# Autómatas y Lenguajes Formales

## Tema 15: Lenguajes de programación equivalentes a la Máquina de Turing


Dr. Favio Ezequiel Miranda Perea

favio@ciencias.unam.mx

Facultad de Ciencias UNAM[1]


27 de abril de 2019

[1] Con el apoyo del proyecto PAPIME PE102117

# Un programa en máquinas de Turing

## Inversión de ceros y unos

```
 1: if 1 then goto 2 else goto 5
 2: write 0
 3: right
 4: goto 1
 5: if 0 then goto 6 else goto 9
 6: write 1
 7: right
 8: goto 1
 9: left
10: if ␣ then 11 else 9
11: halt
```

# Un programa no estructurado

## Uso de goto

El siguiente programa `reverse` implementa la función reversa.

```
read X
1: Y := nil;
2: if X then goto 3 else goto 7;
3: Z := hd X;
4: Y := cons Z Y;
5: X := tl X;
6: goto 2;
write Y
```

# Un programa estructurado

## Uso de secuencias y ciclos

El siguiente programa `reverse` implementa la función reversa.

```
read X;
  Y := nil;
  while X do
    Y := cons (hd X) Y;
    X := tl X;
  end
write Y
```

# La máquina de Turing clásica
## $\mathcal{MT}$

$$T = \langle \Sigma, Q, q_0, q_f, \delta \rangle,$$

donde

- $\Sigma \neq \varnothing$ es un alfabeto finito que contiene un símbolo distinguido $\sqcup$, llamado símbolo blanco,

# La máquina de Turing clásica

## $\mathcal{MT}$

$$T = \langle \Sigma, Q, q_0, q_f, \delta \rangle,$$

donde

- $\Sigma \neq \varnothing$ es un alfabeto finito que contiene un símbolo distinguido $\sqcup$, llamado símbolo blanco,
- $Q \neq \varnothing$ es el conjunto finito de estados, el cual incluye $q_0$ y $q_f$,

# La máquina de Turing clásica

$\mathcal{MT}$

$$T = \langle \Sigma, Q, q_0, q_f, \delta \rangle,$$

donde

- $\Sigma \neq \varnothing$ es un alfabeto finito que contiene un símbolo distinguido $\sqcup$, llamado símbolo blanco,
- $Q \neq \varnothing$ es el conjunto finito de estados, el cual incluye $q_0$ y $q_f$,
- $q_0$ es el estado inicial,

# La máquina de Turing clásica

## $\mathcal{MT}$

$$T = \langle \Sigma, Q, q_0, q_f, \delta \rangle,$$

donde

- $\Sigma \neq \varnothing$ es un alfabeto finito que contiene un símbolo distinguido $\sqcup$, llamado símbolo blanco,
- $Q \neq \varnothing$ es el conjunto finito de estados, el cual incluye $q_0$ y $q_f$,
- $q_0$ es el estado inicial,
- $q_f$ es el estado final de aceptación,

# La máquina de Turing clásica

## $\mathcal{MT}$

$$T = \langle \Sigma, Q, q_0, q_f, \delta \rangle,$$

donde

- $\Sigma \neq \varnothing$ es un alfabeto finito que contiene un símbolo distinguido $\sqcup$, llamado símbolo blanco,
- $Q \neq \varnothing$ es el conjunto finito de estados, el cual incluye $q_0$ y $q_f$,
- $q_0$ es el estado inicial,
- $q_f$ es el estado final de aceptación,
- $\delta : Q \times \Sigma \to Q \times \Sigma \times D$, es la función (parcial) de transición.

# La máquina de Turing clásica

## $\mathcal{MT}$

$$T = \langle \Sigma, Q, q_0, q_f, \delta \rangle,$$

donde

- $\Sigma \neq \varnothing$ es un alfabeto finito que contiene un símbolo distinguido $\sqcup$, llamado símbolo blanco,
- $Q \neq \varnothing$ es el conjunto finito de estados, el cual incluye $q_0$ y $q_f$,
- $q_0$ es el estado inicial,
- $q_f$ es el estado final de aceptación,
- $\delta : Q \times \Sigma \to Q \times \Sigma \times D$, es la función (parcial) de transición.
- $D = \{\leftarrow, \to, -\}$ es el conjunto de movimientos

# Lenguajes de programación

## Modelo teórico

Un lenguaje de programación es una terna

$$\mathcal{L} = \langle \mathcal{P}_{\mathcal{L}}, \mathcal{D}_{\mathcal{L}}, [\![\cdot]\!]_{\mathcal{L}} \rangle$$

donde

- $\mathcal{P}_{\mathcal{L}} \neq \varnothing$ es el conjunto de programas de $\mathcal{L}$

# Lenguajes de programación

## Modelo teórico

Un lenguaje de programación es una terna

$$\mathcal{L} = \langle \mathcal{P}_{\mathcal{L}}, \mathcal{D}_{\mathcal{L}}, [\![\cdot]\!]_{\mathcal{L}} \rangle$$

donde

- $\mathcal{P}_{\mathcal{L}} \neq \varnothing$ es el conjunto de programas de $\mathcal{L}$
- $\mathcal{D}_{\mathcal{L}} \neq \varnothing$ es el conjunto de datos (de entrada y salida) de $\mathcal{L}$

# Lenguajes de programación

## Modelo teórico

Un lenguaje de programación es una terna

$$\mathcal{L} = \langle \mathcal{P}_{\mathcal{L}}, \mathcal{D}_{\mathcal{L}}, [\![\cdot]\!]_{\mathcal{L}} \rangle$$

donde

- $\mathcal{P}_{\mathcal{L}} \neq \varnothing$ es el conjunto de programas de $\mathcal{L}$
- $\mathcal{D}_{\mathcal{L}} \neq \varnothing$ es el conjunto de datos (de entrada y salida) de $\mathcal{L}$
- $[\![\cdot]\!]_{\mathcal{L}}$ es la función semántica de $\mathcal{L}$ tal que

$$[\![\cdot]\!]_{\mathcal{L}} : \mathcal{P}_{\mathcal{L}} \to \mathcal{D}_{\mathcal{L}} \rightharpoonup \mathcal{D}_{\mathcal{L}}$$

# Lenguajes de programación

## Modelo teórico

Un lenguaje de programación es una terna

$$\mathcal{L} = \langle \mathcal{P}_{\mathcal{L}}, \mathcal{D}_{\mathcal{L}}, [\![\cdot]\!]_{\mathcal{L}} \rangle$$

donde

- $\mathcal{P}_{\mathcal{L}} \neq \varnothing$ es el conjunto de programas de $\mathcal{L}$
- $\mathcal{D}_{\mathcal{L}} \neq \varnothing$ es el conjunto de datos (de entrada y salida) de $\mathcal{L}$
- $[\![\cdot]\!]_{\mathcal{L}}$ es la función semántica de $\mathcal{L}$ tal que

$$[\![\cdot]\!]_{\mathcal{L}} : \mathcal{P}_{\mathcal{L}} \to \mathcal{D}_{\mathcal{L}} \rightharpoonup \mathcal{D}_{\mathcal{L}}$$

- $[\![\cdot]\!]_{\mathcal{L}}$ es una función que le asocia a cada programa $p \in \mathcal{P}_{\mathcal{L}}$ una función parcial $[\![p]\!]_{\mathcal{L}} : \mathcal{D}_{\mathcal{L}} \rightharpoonup \mathcal{D}_{\mathcal{L}}$.

# Semántica operacional

$\mathcal{L} = \langle \mathcal{P}_\mathcal{L}, \mathcal{D}_\mathcal{L}, [\![\cdot]\!]_\mathcal{L} \rangle$

Una semántica operacional para $\mathcal{L}$ es una tupla

$$\mathcal{O}_\mathcal{L} = \langle \mathcal{S}, \mathcal{E}, \cdot \triangleright \cdot \to \cdot, \mathsf{final}, \mathsf{init}, \mathsf{output} \rangle$$

tal que

- $\mathcal{S} \neq \varnothing$ es el conjunto de memorias

# Semántica operacional

$\mathcal{L} = \langle \mathcal{P}_\mathcal{L}, \mathcal{D}_\mathcal{L}, [\![\cdot]\!]_\mathcal{L} \rangle$

Una semántica operacional para $\mathcal{L}$ es una tupla

$$\mathcal{O}_\mathcal{L} = \langle \mathcal{S}, \mathcal{E}, \cdot \rhd \cdot \to \cdot, \text{final}, \text{init}, \text{output} \rangle$$

tal que

- $\mathcal{S} \neq \varnothing$ es el conjunto de memorias
- $\mathcal{E} \neq \varnothing$ es el conjunto de estados cuya definición involucra usualmente a $\mathcal{S}$.

# Semántica operacional

$\mathcal{L} = \langle \mathcal{P}_{\mathcal{L}}, \mathcal{D}_{\mathcal{L}}, [\![\cdot]\!]_{\mathcal{L}} \rangle$

Una semántica operacional para $\mathcal{L}$ es una tupla

$$\mathcal{O}_{\mathcal{L}} = \langle \mathcal{S}, \mathcal{E}, \cdot \triangleright \cdot \to \cdot, \text{final}, \text{init}, \text{output} \rangle$$

tal que

- $\mathcal{S} \neq \varnothing$ es el conjunto de memorias
- $\mathcal{E} \neq \varnothing$ es el conjunto de estados cuya definición involucra usualmente a $\mathcal{S}$.
- final : $\mathcal{P}_{\mathcal{L}} \to \mathcal{P}(\mathcal{E})$ es la función que define estados finales. Es decir, un estado $s$ es final para el programa $p$ si y sólo si $s \in \text{final}(p)$.

# Semántica operacional

$\mathcal{L} = \langle \mathcal{P}_{\mathcal{L}}, \mathcal{D}_{\mathcal{L}}, [\![\cdot]\!]_{\mathcal{L}} \rangle$

Una semántica operacional para $\mathcal{L}$ es una tupla

$$\mathcal{O}_{\mathcal{L}} = \langle \mathcal{S}, \mathcal{E}, \cdot \rhd \cdot \to \cdot, \mathsf{final}, \mathsf{init}, \mathsf{output} \rangle$$

tal que

- $\mathcal{S} \neq \varnothing$ es el conjunto de memorias
- $\mathcal{E} \neq \varnothing$ es el conjunto de estados cuya definición involucra usualmente a $\mathcal{S}$.
- final : $\mathcal{P}_{\mathcal{L}} \to \mathcal{P}(\mathcal{E})$ es la función que define estados finales. Es decir, un estado $s$ es final para el programa $p$ si y sólo si $s \in \mathsf{final}(p)$.
- init : $\mathcal{P}_{\mathcal{L}} \times \mathcal{D}_{\mathcal{L}} \to \mathcal{E}$ es la función de inicialización de la ejecución.

# Semántica operacional

## $\mathcal{L} = \langle \mathcal{P}_{\mathcal{L}}, \mathcal{D}_{\mathcal{L}}, [\![\cdot]\!]_{\mathcal{L}} \rangle$

Una semántica operacional para $\mathcal{L}$ es una tupla

$$\mathcal{O}_{\mathcal{L}} = \langle \mathcal{S}, \mathcal{E}, \cdot \triangleright \cdot \to \cdot, \text{final}, \text{init}, \text{output} \rangle$$

tal que

- $\text{output} : \mathcal{P}_{\mathcal{L}} \times \mathcal{E} \to \mathcal{D}_{\mathcal{L}}$ es la función de salida.

# Semántica operacional

$\mathcal{L} = \langle \mathcal{P}_\mathcal{L}, \mathcal{D}_\mathcal{L}, [\![\cdot]\!]_\mathcal{L} \rangle$

Una semántica operacional para $\mathcal{L}$ es una tupla

$$\mathcal{O}_\mathcal{L} = \langle \mathcal{S}, \mathcal{E}, \cdot \rhd \cdot \to \cdot, \text{final}, \text{init}, \text{output} \rangle$$

tal que

- output : $\mathcal{P}_\mathcal{L} \times \mathcal{E} \to \mathcal{D}_\mathcal{L}$ es la función de salida.
- $\cdot \rhd \cdot \to \cdot \subseteq \mathcal{P}_\mathcal{L} \times \mathcal{E} \times \mathcal{E}$ es una relación ternaria de transición entre programas, estados y estados

# Semántica operacional

$\mathcal{L} = \langle \mathcal{P}_\mathcal{L}, \mathcal{D}_\mathcal{L}, [\![\cdot]\!]_\mathcal{L} \rangle$

Una semántica operacional para $\mathcal{L}$ es una tupla

$$\mathcal{O}_\mathcal{L} = \langle \mathcal{S}, \mathcal{E}, \cdot \rhd \cdot \to \cdot, \mathsf{final}, \mathsf{init}, \mathsf{output} \rangle$$

tal que

- output : $\mathcal{P}_\mathcal{L} \times \mathcal{E} \to \mathcal{D}_\mathcal{L}$ es la función de salida.
- $\cdot \rhd \cdot \to \cdot \subseteq \mathcal{P}_\mathcal{L} \times \mathcal{E} \times \mathcal{E}$ es una relación ternaria de transición entre programas, estados y estados
- Significado intencional:

  *$p \rhd s \to s'$ si y sólo si la ejecución del programa $p \in \mathcal{P}_\mathcal{L}$ causa la transición del estado $s$ al estado $s'$*

# El lenguaje LTURING

- Datos: $\mathcal{D}_{\mathcal{L}} = \{0, 1, ␣\}^{\star}$

# El lenguaje LTuring

- Datos: $\mathcal{D}_{\mathcal{L}} = \{0, 1, \sqcup\}^{\star}$
- Programas:

$$\mathcal{P}_{\mathcal{L}} \ni p ::= 1 : \mathcal{I}_1; \ldots; m : \mathcal{I}_m : m+1 : \texttt{halt}$$

# El lenguaje LTuring

- Datos: $\mathcal{D}_{\mathcal{L}} = \{0, 1, \sqcup\}^{\star}$
- Programas:

$$\mathcal{P}_{\mathcal{L}} \ni p ::= 1 : \mathcal{I}_1; \ldots; m : \mathcal{I}_m : m+1 : \mathtt{halt}$$

- Instrucciones

$$\begin{array}{rl} \mathcal{I} ::= & \mathtt{right} \mid \mathtt{left} \mid \mathtt{write}\; s \mid \\ & \mathtt{if}\; s\; \mathtt{then\; goto}\; \ell\; \mathtt{else\; goto}\; \ell' \mid \mathtt{goto}\; \ell \end{array}$$

donde $s \in \{0, 1, \sqcup\}$ y $\ell, \ell' \in \mathbb{N}$.

# Semántica operacional

## LTURING

$$p = 1 : \mathcal{I}_1; \ldots; m : \mathcal{I}_m : m + 1 : \texttt{halt}$$

- Memorias: $\mathcal{S} =_{def} \mathcal{D}_{\mathcal{L}} \times \{0, 1, \llcorner\!\lrcorner\} \times \mathcal{D}_{\mathcal{L}}$.

