

# Autómatas y Lenguajes Formales

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

Dr. Favio Ezequiel Miranda Perea

[favio@ciencias.unam.mx](mailto:favio@ciencias.unam.mx)

Facultad de Ciencias UNAM<sup>1</sup>

3 de febrero de 2020

---

<sup>1</sup>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

- $\Sigma \neq \emptyset$  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

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

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

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

- $\Sigma \neq \emptyset$  es un alfabeto finito que contiene un símbolo distinguido  $\_\_$ , llamado símbolo blanco,
- $Q \neq \emptyset$  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,
- $\delta : Q \times \Sigma \rightarrow Q \times \Sigma \times D$ , es la función (parcial) de transición.



# La máquina de Turing clásica

$\mathcal{MT}$

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

donde

- $\Sigma \neq \emptyset$  es un alfabeto finito que contiene un símbolo distinguido  $\_\_$ , llamado símbolo blanco,
- $Q \neq \emptyset$  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,
- $\delta : Q \times \Sigma \rightarrow Q \times \Sigma \times D$ , es la función (parcial) de transición.
- $D = \{\leftarrow, \rightarrow, -\}$  es el conjunto de movimientos



# Lenguajes de programación

## Modelo teórico

Un lenguaje de programación es una terna

$$\mathcal{L} = \langle \mathcal{P}_{\mathcal{L}}, \mathcal{D}_{\mathcal{L}}, \llbracket \cdot \rrbracket_{\mathcal{L}} \rangle$$

donde

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

# Lenguajes de programación

## Modelo teórico

Un lenguaje de programación es una terna

$$\mathcal{L} = \langle \mathcal{P}_{\mathcal{L}}, \mathcal{D}_{\mathcal{L}}, \llbracket \cdot \rrbracket_{\mathcal{L}} \rangle$$

donde

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

# Lenguajes de programación

## Modelo teórico

Un lenguaje de programación es una terna

$$\mathcal{L} = \langle \mathcal{P}_{\mathcal{L}}, \mathcal{D}_{\mathcal{L}}, \llbracket \cdot \rrbracket_{\mathcal{L}} \rangle$$

donde

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

$$\llbracket \cdot \rrbracket_{\mathcal{L}} : \mathcal{P}_{\mathcal{L}} \rightarrow \mathcal{D}_{\mathcal{L}} \rightharpoonup \mathcal{D}_{\mathcal{L}}$$

# Lenguajes de programación

## Modelo teórico

Un lenguaje de programación es una terna

$$\mathcal{L} = \langle \mathcal{P}_{\mathcal{L}}, \mathcal{D}_{\mathcal{L}}, \llbracket \cdot \rrbracket_{\mathcal{L}} \rangle$$

donde

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

$$\llbracket \cdot \rrbracket_{\mathcal{L}} : \mathcal{P}_{\mathcal{L}} \rightarrow \mathcal{D}_{\mathcal{L}} \rightharpoonup \mathcal{D}_{\mathcal{L}}$$

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



# Semántica operacional

$$\mathcal{L} = \langle \mathcal{P}_{\mathcal{L}}, \mathcal{D}_{\mathcal{L}}, \llbracket \cdot \rrbracket_{\mathcal{L}} \rangle$$

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

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

tal que

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



# Semántica operacional

$$\mathcal{L} = \langle \mathcal{P}_{\mathcal{L}}, \mathcal{D}_{\mathcal{L}}, \llbracket \cdot \rrbracket_{\mathcal{L}} \rangle$$

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

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

tal que

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



# Semántica operacional

$$\mathcal{L} = \langle \mathcal{P}_{\mathcal{L}}, \mathcal{D}_{\mathcal{L}}, \llbracket \cdot \rrbracket_{\mathcal{L}} \rangle$$

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

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

tal que

- $\mathcal{S} \neq \emptyset$  es el conjunto de memorias
- $\mathcal{E} \neq \emptyset$  es el conjunto de estados cuya definición involucra usualmente a  $\mathcal{S}$ .
- $\text{final} : \mathcal{P}_{\mathcal{L}} \rightarrow \mathcal{P}(\mathcal{E})$  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

$$\mathcal{L} = \langle \mathcal{P}_{\mathcal{L}}, \mathcal{D}_{\mathcal{L}}, \llbracket \cdot \rrbracket_{\mathcal{L}} \rangle$$

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

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

tal que

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



# Semántica operacional

$$\mathcal{L} = \langle \mathcal{P}_{\mathcal{L}}, \mathcal{D}_{\mathcal{L}}, \llbracket \cdot \rrbracket_{\mathcal{L}} \rangle$$

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

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

tal que

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



# Semántica operacional

$$\mathcal{L} = \langle \mathcal{P}_{\mathcal{L}}, \mathcal{D}_{\mathcal{L}}, \llbracket \cdot \rrbracket_{\mathcal{L}} \rangle$$

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

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

tal que

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



# Semántica operacional

$$\mathcal{L} = \langle \mathcal{P}_{\mathcal{L}}, \mathcal{D}_{\mathcal{L}}, \llbracket \cdot \rrbracket_{\mathcal{L}} \rangle$$

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

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

tal que

- $\text{output} : \mathcal{P}_{\mathcal{L}} \times \mathcal{E} \rightarrow \mathcal{D}_{\mathcal{L}}$  es la función de salida.
- $\cdot \triangleright \cdot \rightarrow \cdot \subseteq \mathcal{P}_{\mathcal{L}} \times \mathcal{E} \times \mathcal{E}$  es una relación ternaria de transición entre programas, estados y estados
- Significado intencional:  
 $p \triangleright s \rightarrow s'$  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:  $\mathcal{D}_{\mathcal{L}} = \{0, 1, \sqcup\}^*$



