Tema 2: Equivalencia entre modelos de computación

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

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

¿Qué es un modelo de computación?

- Para abordar cuestiones acerca de la computabilidad necesitamos elegir un *modelo de computación*, que nos permita precisar lo que entendemos por *procedimiento efectivo*
- La expresión formal de un algoritmo en un modelo concreto se denomina
  - **Programa**, si el modelo está basado en un *lenguaje de programación* (software) o
  - **Máquina**, (o **autómata**) si el modelo es una representación formal en la que se hace referencia explícita a la gestión de la memoria de trabajo donde se almacenan datos (hardware).

Descripción del modelo

- Debemos describir dos elementos principales:
  - **Especificación:** Definir formalmente dos tipos de elementos:
    - Los **datos de entrada y de salida**: Las estructuras de datos que se manipulan durante la ejecución del programa.
    - Las operaciones básicas permitidas en el modelo: Acciones primitivas, bien definidas, que actúan sobre los datos.
  - **Implementación e Interpretación:**
    - **Sintaxis**: Reglas gramaticales para la construcción, a partir de las operaciones básicas, de los procedimientos definibles en el modelo.
    - **Semántica**: Definición lógico-matemática de las funciones o relaciones que definen los procedimientos entre los datos de entrada y/o de salida.
- Así es posible precisar que un programa es **solución** de un problema concreto y el concepto **función computable**.

Función computable en un modelo de computación

- Supongamos definido un modelo de computación $\mathtt{M}$
- Debemos asociar a cada $\mathtt{M}$–programa $\mathtt{p}$ una función $[\![\mathtt{p}]\!]$ sobre la(s) estructura(s) de dato(s) $\mathbb{D}$

$$[\![\mathtt{p}]\!] : \mathbb{D}- \to \mathbb{D}$$

posiblemente parcial (es posible que $\mathtt{p}$ no pare sobre algún dato de entrada), definida como sigue: para cada $d \in \mathbb{D}$,

$$[\![\mathtt{p}]\!](d) = \textit{ resultado de ejecutar } \mathtt{p} \textit{ con entrada } d$$

- **Definición:** Una función $f : \mathbb{D}- \to \mathbb{D}$ es $\mathtt{M}$–computable si existe un $\mathtt{M}$–programa $\mathtt{p}$ tal que $[\![\mathtt{p}]\!] = f$.

Modos de computación

Dependen de la semántica y del concepto de solución.

1. **Modo secuencial-determinista:** La ejecución de un programa sobre un dato es única. La solución de un problema es *la solución* lógica *exacta* para cada entrada.
2. **Modo paralelo:** El programa especifica diversas tareas que se pueden ejecutar en procesadores distintos, compartiendo información. Solución *exacta*.
3. **Modo no determinista.** La ejecución no es única: existen instrucciones que pueden elegir entre 2 o más opciones.
   - Un programa resuelve un problema si alguna de sus posibles ejecuciones ofrece una solución exacta para el dato de entrada.
4. **Modo probabilista:** Los programas son deterministas, pero contienen instrucciones que se ejecutan sólo con cierta probabilidad. Un programa resuelve un problema si, con cierta probabilidad, obtiene una solución exacta.

Características fundamentales de un modelo

La características fundamentales de un modelo de computación que deben ser estudiadas son:

1. **Potencia del modelo:** Qué problemas puede resolver, y analizar su completitud computacional.
2. **Formalización del modelo:** Interpretabilidad del modelo con respecto a otros, aspectos de complejidad sintáctica, otras semánticas, etc.

El análisis de Turing (I)

- Para obtener una formalización aceptable de la idea de procedimiento efectivo de cálculo, podemos intentar aislar las características generales compartidas por todos los procedimientos de cálculo finito.
- Este fué el camino elegido por Turing para el desarrollo de su modelo de computación.
- Turing procedió aislando las condiciones generales bajo las que trabaja un ser humano (idealizado) que lleva a cabo un cálculo con lápiz y papel siguiendo algún algoritmo prefijado.
- Dicho cálculo consiste esencialmente en la manipulación de símbolos que son escritos sobre el papel y esta manipulación está sujeta a varias condiciones de finitud y de proximidad:

El análisis de Turing (II)

1. El número el símbolos considerados debe ser finito así como el espacio utilizado (papel) en cada momento.
   - Podemos considerar que se trabaja sobre una cinta de papel dividida en casillas, cada una de las cuales puede albergar un único símbolo.
2. El procedimiento seguido durante el cálculo puede descomponerse en operaciones simples que no es posible descomponer en otras más sencillas. Éstas son:
   (a) Cambiar el contenido de una de las casillas observadas (cuyo número no debe superar una cierta cota fija), y
   (b) Desplazar la atención desde el grupo de casillas actualmente observado a otro grupo de casillas (situado a una distancia acotada de antemano).
3. La operación realizada en cada momento sólo depende el estado mental del ser humano que realiza el cálculo y de los símbolos que está observando en ese momento. El número de estados distintos ha de ser finito.