# Semántica operacional

## LTURING

$$p = 1 : \mathcal{I}_1; \ldots; m : \mathcal{I}_m : m + 1 : \texttt{halt}$$

- Memorias: $\mathcal{S} =_{def} \mathcal{D}_{\mathcal{L}} \times \{0, 1, \sqcup\} \times \mathcal{D}_{\mathcal{L}}$.
- Usualmente escribiremos $L\underline{s}R \in \mathcal{S}$ en vez de $(L, s, R) \in \mathcal{S}$.

# Semántica operacional

## LTuring

$$p = 1 : \mathcal{I}_1; \ldots; m : \mathcal{I}_m : m + 1 : \texttt{halt}$$

- Memorias: $\mathcal{S} =_{def} \mathcal{D}_{\mathcal{L}} \times \{0, 1, \sqcup\} \times \mathcal{D}_{\mathcal{L}}$.
- Usualmente escribiremos $L\underline{s}R \in \mathcal{S}$ en vez de $(L, s, R) \in \mathcal{S}$.
- Estados: $\mathcal{E} = \mathbb{N} \times \mathcal{S}$

# Semántica operacional
## LTuring

$$p = 1 : \mathcal{I}_1; \ldots ; m : \mathcal{I}_m : m+1 : \texttt{halt}$$

- Memorias: $\mathcal{S} =_{def} \mathcal{D}_{\mathcal{L}} \times \{0, 1, \sqcup\} \times \mathcal{D}_{\mathcal{L}}$.
- Usualmente escribiremos $L\underline{s}R \in \mathcal{S}$ en vez de $(L, s, R) \in \mathcal{S}$.
- Estados: $\mathcal{E} = \mathbb{N} \times \mathcal{S}$
- Estados finales: $\mathrm{final}(p) = \{\langle m+1, \sigma\rangle | \sigma \in \mathcal{S}\}$.

# Semántica operacional
## LTuring

$$p = 1 : \mathcal{I}_1; \ldots; m : \mathcal{I}_m : m+1 : \texttt{halt}$$

- Memorias: $\mathcal{S} =_{def} \mathcal{D}_{\mathcal{L}} \times \{0, 1, \llcorner\!\lrcorner\} \times \mathcal{D}_{\mathcal{L}}$.
- Usualmente escribiremos $L\underline{s}R \in \mathcal{S}$ en vez de $(L, s, R) \in \mathcal{S}$.
- Estados: $\mathcal{E} = \mathbb{N} \times \mathcal{S}$
- Estados finales: $\text{final}(p) = \{\langle m+1, \sigma\rangle | \sigma \in \mathcal{S}\}$.
- Función de inicialización: $\text{init} : \mathcal{P}_{\mathcal{L}} \times \mathcal{D}_{\mathcal{L}} \to \mathcal{E},\ \text{init}(x) = \langle 1, \underline{\llcorner\!\lrcorner}x\rangle$

# Semántica operacional

## LTURING

$$p = 1 : \mathcal{I}_1; \ldots ; m : \mathcal{I}_m : m+1 : \texttt{halt}$$

- Memorias: $\mathcal{S} =_{def} \mathcal{D}_{\mathcal{L}} \times \{0, 1, \text{␣}\} \times \mathcal{D}_{\mathcal{L}}$.
- Usualmente escribiremos $L\underline{s}R \in \mathcal{S}$ en vez de $(L, s, R) \in \mathcal{S}$.
- Estados: $\mathcal{E} = \mathbb{N} \times \mathcal{S}$
- Estados finales: $\text{final}(p) = \{\langle m+1, \sigma\rangle | \sigma \in \mathcal{S}\}$.
- Función de inicialización: $\text{init} : \mathcal{P}_{\mathcal{L}} \times \mathcal{D}_{\mathcal{L}} \to \mathcal{E}, \ \text{init}(x) = \langle 1, \underline{\text{␣}}x\rangle$
- Función de salida: $\text{output} : \mathcal{P}_{\mathcal{L}} \times \mathcal{E} \to \mathcal{D}_{\mathcal{L}}$,

$$\text{output}(\langle \ell, L\underline{s}R\rangle) = R$$

# Relación de transición

## LTURING

- Si $\mathcal{I}_\ell = \texttt{right}$ entonces $p \triangleright \langle \ell, L\underline{s}s'R\rangle \to \langle \ell+1, Ls\underline{s'}R\rangle$

# Relación de transición

LTuring

- Si $\mathcal{I}_\ell = \texttt{right}$ entonces $p \rhd \langle \ell, L\underline{s}s'R \rangle \to \langle \ell + 1, Ls\underline{s'}R \rangle$
- Si $\mathcal{I}_\ell = \texttt{right}$ entonces $p \rhd \langle \ell, L\underline{s} \rangle \to \langle \ell + 1, Ls\underline{\sqcup} \rangle$

# Relación de transición

## LTURING

- Si $\mathcal{I}_\ell = \texttt{right}$ entonces $p \triangleright \langle \ell, L\underline{s}s'R \rangle \to \langle \ell + 1, Ls\underline{s'}R \rangle$
- Si $\mathcal{I}_\ell = \texttt{right}$ entonces $p \triangleright \langle \ell, L\underline{s} \rangle \to \langle \ell + 1, Ls\underline{\sqcup} \rangle$
- Si $\mathcal{I}_\ell = \texttt{left}$ entonces $p \triangleright \langle \ell, Ls\underline{s'}R \rangle \to \langle \ell + 1, L\underline{s}s'R \rangle$

# Relación de transición

## LTURING

- Si $\mathcal{I}_\ell = \texttt{right}$ entonces $p \triangleright \langle \ell, L\underline{s}s'R\rangle \to \langle \ell + 1, Ls\underline{s'}R\rangle$
- Si $\mathcal{I}_\ell = \texttt{right}$ entonces $p \triangleright \langle \ell, L\underline{s}\rangle \to \langle \ell + 1, Ls\underline{\textvisiblespace}\rangle$
- Si $\mathcal{I}_\ell = \texttt{left}$ entonces $p \triangleright \langle \ell, Ls\underline{s'}R\rangle \to \langle \ell + 1, L\underline{s}s'R\rangle$
- Si $\mathcal{I}_\ell = \texttt{left}$ entonces $p \triangleright \langle \ell, \underline{s}R\rangle \to \langle \ell + 1, \underline{\textvisiblespace}sR\rangle$

# Relación de transición

## LTURING

- Si $\mathcal{I}_\ell = \texttt{right}$ entonces $p \rhd \langle \ell, L\underline{s}s'R \rangle \to \langle \ell + 1, Ls\underline{s'}R \rangle$
- Si $\mathcal{I}_\ell = \texttt{right}$ entonces $p \rhd \langle \ell, L\underline{s} \rangle \to \langle \ell + 1, Ls\underline{\sqcup} \rangle$
- Si $\mathcal{I}_\ell = \texttt{left}$ entonces $p \rhd \langle \ell, Ls\underline{s'}R \rangle \to \langle \ell + 1, L\underline{s}s'R \rangle$
- Si $\mathcal{I}_\ell = \texttt{left}$ entonces $p \rhd \langle \ell, \underline{s}R \rangle \to \langle \ell + 1, \underline{\sqcup}sR \rangle$
- Si $\mathcal{I}_\ell = \texttt{write}\ s$ entonces $p \rhd \langle \ell, L\underline{s'}R \rangle \to \langle \ell + 1, L\underline{s}R \rangle$

# Relación de transición

## LTuring

- Si $\mathcal{I}_\ell = \texttt{right}$ entonces $p \rhd \langle \ell, L\underline{s}s'R\rangle \to \langle \ell + 1, Ls\underline{s'}R\rangle$
- Si $\mathcal{I}_\ell = \texttt{right}$ entonces $p \rhd \langle \ell, L\underline{s}\rangle \to \langle \ell + 1, Ls\underline{\sqcup}\rangle$
- Si $\mathcal{I}_\ell = \texttt{left}$ entonces $p \rhd \langle \ell, Ls\underline{s'}R\rangle \to \langle \ell + 1, L\underline{s}s'R\rangle$
- Si $\mathcal{I}_\ell = \texttt{left}$ entonces $p \rhd \langle \ell, \underline{s}R\rangle \to \langle \ell + 1, \underline{\sqcup}sR\rangle$
- Si $\mathcal{I}_\ell = \texttt{write}\, s$ entonces $p \rhd \langle \ell, L\underline{s'}R\rangle \to \langle \ell + 1, L\underline{s}R\rangle$
- Si $\mathcal{I}_\ell = \texttt{goto}\, \ell'$ entonces $p \rhd \langle \ell, L\underline{s}R\rangle \to \langle \ell', L\underline{s}R\rangle$

# Relación de transición

## LTURING

- Si $\mathcal{I}_\ell = \texttt{right}$ entonces $p \rhd \langle \ell, L\underline{s}s'R \rangle \to \langle \ell + 1, Ls\underline{s'}R \rangle$
- Si $\mathcal{I}_\ell = \texttt{right}$ entonces $p \rhd \langle \ell, L\underline{s} \rangle \to \langle \ell + 1, Ls\underline{\sqcup} \rangle$
- Si $\mathcal{I}_\ell = \texttt{left}$ entonces $p \rhd \langle \ell, Ls\underline{s'}R \rangle \to \langle \ell + 1, L\underline{s}s'R \rangle$
- Si $\mathcal{I}_\ell = \texttt{left}$ entonces $p \rhd \langle \ell, \underline{s}R \rangle \to \langle \ell + 1, \underline{\sqcup}sR \rangle$
- Si $\mathcal{I}_\ell = \texttt{write}\ s$ entonces $p \rhd \langle \ell, L\underline{s'}R \rangle \to \langle \ell + 1, L\underline{s}R \rangle$
- Si $\mathcal{I}_\ell = \texttt{goto}\ \ell'$ entonces $p \rhd \langle \ell, L\underline{s}R \rangle \to \langle \ell', L\underline{s}R \rangle$
- Si $\mathcal{I}_\ell = \texttt{if}\ s\ \texttt{then goto}\ \ell'\ \texttt{else goto}\ \ell''$ entonces $p \rhd \langle \ell, L\underline{s}R \rangle \to \langle \ell', L\underline{s}R \rangle$

# Relación de transición

## LTuring

- Si $\mathcal{I}_\ell = \texttt{right}$ entonces $p \rhd \langle \ell, L\underline{s}s'R \rangle \to \langle \ell + 1, Ls\underline{s'}R \rangle$
- Si $\mathcal{I}_\ell = \texttt{right}$ entonces $p \rhd \langle \ell, L\underline{s} \rangle \to \langle \ell + 1, Ls\underline{\sqcup} \rangle$
- Si $\mathcal{I}_\ell = \texttt{left}$ entonces $p \rhd \langle \ell, Ls\underline{s'}R \rangle \to \langle \ell + 1, L\underline{s}s'R \rangle$
- Si $\mathcal{I}_\ell = \texttt{left}$ entonces $p \rhd \langle \ell, \underline{s}R \rangle \to \langle \ell + 1, \underline{\sqcup}sR \rangle$
- Si $\mathcal{I}_\ell = \texttt{write}\, s$ entonces $p \rhd \langle \ell, L\underline{s'}R \rangle \to \langle \ell + 1, L\underline{s}R \rangle$
- Si $\mathcal{I}_\ell = \texttt{goto}\, \ell'$ entonces $p \rhd \langle \ell, L\underline{s}R \rangle \to \langle \ell', L\underline{s}R \rangle$
- Si $\mathcal{I}_\ell = \texttt{if}\, s\, \texttt{then goto}\, \ell'\, \texttt{else goto}\, \ell''$ entonces $p \rhd \langle \ell, L\underline{s}R \rangle \to \langle \ell', L\underline{s}R \rangle$
- Si $\mathcal{I}_\ell = \texttt{if}\, s\, \texttt{then goto}\, \ell'\, \texttt{else goto}\, \ell''$ entonces $p \rhd \langle \ell, L\underline{s'}R \rangle \to \langle \ell'', L\underline{s'}R \rangle$

# De $\mathcal{MT}$ a LTURING

Sea $T = \langle \Sigma, Q, q_0, q_f, \delta \rangle$ con $\Sigma = \{0, 1, \sqcup\}$ una máquina de Turing clásica.

# De $\mathcal{MT}$ a LTURING

Sea $T = \langle \Sigma, Q, q_0, q_f, \delta \rangle$ con $\Sigma = \{0, 1, \llcorner\!\lrcorner\}$ una máquina de Turing clásica.

- Existe un programa $p_T$ del lenguaje LTURING tal que

$$L(T) \subseteq \{x \in \Sigma^\star \mid [\![p_T]\!](x) \text{ existe}\}$$

# De $\mathcal{MT}$ a LTURING

Sea $T = \langle \Sigma, Q, q_0, q_f, \delta \rangle$ con $\Sigma = \{0, 1, \sqcup\}$ una máquina de Turing clásica.

- Existe un programa $p_T$ del lenguaje LTURING tal que

$$L(T) \subseteq \{x \in \Sigma^\star \mid [\![p_T]\!](x) \text{ existe}\}$$

- Más aún, si $(q_0, \underline{\sqcup}x) \to^\star (q_f, w\underline{s}\,y)$ entonces

$$[\![p_T]\!](x) = y.$$

# De LTURING a $\mathcal{MT}$

Sea $p$ un programa en LTURING

# De LTuring a $\mathcal{MT}$

Sea $p$ un programa en LTuring

- Existe una máquina de Turing clásica $T_p$ tal que:

$$\text{si } [\![p]\!](x) = y \text{ entonces } x \in L(T).$$

# De LTuring a $\mathcal{MT}$

Sea $p$ un programa en LTuring

- Existe una máquina de Turing clásica $T_p$ tal que:

$$\text{si } [\![p]\!](x) = y \text{ entonces } x \in L(T).$$

- Más aún, si $p \rhd_{LTuring} (1, \underline{\sqcup}x) \to^{\star} (m+1, w\underline{s}y)$ entonces

$$(q_1, \underline{\sqcup}x) \vdash^{\star} (q_f, w\underline{s}y).$$

# Árboles binarios

## Tipo de datos

El conjunto de árboles $\mathbb{T}$ se define recursivamente como:

- `nil` es un elemento de $\mathbb{T}$.

# Árboles binarios

## Tipo de datos

El conjunto de árboles $\mathbb{T}$ se define recursivamente como:

- `nil` es un elemento de $\mathbb{T}$.
- Si $t_1,\ t_2 \in \mathbb{T}$ entonces $(t_1.t_2) \in \mathbb{T}$.

# Árboles binarios

## Tipo de datos

El conjunto de árboles $\mathbb{T}$ se define recursivamente como:

- `nil` es un elemento de $\mathbb{T}$.
- Si $t_1,\ t_2 \in \mathbb{T}$ entonces $(t_1 . t_2) \in \mathbb{T}$.
- Son todos.