# El lenguaje LTURING

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

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


# El lenguaje LTURING

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

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

- Instrucciones

$$\begin{aligned}\mathcal{I} ::= & \text{ right } | \text{ left } | \text{ write } s \\ & \text{ if } s \text{ then goto } \ell \text{ else goto } \ell' | \text{ goto } \ell\end{aligned}$$

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



# Semántica operacional

LTURING

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

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



# Semántica operacional

LTURING

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

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



# Semántica operacional

LTURING

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

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



# Semántica operacional

LTURING

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

- Memorias:  $\mathcal{S} =_{\text{def}} \mathcal{D}_{\mathcal{L}} \times \{0, 1, \sqcup\} \times \mathcal{D}_{\mathcal{L}}$ .
- 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:  $\text{final}(p) = \{\langle m + 1, \sigma \rangle | \sigma \in \mathcal{S}\}$ .



# Semántica operacional

LTURING

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

- Memorias:  $\mathcal{S} =_{\text{def}} \mathcal{D}_{\mathcal{L}} \times \{0, 1, \perp\} \times \mathcal{D}_{\mathcal{L}}$ .
- 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:  $\text{init} : \mathcal{P}_{\mathcal{L}} \times \mathcal{D}_{\mathcal{L}} \rightarrow \mathcal{E}$ ,  $\text{init}(x) = \langle 1, \underline{x} \rangle$



# Semántica operacional

LTURING

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

- Memorias:  $\mathcal{S} =_{\text{def}} \mathcal{D}_{\mathcal{L}} \times \{0, 1, \perp\} \times \mathcal{D}_{\mathcal{L}}$ .
- 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:  $\text{init} : \mathcal{P}_{\mathcal{L}} \times \mathcal{D}_{\mathcal{L}} \rightarrow \mathcal{E}$ ,  $\text{init}(x) = \langle 1, \underline{\perp} x \rangle$
- Función de salida:  $\text{output} : \mathcal{P}_{\mathcal{L}} \times \mathcal{E} \rightarrow \mathcal{D}_{\mathcal{L}}$ ,

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



# Relación de transición

## LTURING

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

# Relación de transición

## LTURING

- Si  $\mathcal{I}_\ell = \text{right}$  entonces  $p \triangleright \langle \ell, L\underline{s}s'R \rangle \rightarrow \langle \ell + 1, L\underline{s}s'R \rangle$
- Si  $\mathcal{I}_\ell = \text{right}$  entonces  $p \triangleright \langle \ell, L\underline{s} \rangle \rightarrow \langle \ell + 1, Ls_{\underline{\underline{l}}} \rangle$



# Relación de transición

## LTURING

- Si  $\mathcal{I}_\ell = \text{right}$  entonces  $p \triangleright \langle \ell, L\underline{s}s'R \rangle \rightarrow \langle \ell + 1, L\underline{s}s'R \rangle$
- Si  $\mathcal{I}_\ell = \text{right}$  entonces  $p \triangleright \langle \ell, L\underline{s} \rangle \rightarrow \langle \ell + 1, Ls_{\underline{\underline{l}}} \rangle$
- Si  $\mathcal{I}_\ell = \text{left}$  entonces  $p \triangleright \langle \ell, L\underline{s}s'R \rangle \rightarrow \langle \ell + 1, L\underline{s}s'R \rangle$

# Relación de transición

## L<sub>TURING</sub>

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

# Relación de transición

## L<sub>TURING</sub>

- Si  $\mathcal{I}_\ell = \text{right}$  entonces  $p \triangleright \langle \ell, L\underline{s}s'R \rangle \rightarrow \langle \ell + 1, L\underline{s}s'R \rangle$
- Si  $\mathcal{I}_\ell = \text{right}$  entonces  $p \triangleright \langle \ell, L\underline{s} \rangle \rightarrow \langle \ell + 1, L\underline{s} \underline{\phantom{s}} \rangle$
- Si  $\mathcal{I}_\ell = \text{left}$  entonces  $p \triangleright \langle \ell, L\underline{s}s'R \rangle \rightarrow \langle \ell + 1, L\underline{s}s'R \rangle$
- Si  $\mathcal{I}_\ell = \text{left}$  entonces  $p \triangleright \langle \ell, \underline{s}R \rangle \rightarrow \langle \ell + 1, \underline{\phantom{s}}sR \rangle$
- Si  $\mathcal{I}_\ell = \text{write } s$  entonces  $p \triangleright \langle \ell, L\underline{s}'R \rangle \rightarrow \langle \ell + 1, L\underline{s}R \rangle$

# Relación de transición

## LTURING

- Si  $\mathcal{I}_\ell = \text{right}$  entonces  $p \triangleright \langle \ell, L\underline{s}s'R \rangle \rightarrow \langle \ell + 1, L\underline{s}s'R \rangle$
- Si  $\mathcal{I}_\ell = \text{right}$  entonces  $p \triangleright \langle \ell, L\underline{s} \rangle \rightarrow \langle \ell + 1, L\underline{s} \underline{\phantom{s}} \rangle$
- Si  $\mathcal{I}_\ell = \text{left}$  entonces  $p \triangleright \langle \ell, L\underline{s}s'R \rangle \rightarrow \langle \ell + 1, L\underline{s}s'R \rangle$
- Si  $\mathcal{I}_\ell = \text{left}$  entonces  $p \triangleright \langle \ell, \underline{s}R \rangle \rightarrow \langle \ell + 1, \underline{\phantom{s}}sR \rangle$
- Si  $\mathcal{I}_\ell = \text{write } s$  entonces  $p \triangleright \langle \ell, L\underline{s}'R \rangle \rightarrow \langle \ell + 1, L\underline{s}R \rangle$
- Si  $\mathcal{I}_\ell = \text{goto } \ell'$  entonces  $p \triangleright \langle \ell, L\underline{s}R \rangle \rightarrow \langle \ell', L\underline{s}R \rangle$