Máquinas de Turing

- Son descripciones abstractas de dispositivos mecánicos muy simples que permiten el procesamiento de **cadenas de símbolos** de un alfabeto finito.
- Permiten tomar en cuenta aspectos de una computación:
  1. La manipulación efectiva de números (cálculo) conlleva alguna forma de **representación** de los números mediante sucesiones finitas de símbolos (cifras o dígitos). Un algoritmo trabaja directamente con estas cadenas de símbolos (y no con los números).
  2. La computación consume dos importantes recursos: tiempo y espacio (memoria).
     - El tiempo puede ser identificado con la longitud de la computación (número de operaciones básicas realizadas) siempre que estas operaciones tengan tiempos de ejecución muy parecidos.
     - El espacio se corresponde con el tamaño total de los datos manejados por el algoritmo, y dicho tamaño depende de la representación elegida.

Alfabetos y lenguajes (I)

- Un **alfabeto**, Σ, es un conjunto no vacío finito, cuyos elemento llamamos símbolos.
- Una **palabra** sobre un alfabeto Σ es una sucesión finita de elementos de Σ. Denotaremos por ε la palabra vacía.
- Denotamos por Σ^* al conjunto de las palabras sobre el alfabeto Σ.
  - Formalmente, si $v \in \Sigma^*$ entonces $v : \{1, \ldots, n\} \to \Sigma$, para algún $n \in \mathbb{N}$, es decir, $v \in \Sigma^n$. Por ello,

$$\Sigma^* = \bigcup_{n \in \mathbb{N}} \Sigma^n, \quad \text{siendo } \Sigma^0 = \{\varepsilon\}$$

  - Si $v \in \Sigma^n$ y para cada $j : 1 \leq j \leq n$ se tiene $v(j) = a_j \in \Sigma$, escribiremos $v = a_1 \ldots a_n$.

Alfabetos y lenguajes (I)

- Si $v, w \in \Sigma^*$ y $v = a_1 \cdots a_n$, $w = b_1 \cdots b_m$ entonces la concatenación de v y w es la palabra

$$vw = a_1 \cdots a_n b_1 \cdots b_m$$

- La longitud de $v = a_1 \cdots a_n$ es $|v| = n$ y obviamente $|vw| = |v| + |w|$. Por definición $|\varepsilon| = 0$
- Un **lenguaje** sobre un alfabeto Σ es un subconjunto $L \subseteq \Sigma^*$.
- Las máquinas de Turing permiten definir funciones de Σ^* en Σ^* y también "reconocer" lenguajes, es decir, determinar si una palabra $v \in \Sigma^*$ pertenece o no a un cierto lenguaje $L \subseteq \Sigma^*$.

Máquinas de Turing deterministas

Una máquina de Turing (determinista) M, consta de

1. Un conjunto finito, Q, cuyos elementos se denominan **estados**.
2. Un alfabeto Σ (denominado **alfabeto de entrada**).
3. Un alfabeto Γ tal que $\Sigma \cup \{B\} \subseteq \Gamma$, ($B \notin \Sigma$), denominado **alfabeto de trabajo o de cinta**).
4. $q_0 \in Q$ (se denomina **estado inicial**).
5. $F \subseteq Q$ (sus elementos se llaman **estados finales o de aceptación**).
6. Un conjunto finito $P \subseteq Q \times \Gamma \times (\Gamma \cup \{R, L\}) \times Q$ (con $R, L \notin \Gamma$) tal que:
   Para todo $q \in Q$ y $s \in \Gamma$,

   $$(q, s, t_1, q_1) \in P \text{ y } (q, s, t_2, q_2) \in P \Longrightarrow t_1 = t_2 \text{ y } q_1 = q_2$$

Máquinas de Turing no deterministas

Si en la definición anterior eliminamos la última condición decimos que se trata de una máquina de Turing **no determinista**.

Computaciones de una máquina de Turing (I)

Sea M una máquina de Turing (determinista o no), con programa P.

- Un configuración (o descripción instantánea) de M es un elemento $d = (v, q, w) \in \Gamma^* \times Q \times \Gamma^*$ con $w \neq \varepsilon$.
  - Adicionalmente exigimos que B no sea el primer símbolo de v ni el último de w (salvo cuando $w = B$).
- Una configuración de M, (v, q, su) es de **parada** si no existen $t \in \Gamma$ y $q' \in Q$ tales que $(q, s, t, q') \in P$.
- Una configuración de parada (v, q, w) es de **aceptación** si $q \in F$.
- Una **computación de** M es una sucesión (finita o no) de configuraciones de M, $d_1, \ldots, d_r, \ldots$ tal que para cada $j = 1, \ldots, r-1, \ldots,$ d_{j+1} es una d.i. sucesora de d_j con respecto a M ($d_j \vdash_M d_{j+1}$).
- Una computación finita $d_1, \ldots, d_r$ es una **computación de aceptación** si d_r es una configuración de aceptación.

Computaciones de una máquina de Turing (II)

Sean d y d' configuraciones de M tal que d no es de parada. Decimos que d' es la sucesora de d respecto de M, $d \vdash_M d'$, si $d = (v, q, sx)$ $(s \in \Gamma)$, $d' = (v', q', w')$ y se verifica que:

- $(q, s, t, q') \in P$ con $t \in \Gamma$, $v' = v$ y $w' = tx$.
- $(q, s, R, q') \in P$, $v' = vs$ y $w' = x$ (si $x \neq \varepsilon$) o $w' = B$ (si $x = \varepsilon$).
- $(q, s, L, q') \in P$, $v' = u$ y $w' = rsx$ (si $v = ur$, para algún $r \in \Sigma \cup \{B\}$) o bien $v' = \varepsilon$ y $w' = Bsx$ (si $v = \varepsilon$).