# Árboles binarios

## Codificación de booleanos y naturales

- Los valores de verdad `true` y `false` se definen como:

# Árboles binarios

## Codificación de booleanos y naturales

- Los valores de verdad `true` y `false` se definen como:
  - `false = nil`

# Árboles binarios

## Codificación de booleanos y naturales

- Los valores de verdad `true` y `false` se definen como:
  - `false = nil`
  - `true = (nil.nil)`

# Árboles binarios

## Codificación de booleanos y naturales

- Los valores de verdad `true` y `false` se definen como:
  - `false = nil`
  - `true = (nil.nil)`

- El número natural $n$ se codifica mediante un árbol de tamaño $n+1$, construido de la siguiente forma:

# Árboles binarios

## Codificación de booleanos y naturales

- Los valores de verdad `true` y `false` se definen como:
  - `false = nil`
  - `true = (nil.nil)`
- El número natural $n$ se codifica mediante un árbol de tamaño $n+1$, construido de la siguiente forma:
  - Definimos $\underline{n} = \texttt{nil}^n$ donde

$$\begin{array}{rcl} \texttt{nil}^0 & = & \texttt{nil} \\ \texttt{nil}^{n+1} & = & (\texttt{nil.nil}^{\texttt{n}}) \end{array}$$

# Árboles binarios

## Codificación de booleanos y naturales

- Los valores de verdad `true` y `false` se definen como:
  - `false = nil`
  - `true = (nil.nil)`

- El número natural $n$ se codifica mediante un árbol de tamaño $n+1$, construido de la siguiente forma:
  - Definimos $\underline{n} = \mathtt{nil}^n$ donde

$$\begin{array}{rcl} \mathtt{nil}^0 & = & \mathtt{nil} \\ \mathtt{nil}^{n+1} & = & (\mathtt{nil.nil}^{\mathtt{n}}) \end{array}$$

  - $\mathcal{N} = \{\underline{n} | n \in \mathbb{N}\}$.

# Árboles binarios

## Codificación de booleanos y naturales

- Los valores de verdad `true` y `false` se definen como:
  - `false = nil`
  - `true = (nil.nil)`

- El número natural $n$ se codifica mediante un árbol de tamaño $n+1$, construido de la siguiente forma:
  - Definimos $\underline{n} = \mathtt{nil}^n$ donde

$$
\begin{array}{rcl}
\mathtt{nil}^0 & = & \mathtt{nil} \\
\mathtt{nil}^{n+1} & = & (\mathtt{nil}.\mathtt{nil}^{\mathtt{n}})
\end{array}
$$

  - $\mathcal{N} = \{\underline{n} | n \in \mathbb{N}\}$.
  - Por ejemplo, el numeral correspondiente al número natural 4 es $\underline{4} = (\mathtt{nil}.(\mathtt{nil}.(\mathtt{nil}.(\mathtt{nil}.\mathtt{nil}))))$.

# Árboles binarios

## Codificación de booleanos y naturales

- Los valores de verdad `true` y `false` se definen como:
  - `false = nil`
  - `true = (nil.nil)`

- El número natural $n$ se codifica mediante un árbol de tamaño $n+1$, construido de la siguiente forma:
  - Definimos $\underline{n} = \mathtt{nil}^n$ donde

$$\begin{array}{rcl} \mathtt{nil}^0 & = & \mathtt{nil} \\ \mathtt{nil}^{n+1} & = & (\mathtt{nil}.\mathtt{nil}^n) \end{array}$$

  - $\mathcal{N} = \{\underline{n} | n \in \mathbb{N}\}$.
  - Por ejemplo, el numeral correspondiente al número natural 4 es $\underline{4} = (\mathtt{nil}.(\mathtt{nil}.(\mathtt{nil}.(\mathtt{nil}.\mathtt{nil}))))$.
  - Por simplicidad escribiremos $0, 1, 2, \ldots$ en vez de $\underline{0}, \underline{1}, \underline{2}, \ldots$ o $\mathtt{nil}^0, \mathtt{nil}^1, \mathtt{nil}^2, \ldots$.

# El lenguaje WHILE

- Datos: $\mathcal{D}_{\mathcal{L}} = \mathbb{T}$.

# El lenguaje WHILE

- Datos: $\mathcal{D}_{\mathcal{L}} = \mathbb{T}$.
- Programas:

$$\mathcal{P}_{\mathcal{L}} \ni p ::= \texttt{read}\, X;\ C_1, \ldots, C_m\,; \texttt{write}\, Y$$

# El lenguaje WHILE

- Datos: $\mathcal{D}_{\mathcal{L}} = \mathbb{T}$.
- Programas:

$$\mathcal{P}_{\mathcal{L}} \ni p ::= \texttt{read}\, X;\ C_1, \ldots, C_m\, ;\texttt{write}\, Y$$

- Comandos:

$$C ::= X := e \mid \texttt{while}\, e\, \texttt{do}\ \ \vec{C}\ \texttt{end}$$

# El lenguaje WHILE

- Datos: $\mathcal{D}_{\mathcal{L}} = \mathbb{T}$.
- Programas:

$$\mathcal{P}_{\mathcal{L}} \ni p ::= \texttt{read}\, X;\ C_1, \ldots, C_m\, ;\texttt{write}\ Y$$

- Comandos:

$$C ::= X := e \mid \texttt{while}\, e\, \texttt{do}\ \ \vec{C}\ \texttt{end}$$

- Expresiones:

$$e, f ::= X \mid \mathsf{d} \mid \texttt{cons}\ \ e\ f \mid \texttt{hd}\, e \mid \texttt{tl}\ e \mid\ =?\, e\ f$$

# El lenguaje WHILE

- Datos: $\mathcal{D}_{\mathcal{L}} = \mathbb{T}$.
- Programas:

$$\mathcal{P}_{\mathcal{L}} \ni p ::= \texttt{read}\, X;\ C_1, \ldots, C_m\, ; \texttt{write}\, Y$$

- Comandos:

$$C ::= X := e \mid \texttt{while}\, e\, \texttt{do}\ \ \vec{C}\ \texttt{end}$$

- Expresiones:

$$e, f ::= X \mid \mathrm{d} \mid \texttt{cons}\ \ e\, f \mid \texttt{hd}\, e \mid \texttt{tl}\, e \mid =?\, e\, f$$

- Longitud o número de lineas en una secuencia de comandos, denotada

# El lenguaje WHILE

- Datos: $\mathcal{D}_{\mathcal{L}} = \mathbb{T}$.
- Programas:

$$\mathcal{P}_{\mathcal{L}} \ni p ::= \texttt{read}\, X;\ C_1, \ldots, C_m\, ; \texttt{write}\ Y$$

- Comandos:

$$C ::= X := e \mid \texttt{while}\, e\, \texttt{do}\ \ \vec{C}\ \texttt{end}$$

- Expresiones:

$$e, f ::= X \mid \mathrm{d} \mid \texttt{cons}\ \ e\ f \mid \texttt{hd}\, e \mid \texttt{tl}\ e \mid =?\, e\ f$$

- Longitud o número de lineas en una secuencia de comandos, denotada
  - $|X := e| = 1$

# El lenguaje WHILE

- Datos: $\mathcal{D}_{\mathcal{L}} = \mathbb{T}$.
- Programas:

$$\mathcal{P}_{\mathcal{L}} \ni p ::= \texttt{read}\, X;\ C_1, \ldots, C_m\, ;\texttt{write}\, Y$$

- Comandos:

$$C ::= X := e \mid \texttt{while}\, e\, \texttt{do}\ \ \vec{C}\ \texttt{end}$$

- Expresiones:

$$e, f ::= X \mid \mathrm{d} \mid \texttt{cons}\ \ e\ f \mid \texttt{hd}\, e \mid \texttt{tl}\ e \mid =?\, e\ f$$

- Longitud o número de lineas en una secuencia de comandos, denotada
  - $|X := e| = 1$
  - $|\texttt{while}\, e\, \texttt{do}\ \ \vec{C}\ \texttt{end}| = |\vec{C}| + 1$

# El lenguaje WHILE

- Datos: $\mathcal{D}_{\mathcal{L}} = \mathbb{T}$.
- Programas:

$$\mathcal{P}_{\mathcal{L}} \ni p ::= \texttt{read}\, X;\ C_1, \ldots, C_m\, ; \texttt{write}\, Y$$

- Comandos:

$$C ::= X := e \mid \texttt{while}\, e\, \texttt{do}\ \ \vec{C}\ \texttt{end}$$

- Expresiones:

$$e, f ::= X \mid \mathrm{d} \mid \texttt{cons}\ \ e\ f \mid \texttt{hd}\, e \mid \texttt{tl}\ e \mid =?\, e\ f$$

- Longitud o número de lineas en una secuencia de comandos, denotada
  - $|X := e| = 1$
  - $|\texttt{while}\, e\, \texttt{do}\ \ \vec{C}\ \texttt{end}| = |\vec{C}| + 1$
  - $|C_1; \ldots; C_n| = |C_1| + \ldots + |C_n|$

# Ejemplo

## WHILE

El siguiente programa realiza la suma dos numerales. Aquí $XY$ es una variable que recibe a un dato de entrada el cual debe ser de la forma $(n.m)$ para devolver $n + m$.

```
read XY;  (* add X Y *)
  X := hd XY;
  Y := tl XY;
  while X do
    Y := cons nil Y;
    X := tl X;
  end
write Y
```

# Semántica Operacional

## WHILE

- Memorias: $\mathcal{S} = \{\sigma | \ \sigma : \mathit{Var} \to \mathbb{T}\}$

# Semántica Operacional

## WHILE

- Memorias: $\mathcal{S} = \{\sigma | \ \sigma : \mathit{Var} \to \mathbb{T}\}$
- Estados: $\mathcal{E} = \mathbb{N} \times \mathcal{S}$.

# Semántica Operacional

## WHILE

- Memorias: $\mathcal{S} = \{\sigma | \ \sigma : \mathit{Var} \to \mathbb{T}\}$
- Estados: $\mathcal{E} = \mathbb{N} \times \mathcal{S}$.
- Evaluación de expresiones: la relación de transición se servirá de la siguiente función $\mathsf{ev} : \mathcal{E} \to \mathit{Expr} \to \mathbb{T}$

# Semántica Operacional

## While

- Memorias: $\mathcal{S} = \{\sigma |\ \sigma : \mathit{Var} \to \mathbb{T}\}$
- Estados: $\mathcal{E} = \mathbb{N} \times \mathcal{S}$.
- Evaluación de expresiones: la relación de transición se servirá de la siguiente función $\mathrm{ev} : \mathcal{E} \to \mathit{Expr} \to \mathbb{T}$
  - $\mathrm{ev}_s(X) = \sigma(X)$ donde $s = \langle \ell, \sigma \rangle$.

## Semántica Operacional
### WHILE

- Memorias: $\mathcal{S} = \{\sigma |\ \sigma : \mathit{Var} \to \mathbb{T}\}$
- Estados: $\mathcal{E} = \mathbb{N} \times \mathcal{S}$.
- Evaluación de expresiones: la relación de transición se servirá de la siguiente función $\mathrm{ev} : \mathcal{E} \to \mathit{Expr} \to \mathbb{T}$
  - $\mathrm{ev}_s(X) = \sigma(X)$ donde $s = \langle \ell, \sigma \rangle$.
  - $\mathrm{ev}_s(d) = d$

# Semántica Operacional

## WHILE

- Memorias: $\mathcal{S} = \{\sigma | \ \sigma : \mathit{Var} \to \mathbb{T}\}$
- Estados: $\mathcal{E} = \mathbb{N} \times \mathcal{S}$.
- Evaluación de expresiones: la relación de transición se servirá de la siguiente función $\mathsf{ev} : \mathcal{E} \to \mathit{Expr} \to \mathbb{T}$
  - $\mathsf{ev}_s(X) = \sigma(X)$ donde $s = \langle \ell, \sigma \rangle$.
  - $\mathsf{ev}_s(d) = d$
  - $\mathsf{ev}_s(\mathtt{cons}\, e\, f) = ((\mathsf{ev}_\sigma\ e).(\mathsf{ev}_\sigma\ f))$

- Memorias: $\mathcal{S} = \{\sigma |\ \sigma : \mathit{Var} \to \mathbb{T}\}$
- Estados: $\mathcal{E} = \mathbb{N} \times \mathcal{S}$.
- Evaluación de expresiones: la relación de transición se servirá de la siguiente función $\mathrm{ev} : \mathcal{E} \to \mathit{Expr} \to \mathbb{T}$
  - $\mathrm{ev}_s(X) = \sigma(X)$ donde $s = \langle \ell, \sigma \rangle$.
  - $\mathrm{ev}_s(d) = d$
  - $\mathrm{ev}_s(\mathtt{cons}\, e\, f) = ((\mathrm{ev}_\sigma\, e).(\mathrm{ev}_\sigma\, f))$
  - $\mathrm{ev}_s(\mathtt{hd}\, e) = \mathit{nil}$, si $\mathrm{ev}_\sigma\, e = \mathit{nil}$

## Semántica Operacional

### While

- Memorias: $\mathcal{S} = \{\sigma | \ \sigma : \mathit{Var} \to \mathbb{T}\}$
- Estados: $\mathcal{E} = \mathbb{N} \times \mathcal{S}$.
- Evaluación de expresiones: la relación de transición se servirá de la siguiente función $\mathsf{ev} : \mathcal{E} \to \mathit{Expr} \to \mathbb{T}$
  - $\mathsf{ev}_s(X) = \sigma(X)$ donde $s = \langle \ell, \sigma \rangle$.
  - $\mathsf{ev}_s(d) = d$
  - $\mathsf{ev}_s(\texttt{cons}\, e\, f) = ((\mathsf{ev}_\sigma\ e).(\mathsf{ev}_\sigma\ f))$
  - $\mathsf{ev}_s(\texttt{hd}\, e) = \mathit{nil}$, si $\mathsf{ev}_\sigma\ e = \mathit{nil}$
  - $\mathsf{ev}_s(\texttt{hd}\, e) = t$, si $\mathsf{ev}_\sigma\ e = (t.r)$