# Relación de transición

## LTURING

- Si  $\mathcal{I}_\ell = \text{right}$  entonces  $p \triangleright \langle \ell, L\underline{s}s'R \rangle \rightarrow \langle \ell + 1, L\underline{s}s'R \rangle$
- Si  $\mathcal{I}_\ell = \text{right}$  entonces  $p \triangleright \langle \ell, L\underline{s} \rangle \rightarrow \langle \ell + 1, L\underline{s} \underline{\phantom{s}} \rangle$
- Si  $\mathcal{I}_\ell = \text{left}$  entonces  $p \triangleright \langle \ell, L\underline{s}s'R \rangle \rightarrow \langle \ell + 1, L\underline{s}s'R \rangle$
- Si  $\mathcal{I}_\ell = \text{left}$  entonces  $p \triangleright \langle \ell, \underline{s}R \rangle \rightarrow \langle \ell + 1, \underline{\phantom{s}}R \rangle$
- Si  $\mathcal{I}_\ell = \text{write } s$  entonces  $p \triangleright \langle \ell, L\underline{s}'R \rangle \rightarrow \langle \ell + 1, L\underline{s}R \rangle$
- Si  $\mathcal{I}_\ell = \text{goto } \ell'$  entonces  $p \triangleright \langle \ell, L\underline{s}R \rangle \rightarrow \langle \ell', L\underline{s}R \rangle$
- Si  $\mathcal{I}_\ell = \text{if } s \text{ then goto } \ell' \text{ else goto } \ell''$  entonces  $p \triangleright \langle \ell, L\underline{s}R \rangle \rightarrow \langle \ell', L\underline{s}R \rangle$



# Relación de transición

## LTURING

- Si  $\mathcal{I}_\ell = \text{right}$  entonces  $p \triangleright \langle \ell, L\underline{s}s'R \rangle \rightarrow \langle \ell + 1, L\underline{s}s'R \rangle$
- Si  $\mathcal{I}_\ell = \text{right}$  entonces  $p \triangleright \langle \ell, L\underline{s} \rangle \rightarrow \langle \ell + 1, L\underline{s} \underline{\phantom{s}} \rangle$
- Si  $\mathcal{I}_\ell = \text{left}$  entonces  $p \triangleright \langle \ell, L\underline{s}s'R \rangle \rightarrow \langle \ell + 1, L\underline{s}s'R \rangle$
- Si  $\mathcal{I}_\ell = \text{left}$  entonces  $p \triangleright \langle \ell, \underline{s}R \rangle \rightarrow \langle \ell + 1, \underline{\phantom{s}}R \rangle$
- Si  $\mathcal{I}_\ell = \text{write } s$  entonces  $p \triangleright \langle \ell, L\underline{s}'R \rangle \rightarrow \langle \ell + 1, L\underline{s}R \rangle$
- Si  $\mathcal{I}_\ell = \text{goto } \ell'$  entonces  $p \triangleright \langle \ell, L\underline{s}R \rangle \rightarrow \langle \ell', L\underline{s}R \rangle$
- Si  $\mathcal{I}_\ell = \text{if } s \text{ then goto } \ell' \text{ else goto } \ell''$  entonces  $p \triangleright \langle \ell, L\underline{s}R \rangle \rightarrow \langle \ell', L\underline{s}R \rangle$
- Si  $\mathcal{I}_\ell = \text{if } s \text{ then goto } \ell' \text{ else goto } \ell''$  entonces  $p \triangleright \langle \ell, L\underline{s}'R \rangle \rightarrow \langle \ell'', L\underline{s}'R \rangle$



# 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  $\Sigma = \{0, 1, \sqcup\}$  una máquina de Turing clásica.

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

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

## De $\mathcal{MT}$ a LTURING

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

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

$$L(T) \subseteq \{x \in \Sigma^* \mid \llbracket p_T \rrbracket(x) \text{ existe}\}$$

- Más aún, si  $(q_0, \underline{\_}x) \rightarrow^* (q_f, ws y)$  entonces

$$\llbracket p_T \rrbracket(x) = y.$$



# De LTURING a $\lambda T$

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:

si  $\llbracket p \rrbracket(x) = y$  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:

si  $\llbracket p \rrbracket(x) = y$  entonces  $x \in L(T)$ .

- Más aún, si  $p \triangleright_{LTURING} (1, \underline{\underline{x}}) \rightarrow^* (m + 1, w\underline{s}y)$  entonces

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

# 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:

- $\text{nil}$  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:

- $\text{nil}$  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  $\underline{n} = \text{nil}^n$  donde

$$\begin{aligned}\text{nil}^0 &= \text{nil} \\ \text{nil}^{n+1} &= (\text{nil}.\text{nil}^n)\end{aligned}$$

# 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  $\underline{n} = \text{nil}^n$  donde

$$\begin{aligned}\text{nil}^0 &= \text{nil} \\ \text{nil}^{n+1} &= (\text{nil}.\text{nil}^n)\end{aligned}$$

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

# 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  $\underline{n} = \text{nil}^n$  donde

$$\begin{aligned}\text{nil}^0 &= \text{nil} \\ \text{nil}^{n+1} &= (\text{nil}.\text{nil}^n)\end{aligned}$$

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



# 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  $\underline{n} = \text{nil}^n$  donde

$$\begin{aligned}\text{nil}^0 &= \text{nil} \\ \text{nil}^{n+1} &= (\text{nil}.\text{nil}^n)\end{aligned}$$