Lenguaje aceptado y función calculada por una MT

Dada una máquina de Turing M definimos

- Si M es determinista, la función de aridad n calculada por M es $f : (\Sigma^*)^n - \to \Sigma^*$ definida como sigue: Dada $(v_1, \ldots, v_n) \in (\Sigma^*)^n$,
  - $f(v_1, \ldots, v_n) = y$ si existe un computación finita $d_1, \ldots, d_r$ de P tal que d_r es de parada

$$d_1 = (\varepsilon, q_0, Bv_1B \ldots Bv_n) \text{ y } d_r = (\varepsilon, q, By)$$

  - En caso contrario, $f(v_1, \ldots, v_n) \uparrow$.
- El lenguaje aceptado por M es el conjunto $L(M) \subseteq \Sigma^*$ tal que para todo $v \in \Sigma^*$,

$$v \in L(M) \iff \begin{cases} \text{existe un computación de aceptación de } M \\ d_1, \ldots, d_r, \text{ con } d_1 = (\varepsilon, q_0, Bv) \end{cases}$$

Lenguaje aceptado y función calculada por una MT

- Una función $f : (\Sigma^*)^n - \rightarrow \Sigma^*$ es **Turing–computable** si existe una máquina de Turing determinista que la calcula.
- Un lenguaje $A \subseteq \Sigma^*$ es **recursivamente enumerable** si existe una máquina de Turing M tal que $A = L(M)$.
- Un lenguaje $A \subseteq \Sigma^*$ es **recursivo** si existe una máquina de Turing M tal que $A = L(M)$ y M para sobre toda palabra de Σ^*. Es decir, para toda $v \in \Sigma^*$, existe una computación de M, $d_1, \ldots, d_r$, donde $d_1 = (\varepsilon, q_0, Bv)$ y d_r es de parada.
  - En las condiciones anteriores se dice que L es el lenguaje **decidido** por M, o también, que M **decide** L.

Otros modelos de computación

- El análisis de Turing es la base del modelo de computación propuesto por Turing en 1936: las máquinas de Turing.
- Desde entonces se han propuesto muchos otros modelos de computación basados en distintas perspectivas del proceso de computación:
  1. λ–cálculo: la computación se lleva a cabo aplicando reglas que permiten reescribir expresiones hasta su forma más simple.
  2. Funciones computables en el sentido de Herbrand–Godel: las funciones quedan definidas mediante sistemas de ecuaciones.
  3. Diversos lenguajes de computación (GOTO, WHILE,...)
  4. Máquinas de registros ilimitados (URM).
  5. Algoritmos de Markov ...

La tesis de Church–Turing

- La tesis de Church–Turing afirma que la clase de las funciones numéricas cuyos valores pueden calcularse siguiendo algún algoritmo coincide con la clase de las funciones Turing–computables (o recursivas).
- Esta tesis sostiene que el concepto **informal** "función computable mediante un algoritmo" coincide con un concepto **formal** "función recursiva".
- Por tanto, esta afirmación no puede ser demostrada formalmente, sólo podemos validarla en la práctica demostrando que todos los modelos de computación propuestos son equivalentes.
- Esto es lo que ha ocurrido hasta la fecha, lo que muestra que la noción de función computable es muy estable y, a la vez, es interpretado por muchos investigadores como un respaldo a la tesis.

Máquinas de Turing (descripción alternativa)

Una máquina de Turing M (con función de transición) consta de:

1. Un conjunto finito, Q, cuyos elementos de llaman **estados**.
2. Un alfabeto Σ (denominado **alfabeto de entrada**).
3. Un alfabeto Γ con $\Sigma \cup \{B\} \subseteq \Gamma$ (denominado **alfabeto de trabajo**).
4. $q_0 \in Q$ (se denomina **estado inicial**).
5. $F \subseteq Q$ (sus elementos se llaman **estados finales**).
6. Una función **parcial** δ de $Q \times \Gamma$ en $Q \times \Gamma \times \{L, R\}$, llamada **función de transición**.

Computaciones de una máquina de Turing (II)

Sean d y d' configuraciones de una MT (con función de transición, δ) tal que $d = (v, q, sx)$ no es de parada. Decimos que d' es la sucesora de d respecto de M, $d \vdash_M d'$, si $d = (v, q, sx)$ $(s \in \Gamma)$, $d' = (v', q', w')$ y se verifica que:

- $\delta(q, s) = (q', t, R)$, $v' = vt$ y $w' = x$ (si $x \neq \varepsilon$) o $w' = B$ (si $x = \varepsilon$).
- $\delta(q, s) = (q', t, L)$, $v' = u$ y $w' = rtx$ (si $v = ur$, para algún $r \in \Sigma \cup \{B\}$) o bien $v' = \varepsilon$ y $w' = Btx$ (si $v = \varepsilon$).

Utilizando esta noción de configuración sucesora podemos definir las nociones de lenguaje aceptado y función calculada para estas máquinas de Turing.

Programas de Post–Turing

Consideremos el siguiente lenguaje de programación (formal), $\mathcal{T}$, para el cálculo con palabras sobre un alfabeto finito Σ (aunque el alfabeto de trabajo del lenguaje contendrá un símbolo extra $B \notin \Sigma$).