# Semántica Operacional

## WHILE

- Memorias: $\mathcal{S} = \{\sigma | \ \sigma : \mathit{Var} \to \mathbb{T}\}$
- Estados: $\mathcal{E} = \mathbb{N} \times \mathcal{S}$.
- Evaluación de expresiones: la relación de transición se servirá de la siguiente función $\mathrm{ev} : \mathcal{E} \to \mathit{Expr} \to \mathbb{T}$
  - $\mathrm{ev}_s(X) = \sigma(X)$ donde $s = \langle \ell, \sigma \rangle$.
  - $\mathrm{ev}_s(d) = d$
  - $\mathrm{ev}_s(\mathtt{cons}\, e\, f) = ((\mathrm{ev}_\sigma\ e).(\mathrm{ev}_\sigma\ f))$
  - $\mathrm{ev}_s(\mathtt{hd}\, e) = \mathit{nil}$, si $\mathrm{ev}_\sigma\ e = \mathit{nil}$
  - $\mathrm{ev}_s(\mathtt{hd}\, e) = t$, si $\mathrm{ev}_\sigma\ e = (t.r)$
  - $\mathrm{ev}_s(\mathtt{tl}\ e) = \mathit{nil}$, si $\mathrm{ev}_\sigma\ e = \mathit{nil}$

# Semántica Operacional

## While

- Memorias: $\mathcal{S} = \{\sigma | \ \sigma : \mathit{Var} \to \mathbb{T}\}$
- Estados: $\mathcal{E} = \mathbb{N} \times \mathcal{S}$.
- Evaluación de expresiones: la relación de transición se servirá de la siguiente función $\mathrm{ev} : \mathcal{E} \to \mathit{Expr} \to \mathbb{T}$
  - $\mathrm{ev}_s(X) = \sigma(X)$ donde $s = \langle \ell, \sigma \rangle$.
  - $\mathrm{ev}_s(d) = d$
  - $\mathrm{ev}_s(\mathtt{cons}\, e\, f) = ((\mathrm{ev}_\sigma\ e).(\mathrm{ev}_\sigma\ f))$
  - $\mathrm{ev}_s(\mathtt{hd}\, e) = \mathit{nil}$, si $\mathrm{ev}_\sigma\ e = \mathit{nil}$
  - $\mathrm{ev}_s(\mathtt{hd}\, e) = t$, si $\mathrm{ev}_\sigma\ e = (t.r)$
  - $\mathrm{ev}_s(\mathtt{tl}\ e) = \mathit{nil}$, si $\mathrm{ev}_\sigma\ e = \mathit{nil}$
  - $\mathrm{ev}_s(\mathtt{tl}\ e) = r$, si $\mathrm{ev}_\sigma\ e = (t.r)$

- Memorias: $\mathcal{S} = \{\sigma |\ \sigma : \mathit{Var} \to \mathbb{T}\}$
- Estados: $\mathcal{E} = \mathbb{N} \times \mathcal{S}$.
- Evaluación de expresiones: la relación de transición se servirá de la siguiente función $\mathrm{ev} : \mathcal{E} \to \mathit{Expr} \to \mathbb{T}$
  - $\mathrm{ev}_s(X) = \sigma(X)$ donde $s = \langle \ell, \sigma \rangle$.
  - $\mathrm{ev}_s(d) = d$
  - $\mathrm{ev}_s(\texttt{cons}\, e\, f) = ((\mathrm{ev}_\sigma\ e).(\mathrm{ev}_\sigma\ f))$
  - $\mathrm{ev}_s(\texttt{hd}\, e) = \mathit{nil}$, si $\mathrm{ev}_\sigma\ e = \mathit{nil}$
  - $\mathrm{ev}_s(\texttt{hd}\, e) = t$, si $\mathrm{ev}_\sigma\ e = (t.r)$
  - $\mathrm{ev}_s(\texttt{tl}\, e) = \mathit{nil}$, si $\mathrm{ev}_\sigma\ e = \mathit{nil}$
  - $\mathrm{ev}_s(\texttt{tl}\, e) = r$, si $\mathrm{ev}_\sigma\ e = (t.r)$
  - $\mathrm{ev}_s(=?\ e\ f) = \mathit{true}$, si $\mathrm{ev}_\sigma\ e = \mathrm{ev}_\sigma\ f$

# Semántica Operacional

## WHILE

- Memorias: $\mathcal{S} = \{\sigma |\ \sigma : \mathit{Var} \to \mathbb{T}\}$
- Estados: $\mathcal{E} = \mathbb{N} \times \mathcal{S}$.
- Evaluación de expresiones: la relación de transición se servirá de la siguiente función $\mathsf{ev} : \mathcal{E} \to \mathit{Expr} \to \mathbb{T}$
  - $\mathsf{ev}_s(X) = \sigma(X)$ donde $s = \langle \ell, \sigma \rangle$.
  - $\mathsf{ev}_s(d) = d$
  - $\mathsf{ev}_s(\mathtt{cons}\, e\, f) = ((\mathsf{ev}_\sigma\, e).(\mathsf{ev}_\sigma\, f))$
  - $\mathsf{ev}_s(\mathtt{hd}\, e) = \mathit{nil}$, si $\mathsf{ev}_\sigma\, e = \mathit{nil}$
  - $\mathsf{ev}_s(\mathtt{hd}\, e) = t$, si $\mathsf{ev}_\sigma\, e = (t.r)$
  - $\mathsf{ev}_s(\mathtt{tl}\, e) = \mathit{nil}$, si $\mathsf{ev}_\sigma\, e = \mathit{nil}$
  - $\mathsf{ev}_s(\mathtt{tl}\, e) = r$, si $\mathsf{ev}_\sigma\, e = (t.r)$
  - $\mathsf{ev}_s(=?\, e\ f) = \mathit{true}$, si $\mathsf{ev}_\sigma\, e = \mathsf{ev}_\sigma\, f$
  - $\mathsf{ev}_s(=?\, e\ f) = \mathit{false}$, si $\mathsf{ev}_\sigma\, e \neq \mathsf{ev}_\sigma\, f$

# Relación de transición

## While

- Si $\mathcal{I}_\ell = X := e$ y $\mathrm{ev}_{\langle \ell, \sigma \rangle}(e) = d$ entonces

$$p \triangleright \langle \ell, \sigma \rangle \to \langle \ell + 1, \sigma[X/d] \rangle$$

donde $\sigma[X/d]$ denota a la actualización de $\sigma$ en $X$ por $d$.

## Relación de transición

### WHILE

- Si $\mathcal{I}_\ell = X := e$ y $\mathsf{ev}_{\langle \ell,\sigma\rangle}(e) = d$ entonces

$$p \rhd \langle \ell, \sigma\rangle \to \langle \ell + 1, \sigma[X/d]\rangle$$

  donde $\sigma[X/d]$ denota a la actualización de $\sigma$ en $X$ por $d$.

- Si $\mathcal{I}_\ell = \texttt{while}\ e\ \texttt{do}\ \vec{D}\ \texttt{end}$ con $|\vec{D}| = k$ y $\mathsf{ev}_s(e) = \mathit{nil}$, entonces

$$p \rhd \langle \ell, \sigma\rangle \to \langle \ell + k + 1, \sigma\rangle$$

# Relación de transición

## While

- Si $\mathcal{I}_\ell = X := e$ y $\mathrm{ev}_{\langle \ell,\sigma\rangle}(e) = d$ entonces

$$p \rhd \langle \ell, \sigma \rangle \to \langle \ell + 1, \sigma[X/d] \rangle$$

  donde $\sigma[X/d]$ denota a la actualización de $\sigma$ en $X$ por $d$.

- Si $\mathcal{I}_\ell = \texttt{while } e \texttt{ do } \vec{D} \texttt{ end}$ con $|\vec{D}| = k$ y $\mathrm{ev}_s(e) = nil$, entonces

$$p \rhd \langle \ell, \sigma \rangle \to \langle \ell + k + 1, \sigma \rangle$$

- Si $\mathcal{I}_\ell = \texttt{while } e \texttt{ do } \vec{D} \texttt{ end}$ con $|\vec{D}| = k$ y $\mathrm{ev}_s(e) \neq nil$, entonces

$$p \rhd \langle \ell, \sigma \rangle \to \langle \ell + k + 1, \sigma'' \rangle,$$

  donde

## Relación de transición

### WHILE

- Si $\mathcal{I}_\ell = X := e$ y $\text{ev}_{\langle \ell,\sigma\rangle}(e) = d$ entonces

$$p \rhd \langle \ell, \sigma \rangle \to \langle \ell + 1, \sigma[X/d] \rangle$$

  donde $\sigma[X/d]$ denota a la actualización de $\sigma$ en $X$ por $d$.

- Si $\mathcal{I}_\ell = \texttt{while}\ e\ \texttt{do}\ \ \vec{D}\ \texttt{end}$ con $|\vec{D}| = k$ y $\text{ev}_s(e) = nil$, entonces

$$p \rhd \langle \ell, \sigma \rangle \to \langle \ell + k + 1, \sigma \rangle$$

- Si $\mathcal{I}_\ell = \texttt{while}\ e\ \texttt{do}\ \ \vec{D}\ \texttt{end}$ con $|\vec{D}| = k$ y $\text{ev}_s(e) \neq nil$, entonces

$$p \rhd \langle \ell, \sigma \rangle \to \langle \ell + k + 1, \sigma'' \rangle,$$

  donde

  - $p \rhd \langle \ell + 1, \sigma \rangle \to^\star \langle \ell + k, \sigma' \rangle$

# Relación de transición

## WHILE

- Si $\mathcal{I}_\ell = X := e$ y $\text{ev}_{\langle \ell, \sigma \rangle}(e) = d$ entonces

$$p \rhd \langle \ell, \sigma \rangle \to \langle \ell + 1, \sigma[X/d] \rangle$$

  donde $\sigma[X/d]$ denota a la actualización de $\sigma$ en $X$ por $d$.

- Si $\mathcal{I}_\ell = \texttt{while}\ e\ \texttt{do}\ \ \vec{D}\ \texttt{end}$ con $|\vec{D}| = k$ y $\text{ev}_s(e) = nil$, entonces

$$p \rhd \langle \ell, \sigma \rangle \to \langle \ell + k + 1, \sigma \rangle$$

- Si $\mathcal{I}_\ell = \texttt{while}\ e\ \texttt{do}\ \ \vec{D}\ \texttt{end}$ con $|\vec{D}| = k$ y $\text{ev}_s(e) \neq nil$, entonces

$$p \rhd \langle \ell, \sigma \rangle \to \langle \ell + k + 1, \sigma'' \rangle,$$

  donde
    - $p \rhd \langle \ell + 1, \sigma \rangle \to^\star \langle \ell + k, \sigma' \rangle$
    - $p \rhd \langle \ell, \sigma' \rangle \to \langle \ell + k + 1, \sigma'' \rangle$

# Semántica Operacional

## WHILE

$$p = \texttt{read}\, X; \vec{C}; \texttt{write}\, Y$$

- Estados finales: $\text{final}(p) = \{\langle m+1, \sigma\rangle \mid \sigma \in \mathcal{S}\}$, donde $m = |\vec{C}|$.

# Semántica Operacional

## WHILE

$$p = \texttt{read}\, X; \vec{C}; \texttt{write}\, Y$$

- Estados finales: $\mathrm{final}(p) = \{\langle m+1, \sigma\rangle \mid \sigma \in \mathcal{S}\}$, donde $m = |\vec{C}|$.
- Función de inicialización: $\mathrm{init} : \mathcal{P}_{\mathcal{L}} \times \mathcal{D}_{\mathcal{L}} \to \mathcal{E}$,

$$\mathrm{init}(p, d) = \big\langle 1, [X \mapsto d, Y \mapsto \texttt{nil}, Z_1 \mapsto \texttt{nil}, \ldots, Z_n \mapsto \texttt{nil}]\big\rangle$$

donde $Vars(p) = \{X, Y, Z_1, \ldots, Z_n\}$.

# Semántica Operacional

## WHILE

$$p = \texttt{read}\, X; \vec{C}; \texttt{write}\, Y$$

- Estados finales: $\mathrm{final}(p) = \{\langle m+1, \sigma\rangle \mid \sigma \in \mathcal{S}\}$, donde $m = |\vec{C}|$.
- Función de inicialización: $\mathrm{init} : \mathcal{P}_{\mathcal{L}} \times \mathcal{D}_{\mathcal{L}} \to \mathcal{E}$,

$$\mathrm{init}(p, d) = \big\langle 1, [X \mapsto d, Y \mapsto \texttt{nil}, Z_1 \mapsto \texttt{nil}, \ldots, Z_n \mapsto \texttt{nil}]\big\rangle$$

  donde $\mathit{Vars}(p) = \{X, Y, Z_1, \ldots, Z_n\}$.
- Función de salida: $\mathrm{output} : \mathcal{P}_{\mathcal{L}} \times \mathcal{E} \to \mathcal{D}_{\mathcal{L}}$,

$$\mathrm{output}(p, \sigma) = \sigma(Y)$$

# El lenguaje GOTO

- $\mathcal{D}_{\mathcal{L}} = \mathbb{T}$

# El lenguaje GOTO

- $\mathcal{D}_{\mathcal{L}} = \mathbb{T}$
- $\mathcal{P}_{\mathcal{L}}$ se define mediante la siguiente gramática

# El lenguaje GOTO

- $\mathcal{D}_{\mathcal{L}} = \mathbb{T}$
- $\mathcal{P}_{\mathcal{L}}$ se define mediante la siguiente gramática

$$\mathcal{P}_{\mathcal{L}} \ni p ::= \texttt{read}\, X; 1 : \mathcal{I}_1; \ldots; m : \mathcal{I}_m; \texttt{write}\, Y$$

# El lenguaje GOTO

- $\mathcal{D}_{\mathcal{L}} = \mathbb{T}$
- $\mathcal{P}_{\mathcal{L}}$ se define mediante la siguiente gramática

$$\mathcal{P}_{\mathcal{L}} \ni p ::= \texttt{read}\, X; 1 : \mathcal{I}_1; \ldots ; m : \mathcal{I}_m; \texttt{write}\, Y$$

$$\mathcal{I} ::= X := e \mid \texttt{if}\, X\, \texttt{then goto}\, \ell\, \texttt{else goto}\, \ell' \mid \texttt{goto}\, \ell$$

# El lenguaje GOTO

- $\mathcal{D}_{\mathcal{L}} = \mathbb{T}$
- $\mathcal{P}_{\mathcal{L}}$ se define mediante la siguiente gramática

$$\mathcal{P}_{\mathcal{L}} \ni p ::= \texttt{read}\, X; 1 : \mathcal{I}_1; \ldots ; m : \mathcal{I}_m; \texttt{write}\, Y$$

$$\mathcal{I} ::= X := e \mid \texttt{if}\, X\, \texttt{then goto}\, \ell\, \texttt{else goto}\, \ell' \mid \texttt{goto}\, \ell$$

$$e, f ::= X \mid \mathsf{d} \mid \texttt{cons}\ \ Y\, Z \mid \texttt{hd}\, X \mid \texttt{tl}\, X \mid =?\, Y\, Z$$

# El lenguaje GOTO

- $\mathcal{D}_{\mathcal{L}} = \mathbb{T}$
- $\mathcal{P}_{\mathcal{L}}$ se define mediante la siguiente gramática

  $$\mathcal{P}_{\mathcal{L}} \ni p ::= \texttt{read}\, X; 1 : \mathcal{I}_1; \ldots ; m : \mathcal{I}_m; \texttt{write}\, Y$$

  $$\mathcal{I} ::= X := e \mid \texttt{if}\, X\, \texttt{then goto}\, \ell\, \texttt{else goto}\, \ell' \mid \texttt{goto}\, \ell$$

  $$e, f ::= X \mid \mathsf{d} \mid \texttt{cons}\;\; Y\, Z \mid \texttt{hd}\, X \mid \texttt{tl}\, X \mid =?\, Y\, Z$$

  donde $\ell, \ell' \in \mathbb{N}$.