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

# El lenguaje WHILE

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



# El lenguaje WHILE

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

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

# El lenguaje WHILE

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

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

- Comandos:

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


# El lenguaje WHILE

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

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

- Comandos:

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

- Expresiones:

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

# El lenguaje WHILE

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

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

- Comandos:

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

- Expresiones:

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

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



# El lenguaje WHILE

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

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

- Comandos:

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

- Expresiones:

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

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



# El lenguaje WHILE

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

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

- Comandos:

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

- Expresiones:

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

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

- ▶  $|X := e| = 1$
- ▶  $|\text{while } e \text{ do } \vec{C} \text{ end}| = |\vec{C}| + 1$

# El lenguaje WHILE

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

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

- Comandos:

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

- Expresiones:

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

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

- ▶  $|X := e| = 1$
- ▶  $|\text{while } e \text{ do } \vec{C} \text{ end}| = |\vec{C}| + 1$
- ▶  $|C_1; \dots; C_n| = |C_1| + \dots + |C_n|$



# 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:  $\mathcal{S} = \{\sigma \mid \sigma : Var \rightarrow \mathbb{T}\}$



# Semántica Operacional

## WHILE

- Memorias:  $\mathcal{S} = \{\sigma \mid \sigma : Var \rightarrow \mathbb{T}\}$
- Estados:  $\mathcal{E} = \mathbb{N} \times \mathcal{S}$ .

# Semántica Operacional

## WHILE

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



# Semántica Operacional

## WHILE

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



# Semántica Operacional

## WHILE

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



# Semántica Operacional

## WHILE

- Memorias:  $\mathcal{S} = \{\sigma \mid \sigma : Var \rightarrow \mathbb{T}\}$
- Estados:  $\mathcal{E} = \mathbb{N} \times \mathcal{S}$ .
- Evaluación de expresiones: la relación de transición se servirá de la siguiente función  $ev : \mathcal{E} \rightarrow Expr \rightarrow \mathbb{T}$ 
  - ▶  $ev_s(X) = \sigma(X)$  donde  $s = \langle \ell, \sigma \rangle$ .
  - ▶  $ev_s(d) = d$
  - ▶  $ev_s(\text{cons } e f) = ((ev_\sigma e).(ev_\sigma f))$



# Semántica Operacional

## WHILE

- Memorias:  $\mathcal{S} = \{\sigma \mid \sigma : Var \rightarrow \mathbb{T}\}$
- Estados:  $\mathcal{E} = \mathbb{N} \times \mathcal{S}$ .
- Evaluación de expresiones: la relación de transición se servirá de la siguiente función  $ev : \mathcal{E} \rightarrow Expr \rightarrow \mathbb{T}$ 
  - ▶  $ev_s(X) = \sigma(X)$  donde  $s = \langle \ell, \sigma \rangle$ .
  - ▶  $ev_s(d) = d$
  - ▶  $ev_s(\text{cons } e f) = ((ev_\sigma e).(ev_\sigma f))$
  - ▶  $ev_s(\text{hd } e) = nil$ , si  $ev_\sigma e = nil$

# Semántica Operacional

## WHILE

- Memorias:  $\mathcal{S} = \{\sigma \mid \sigma : Var \rightarrow \mathbb{T}\}$
- Estados:  $\mathcal{E} = \mathbb{N} \times \mathcal{S}$ .
- Evaluación de expresiones: la relación de transición se servirá de la siguiente función  $ev : \mathcal{E} \rightarrow Expr \rightarrow \mathbb{T}$ 
  - ▶  $ev_s(X) = \sigma(X)$  donde  $s = \langle \ell, \sigma \rangle$ .
  - ▶  $ev_s(d) = d$
  - ▶  $ev_s(\text{cons } e f) = ((ev_\sigma e).(ev_\sigma f))$
  - ▶  $ev_s(\text{hd } e) = nil$ , si  $ev_\sigma e = nil$
  - ▶  $ev_s(\text{hd } e) = t$ , si  $ev_\sigma e = (t.r)$

# Semántica Operacional

## WHILE

- Memorias:  $\mathcal{S} = \{\sigma \mid \sigma : Var \rightarrow \mathbb{T}\}$
- Estados:  $\mathcal{E} = \mathbb{N} \times \mathcal{S}$ .
- Evaluación de expresiones: la relación de transición se servirá de la siguiente función  $ev : \mathcal{E} \rightarrow Expr \rightarrow \mathbb{T}$ 
  - ▶  $ev_s(X) = \sigma(X)$  donde  $s = \langle \ell, \sigma \rangle$ .
  - ▶  $ev_s(d) = d$
  - ▶  $ev_s(\text{cons } e f) = ((ev_\sigma e).(ev_\sigma f))$
  - ▶  $ev_s(\text{hd } e) = nil$ , si  $ev_\sigma e = nil$
  - ▶  $ev_s(\text{hd } e) = t$ , si  $ev_\sigma e = (t.r)$
  - ▶  $ev_s(\text{tl } e) = nil$ , si  $ev_\sigma e = nil$

# Semántica Operacional

## WHILE

- Memorias:  $\mathcal{S} = \{\sigma \mid \sigma : Var \rightarrow \mathbb{T}\}$
- Estados:  $\mathcal{E} = \mathbb{N} \times \mathcal{S}$ .
- Evaluación de expresiones: la relación de transición se servirá de la siguiente función  $ev : \mathcal{E} \rightarrow Expr \rightarrow \mathbb{T}$ 
  - ▶  $ev_s(X) = \sigma(X)$  donde  $s = \langle \ell, \sigma \rangle$ .
  - ▶  $ev_s(d) = d$
  - ▶  $ev_s(\text{cons } e f) = ((ev_\sigma e).(ev_\sigma f))$
  - ▶  $ev_s(\text{hd } e) = nil$ , si  $ev_\sigma e = nil$
  - ▶  $ev_s(\text{hd } e) = t$ , si  $ev_\sigma e = (t.r)$
  - ▶  $ev_s(\text{tl } e) = nil$ , si  $ev_\sigma e = nil$
  - ▶  $ev_s(\text{tl } e) = r$ , si  $ev_\sigma e = (t.r)$