- Un programa de $\mathcal{T}$, es una sucesión finita de instrucciones $I_1, \ldots, I_n$.
- Instrucciones (algunas pueden estar etiquetadas):
  - Para cada símbolo $s \in \Sigma \cup \{B\}$ y cada etiqueta L, la instrucción

    $$\text{IF } s \text{ GOTO } L$$

  - Para cada símbolo $s \in \Sigma \cup \{B\}$, la instrucción

    $$\text{PRINT } s$$

  - Instrucciones de desplazamiento: RIGHT y LEFT.

Programas de Post–Turing (II)

- Un programa controla un dispositivo que inspecciona una cinta infinita por ambos extremos, dividida en celdas.
- En cada momento la cabeza lectora puede leer el contenido de una celda (un símbolo de $\Sigma \cup \{B\}$).
  - El dispositivo puede sobreescribirlo (PRINT), moverse a una celda adyacente (RIGHT, LEFT) o cambiar de instrucción (IF s GOTO L).
- Para calcular una función $f : \Sigma^* \times \overset{(n)}{\cdots} \times \Sigma^* - \ \to \Sigma^*$, dada la n–tupla $(v_1, \ldots, v_n)$ escribimos en la cinta

$$B\, v_1\, B\, v_2\, B \cdots B\, v_n\, B$$

  con la cabeza lectora en la primera celda de la cadena.
- Y a continuación ...

Programas de Post–Turing (II)

- El programa ejecuta secuencialmente cada instrucción pasando a la siguiente.
- En el caso de una instrucción IF s GOTO A, si la cabeza lectora lee el símbolo s, se pasa a la primera instrucción etiquetada por A (si existe). En caso contrario se pasa a la siguiente instrucción (si existe).
- El programa se detiene si el proceso de ejecución descrito le envía a una instrucción inexistente.
- Si $f(v_1, \ldots, v_n) = x$, al detenerse el programa, el contenido de la cinta debe ser Bx y la cabeza lectora debe estar en la celda marcada con B.

Ejemplos (I)

Consideremos el alfabeto $\Sigma = \{a, b\}$.

1. El siguiente programa calcula la función $f : \Sigma^* \to \Sigma^*$, $f(v) = abv$.

```
PRINT b
LEFT
PRINT a
LEFT
```

2. El siguiente programa calcula la función $f : \Sigma^* \to \Sigma^*$, $f(v) = avb$.

```
      PRINT a
[A1]  RIGHT
      IF a GOTO A1
      IF b GOTO A1
      PRINT b
[A2]  LEFT
      IF a GOTO A2
      IF b GOTO A2
```

Ejemplos (II)

- El siguiente programa calcula la función $f : \Sigma^* \times \Sigma^* \to \Sigma^*$ que concatena dos palabras de Σ:

$$f(v, w) = vw$$

```
[A]    RIGHT
       IF a GOTO A
       IF b GOTO A
[C1]   RIGHT
       IF a GOTO C2
       IF b GOTO C3
       IF B GOTO D
[C2]   PRINT B
       LEFT
       PRINT a
       RIGHT
       IF B GOTO C1
[C3]   PRINT B
       LEFT
       PRINT b
       RIGHT
       IF B GOTO C1
[D]    LEFT
       IF a GOTO D
       IF b GOTO D
```

Semántica de $\mathcal{T}$ (I)

Dado un programa P de Post–Turing veamos como definir de manera precisa la función de aridad $n \geq 1$ calculada por P.

- Una **configuración de cinta** es un par $(v, w) \in (\Sigma \cup \{B\})^* \times (\Sigma \cup \{B\})^*$, con $w \neq \varepsilon$.
- Un **descripción instantánea** para P es un par (j, c) donde ($|P|$ es el número de instrucciones de P):
  - $j \in \mathbb{N}$, verifica $1 \leq j \leq |P| + 1$, y
  - c es una configuración de cinta.
- Una descripción instantánea $d = (j, c)$ es terminal si $j = |P| + 1$.
- Una computación de P es una sucesión $d_1, \ldots, d_r, \ldots$ de descripciones instantáneas para P tal que para cada $i = 1, \ldots, r-1, \ldots$, $d_i \vdash_P d_{i+1}$, es decir, d_{i+1} es la descripción instantanea sucesora de d_i con respecto a P (ver la definición a continuación).

Semántica de $\mathcal{T}$ (II)

Sean d y d' descripciones instantáneas para $P = I_1, \ldots, I_m$ tales que d no es terminal. Decimos que d' es la sucesora de d respecto de P, $d \vdash_P d'$, si $d = (j, (v, w))$, $d' = (j', (v', w'))$ y se verifica que:

- Si I_j es PRINT s y $w = tu$ para cierto $t \in \Sigma \cup \{B\}$, entonces $j' = j + 1$, $v' = v$ y $w' = su$.
- Si I_j es RIGHT y $w = tu$ para cierto $t \in \Sigma \cup \{B\}$, entonces $j' = j + 1$, $v' = vt$ y $w' = u$ (si $u \neq \varepsilon$) y $w' = B$ (si $u = \varepsilon$).
- Si I_j es LEFT, entonces $j' = j + 1$, $v' = u$ y $w' = tw$ (si $v = ut$, para algún $t \in \Sigma \cup \{B\}$) o bien $v = \varepsilon$ y $w' = Bw$ (si $v = \varepsilon$).
- Si I_j es IF s GOTO L entonces $v' = v$, $w' = w$ y j' es: el menor i tal que I_i está etiquetada con L (si existe) o $j' = |P| + 1$ en otro caso.