# El lenguaje GOTO

- $\mathcal{D}_{\mathcal{L}} = \mathbb{T}$
- $\mathcal{P}_{\mathcal{L}}$ se define mediante la siguiente gramática

  $\mathcal{P}_{\mathcal{L}} \ni p ::= \texttt{read}\, X; 1 : \mathcal{I}_1; \ldots; m : \mathcal{I}_m; \texttt{write}\, Y$

  $\mathcal{I} ::= X := e \mid \texttt{if}\, X\, \texttt{then goto}\, \ell\, \texttt{else goto}\, \ell' \mid \texttt{goto}\, \ell$

  $e, f ::= X \mid \mathsf{d} \mid \texttt{cons}\ \ Y\ Z \mid \texttt{hd}\, X \mid \texttt{tl}\, X \mid =?\, Y\ Z$

  donde $\ell, \ell' \in \mathbb{N}$.
- Obsérvese que, a diferencia de WHILE , en este lenguaje en cada expresión $e$ hay exactamente una presencia de operador

# El lenguaje GOTO

- $\mathcal{D}_{\mathcal{L}} = \mathbb{T}$
- $\mathcal{P}_{\mathcal{L}}$ se define mediante la siguiente gramática

$$\mathcal{P}_{\mathcal{L}} \ni p ::= \texttt{read}\, X; 1 : \mathcal{I}_1; \ldots ; m : \mathcal{I}_m; \texttt{write}\, Y$$

$$\mathcal{I} ::= X := e \mid \texttt{if}\, X\, \texttt{then goto}\, \ell\, \texttt{else goto}\, \ell' \mid \texttt{goto}\, \ell$$

$$e, f ::= X \mid \mathsf{d} \mid \texttt{cons}\ \ Y\ Z \mid \texttt{hd}\, X \mid \texttt{tl}\, X \mid =?\, Y\ Z$$

donde $\ell, \ell' \in \mathbb{N}$.

- Obsérvese que, a diferencia de WHILE , en este lenguaje en cada expresión $e$ hay exactamente una presencia de operador
- Además la longitud $|\vec{\mathcal{I}}|$ de una secuencia de instrucciones es simplemente el número de instrucciones que la componen.

# Suma de dos numerales

## GOTO

```
read XY;
 1: X:= hd XY;
 2: Y:= tl XY;
 3: if X then goto 4 else goto 7;
 4: Y:= cons nil Y;
 5: X:= tl X;
 6: goto 3;
write Y.
```

# Semántica operacional

## GOTO

- Memorias: $\mathcal{S} = \{\sigma |\ \sigma : \textit{Var} \to \mathbb{T}\}$

# Semántica operacional

## GOTO

- Memorias: $\mathcal{S} = \{\sigma | \ \sigma : \mathit{Var} \to \mathbb{T}\}$
- Estados: $\mathcal{E} = \mathbb{N} \times \mathcal{S}$.

# Semántica operacional

## GOTO

- Memorias: $\mathcal{S} = \{\sigma |\ \sigma : \mathit{Var} \to \mathbb{T}\}$
- Estados: $\mathcal{E} = \mathbb{N} \times \mathcal{S}$.
- Evaluación de expresiones: si $s = \langle \ell, \sigma \rangle$ entonces definimos la función de evaluación $ev_s : \mathit{Expr} \to \mathcal{D}_{\mathcal{L}}$ como sigue:

# Semántica operacional

## GOTO

- Memorias: $\mathcal{S} = \{\sigma |\ \sigma : \mathit{Var} \to \mathbb{T}\}$
- Estados: $\mathcal{E} = \mathbb{N} \times \mathcal{S}$.
- Evaluación de expresiones: si $s = \langle \ell, \sigma \rangle$ entonces definimos la función de evaluación $ev_s : \mathit{Expr} \to \mathcal{D}_{\mathcal{L}}$ como sigue:
  - $ev_s(X) = \sigma(X)$

# Semántica operacional

## GOTO

- Memorias: $\mathcal{S} = \{\sigma | \ \sigma : \mathit{Var} \to \mathbb{T}\}$
- Estados: $\mathcal{E} = \mathbb{N} \times \mathcal{S}$.
- Evaluación de expresiones: si $s = \langle \ell, \sigma \rangle$ entonces definimos la función de evaluación $ev_s : \mathit{Expr} \to \mathcal{D}_{\mathcal{L}}$ como sigue:
  - $ev_s(X) = \sigma(X)$
  - $ev_s(d) = d$

# Semántica operacional

## GOTO

- Memorias: $\mathcal{S} = \{\sigma |\ \sigma : \mathit{Var} \to \mathbb{T}\}$
- Estados: $\mathcal{E} = \mathbb{N} \times \mathcal{S}$.
- Evaluación de expresiones: si $s = \langle \ell, \sigma \rangle$ entonces definimos la función de evaluación $ev_s : \mathit{Expr} \to \mathcal{D}_{\mathcal{L}}$ como sigue:
  - $ev_s(X) = \sigma(X)$
  - $ev_s(d) = d$
  - $ev_s(\texttt{hd}\, X) = nil$, si $\sigma(X) = nil$

# Semántica operacional

## GOTO

- Memorias: $\mathcal{S} = \{\sigma |\ \sigma : \mathit{Var} \to \mathbb{T}\}$
- Estados: $\mathcal{E} = \mathbb{N} \times \mathcal{S}$.
- Evaluación de expresiones: si $s = \langle \ell, \sigma \rangle$ entonces definimos la función de evaluación $ev_s : \mathit{Expr} \to \mathcal{D}_{\mathcal{L}}$ como sigue:
  - $ev_s(X) = \sigma(X)$
  - $ev_s(d) = d$
  - $ev_s(\mathtt{hd}\, X) = nil$, si $\sigma(X) = nil$
  - $ev_s(\mathtt{hd}\, X) = d$, si $\sigma(X) = (d.e)$

# Semántica operacional

## Goto

- Memorias: $\mathcal{S} = \{\sigma | \ \sigma : \mathit{Var} \to \mathbb{T}\}$
- Estados: $\mathcal{E} = \mathbb{N} \times \mathcal{S}$.
- Evaluación de expresiones: si $s = \langle \ell, \sigma \rangle$ entonces definimos la función de evaluación $ev_s : \mathit{Expr} \to \mathcal{D}_{\mathcal{L}}$ como sigue:
  - $ev_s(X) = \sigma(X)$
  - $ev_s(d) = d$
  - $ev_s(\mathtt{hd}\, X) = nil$, si $\sigma(X) = nil$
  - $ev_s(\mathtt{hd}\, X) = d$, si $\sigma(X) = (d.e)$
  - $ev_s(\mathtt{tl}\, X) = nil$, si $\sigma(X) = nil$

# Semántica operacional

## GOTO

- Memorias: $\mathcal{S} = \{\sigma |\ \sigma : \mathit{Var} \to \mathbb{T}\}$
- Estados: $\mathcal{E} = \mathbb{N} \times \mathcal{S}$.
- Evaluación de expresiones: si $s = \langle \ell, \sigma \rangle$ entonces definimos la función de evaluación $ev_s : \mathit{Expr} \to \mathcal{D}_{\mathcal{L}}$ como sigue:
  - $ev_s(X) = \sigma(X)$
  - $ev_s(d) = d$
  - $ev_s(\mathtt{hd}\, X) = nil$, si $\sigma(X) = nil$
  - $ev_s(\mathtt{hd}\, X) = d$, si $\sigma(X) = (d.e)$
  - $ev_s(\mathtt{tl}\, X) = nil$, si $\sigma(X) = nil$
  - $ev_s(\mathtt{tl}\, X) = e$, si $\sigma(X) = (d.e)$

# Semántica operacional

## GOTO

- Memorias: $\mathcal{S} = \{\sigma | \ \sigma : \mathit{Var} \to \mathbb{T}\}$
- Estados: $\mathcal{E} = \mathbb{N} \times \mathcal{S}$.
- Evaluación de expresiones: si $s = \langle \ell, \sigma \rangle$ entonces definimos la función de evaluación $ev_s : \mathit{Expr} \to \mathcal{D}_{\mathcal{L}}$ como sigue:
  - $ev_s(X) = \sigma(X)$
  - $ev_s(d) = d$
  - $ev_s(\mathtt{hd}\, X) = nil$, si $\sigma(X) = nil$
  - $ev_s(\mathtt{hd}\, X) = d$, si $\sigma(X) = (d.e)$
  - $ev_s(\mathtt{tl}\, X) = nil$, si $\sigma(X) = nil$
  - $ev_s(\mathtt{tl}\, X) = e$, si $\sigma(X) = (d.e)$
  - $ev_s(\mathtt{cons}\, X\ Y) = (d.e)$ si $\sigma(X) = d,\ \sigma(Y) = e$

# Semántica operacional

## GOTO

- Memorias: $\mathcal{S} = \{\sigma |\ \sigma : \mathit{Var} \to \mathbb{T}\}$
- Estados: $\mathcal{E} = \mathbb{N} \times \mathcal{S}$.
- Evaluación de expresiones: si $s = \langle \ell, \sigma \rangle$ entonces definimos la función de evaluación $ev_s : \mathit{Expr} \to \mathcal{D}_{\mathcal{L}}$ como sigue:
  - $ev_s(X) = \sigma(X)$
  - $ev_s(d) = d$
  - $ev_s(\texttt{hd}\, X) = nil$, si $\sigma(X) = nil$
  - $ev_s(\texttt{hd}\, X) = d$, si $\sigma(X) = (d.e)$
  - $ev_s(\texttt{tl}\, X) = nil$, si $\sigma(X) = nil$
  - $ev_s(\texttt{tl}\, X) = e$, si $\sigma(X) = (d.e)$
  - $ev_s(\texttt{cons}\, X\ Y) = (d.e)$ si $\sigma(X) = d,\ \sigma(Y) = e$
  - $ev_s(=?X\ Y) = \mathit{true}$ si $\sigma(X) = \sigma(Y)$.

# Semántica operacional

## GOTO

- Memorias: $\mathcal{S} = \{\sigma |\ \sigma : \mathit{Var} \to \mathbb{T}\}$
- Estados: $\mathcal{E} = \mathbb{N} \times \mathcal{S}$.
- Evaluación de expresiones: si $s = \langle \ell, \sigma \rangle$ entonces definimos la función de evaluación $ev_s : \mathit{Expr} \to \mathcal{D}_{\mathcal{L}}$ como sigue:
  - $ev_s(X) = \sigma(X)$
  - $ev_s(d) = d$
  - $ev_s(\texttt{hd}\, X) = nil$, si $\sigma(X) = nil$
  - $ev_s(\texttt{hd}\, X) = d$, si $\sigma(X) = (d.e)$
  - $ev_s(\texttt{tl}\, X) = nil$, si $\sigma(X) = nil$
  - $ev_s(\texttt{tl}\, X) = e$, si $\sigma(X) = (d.e)$
  - $ev_s(\texttt{cons}\, X\ Y) = (d.e)$ si $\sigma(X) = d,\ \sigma(Y) = e$
  - $ev_s(=?X\ Y) = \mathit{true}$ si $\sigma(X) = \sigma(Y)$.
  - $ev_s(=?X\ Y) = \mathit{false}$ si $\sigma(X) \neq \sigma(Y)$.

# Semántica operacional

## GOTO

- Memorias: $\mathcal{S} = \{\sigma | \ \sigma : \mathit{Var} \to \mathbb{T}\}$
- Estados: $\mathcal{E} = \mathbb{N} \times \mathcal{S}$.
- Evaluación de expresiones: si $s = \langle \ell, \sigma \rangle$ entonces definimos la función de evaluación $ev_s : \mathit{Expr} \to \mathcal{D}_{\mathcal{L}}$ como sigue:
  - $ev_s(X) = \sigma(X)$
  - $ev_s(d) = d$
  - $ev_s(\mathtt{hd}\, X) = \mathit{nil}$, si $\sigma(X) = \mathit{nil}$
  - $ev_s(\mathtt{hd}\, X) = d$, si $\sigma(X) = (d.e)$
  - $ev_s(\mathtt{tl}\, X) = \mathit{nil}$, si $\sigma(X) = \mathit{nil}$
  - $ev_s(\mathtt{tl}\, X) = e$, si $\sigma(X) = (d.e)$
  - $ev_s(\mathtt{cons}\, X \ \ Y) = (d.e)$ si $\sigma(X) = d, \ \sigma(Y) = e$
  - $ev_s(=?X \ \ Y) = \mathit{true}$ si $\sigma(X) = \sigma(Y)$.
  - $ev_s(=?X \ \ Y) = \mathit{false}$ si $\sigma(X) \neq \sigma(Y)$.
- Obsérvese que la función $ev_s$ no es recursiva a diferencia de la función correspondiente en el lenguaje WHILE .