# Semántica Operacional

## WHILE

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



# Semántica Operacional

## WHILE

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



# Relación de transición

WHILE

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

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

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  $\text{ev}_{\langle \ell, \sigma \rangle}(e) = d$  entonces

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

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

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

$$p \triangleright \langle \ell, \sigma \rangle \rightarrow \langle \ell + k + 1, \sigma \rangle$$

# Relación de transición

## WHILE

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

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

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

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

$$p \triangleright \langle \ell, \sigma \rangle \rightarrow \langle \ell + k + 1, \sigma \rangle$$

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

$$p \triangleright \langle \ell, \sigma \rangle \rightarrow \langle \ell + k + 1, \sigma'' \rangle,$$

donde



# Relación de transición

## WHILE

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

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

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

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

$$p \triangleright \langle \ell, \sigma \rangle \rightarrow \langle \ell + k + 1, \sigma \rangle$$

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

$$p \triangleright \langle \ell, \sigma \rangle \rightarrow \langle \ell + k + 1, \sigma'' \rangle,$$

donde

►  $p \triangleright \langle \ell + 1, \sigma \rangle \rightarrow^* \langle \ell + k, \sigma' \rangle$



# Relación de transición

## WHILE

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

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

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

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

$$p \triangleright \langle \ell, \sigma \rangle \rightarrow \langle \ell + k + 1, \sigma \rangle$$

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

$$p \triangleright \langle \ell, \sigma \rangle \rightarrow \langle \ell + k + 1, \sigma'' \rangle,$$

donde

- ▶  $p \triangleright \langle \ell + 1, \sigma \rangle \rightarrow^* \langle \ell + k, \sigma' \rangle$
- ▶  $p \triangleright \langle \ell, \sigma' \rangle \rightarrow \langle \ell + k + 1, \sigma'' \rangle$

# Semántica Operacional

## WHILE

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

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

# Semántica Operacional

## WHILE

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

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

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

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

# Semántica Operacional

## WHILE

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

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

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

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

- Función de salida:  $\text{output} : \mathcal{P}_{\mathcal{L}} \times \mathcal{E} \rightarrow \mathcal{D}_{\mathcal{L}}$ ,

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



# 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

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

# El lenguaje GOTO

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

$$\mathcal{P}_{\mathcal{L}} \ni p ::= \text{read } X; 1 : \mathcal{I}_1; \dots; m : \mathcal{I}_m; \text{write } Y$$
$$\mathcal{I} ::= X := e \mid \text{if } X \text{ then goto } \ell \text{ else goto } \ell' \mid \text{goto } \ell$$

# El lenguaje GOTO

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

$$\mathcal{P}_{\mathcal{L}} \ni p ::= \text{read } X; 1 : \mathcal{I}_1; \dots; m : \mathcal{I}_m; \text{write } Y$$
$$\mathcal{I} ::= X := e \mid \text{if } X \text{ then goto } \ell \text{ else goto } \ell' \mid \text{goto } \ell$$
$$e, f ::= X \mid d \mid \text{cons } Y Z \mid \text{hd } X \mid \text{tl } X \mid=? Y Z$$

# El lenguaje GOTO

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

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

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

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

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

$$\mathcal{P}_{\mathcal{L}} \ni p ::= \text{read } X; 1 : \mathcal{I}_1; \dots; m : \mathcal{I}_m; \text{write } Y$$
$$\mathcal{I} ::= X := e \mid \text{if } X \text{ then goto } \ell \text{ else goto } \ell' \mid \text{goto } \ell$$
$$e, f ::= X \mid d \mid \text{cons } Y Z \mid \text{hd } X \mid \text{tl } X \mid =? Y Z$$

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

$$\mathcal{P}_{\mathcal{L}} \ni p ::= \text{read } X; 1 : \mathcal{I}_1; \dots; m : \mathcal{I}_m; \text{write } Y$$
$$\mathcal{I} ::= X := e \mid \text{if } X \text{ then goto } \ell \text{ else goto } \ell' \mid \text{goto } \ell$$
$$e, f ::= X \mid d \mid \text{cons } Y Z \mid \text{hd } X \mid \text{tl } X \mid=? Y Z$$

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:  $\mathcal{S} = \{\sigma \mid \sigma : Var \rightarrow \mathbb{T}\}$



# Semántica operacional

GOTO

- Memorias:  $\mathcal{S} = \{\sigma \mid \sigma : Var \rightarrow \mathbb{T}\}$
- Estados:  $\mathcal{E} = \mathbb{N} \times \mathcal{S}$ .