Semántica de $\mathcal{T}$ (III)

Dado un programa P de Post–Turing la función de aridad n calculada por P es la función $f : (\Sigma^*)^n - \to \Sigma^*$ definida como sigue:
Dada $(v_1, \ldots, v_n) \in (\Sigma^*)^n$,

- $f(v_1, \ldots, v_n) = y$ si existe un computación finita $d_1, \ldots, d_r$ de P tal que

$$d_1 = (1, (\varepsilon, Bv_1B \ldots Bv_n)) \text{ y } d_r = (|P| + 1, (\varepsilon, By))$$

- En caso contrario, $f(v_1, \ldots, v_n) \uparrow$.

Equivalencia de modelos

- **Teorema**: Sea Σ un alfabeto, $\Gamma = \Sigma \cup \{B\}$ y $f : (\Sigma^*)^n - \rightarrow \Sigma^*$. Son equivalentes:
  1. f es computable mediante un programa de Post–Turing.
  2. f es Turing–computable (con alfabeto de cinta Γ).
  3. f es computable mediante una MT con función de transición (y alfabeto de cinta Γ).
- (1. $\Rightarrow$ 2.) Sea P un programa que calcula f. Sea M la máquina de Turing que tiene un estado q_j por cada instrucción I_j de P, Γ como lenguaje de cinta y su programa contiene las siguientes tuplas
  - (q_j, t, s, q_{j+1}) para cada $t \in \Sigma$, si I_j es PRINT s.
  - (q_j, t, L, q_{j+1}) para cada $t \in \Sigma$, si I_j es LEFT.
  - (q_j, t, R, q_{j+1}) para cada $t \in \Sigma$, si I_j es RIGHT.
  - (q_j, s, s, q_k) para cada si I_j es IF s GOTO L y k es el menor número tal que I_k está etiquetada con L, si existe alguna, y $k = |P| + 1$ en caso contrario.

  Entonces, tomando q_1 como estado inicial, M calcula f.

Equivalencia de modelos (cont.)

- (2. $\Rightarrow$ 3.) Si M calcula f (y tiene programa P), sea M' la MT que tiene como función de transición la siguiente:
  - $\delta(q, s) = (q', s, R)$, si $(q, s, R, q') \in P$.
  - $\delta(q, s) = (q', s, L)$, si $(q, s, L, q') \in P$.
  - Si $t \in \Gamma$ y $(q, s, t, q') \in P$, añadimos un nuevo estado $\hat{q}_t$ y definimos

$$\delta(q, s) = (\hat{q}_t, t, R), \text{ y}$$

$$\delta(\hat{q}_t, r) = (q', r, L), \text{ para cada } r \in \Gamma$$

  Es directo comprobar que M' calcula f.

Equivalencia de modelos (cont.)

- (3. $\Rightarrow$ 1.) Sea M una MT, con función de transición δ, que calcula f. Enumeremos los estados de M y el alfabeto Γ:

$$Q = \{q_0, q_1, \ldots, q_K\}, \quad \Gamma = \{s_0, \ldots, s_n\}$$

Asociemos a cada q_j una etiqueta S_j y a cada par (q_i, s_j) otra etiqueta $F_{i,j}$. Sea P el programa de Post–Turing que empieza con bloques de instrucciones consecutivos ($i = 0, \ldots, K$):

$$\begin{array}{ll} [S_i] & \text{IF } s_0 \text{ GOTO } F_{i,0} \\ & \vdots \\ & \text{IF } s_n \text{ GOTO } F_{i,n} \end{array}$$

y continúa con los bloques ($i = 0, \ldots, K; j = 0, \ldots, n$)

Equivalencia de modelos (cont.)

- Si $\delta(q_i, s_j) = (q_h, s_k, R)$,

$$[F_{i,j}] \quad \begin{array}{l} \text{PRINT } s_k \\ \text{RIGHT} \\ \text{GOTO } S_h \end{array}$$

- Si $\delta(q_i, s_j) = (q_h, s_k, L)$,

$$[F_{i,j}] \quad \begin{array}{l} \text{PRINT } s_k \\ \text{LEFT} \\ \text{GOTO } S_h \end{array}$$

Entonces la función de aridad n calculada por P es f.

Procesos y gramáticas

- Hasta ahora sólo hemos considerado el problema de **reconocer** si una palabra sobre un alfabeto σ pertenece o no a un lenguaje L.
- Ahora pasaremos a considerar el problema de **generar** automáticamente palabras de un lenguaje dado.
- Para ello necesitaremos algún modo de hacer explícita la forma "sintáctica" común a todas estas palabras.
- Las gramáticas formales proporcionan una herramienta para esta tarea.

Procesos semi–Thue

- Una **producción** (o **regla de reescritura**) sobre un alfabeto Σ es un par de palabras sobre Σ, $(g, \bar{g})$.
  - La regla $(g, \bar{g})$ se representará por $g \to \bar{g}$.
- Si P es un producción $g \to \bar{g}$, y $u, v \in \Sigma^*$, escribiremos $u \Rightarrow_P v$ para expresar que

$$\text{Existen } r, s \in \Sigma^* \text{ tales que } u = rgs \text{ y } v = r\bar{g}s$$

- Un **proceso semi–Thue** es un conjunto finito de producciones.
- Si Φ es un proceso semi–Thue y $u, v \in \Sigma^*$ escribimos:
  - $u \Rightarrow_\Phi v$ si existe $P \in \Phi$ tal que $u \Rightarrow_P v$.
  - $u \overset{\star}{\Rightarrow}_\Phi v$ si existe una sucesión $w_1, \ldots, w_n \in \Sigma^*$ tal que $w_1 = u$, $w_n = v$ y

$$\text{para cada } j = 1, \ldots, n-1, \quad w_j \Rightarrow_\Phi w_{j+1}$$

(decimos que $w_1, \ldots, w_n$ es una derivación de v a partir de u.