# Relación de transición

## Goto

- Si $\mathcal{I}_\ell =_{def} X := e$ y $\text{ev}_{\langle \ell,\sigma \rangle}(e) = d$ entonces

$$p \rhd \langle \ell, \sigma \rangle \to \langle \ell + 1, \sigma[X/d] \rangle$$

# Relación de transición

## GOTO

- Si $\mathcal{I}_\ell =_{def} X := e$ y $\mathsf{ev}_{\langle \ell,\sigma\rangle}(e) = d$ entonces

$$p \rhd \langle \ell, \sigma\rangle \to \langle \ell + 1, \sigma[X/d]\rangle$$

- Si $\mathcal{I}_\ell = \texttt{if}\ X\ \texttt{then goto}\ \ell'\ \texttt{else goto}\ \ell''$ y $\sigma(X) = \texttt{nil}$ entonces

$$p \rhd \langle \ell, \sigma\rangle \to \langle \ell'', \sigma\rangle$$

# Relación de transición

## Goto

- Si $\mathcal{I}_\ell =_{def} X := e$ y $\text{ev}_{\langle \ell,\sigma\rangle}(e) = d$ entonces

$$p \triangleright \langle \ell, \sigma \rangle \to \langle \ell + 1, \sigma[X/d] \rangle$$

- Si $\mathcal{I}_\ell = \texttt{if}\ X\ \texttt{then goto}\ \ell'\ \texttt{else goto}\ \ell''$ y $\sigma(X) = \texttt{nil}$ entonces

$$p \triangleright \langle \ell, \sigma \rangle \to \langle \ell'', \sigma \rangle$$

- Si $\mathcal{I}_\ell = \texttt{if}\ X\ \texttt{then goto}\ \ell'\ \texttt{else goto}\ \ell''$ y $\sigma(X) \neq \texttt{nil}$ entonces

$$p \triangleright \langle \ell, \sigma \rangle \to \langle \ell', \sigma \rangle$$

# Relación de transición

## GOTO

- Si $\mathcal{I}_\ell =_{def} X := e$ y $\mathsf{ev}_{\langle \ell,\sigma\rangle}(e) = d$ entonces

$$p \triangleright \langle \ell, \sigma\rangle \to \langle \ell + 1, \sigma[X/d]\rangle$$

- Si $\mathcal{I}_\ell = \texttt{if}\, X\, \texttt{then goto}\, \ell' \,\texttt{else goto}\, \ell''$ y $\sigma(X) = \texttt{nil}$ entonces

$$p \triangleright \langle \ell, \sigma\rangle \to \langle \ell'', \sigma\rangle$$

- Si $\mathcal{I}_\ell = \texttt{if}\, X\, \texttt{then goto}\, \ell' \,\texttt{else goto}\, \ell''$ y $\sigma(X) \neq \texttt{nil}$ entonces

$$p \triangleright \langle \ell, \sigma\rangle \to \langle \ell', \sigma\rangle$$

- Si $\mathcal{I}_\ell = \texttt{goto}\, \ell'$ entonces

$$p \triangleright \langle \ell, \sigma\rangle \to \langle \ell', \sigma\rangle$$

# Semántica operacional

## GOTO

$$p = \texttt{read}\, X; 1 : \mathcal{I}_1; \ldots ; m : \mathcal{I}_m; \texttt{write}\, Y.$$

- Estados finales: $\text{final}(p) = \{\langle m+1, \sigma\rangle \mid \sigma \in \mathcal{S}\}$.

# Semántica operacional

## GOTO

$$p = \texttt{read}\, X; 1 : \mathcal{I}_1; \ldots ; m : \mathcal{I}_m; \texttt{write}\, Y.$$

- Estados finales: final$(p) = \{\langle m+1, \sigma\rangle \mid \sigma \in \mathcal{S}\}$.
- Función de inicialización: init : $\mathcal{P}_{\mathcal{L}} \times \mathcal{D}_{\mathcal{L}} \to \mathcal{E}$

$$\text{init}(p, d) = \big\langle 1, [X \mapsto d, Z_1 \mapsto \texttt{nil}, \ldots, Z_n \mapsto \texttt{nil}]\big\rangle$$

donde $Vars(p) = \{X, Y, Z_1, \ldots, Z_n\}$.

# Semántica operacional

## Goto

$$p = \texttt{read}\, X; 1 : \mathcal{I}_1; \ldots; m : \mathcal{I}_m; \texttt{write}\, Y.$$

- Estados finales: $\text{final}(p) = \{\langle m+1, \sigma\rangle \mid \sigma \in \mathcal{S}\}$.
- Función de inicialización: $\text{init} : \mathcal{P}_{\mathcal{L}} \times \mathcal{D}_{\mathcal{L}} \to \mathcal{E}$

$$\text{init}(p, d) = \big\langle 1, [X \mapsto d, Z_1 \mapsto \texttt{nil}, \ldots, Z_n \mapsto \texttt{nil}]\big\rangle$$

donde $\mathit{Vars}(p) = \{X, Y, Z_1, \ldots, Z_n\}$.

- Función de salida: $\text{output} : \mathcal{P}_{\mathcal{L}} \times \mathcal{E} \to \mathcal{D}_{\mathcal{L}},\ \text{output}(p, \sigma) = \sigma(Y)$

# Compilador

## Equivalencia de lenguajes de programación

Sean $\mathcal{S} = \langle \mathcal{P}_\mathcal{S}, \mathcal{D}_\mathcal{S}, [\![\cdot]\!]_\mathcal{S} \rangle$ y $\mathcal{T} = \langle \mathcal{P}_\mathcal{T}, \mathcal{D}_\mathcal{T}, [\![\cdot]\!]_\mathcal{T} \rangle$ dos lenguajes de programación.

- Un compilador de $\mathcal{S}$ en $\mathcal{T}$ es un par de funciones $\mathcal{C} = \langle F, c \rangle$ tal que $F : \mathcal{P}_\mathcal{S} \to \mathcal{P}_\mathcal{T},\ c : \mathcal{D}_\mathcal{S} \to \mathcal{D}_\mathcal{T}$ y

# Compilador

## Equivalencia de lenguajes de programación

Sean $\mathcal{S} = \langle \mathcal{P}_\mathcal{S}, \mathcal{D}_\mathcal{S}, [\![\cdot]\!]_\mathcal{S} \rangle$ y $\mathcal{T} = \langle \mathcal{P}_\mathcal{T}, \mathcal{D}_\mathcal{T}, [\![\cdot]\!]_\mathcal{T} \rangle$ dos lenguajes de programación.

- Un compilador de $\mathcal{S}$ en $\mathcal{T}$ es un par de funciones $\mathcal{C} = \langle F, c \rangle$ tal que $F : \mathcal{P}_\mathcal{S} \to \mathcal{P}_\mathcal{T},\ c : \mathcal{D}_\mathcal{S} \to \mathcal{D}_\mathcal{T}$ y
- se cumple que

$$\forall p \in \mathcal{P}_\mathcal{S} \forall x \in \mathcal{D}_\mathcal{S} \big( c([\![p]\!]_\mathcal{S}(x)) = [\![F(p)]\!]_\mathcal{T}(c(x)) \big)$$

# Compilador

## Equivalencia de lenguajes de programación

Sean $\mathcal{S} = \langle \mathcal{P}_\mathcal{S}, \mathcal{D}_\mathcal{S}, [\![\cdot]\!]_\mathcal{S} \rangle$ y $\mathcal{T} = \langle \mathcal{P}_\mathcal{T}, \mathcal{D}_\mathcal{T}, [\![\cdot]\!]_\mathcal{T} \rangle$ dos lenguajes de programación.

- Un compilador de $\mathcal{S}$ en $\mathcal{T}$ es un par de funciones $\mathcal{C} = \langle F, c \rangle$ tal que $F : \mathcal{P}_\mathcal{S} \to \mathcal{P}_\mathcal{T}, \ c : \mathcal{D}_\mathcal{S} \to \mathcal{D}_\mathcal{T}$ y
- se cumple que

$$\forall p \in \mathcal{P}_\mathcal{S} \forall x \in \mathcal{D}_\mathcal{S} \big( c([\![p]\!]_\mathcal{S}(x)) = [\![F(p)]\!]_\mathcal{T}(c(x)) \big)$$

- o equivalentemente: $\forall p \in \mathcal{P}_\mathcal{S} \forall x \in \mathcal{D}_\mathcal{S} \forall y \in \mathcal{D}_\mathcal{T}$

$$[\![p]\!]_\mathcal{S}(x) = y \text{ si y sólo si } [\![F(p)]\!]_\mathcal{T}(c(x)) = c(y)$$

# De *While* a *Goto*

- *While*1: restricción de *While* de modo que cada expresión contiene un único operador.

# De *While* a *Goto*

- *While*1: restricción de *While* de modo que cada expresión contiene un único operador.
- *While*1 es equivalente a *While*.

# De *While* a *Goto*

- *While*1: restricción de *While* de modo que cada expresión contiene un único operador.
- *While*1 es equivalente a *While*.
- Basta transformar instrucciones que contengan expresiones con más de un operador mediante el uso de variables auxilares.

# De *While* a *Goto*

- *While*1: restricción de *While* de modo que cada expresión contiene un único operador.
- *While*1 es equivalente a *While*.
- Basta transformar instrucciones que contengan expresiones con más de un operador mediante el uso de variables auxilares.
- Por ejemplo $X := \mathtt{cons}\, X(\mathtt{hd}\, Y)$ se convierte en

$$W := \mathtt{hd}\, Y; X := \mathtt{cons}\, X\ W$$

# Traducción de comandos de WHILE1 a GOTO

## $C \mapsto \widehat{\ell} : C^\star$

Sea $C$ un comando del lenguaje WHILE1 y $\ell$ una etiqueta. Definimos la secuencia de comandos etiquetados del lenguaje GOTO $\widehat{\ell} : C^\star$como sigue:

- Si $C =_{def} X := e$ entonces $\widehat{\ell} : C^\star =_{def} X := e$.

# Traducción de comandos de WHILE1 a GOTO

$C \mapsto \widehat{\ell} : C^\star$

Sea $C$ un comando del lenguaje WHILE1 y $\ell$ una etiqueta. Definimos la secuencia de comandos etiquetados del lenguaje GOTO $\widehat{\ell} : C^\star$como sigue:

- Si $C =_{def} X := e$ entonces $\widehat{\ell} : C^\star =_{def} X := e$.
- Si $C =_{def} \mathtt{while}\, X\, \mathtt{do}\;\; D_1; \ldots; D_k\; \mathtt{end}$ entonces

$$
\begin{aligned}
C^\star \quad =_{def} \quad & \widehat{\ell} : \mathtt{if}\, X\, \mathtt{then}\; \mathtt{goto}\, \widehat{\ell} + 1\; \mathtt{else}\; \mathtt{goto}\, \widehat{\ell_{k+1}} + 1; \\
& \ell + 1 : D_1^\star; \\
& \widehat{\ell_2} : D_2^\star; \\
& \vdots \\
& \widehat{\ell_k} : D_k^\star \\
& \widehat{\ell_{k+1}} : \mathtt{goto}\, \ell;
\end{aligned}
$$

# Traducción de programas de WHILE1 a GOTO

$p \mapsto p_G$

- Dado $p$ en WHILE1 :

$$p =_{def} \texttt{read}\, X; C_1; \ldots; C_m; \texttt{write}\, Y$$

# Traducción de programas de WHILE1 a GOTO

*$p \mapsto p_G$*

- Dado $p$ en WHILE1 :

$$p =_{def} \texttt{read}\, X; C_1; \ldots; C_m; \texttt{write}\, Y$$

- Definimos el programa $p_G$ en GOTO como:

$$p_G =_{def} \texttt{read}\, X; 1 : C_1^\star; \widehat{\ell_2} : C_2^\star; \ldots; \widehat{\ell_m} : C_m^\star; \texttt{write}\, Y$$

# Traducción de programas de WHILE1 a GOTO

$p \mapsto p_G$

- Dado $p$ en WHILE1 :

$$p =_{def} \texttt{read}\, X; C_1; \ldots; C_m; \texttt{write}\, Y$$

- Definimos el programa $p_G$ en GOTO como:

$$p_G =_{def} \texttt{read}\, X; 1 : C_1^\star; \widehat{\ell_2} : C_2^\star; \ldots; \widehat{\ell_m} : C_m^\star; \texttt{write}\, Y$$

- donde $\widehat{\ell_2} = |C_1^\star| + 1$ y $\widehat{\ell_{j+1}} = \widehat{\ell_j} + |C_j^\star|$ para $1 < j < m$.

# De WHILE1 a GOTO

- Traducción de estados: sea $s = \langle \ell, \sigma \rangle$ un estado en WHILE1 . Definimos el estado $\bar{s}$ en GOTO como $\bar{s} =_{def} \langle \widehat{l}, \sigma \rangle$

# De WHILE1 a GOTO

- Traducción de estados: sea $s = \langle \ell, \sigma \rangle$ un estado en WHILE1 . Definimos el estado $\bar{s}$ en GOTO como $\bar{s} =_{def} \langle \widehat{l}, \sigma \rangle$
- Simulación:

$$\text{Si } p \rhd_{\text{WHILE1}} s \to s' \text{ entonces } p_G \rhd_{\text{GOTO}} \bar{s} \to^{\star} \bar{s}'$$

## De WHILE1 a GOTO

- Traducción de estados: sea $s = \langle \ell, \sigma \rangle$ un estado en WHILE1 . Definimos el estado $\bar{s}$ en GOTO como $\bar{s} =_{def} \langle \widehat{l}, \sigma \rangle$
- Simulación:

$$\text{Si } p \rhd_{\text{WHILE1}} s \to s' \text{ entonces } p_G \rhd_{\text{GOTO}} \bar{s} \to^{\star} \bar{s'}$$

- Compilación: Sea $\mathcal{C} = \langle F, c \rangle$ con $F : \mathcal{P}_{\text{WHILE1}} \to \mathcal{P}_{\text{GOTO}}$ tal que $F(p) = p_G$ y $c = id$ la función identidad. $\mathcal{C}$ es un compilador de WHILE1 a GOTO .

# Transformación de Böhm-Jacopini $\ell : \mathcal{I}_\ell \mapsto \mathcal{I}_\ell^\star$

## De GOTO a WHILE

Dada una instrucción etiquetada $\ell : \mathcal{I}_\ell$ del lenguaje GOTO . Definimos la secuencia de comandos $\mathcal{I}_\ell^\star$ del lenguaje WHILE de acuerdo a los siguientes casos:

- Si $\mathcal{I}_\ell =_{def} X := e$ entonces

$$\mathcal{I}_\ell^\star =_{def} X := e; PC := \ell + 1$$

# Transformación de Böhm-Jacopini $\ell : \mathcal{I}_\ell \mapsto \mathcal{I}_\ell^\star$

## De GOTO a WHILE

Dada una instrucción etiquetada $\ell : \mathcal{I}_\ell$ del lenguaje GOTO . Definimos la secuencia de comandos $\mathcal{I}_\ell^\star$ del lenguaje WHILE de acuerdo a los siguientes casos:

- Si $\mathcal{I}_\ell =_{def} X := e$ entonces

$$\mathcal{I}_\ell^\star =_{def} X := e; PC := \ell + 1$$

- Si $\mathcal{I}_\ell =_{def} \texttt{if}\, X\, \texttt{then goto}\, \ell'\, \texttt{else goto}\, \ell''$ entonces

$$\mathcal{I}_\ell^\star =_{def} \texttt{if}\, X\, \texttt{then}\, PC := \ell'\, \texttt{else}\ \ PC := \ell''$$

# Transformación de Böhm-Jacopini $\ell : \mathcal{I}_\ell \mapsto \mathcal{I}_\ell^\star$

## De GOTO a WHILE

Dada una instrucción etiquetada $\ell : \mathcal{I}_\ell$ del lenguaje GOTO . Definimos la secuencia de comandos $\mathcal{I}_\ell^\star$ del lenguaje WHILE de acuerdo a los siguientes casos:

- Si $\mathcal{I}_\ell =_{def} X := e$ entonces

$$\mathcal{I}_\ell^\star =_{def} X := e; PC := \ell + 1$$

- Si $\mathcal{I}_\ell =_{def}$ `if` $X$ `then goto` $\ell'$ `else goto` $\ell''$ entonces

$$\mathcal{I}_\ell^\star =_{def} \texttt{if}\ X\ \texttt{then}\ PC := \ell'\ \texttt{else}\ \ PC := \ell''$$