# Semántica operacional

GOTO

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



# Semántica operacional

GOTO

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



# Semántica operacional

GOTO

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

# Semántica operacional

GOTO

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



# Semántica operacional

GOTO

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

# Semántica operacional

GOTO

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

# Semántica operacional

GOTO

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

# Semántica operacional

GOTO

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

# Semántica operacional

GOTO

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



# Semántica operacional

GOTO

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



# Semántica operacional

GOTO

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

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

# Relación de transición

GOTO

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

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

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

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



# Relación de transición

GOTO

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

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

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

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

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

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



# Relación de transición

GOTO

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

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

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

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

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

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

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

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



# Semántica operacional

GOTO

$p = \text{read } X; 1 : \mathcal{I}_1; \dots; m : \mathcal{I}_m; \text{write } Y.$

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

# Semántica operacional

GOTO

$p = \text{read } X; 1 : \mathcal{I}_1; \dots; m : \mathcal{I}_m; \text{write } Y.$

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

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

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



# Semántica operacional

GOTO

$p = \text{read } X; 1 : \mathcal{I}_1; \dots; m : \mathcal{I}_m; \text{write } Y.$

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

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

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

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



# Compilador

## Equivalencia de lenguajes de programación

Sean  $\mathcal{S} = \langle \mathcal{P}_{\mathcal{S}}, \mathcal{D}_{\mathcal{S}}, \llbracket \cdot \rrbracket_{\mathcal{S}} \rangle$  y  $\mathcal{T} = \langle \mathcal{P}_{\mathcal{T}}, \mathcal{D}_{\mathcal{T}}, \llbracket \cdot \rrbracket_{\mathcal{T}} \rangle$  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  $F : \mathcal{P}_{\mathcal{S}} \rightarrow \mathcal{P}_{\mathcal{T}}$ ,  $c : \mathcal{D}_{\mathcal{S}} \rightarrow \mathcal{D}_{\mathcal{T}}$  y



# Compilador

## Equivalencia de lenguajes de programación

Sean  $\mathcal{S} = \langle \mathcal{P}_{\mathcal{S}}, \mathcal{D}_{\mathcal{S}}, \llbracket \cdot \rrbracket_{\mathcal{S}} \rangle$  y  $\mathcal{T} = \langle \mathcal{P}_{\mathcal{T}}, \mathcal{D}_{\mathcal{T}}, \llbracket \cdot \rrbracket_{\mathcal{T}} \rangle$  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  $F : \mathcal{P}_{\mathcal{S}} \rightarrow \mathcal{P}_{\mathcal{T}}$ ,  $c : \mathcal{D}_{\mathcal{S}} \rightarrow \mathcal{D}_{\mathcal{T}}$  y
- se cumple que

$$\forall p \in \mathcal{P}_{\mathcal{S}} \forall x \in \mathcal{D}_{\mathcal{S}} (c(\llbracket p \rrbracket_{\mathcal{S}}(x)) = \llbracket F(p) \rrbracket_{\mathcal{T}}(c(x)))$$



# Compilador

## Equivalencia de lenguajes de programación

Sean  $\mathcal{S} = \langle \mathcal{P}_{\mathcal{S}}, \mathcal{D}_{\mathcal{S}}, \llbracket \cdot \rrbracket_{\mathcal{S}} \rangle$  y  $\mathcal{T} = \langle \mathcal{P}_{\mathcal{T}}, \mathcal{D}_{\mathcal{T}}, \llbracket \cdot \rrbracket_{\mathcal{T}} \rangle$  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  $F : \mathcal{P}_{\mathcal{S}} \rightarrow \mathcal{P}_{\mathcal{T}}$ ,  $c : \mathcal{D}_{\mathcal{S}} \rightarrow \mathcal{D}_{\mathcal{T}}$  y
- se cumple que

$$\forall p \in \mathcal{P}_{\mathcal{S}} \forall x \in \mathcal{D}_{\mathcal{S}} (c(\llbracket p \rrbracket_{\mathcal{S}}(x)) = \llbracket F(p) \rrbracket_{\mathcal{T}}(c(x)))$$

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

$$\llbracket p \rrbracket_{\mathcal{S}}(x) = y \text{ si y sólo si } \llbracket F(p) \rrbracket_{\mathcal{T}}(c(x)) = c(y)$$



# De *While* a *Goto*

- *While1*: restricción de *While* de modo que cada expresión contiene un único operador.



# De *While* a *Goto*

- *While1*: restricción de *While* de modo que cada expresión contiene un único operador.
- *While1* es equivalente a *While*.



## De *While* a *Goto*

- *While1*: restricción de *While* de modo que cada expresión contiene un único operador.
- *While1* es equivalente a *While*.
- Basta transformar instrucciones que contengan expresiones con más de un operador mediante el uso de variables auxiliares.



## De *While* a *Goto*

- *While1*: restricción de *While* de modo que cada expresión contiene un único operador.
- *While1* es equivalente a *While*.
- Basta transformar instrucciones que contengan expresiones con más de un operador mediante el uso de variables auxiliares.
- Por ejemplo  $X := \text{cons } X(\text{hd } Y)$  se convierte en

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



# Traducción de comandos de WHILE1 a GOTO

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

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

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



## Traducción de comandos de WHILE1 a GOTO

$C \mapsto \hat{\ell} : C^*$

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

- Si  $C =_{def} X := e$  entonces  $\hat{\ell} : C^* =_{def} X := e$ .
- Si  $C =_{def} \text{while } X \text{ do } D_1; \dots; D_k \text{ end}$  entonces

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

# Traducción de programas de WHILE1 a GOTO

$p \mapsto p_G$

- Dado  $p$  en WHILE1 :

$$p =_{def} \text{read } X; C_1; \dots; C_m; \text{write } Y$$

# Traducción de programas de WHILE1 a GOTO

$p \mapsto p_G$

- Dado  $p$  en WHILE1 :

$$p =_{def} \text{read } X; C_1; \dots; C_m; \text{write } Y$$

- Definimos el programa  $p_G$  en GOTO como:

$$p_G =_{def} \text{read } X; 1 : C_1^*; \widehat{\ell_2} : C_2^*; \dots; \widehat{\ell_m} : C_m^*; \text{write } Y$$