Ejemplo

- Si $\Phi = \{ab \to aa,\ ba \to bb\}$, entonces

$$aba \Rightarrow abb \Rightarrow aab \Rightarrow aaa$$

- La sucesión aba, abb, aab, aaa es una derivación de aaa a partir de aba

Gramáticas

- Una **gramática** es un proceso semi–Thue en el que los símbolos del alfabeto se han dividido en dos clases: **terminales** y **no terminales** (o variables), y existe un símbolo no terminal distinguido que llamamos **símbolo inicial**.
- Una gramática $\Gamma = \langle V, T, \Phi, S \rangle$ está determinada por:
  - Un alfabeto $\Sigma = V \cup T$ con $V \cap T = \emptyset$ (V es el conjunto de variables, T el de símbolos terminales)
  - Un símbolo $S \in V$ (símbolo inicial).
  - Un conjunto de reglas de reescritura, Φ.
- El **lenguaje generado** por la gramática $\Gamma = \langle V, T, \Phi, S \rangle$ es

$$L(\Gamma) = \{u \in T^* : \ S \stackrel{\star}{\Rightarrow}_{\Phi} u\}$$

Ejemplos

- Si Γ_1 es la gramática dada por:

$$S \to aRb, \quad R \to aRb, \quad R \to ab, \quad R \to \varepsilon$$

  Entonces $L(\Gamma_1) = \{a^n b^n : \ n \geq 1\}$

- Si Γ_2 es la gramática dada por:

$$S \to aRa, \quad S \to bRb, \quad R \to aRa, \quad R \to bRb$$

$$S \to \varepsilon, \quad R \to \varepsilon$$

  Entonces $L(\Gamma_2) = \{ww^R : \ w \in \{a, b\}^*\}$

Gramáticas y lenguajes

- **Teorema**: Un lenguaje es aceptado por una máquina de Turing si y sólo si es generado por alguna gramática.

Máquinas de Turing con cintas semi-infinitas

Una máquina de Turing de una cinta semi-infinita consta de:

1. Un conjunto finito, Q, de **estados**.
2. Un alfabeto Σ (denominado **alfabeto de entrada**).
3. Un alfabeto Γ con $\Sigma \cup \{\triangleright, B\} \subseteq \Gamma$ (denominado **alfabeto de trabajo**) siendo $\triangleright \neq B$.
   - Los símbolos $\triangleright$ y B se denominan, repectivamente, **marcador de inicio** y **blanco**
4. $q_0 \in Q$ (se denomina **estado inicial**).
5. Dos estados $q_A, q_R \in Q$, $q_A \neq q_R$.
   - q_A es el estado de **aceptación** y q_R es el de **rechazo**.
6. Una función $\delta : (Q - \{q_A, q_R\}) \times \Gamma \to Q \times \Gamma \times \{L, N, R\}$, llamada **función de transición**.
   - Exigimos que si $\delta(q, \triangleright) = (p, \sigma, D)$ entonces $\sigma = \triangleright$ y $D = R$.

Computaciones de una máquina determinista (I)

- Una configuración de una MT de un cinta es un triple $d = (q, v, w) \in Q \times \Gamma^* \times \Gamma^*$.
  - Se representa con ella que el contenido de la cinta es la palabra vw y que la cabeza lectora se encuentra en el primer símbolo de w (si $w = \varepsilon$, la cabeza lee B).
- Sean d y d' configuraciones tales que $d = (q, v, sw)$ con $s \in \Gamma$, $q \neq q_A$ y $q \neq q_R$. Decimos que $d' = (q', v', w')$ es la **sucesora** de d respecto de M, $d \vdash_M d'$, cuando
  - Si $\delta(q, s) = (q', t, N)$, entonces $v' = v$ y $w' = tw$.
  - Si $\delta(q, s) = (q', t, R)$, entonces $v' = vt$ y $w' = w$.
  - Si $\delta(q, s) = (q', t, L)$, entonces $v' = u$ y $w' = rtw$ (si $v = ur$, para algún $r \in \Gamma$).
- Si $d = (q, v, \varepsilon)$, entonces $d \vdash_M d'$ si $(q, w, B) \vdash d'$.
- Decimos que una configuración de M, $d = (q, v, sw)$ es de **parada** si $q = q_A$ o bien $q = q_R$.
  - Si $q = q_A$, d es una configuración de **aceptación**.
  - Si $q = q_R$ decimos que es de **rechazo**.

Computaciones de una máquina determinista (II)

Sea $M = \langle Q, \Sigma, \Gamma, q_0, q_A, q_R, \delta \rangle$ una MT con una cinta.