- Si $\mathcal{I}_\ell =_{def}$ `goto` $\ell'$ entonces $\mathcal{I}_\ell^\star =_{def} PC := \ell'$

# Transformación de Böhm-Jacopini $\ell : \mathcal{I}_\ell \mapsto \mathcal{I}_\ell^\star$



Dada una instrucción etiquetada $\ell : \mathcal{I}_\ell$ del lenguaje GOTO . Definimos la secuencia de comandos $\mathcal{I}_\ell^\star$ del lenguaje WHILE de acuerdo a los siguientes casos:

- Si $\mathcal{I}_\ell =_{def} X := e$ entonces

$$\mathcal{I}_\ell^\star =_{def} X := e; PC := \ell + 1$$

- Si $\mathcal{I}_\ell =_{def} \texttt{if}\ X\ \texttt{then goto}\ \ell'\ \texttt{else goto}\ \ell''$ entonces

$$\mathcal{I}_\ell^\star =_{def} \texttt{if}\ X\ \texttt{then}\ PC := \ell'\ \texttt{else}\ \ PC := \ell''$$

- Si $\mathcal{I}_\ell =_{def} \texttt{goto}\ \ell'$ entonces $\mathcal{I}_\ell^\star =_{def} PC := \ell'$
- Aquí $PC$ es una variable nueva

# Traducción de GOTO a WHILE

Si $p$ es un programa en GOTO ,

$$p =_{def} \texttt{read}\, X; 1 : \mathcal{I}_1; \ldots ; m : \mathcal{I}_m; \texttt{write}\, Y$$

definimos el programa $p_W$ en WHILE como:

# Traducción de GOTO a WHILE

Si $p$ es un programa en GOTO ,

$$p =_{def} \texttt{read}\, X; 1 : \mathcal{I}_1; \ldots ; m : \mathcal{I}_m; \texttt{write}\, Y$$

definimos el programa $p_W$ en WHILE como:

$$\begin{array}{lll} p_W & =_{def} & \texttt{read}\, X; \\ & & PC := 1; \\ & & \texttt{while}\ PC\ \texttt{do} \\ & & \quad \texttt{case}\ PC\ \texttt{of} \\ & & \qquad 1 \Rightarrow \mathcal{I}_1^\star; \\ & & \qquad \vdots \\ & & \qquad m \Rightarrow \mathcal{I}_m^\star; \\ & & \qquad \texttt{otherwise} \Rightarrow PC := 0; \\ & & \texttt{end}; \\ & & \texttt{write}\, Y \end{array}$$

# Ejemplo

## Reversa

```
read X;
  PC := 1;
  while PC do
   case PC of
    1 => Y:=nil;PC:=2;
    2 => if X then PC:=3 else PC:=7;
    3 => Z:=hd x; PC:=4;
    4 => Y:=cons Z Y;PC:=5;
    5 => X:=tl X; PC:= 6;
    6 => PC:=2;
    otherwise => PC:=0
  end
write Y
```

# De GOTO a WHILE1

- Traducción de estados: Sean $p = \texttt{read}\, X, 1 : \mathcal{I}_1, \ldots, m : \mathcal{I}_m, \texttt{write}\, Y$ un programa y $s = \langle \ell, \sigma \rangle$ un estado en GOTO . Definimos el estado $\bar{s}$ en WHILE como $\bar{s} =_{def} \langle \widetilde{\ell}, \widetilde{\sigma} \rangle$ donde $\widetilde{1} = 4$, $\widetilde{\ell + 1} = \widetilde{\ell} + |\widetilde{\mathcal{I}_\ell}|$ para $1 < \ell \leq m$ y $\widetilde{\sigma} = \sigma[PC/n]$ donde $n$ es un valor único que queda determinado hasta el momento de la ejecución.

# De GOTO a WHILE1

- Traducción de estados: Sean $p = \texttt{read}\, X, 1 : \mathcal{I}_1, \ldots, m : \mathcal{I}_m, \texttt{write}\, Y$ un programa y $s = \langle \ell, \sigma \rangle$ un estado en GOTO . Definimos el estado $\bar{s}$ en WHILE como $\bar{s} =_{def} \langle \widetilde{\ell}, \widetilde{\sigma} \rangle$ donde $\widetilde{1} = 4$, $\widetilde{\ell + 1} = \widetilde{\ell} + |\widetilde{\mathcal{I}_\ell}|$ para $1 < \ell \leq m$ y $\widetilde{\sigma} = \sigma[PC/n]$ donde $n$ es un valor único que queda determinado hasta el momento de la ejecución.

- Simulación: Si $p \triangleright_{\text{GOTO}}\ s \to s'$ entonces $p_W \triangleright_{\text{WHILE}}\ \bar{s} \to^{+} \bar{s'}$

# De GOTO a WHILE1

- Traducción de estados: Sean $p = \texttt{read}\, X, 1 : \mathcal{I}_1, \ldots, m : \mathcal{I}_m, \texttt{write}\, Y$ un programa y $s = \langle \ell, \sigma \rangle$ un estado en GOTO . Definimos el estado $\bar{s}$ en WHILE como $\bar{s} =_{def} \langle \widetilde{\ell}, \widetilde{\sigma} \rangle$ donde $\widetilde{1} = 4$, $\widetilde{\ell + 1} = \widetilde{\ell} + |\widetilde{\mathcal{I}_\ell}|$ para $1 < \ell \leq m$ y $\widetilde{\sigma} = \sigma[PC/n]$ donde $n$ es un valor único que queda determinado hasta el momento de la ejecución.

- Simulación: Si $p \rhd_{\text{GOTO}}\ s \to s'$ entonces $p_W \rhd_{\text{WHILE}}\ \bar{s} \to^+ \bar{s'}$

- Compilación: Sea $\mathcal{C} = \langle F, c \rangle$ con $F : \mathcal{P}_{\text{GOTO}} \to \mathcal{P}_{\text{WHILE1}}$ tal que $F(p) = p_W$ y $c = id$ es la función identidad. $\mathcal{C}$ es un compilador de GOTO a WHILE1

# Teorema de la programación estructurada

## Böhm-Jacopini

Cualquier programa no estructurado puede implementarse en un lenguaje estrictamente secuencial que contenga las siguientes instrucciones: secuencia (;) ,condicional (`if`) e iteración (`while`)

# De LTURING a GOTO

## Codificación de datos

La función de codificación $(\cdot)^\dagger : \{0, 1, \sqcup\}^\star \to \mathbb{T}$ se define como $w^\dagger = s_1^\dagger s_2^\dagger \dots s_n^\dagger$. donde $w = s_1 s_2 \dots s_n$ y

$$\sqcup^\dagger = nil, \;\; 0^\dagger = nil.nil, \;\; 1^\dagger = nil.(nil.nil)$$

# Traducción de instrucciones de LTURING a GOTO

$\ell : \mathcal{I}_\ell \mapsto \widehat{\ell} : \widetilde{\mathcal{I}_\ell}$

- Si $\mathcal{I}_\ell =_{def}$ `right` entonces

$$
\begin{array}{rcl}
\widehat{\ell} : \widetilde{\mathcal{I}_\ell} & =_{def} & \widehat{\ell} : A_1 := nil \\
& & \widehat{\ell}+1 : A_2 := (=?\ Rt\ A_1) \\
& & \widehat{\ell}+2 : \texttt{if}\ A_2\ \texttt{then goto}\ \widehat{\ell}+3\ \texttt{else goto}\ \widehat{\ell}+6 \\
& & \widehat{\ell}+3 : Lf := \texttt{cons}\ \ C\ Lf \\
& & \widehat{\ell}+4 : C := \texttt{\char32} \\
& & \widehat{\ell}+5 : \texttt{goto}\ \widehat{l+1} \\
& & \widehat{\ell}+6 : Lf := \texttt{cons}\ \ C\ Lf \\
& & \widehat{\ell}+7 : C := \texttt{hd}\ Rt \\
& & \widehat{\ell}+8 : Rt := \texttt{tl}\ Rt
\end{array}
$$

donde las variables $A_1, A_2$ son nuevas.

# Traducción de instrucciones de LTURING a GOTO

$\ell : \mathcal{I}_\ell \mapsto \widehat{\ell} : \widetilde{\mathcal{I}_\ell}$

- Si $\mathcal{I}_\ell =_{def}$ `right` entonces

$$
\begin{array}{lll}
\widehat{\ell} : \widetilde{\mathcal{I}_\ell} & =_{def} & \widehat{\ell} : A_1 := nil \\
& & \widehat{\ell} + 1 : A_2 := (=?\ Rt\ A_1) \\
& & \widehat{\ell} + 2 : \texttt{if } A_2 \texttt{ then goto } \widehat{\ell} + 3 \texttt{ else goto } \widehat{\ell} + 6 \\
& & \widehat{\ell} + 3 : Lf := \texttt{cons }\ C\ Lf \\
& & \widehat{\ell} + 4 : C := \textvisiblespace \\
& & \widehat{\ell} + 5 : \texttt{goto } \widehat{l+1} \\
& & \widehat{\ell} + 6 : Lf := \texttt{cons }\ C\ Lf \\
& & \widehat{\ell} + 7 : C := \texttt{hd } Rt \\
& & \widehat{\ell} + 8 : Rt := \texttt{tl } Rt
\end{array}
$$

  donde las variables $A_1, A_2$ son nuevas.
- Si $\mathcal{I}_\ell =_{def}$ `left` entonces $\widehat{\ell} : \widetilde{\mathcal{I}_\ell}$ se define análogamente al caso anterior.

# Traducción de instrucciones de LTURING a GOTO

$\ell : \mathcal{I}_\ell \mapsto \widehat{\ell} : \widetilde{\mathcal{I}_\ell}$

- Si $\mathcal{I}_\ell =_{def}$ `write` $s$ entonces $\widehat{\ell} : \widetilde{\mathcal{I}_\ell} =_{def} \widehat{\ell} : C := s$

## Traducción de instrucciones de LTURING a GOTO

$\ell : \mathcal{I}_\ell \mapsto \widehat{\ell} : \widetilde{\mathcal{I}_\ell}$

- Si $\mathcal{I}_\ell =_{def}$ `write` $s$ entonces $\widehat{\ell} : \widetilde{\mathcal{I}_\ell} =_{def} \widehat{\ell} : C := s$
- Si $\mathcal{I}_\ell =_{def}$ `if` $s$ `then goto` $\ell'$ `else goto` $\ell''$ entonces

$$\begin{array}{lll} \widehat{\ell} : \widetilde{\mathcal{I}_\ell} & =_{def} & \widehat{\ell} : A_1 := s \\ & & \widehat{\ell}+1 : A_2 := (=?\ C\ A_1) \\ & & \widehat{\ell}+2 : \texttt{if}\ A_2\ \texttt{then goto}\ \widehat{\ell'}\ \texttt{else goto}\ \widehat{\ell''} \end{array}$$

donde las variables $A_1, A_2$ son nuevas.

# Traducción de instrucciones de LTURING a GOTO

$\ell : \mathcal{I}_\ell \mapsto \widehat{\ell} : \widetilde{\mathcal{I}_\ell}$

- Si $\mathcal{I}_\ell =_{def}$ `write` $s$ entonces $\widehat{\ell} : \widetilde{\mathcal{I}_\ell} =_{def} \widehat{\ell} : C := s$
- Si $\mathcal{I}_\ell =_{def}$ `if` $s$ `then goto` $\ell'$ `else goto` $\ell''$ entonces

$$\begin{array}{lll} \widehat{\ell} : \widetilde{\mathcal{I}_\ell} & =_{def} & \widehat{\ell} : A_1 := s \\ & & \widehat{\ell} + 1 : A_2 := (=?\ C\ A_1) \\ & & \widehat{\ell} + 2 : \texttt{if}\ A_2\ \texttt{then goto}\ \widehat{\ell'}\ \texttt{else goto}\ \widehat{\ell''} \end{array}$$

  donde las variables $A_1, A_2$ son nuevas.

- Si $\mathcal{I}_\ell =_{def}$ `goto` $\ell'$ entonces $\widetilde{\mathcal{I}_\ell} =_{def}$ `goto` $\widehat{\ell'}$

# Traducción de LTURING a GOTO

Dado un programa $p$ en LTURING

$$p =_{def} 1 : \mathcal{I}_1, 2 : \mathcal{I}_2, \ldots, m : \mathcal{I}_m; m + 1 : \texttt{halt}$$

# Traducción de LTuring a Goto

Dado un programa $p$ en LTuring

$$p =_{def} 1 : \mathcal{I}_1, 2 : \mathcal{I}_2, \ldots, m : \mathcal{I}_m; m+1 : \texttt{halt}$$

definimos el programa $p_G$ en Goto como sigue:

$$p_G =_{def} \texttt{read}\, Rt; 1 : \widetilde{\mathcal{I}_1}, \widehat{2} : \widetilde{\mathcal{I}_2}, \ldots, \widehat{m} : \widetilde{\mathcal{I}_m}; \texttt{write}\, Rt$$

donde $\widehat{2} = 1 + |\widetilde{\mathcal{I}_1}|$ y $\widehat{j+1} = |\widetilde{\mathcal{I}_j}| + \widehat{j},\ 1 < j < m$.

# De LTURING a GOTO

- Traducción de estados: Dado un estado $s =_{def} \langle \ell, L\underline{s}R \rangle$ en LTURING definimos el estado $\bar{s}$ en GOTO como $\bar{s} =_{def} \langle \widehat{\ell}, \sigma \rangle$ donde

$$\sigma =_{def} [Lf \mapsto (L^{rev})^{\dagger},\ C \mapsto s^{\dagger},\ Rt \mapsto R^{\dagger}]$$

  y $L^{rev}$ denota a la reversa de la cadena $L$.

# De LTURING a GOTO

- Traducción de estados: Dado un estado $s =_{def} \langle \ell, L\underline{s}R \rangle$ en LTURING definimos el estado $\bar{s}$ en GOTO como $\bar{s} =_{def} \langle \widehat{\ell}, \sigma \rangle$ donde

$$\sigma =_{def} [Lf \mapsto (L^{rev})^{\dagger},\ C \mapsto s^{\dagger},\ Rt \mapsto R^{\dagger}]$$

  y $L^{rev}$ denota a la reversa de la cadena $L$.

- Simulación: Si $p \triangleright_{\text{LTURING}}\ s \to s'$ entonces $p_G \triangleright_{\text{GOTO}}\ \bar{s} \to^{+} \bar{s'}$

# De LTURING a GOTO

- Traducción de estados: Dado un estado $s =_{def} \langle \ell, L\underline{s}R \rangle$ en LTURING definimos el estado $\bar{s}$ en GOTO como $\bar{s} =_{def} \langle \widehat{\ell}, \sigma \rangle$ donde

$$\sigma =_{def} [Lf \mapsto (L^{rev})^{\dagger},\ C \mapsto s^{\dagger},\ Rt \mapsto R^{\dagger}]$$

  y $L^{rev}$ denota a la reversa de la cadena $L$.