# Traducción de programas de WHILE1 a GOTO

$p \mapsto p_G$

- Dado  $p$  en WHILE1 :

$$p =_{def} \text{read } X; C_1; \dots; C_m; \text{write } Y$$

- Definimos el programa  $p_G$  en GOTO como:

$$p_G =_{def} \text{read } X; 1 : C_1^*; \widehat{\ell_2} : C_2^*; \dots; \widehat{\ell_m} : C_m^*; \text{write } Y$$

- donde  $\widehat{\ell_2} = |C_1^*| + 1$  y  $\widehat{\ell_{j+1}} = \widehat{\ell_j} + |C_j^*|$  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  $\bar{s} =_{def} \langle \hat{\ell}, \sigma \rangle$



# 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  $\bar{s} =_{def} \langle \hat{\ell}, \sigma \rangle$
- Simulación:

Si  $p \triangleright_{\text{WHILE1}} s \rightarrow s'$  entonces  $p_G \triangleright_{\text{GOTO}} \bar{s} \rightarrow^* \bar{s}'$

# 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  $\bar{s} =_{def} \langle \hat{\ell}, \sigma \rangle$
- Simulación:

Si  $p \triangleright_{\text{WHILE1}} s \rightarrow s'$  entonces  $p_G \triangleright_{\text{GOTO}} \bar{s} \rightarrow^* \bar{s}'$

- Compilación: Sea  $\mathcal{C} = \langle F, c \rangle$  con  $F : \mathcal{P}_{\text{WHILE1}} \rightarrow \mathcal{P}_{\text{GOTO}}$  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 $\ell : \mathcal{I}_\ell \mapsto \mathcal{I}_\ell^*$

## De GOTO a WHILE

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

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

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



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

## De GOTO a WHILE

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

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

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

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

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

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

## De GOTO a WHILE

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

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

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

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

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

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



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

## De GOTO a WHILE

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

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

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

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

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

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

- Aquí  $PC$  es una variable nueva

# Traducción de GOTO a WHILE

Si  $p$  es un programa en GOTO ,

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

definimos el programa  $p_W$  en WHILE como:



# Traducción de GOTO a WHILE

Si  $p$  es un programa en GOTO ,

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

definimos el programa  $p_W$  en WHILE como:

```
 $p_W =_{def}$  read  $X$ ;  
 $PC := 1$ ;  
while  $PC$  do  
    case  $PC$  of  
         $1 \Rightarrow \mathcal{I}_1^*$ ;  
         $\vdots$   
         $m \Rightarrow \mathcal{I}_m^*$ ;  
        otherwise  $\Rightarrow PC := 0$ ;  
    end;  
    write  $Y$ 
```



# 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

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

# De GOTO a WHILE1

- Traducción de estados: Sean  $p = \text{read } X, 1 : \mathcal{I}_1, \dots, m : \mathcal{I}_m, \text{write } Y$  un programa y  $s = \langle \ell, \sigma \rangle$  un estado en GOTO. Definimos el estado  $\bar{s}$  en WHILE como  $\bar{s} =_{\text{def}} \langle \widetilde{\ell}, \widetilde{\sigma} \rangle$  donde  $\widetilde{1} = 4$ ,  $\widetilde{\ell + 1} = \widetilde{\ell} + |\mathcal{I}_{\ell}|$  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  $p \triangleright_{\text{GOTO}} s \rightarrow s'$  entonces  $p_W \triangleright_{\text{WHILE}} \bar{s} \rightarrow^+ \bar{s}'$

# De GOTO a WHILE1

- Traducción de estados: Sean  $p = \text{read } X, 1 : \mathcal{I}_1, \dots, m : \mathcal{I}_m, \text{write } Y$  un programa y  $s = \langle \ell, \sigma \rangle$  un estado en GOTO. Definimos el estado  $\bar{s}$  en WHILE como  $\bar{s} =_{\text{def}} \langle \widetilde{\ell}, \widetilde{\sigma} \rangle$  donde  $\widetilde{1} = 4$ ,  $\widetilde{\ell + 1} = \widetilde{\ell} + |\mathcal{I}_{\ell}|$  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  $p \triangleright_{\text{GOTO}} s \rightarrow s'$  entonces  $p_w \triangleright_{\text{WHILE}} \bar{s} \rightarrow^+ \bar{s}'$
- Compilación: Sea  $\mathcal{C} = \langle F, c \rangle$  con  $F : \mathcal{P}_{\text{GOTO}} \rightarrow \mathcal{P}_{\text{WHILE1}}$  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  $(\cdot)^\dagger : \{0, 1, \sqcup\}^* \rightarrow \mathbb{T}$  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

$$\sqcup^\dagger = \text{nil}, \quad 0^\dagger = \text{nil.nil}, \quad 1^\dagger = \text{nil.(nil.nil)}$$

# Traducción de instrucciones de LTURING a GOTO

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

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

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

donde las variables  $A_1, A_2$  son nuevas.



# Traducción de instrucciones de LTURING a GOTO

$$\ell : \mathcal{I}_\ell \mapsto \hat{\ell} : \tilde{\mathcal{I}}_\ell$$

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

$$\begin{aligned}\hat{\ell} : \tilde{\mathcal{I}}_\ell &=_{def} \hat{\ell} : A_1 := \text{nil} \\ &\quad \hat{\ell} + 1 : A_2 := (=? Rt A_1) \\ &\quad \hat{\ell} + 2 : \text{if } A_2 \text{ then goto } \hat{\ell} + 3 \text{ else goto } \hat{\ell} + 6 \\ &\quad \hat{\ell} + 3 : Lf := \text{cons } C Lf \\ &\quad \hat{\ell} + 4 : C := \sqcup \\ &\quad \hat{\ell} + 5 : \text{goto } \widehat{I+1} \\ &\quad \hat{\ell} + 6 : Lf := \text{cons } C Lf \\ &\quad \hat{\ell} + 7 : C := \text{hd } Rt \\ &\quad \hat{\ell} + 8 : Rt := \text{tl } Rt\end{aligned}$$

donde las variables  $A_1, A_2$  son nuevas.