- Dada $x \in \Sigma^*$ la computación de M sobre x es una sucesión (posiblemente infinita) de configuraciones de M, d_0, d_1, $d_2, \ldots$ tal que:
  - $d_0 = (q_0, \triangleright, x)$.
  - Para cada i, si $d_i = (q, v, w)$ no es de parada entonces, d_{i+1} está definida y $d_i \vdash_M d_{i+1}$.

Máquinas de Turing con varias cintas (II)

Una máquina de Turing de k cintas consta de:

1. Un conjunto finito, Q, de **estados**.
2. Un alfabeto Σ (denominado **alfabeto de entrada**).
3. Un alfabeto Γ con $\Sigma \cup \{\triangleright, B\} \subseteq \Gamma$ (denominado **alfabeto de trabajo**) siendo $\triangleright \neq B$.
4. $q_0 \in Q$ (se denomina **estado inicial**).
5. Dos estados $q_A, q_R \in Q$, $q_A \neq q_R$.
6. Una función $\delta : (Q - \{q_A, q_R\}) \times \Gamma^k \to Q \times \Gamma^k \times \{L, N, R\}^k$, llamada **función de transición**.
   - Exigimos que si

$$\delta(q, \sigma_1, \ldots, \sigma_k) = (p, \tau_1, \ldots, \tau_k, D_1, \ldots, D_k)$$

y para algún j, $\sigma_j = \triangleright$, entonces $D_j = R$ y $\tau_j = \triangleright$.

Computaciones de una MT con k cintas

- Una configuración de una MT con k cintas es una $(2k+1)$–tupla $d = (q, v_1, w_1, \ldots, v_k, w_k) \in Q \times (\Gamma^*)^{2k}$.
  - Se representa con ella que el contenido de la j–ésima cinta es la palabra $v_j w_j$ y que la cabeza lectora se encuentra en el primer símbolo de w_j (si $w_j = \varepsilon$, la cabeza lee B).
- Sean d y d' configuraciones tales que $d = (q, v_1, s_1 w_1, \ldots, v_k, s_k w_k)$ con $q \neq q_A$ y $q \neq q_R$. Decimos que $d' = (q', v_1', w_1', \ldots, v_k', w_k')$ es **la sucesora de** d respecto de M, $d \vdash_M d'$, si se verifica que si $\delta(q, s_1, \ldots, s_k) = (q', t_1, \ldots, t_k, D_1, \ldots, D_k)$ entonces para cada $j = 1, \ldots, k$,
  - Si $D_j = N$, entonces $v_j' = v_j$ y $w_j' = t_j w_j$.
  - Si $D_j = R$, entonces $v_j' = v_j t_j$ y $w_j' = w_j$.
  - Si $D_j = L$, entonces $v_j' = u_j$ y $w_j' = r_j t_j w_j$ (si $v_j = u_j r_j$, para algún $r_j \in \Gamma$).
- Si $d = (q, v_1, w_1, \ldots, v_k, w_k)$ con alguna $w_j = \varepsilon$ procedemos como en el caso de una sola cinta.

Computaciones de una MT con k cintas (II)

Sea $M = \langle Q, \Sigma, \Gamma, q_0, q_A, q_R, \delta \rangle$ un MT con k cintas.

- Las nociones de configuración de parada, rechazo y aceptación se definen como en el caso de una sóla cinta.
- Dada $x \in \Sigma^*$ la computación de M sobre x es una sucesión (posiblemente infinita) de configuraciones de M, d_0, d_1, $d_2, \ldots$ tal que:
  - $d_0 = (q_0, \triangleright, x, \triangleright, \varepsilon, \ldots, \triangleright, \varepsilon)$.
  - Para cada i, si d_i no es de parada entonces, d_{i+1} está definida y $d_i \vdash_M d_{i+1}$.
- Una **computación de parada** es una computación finita $d_0, d_1, \ldots, d_n$. Si $d_n = (q, v_1, w_1, \ldots, v_k.w_k)$ diremos que la computación es de **aceptación** si $q = q_A$ y de **rechazo** si $q = q_R$.
- Sea $L \subseteq \Sigma^*$. Decimos que una MT, M, **decide** L si para cada $w \in \Sigma^*$ se tiene:
  - Si $w \in L$ entonces la computación de M sobre w es de aceptación.
  - Si $w \notin L$ entonces la computación de M sobre w es de rechazo.

Máquinas no deterministas con varias cintas

Una máquina de Turing no determinista con k cintas (semiinfinitas) consta de:

1. Un conjunto finito, Q, de **estados**.
2. Un alfabeto Σ (denominado **alfabeto de entrada**).
3. Un alfabeto Γ con $\Sigma \cup \{\triangleright, B\} \subseteq \Gamma$ (denominado **alfabeto de trabajo**) siendo $\triangleright \neq B$.
4. $q_0 \in Q$ (se denomina **estado inicial**).
5. Dos estados $q_A, q_R \in Q$, $q_A \neq q_R$.
6. Una función $\delta : (Q - \{q_A, q_R\}) \times \Gamma^k \to \mathcal{P}(Q \times \Gamma^k \times \{L, N, R\}^k)$, llamada **función de transición**.
   - Para cada $(q, \sigma_1, \ldots, \sigma_k) \in (Q - \{q_A, q_R\}) \times \Gamma^k$, exigimos que
     - $\delta(q, \sigma_1, \ldots, \sigma_k) \neq \emptyset$ y
     - Si se tiene

       $$(p, \tau_1, \ldots, \tau_k, D_1, \ldots, D_k) \in \delta(q, \sigma_1, \ldots, \sigma_k)$$

       y para algún j, $\sigma_j = \triangleright$, entonces $D_j = R$ y $\tau_j = \triangleright$.