- Simulación: Si $p \rhd_{\text{LTURING}} s \to s'$ entonces $p_G \rhd_{\text{GOTO}} \bar{s} \to^{+} \bar{s}'$
- Compilación: Sea $\mathcal{C} = \langle F, c \rangle$ con

$$F(p) = p_G \text{ y } c(x) = x^{\dagger}.$$

  $\mathcal{C}$ es un compilador de LTURING a GOTO

# De GOTO a $\mathcal{MT}$

## Codificación de datos

Sea $\Sigma = \{0, (, ), \cdot, \llcorner\!\lrcorner\}$ El conjunto de datos de GOTO , $\mathcal{D}_{Goto} = \mathbb{T}$ se codifica en $\Sigma$ mediante la función $(\cdot)^{\ddagger} : \mathbb{T} \to \Sigma^{\star}$ como sigue:

- $nil^{\ddagger} = 0$

# De GOTO a $\mathcal{MT}$

## Codificación de datos

Sea $\Sigma = \{0, (, ), \cdot, \sqcup\}$ El conjunto de datos de GOTO , $\mathcal{D}_{Goto} = \mathbb{T}$ se codifica en $\Sigma$ mediante la función $(\cdot)^{\ddagger} : \mathbb{T} \to \Sigma^{\star}$ como sigue:

- $nil^{\ddagger} = 0$
- $(x.y)^{\ddagger} = (x^{\ddagger} \, . \, y^{\ddagger})$

# De GOTO a $\mathcal{MT}$

## Codificación de datos

Sea $\Sigma = \{0, (, ), \cdot, \llcorner\!\lrcorner\}$ El conjunto de datos de GOTO , $\mathcal{D}_{Goto} = \mathbb{T}$ se codifica en $\Sigma$ mediante la función $(\cdot)^{\ddagger} : \mathbb{T} \to \Sigma^{\star}$ como sigue:

- $nil^{\ddagger} = 0$
- $(x.y)^{\ddagger} = (x^{\ddagger} \,.\, y^{\ddagger})$
- Por ejemplo el árbol $(nil.(nil.nil))$ se codifica con la cadena $(0.(0.0))$

# Macros en $\mathcal{MT}$ multicinta

Sea $M$ una máquina de Turing multicinta cuyo alfabeto codifica expresiones del lenguaje GOTO . Los siguientes macros son implementables.

1. `C.E`: mover la cabeza de todas las cintas al primer blanco a la izquierda de la cadena actual (configuración estandar).

# Macros en $\mathcal{MT}$ multicinta

Sea $M$ una máquina de Turing multicinta cuyo alfabeto codifica expresiones del lenguaje GOTO . Los siguientes macros son implementables.

1. `C.E`: mover la cabeza de todas las cintas al primer blanco a la izquierda de la cadena actual (configuración estandar).
2. `copy C to D`: copia a la cinta `D` el contenido de la cinta `C`.

# Macros en $\mathcal{MT}$ multicinta

Sea $M$ una máquina de Turing multicinta cuyo alfabeto codifica expresiones del lenguaje GOTO . Los siguientes macros son implementables.

1. `C.E`: mover la cabeza de todas las cintas al primer blanco a la izquierda de la cadena actual (configuración estandar).
2. `copy C to D`: copia a la cinta `D` el contenido de la cinta `C`.
3. `erase` *C*: borra el contenido de la cinta C.

# Macros en $\mathcal{MT}$ multicinta

Sea *M* una máquina de Turing multicinta cuyo alfabeto codifica expresiones del lenguaje GOTO . Los siguientes macros son implementables.

1. `C.E`: mover la cabeza de todas las cintas al primer blanco a la izquierda de la cadena actual (configuración estandar).
2. `copy C to D`: copia a la cinta `D` el contenido de la cinta `C`.
3. `erase` *C*: borra el contenido de la cinta C.
4. `compute e in C`: calcula el valor de la GOTO-expresión *e* (codificada), escribiendo el resultado en la cinta C

# Macros en $\mathcal{MT}$ multicinta

Sea *M* una máquina de Turing multicinta cuyo alfabeto codifica expresiones del lenguaje GOTO . Los siguientes macros son implementables.

1. `C.E`: mover la cabeza de todas las cintas al primer blanco a la izquierda de la cadena actual (configuración estandar).
2. `copy C to D`: copia a la cinta `D` el contenido de la cinta `C`.
3. `erase` *C*: borra el contenido de la cinta C.
4. `compute e in C`: calcula el valor de la GOTO-expresión *e* (codificada), escribiendo el resultado en la cinta C
5. `if C then goto q1 else goto q2` : verifica si el contenido de la cinta C no es 0 (el código de nil) en cuyo caso se cambia el estado a `q1` y en caso contrario se cambia el estado a `q2`.

# Macros en $\mathcal{MT}$ multicinta

Sea *M* una máquina de Turing multicinta cuyo alfabeto codifica expresiones del lenguaje GOTO . Los siguientes macros son implementables.

1. `C.E`: mover la cabeza de todas las cintas al primer blanco a la izquierda de la cadena actual (configuración estandar).
2. `copy C to D`: copia a la cinta `D` el contenido de la cinta `C`.
3. `erase` *C*: borra el contenido de la cinta C.
4. `compute e in C`: calcula el valor de la GOTO-expresión *e* (codificada), escribiendo el resultado en la cinta C
5. `if C then goto q1 else goto q2` : verifica si el contenido de la cinta C no es 0 (el código de nil) en cuyo caso se cambia el estado a `q1` y en caso contrario se cambia el estado a `q2`.
6. `goto` `q`: cambia el estado a `q`

# Traducción de LTURING a $\mathcal{MT}$

Sea $p = \texttt{read}\, X; 1 : \mathcal{I}_1, \ldots, m : \mathcal{I}_m; \texttt{write}\, Y$ un programa en GOTO tal que $Vars(p) = \{X, Y, Z_1, \ldots, Z_n\}$. Definimos la máquina de Turing multicinta $T_p = \langle \Sigma, Q, q_1, q_f, \delta \rangle$ como sigue:

- Existen $n + 3$ cintas, una por cada variable de $p$, denotada exactamente igual, más una cinta auxiliar denotada $\mathcal{A}$.

# Traducción de LTURING a $\mathcal{MT}$

Sea $p = \texttt{read}\, X; 1 : \mathcal{I}_1, \ldots, m : \mathcal{I}_m; \texttt{write}\, Y$ un programa en GOTO tal que $Vars(p) = \{X, Y, Z_1, \ldots, Z_n\}$. Definimos la máquina de Turing multicinta $T_p = \langle \Sigma, Q, q_1, q_f, \delta \rangle$ como sigue:

- Existen $n + 3$ cintas, una por cada variable de $p$, denotada exactamente igual, más una cinta auxiliar denotada $\mathcal{A}$.
- $\Sigma = \{(, ), \cdot, 0, \llcorner\!\lrcorner\}$

# Traducción de LTURING a $\mathcal{MT}$

Sea $p = \texttt{read}\, X; 1 : \mathcal{I}_1, \ldots, m : \mathcal{I}_m; \texttt{write}\, Y$ un programa en GOTO tal que $Vars(p) = \{X, Y, Z_1, \ldots, Z_n\}$. Definimos la máquina de Turing multicinta $T_p = \langle \Sigma, Q, q_1, q_f, \delta \rangle$ como sigue:

- Existen $n + 3$ cintas, una por cada variable de $p$, denotada exactamente igual, más una cinta auxiliar denotada $\mathcal{A}$.
- $\Sigma = \{(, ), \cdot, 0, \llcorner\!\lrcorner\}$
- $Q = \{q_1 \ldots, q_m, q_f\} \cup Q'$ donde existe un estado $q_\ell$ para cada instrucción $\mathcal{I}_\ell$ del programa GOTO y $Q'$ es un conjunto de estados auxiliares utilizados únicamente en los macros anteriores.

# Traducción de LTURING a $\mathcal{MT}$

Sea $p = \texttt{read}\,X; 1 : \mathcal{I}_1, \ldots, m : \mathcal{I}_m; \texttt{write}\,Y$ un programa en GOTO tal que $Vars(p) = \{X, Y, Z_1, \ldots, Z_n\}$. Definimos la máquina de Turing multicinta $T_p = \langle \Sigma, Q, q_1, q_f, \delta \rangle$ como sigue:

- Existen $n + 3$ cintas, una por cada variable de $p$, denotada exactamente igual, más una cinta auxiliar denotada $\mathcal{A}$.
- $\Sigma = \{(, ), \cdot, 0, \sqcup\}$
- $Q = \{q_1 \ldots, q_m, q_f\} \cup Q'$ donde existe un estado $q_\ell$ para cada instrucción $\mathcal{I}_\ell$ del programa GOTO y $Q'$ es un conjunto de estados auxiliares utilizados únicamente en los macros anteriores.
- El estado inicial es $q_1$ y el estado final es $q_f$.

# Traducción de LTURING a $\mathcal{MT}$

- Las acciones de la máquina al llegar a un estado correspondiente a una instruccion del programa GOTO son las siguientes:

# Traducción de LTuring a $\mathcal{MT}$

- Las acciones de la máquina al llegar a un estado correspondiente a una instruccion del programa Goto son las siguientes:
  - Si $\mathcal{I}_\ell =_{def} X := e$ entonces

$q_\ell$: `compute e in` $\mathcal{A}$; `erase` $X$; `copy` $\mathcal{A}$ `to X`; `C.E`; `goto` $q_{\ell+1}$

# Traducción de LTuring a $\mathcal{MT}$

- Las acciones de la máquina al llegar a un estado correspondiente a una instruccion del programa Goto son las siguientes:
  - Si $\mathcal{I}_\ell =_{def} X := e$ entonces

    $q_\ell$: `compute e in` $\mathcal{A}$; `erase` $X$; `copy` $\mathcal{A}$ `to X`; `C.E`; `goto` $q_{\ell+1}$

  - Si $\mathcal{I}_\ell =_{def}$ `if` $X$ `then goto` $\ell'$ `else goto` $\ell''$ entonces

    $q_\ell$: `compute X in` $\mathcal{A}$; `C.E`; `if` $\mathcal{A}$ `then goto` $q_{\ell'}$ `else goto` $q_{\ell''}$

# Traducción de LTURING a $\mathcal{MT}$

- Las acciones de la máquina al llegar a un estado correspondiente a una instruccion del programa GOTO son las siguientes:
  - Si $\mathcal{I}_\ell =_{def} X := e$ entonces

    $q_\ell$: `compute e in` $\mathcal{A}$; `erase` $X$; `copy` $\mathcal{A}$ `to X`; `C.E`; `goto` $q_{\ell+1}$

  - Si $\mathcal{I}_\ell =_{def}$ `if` $X$ `then goto` $\ell'$ `else goto` $\ell''$ entonces

    $q_\ell$: `compute X in` $\mathcal{A}$; `C.E`; `if` $\mathcal{A}$ `then goto` $q_{\ell'}$ `else goto` $q_{\ell''}$

  - Si $\mathcal{I}_\ell =_{def}$ `goto` $q_{\ell'}$ entonces

$$q_\ell : \texttt{goto}\, q_{\ell'}$$

# Ejemplo

## $\mathcal{MT}$

La máquina $T_{\texttt{reverse}}$ asociada al programa goto para REVERSE se describe como sigue:

```
q1: compute nil in A;erase Y;copy A to Y;
    C.E.;goto q2
q2: if X then goto q3 else goto q7
q3: compute hd X in A;erase Z;copy A to Z;
    C.E.;goto q4
q4: compute cons Z Y in A;erase Y;copy A to Y;
    C.E.;goto q5
q5: compute tl X in A;erase X;copy A to X;
    C.E.;goto q6
q6: goto q2
```

# Corrección de la transformación de LTURING a $\mathcal{MT}$

- Si $p$ es un programa en *LTuring* tal que $[\![p]\!](x) = y$ entonces $y \in L(T_p)$.

# Corrección de la transformación de LTURING a $\mathcal{MT}$

- Si $p$ es un programa en *LTuring* tal que $[\![p]\!](x) = y$ entonces $y \in L(T_p)$.
- Más aún, si la tupla

$$(q, w_X, w_Y, w_{Z_1}, \ldots, w_{Z_n}, w_{\mathcal{A}})$$

es una configuración de $T_p$ indicando que $T_p$ se encuentra en el estado $q$ siendo $w_V$ el contenido de la cinta correspondiente a cada variable $V$ entonces

$$(q_1, (x^{\ddagger}, 0, \ldots, 0)) \vdash^{\star} (q_f, (w_X, y^{\ddagger}, \ldots, w_{Z_n}, w_{\mathcal{A}}))$$

# Equivalencia de nociones de computabilidad

**Proposición**

*Los lenguajes* WHILE , GOTO *y* LTURING *son equivalentes a* $\mathcal{MT}$ *.*

# Equivalencia de nociones de computabilidad

## Proposición

*Los lenguajes* WHILE , GOTO *y* LTURING *son equivalentes a* $\mathcal{MT}$ *.*

## Demostración.

Hemos mostrado que:

□

# Equivalencia de nociones de computabilidad

## Proposición

*Los lenguajes* WHILE , GOTO *y* LTURING *son equivalentes a* $\mathcal{MT}$.

## Demostración.

Hemos mostrado que:

- WHILE $\Leftrightarrow$ GOTO

$\square$

# Equivalencia de nociones de computabilidad

## Proposición

*Los lenguajes* WHILE , GOTO *y* LTURING *son equivalentes a* $\mathcal{MT}$.

## Demostración.

Hemos mostrado que:

- WHILE $\Leftrightarrow$ GOTO
- GOTO $\Rightarrow$ $\mathcal{MT}$

$\square$

# Equivalencia de nociones de computabilidad

## Proposición

*Los lenguajes* WHILE , GOTO *y* LTURING *son equivalentes a* $\mathcal{MT}$ .

## Demostración.

Hemos mostrado que:

- WHILE $\Leftrightarrow$ GOTO
- GOTO $\Rightarrow$ $\mathcal{MT}$
- LTURING $\Rightarrow$GOTO .

$\square$

## Equivalencia de nociones de computabilidad

### Proposición

*Los lenguajes* WHILE , GOTO *y* LTURING *son equivalentes a* $\mathcal{MT}$.

### Demostración.

Hemos mostrado que:

- WHILE $\Leftrightarrow$ GOTO
- GOTO $\Rightarrow$ $\mathcal{MT}$
- LTURING $\Rightarrow$GOTO .
- LTURING $\Leftrightarrow$ $\mathcal{MT}$. De donde también tenemos que GOTO $\Leftrightarrow$ $\mathcal{MT}$.

$\square$