Computaciones de una máquina no determinista

- Como en el caso determinista, una configuración de una MT no determinista con k cintas es una $(2k+1)$–tupla $d = (q, v_1, w_1, \ldots, v_k, w_k) \in Q \times (\Gamma^*)^{2k}$
- Sean d y d' configuraciones tales que $d = (q, v_1, s_1 w_1, \ldots, v_k, s_k w_k)$ con $q \neq q_A$ y $q \neq q_R$. Decimos que d' es una sucesora de d respecto de M, $d \vdash_M d'$, $d' = (q', v_1', w_1', \ldots, v_k', w_k')$ si existe $(q', t_1, \ldots, t_k, D_1, \ldots, D_k) \in \delta(q, s_1, \ldots, s_k)$ tal que para cada $j = 1, \ldots, k$,
  - Si $D_j = N$, entonces $v_j' = v_j$ y $w_j' = t_j w_j$.
  - Si $D_j = R$, entonces $v_j' = v_j t_j$ y $w_j' = w_j$.
  - Si $D_j = L$, entonces $v_j' = u_j$ y $w_j' = r_j t_j w_j$ (si $v_j = u_j r_j$, para algún $r_j \in \Gamma$).
- Si $d = (q, v_1, w_1, \ldots, v_k, w_k)$ con alguna $w_j = \varepsilon$ procedemos como en el caso determinista.
- Las nociones de configuración de parada, rechazo y aceptación se definen como en el caso determinista.

Computaciones de una máquina no determinista (II)

Sea $M = \langle Q, \Sigma, \Gamma, q_0, q_A, q_R, \delta \rangle$ un MT no determinista con k cintas.

- Dada $x \in \Sigma^*$ la computación de M sobre x es una sucesión (posiblemente infinita) de configuraciones de M, d_0, d_1, $d_2, \ldots$ tal que:
  - $d_0 = (q_0, \triangleright, x, \triangleright, \varepsilon, \ldots, \triangleright, \varepsilon)$.
  - Para cada i, si d_i no es de parada entonces, d_{i+1} está definida y $d_i \vdash_M d_{i+1}$.
- Sea $L \subseteq \Sigma^*$. Decimos que una máquina de Turing no determinista, M, **decide** L, si para cada $w \in \Sigma^*$ se tiene:
  - Si $w \in L$ entonces existe una computación de M sobre w que es de aceptación.
  - Si $w \notin L$ entonces toda computación de M sobre w es de rechazo

Máquinas con entrada y salida

- Dado $k > 2$, una máquina de Turing (determinista o no) de k cintas con entrada y salida es una máquina de Turing con k cintas tal que:
  Si $\delta(q, s_1, \ldots, s_k) = (q', t_1, \ldots, t_k, D_1, \ldots, D_k)$
  - (En el caso no determinista, $(q', t_1, \ldots, t_k, D_1, \ldots, D_k) \in \delta(q, s_1, \ldots, s_k)$)

  entonces
  1. $s_1 = t_1$.
  2. $D_k \neq L$.
  3. Si $s_1 = B$ entonces $D_1 = L$.
- La máquina M tiene un estado de parada (adicional) h.
- Decimos que $f : \Sigma^* \to \Sigma^*$ es computable por M si para cada $x \in \Sigma^*$ existe una computación finita de M en la que x es el contenido de la cinta 1 en la configuración inicial y $f(x)$ es el contenido de la k–ésima cinta en la configuración final $d = (h, w_1, ..., w_{k-1}, f(x))$.

Complejidad en tiempo

Sea $f : \mathbb{N} \to \mathbb{N}$, $L \subseteq \Sigma^*$ y M una máquina de Turing (determinista o no) con alfabeto de entrada Σ.

- Dada $x \in \Sigma^*$ la longitud de una computación de M sobre x de define como:
  - ∞, si es una computación infinita.
  - n, si es una computación finita $d_0, d_1, \ldots, d_n$.
- Decimos que M **opera en tiempo** $f(n)$ si para toda palabra $x \in \Sigma^*$, todas las computaciones de M sobre x son finitas y sus longitudes menores o iguales que $f(|x|)$.
- Decimos que L es decidido por una máquina de Turing M, **en tiempo** $f(n)$, si M decide L y opera en tiempo $f(n)$.

Complejidad en espacio

Sea $f : \mathbb{N} \to \mathbb{N}$, $L \subseteq \Sigma^*$ y M una máquina de Turing (determinista o no) con alfabeto de entrada Σ, de $k > 2$ cintas y con entrada y salida.

- Dada $x \in \Sigma^*$, el **espacio utilizado** por una computación finita de M sobre x, d_0, $d_1, \ldots, d_n$, se define como

$$\sum_{j=2}^{k-1} |v_j w_j|$$

  siendo $d_n = (q, v_1, w_1, \ldots, v_k, w_k)$.
- Decimos que M **opera en espacio** $f(n)$ si para toda palabra $x \in \Sigma^*$, todas las computaciones de M sobre x son finitas y el espacio utilizado por ellas siempre es menor o igual que $f(|x|)$.
- Decimos que L es **decidido** por una máquina de Turing M, **en espacio** $f(n)$, si M decide L y opera en tiempo $f(n)$.