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

$$\widehat{\ell} : \widetilde{\mathcal{I}}_\ell =_{def} \widehat{\ell} : A_1 := s$$

$$\widehat{\ell} + 1 : A_2 := (=? C A_1)$$

$$\widehat{\ell} + 2 : \text{if } A_2 \text{ then goto } \widehat{\ell}' \text{ else goto } \widehat{\ell}''$$

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

$$\widehat{\ell} : \widetilde{\mathcal{I}}_\ell =_{def} \widehat{\ell} : A_1 := s$$

$$\widehat{\ell} + 1 : A_2 := (=? C A_1)$$

$$\widehat{\ell} + 2 : \text{if } A_2 \text{ then goto } \widehat{\ell}' \text{ else goto } \widehat{\ell}''$$

donde las variables  $A_1, A_2$  son nuevas.

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



# Traducción de LTURING a GOTO

Dado un programa  $p$  en LTURING

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



# Traducción de LTURING a GOTO

Dado un programa  $p$  en LTURING

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

definimos el programa  $p_G$  en GOTO como sigue:

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

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



# De LTURING a GOTO

- Traducción de estados: Dado un estado  $s =_{def} \langle \ell, LsR \rangle$  en LTURING definimos el estado  $\bar{s}$  en GOTO como  $\bar{s} =_{def} \langle \hat{\ell}, \sigma \rangle$  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  $s =_{def} \langle \ell, LsR \rangle$  en LTURING definimos el estado  $\bar{s}$  en GOTO como  $\bar{s} =_{def} \langle \hat{\ell}, \sigma \rangle$  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  $p \triangleright_{LTURING} s \rightarrow s'$  entonces  $p_G \triangleright_{GOTO} \bar{s} \rightarrow^+ \bar{s}'$

# De LTURING a GOTO

- Traducción de estados: Dado un estado  $s =_{def} \langle \ell, LsR \rangle$  en LTURING definimos el estado  $\bar{s}$  en GOTO como  $\bar{s} =_{def} \langle \hat{\ell}, \sigma \rangle$  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  $p \triangleright_{LTURING} s \rightarrow s'$  entonces  $p_G \triangleright_{GOTO} \bar{s} \rightarrow^+ \bar{s}'$
- 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  $\Sigma = \{0, (,), \cdot, \lfloor \rfloor\}$  El conjunto de datos de GOTO ,  $\mathcal{D}_{Goto} = \mathbb{T}$  se codifica en  $\Sigma$  mediante la función  $(\cdot)^\ddagger : \mathbb{T} \rightarrow \Sigma^*$  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  $(\cdot)^\ddagger : \mathbb{T} \rightarrow \Sigma^*$  como sigue:

- $nil^\ddagger = 0$
- $(x.y)^\ddagger = (x^\ddagger \cdot y^\ddagger)$

# De GOTO a $\mathcal{MT}$

## Codificación de datos

Sea  $\Sigma = \{0, (, ), \cdot, \lfloor \rfloor\}$  El conjunto de datos de GOTO ,  $\mathcal{D}_{Goto} = \mathbb{T}$  se codifica en  $\Sigma$  mediante la función  $(\cdot)^\ddagger : \mathbb{T} \rightarrow \Sigma^*$  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 standar).
- 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  $q_1$  y en caso contrario se cambia el estado a  $q_2$ .



# 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 standar).
- 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  $q_1$  y en caso contrario se cambia el estado a  $q_2$ .
- ⑥ goto q: cambia el estado a  $q$



# Traducción de LTURING a $MT$

Sea  $p = \text{read } X; 1 : I_1, \dots, m : I_m; \text{write } Y$  un programa en GOTO tal que  $\text{Vars}(p) = \{X, Y, Z_1, \dots, Z_n\}$ . 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 denominada  $\mathcal{A}$ .



# Traducción de LTURING a $MT$

Sea  $p = \text{read } X; 1 : I_1, \dots, m : I_m; \text{write } Y$  un programa en GOTO tal que  $\text{Vars}(p) = \{X, Y, Z_1, \dots, Z_n\}$ . 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  $\mathcal{A}$ .
- $\Sigma = \{((), \cdot, 0, \sqcup\}$

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

Sea  $p = \text{read } X; 1 : \mathcal{I}_1, \dots, m : \mathcal{I}_m; \text{write } Y$  un programa en GOTO tal que  $\text{Vars}(p) = \{X, Y, Z_1, \dots, Z_n\}$ . 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 denominada  $\mathcal{A}$ .
- $\Sigma = \{(., \cdot, 0, \sqcup\}$
- $Q = \{q_1, \dots, q_m, q_f\} \cup Q'$  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  $p = \text{read } X; 1 : \mathcal{I}_1, \dots, m : \mathcal{I}_m; \text{write } Y$  un programa en GOTO tal que  $\text{Vars}(p) = \{X, Y, Z_1, \dots, Z_n\}$ . 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 denominada  $\mathcal{A}$ .
- $\Sigma = \{(., \cdot, 0, \sqcup\}$
- $Q = \{q_1, \dots, q_m, q_f\} \cup Q'$  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 $MT$

- Las acciones de la máquina al llegar a un estado correspondiente a una instrucción 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 instrucción 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 instrucción 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  $\mathcal{I}_\ell =_{def} \text{if } X \text{ then goto } \ell' \text{ else goto } \ell''$  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 instrucción del programa GOTO son las siguientes:
  - Si  $I_\ell =_{def} X := e$  entonces  
 $q_\ell$ : compute  $e$  in  $\mathcal{A}$ ; erase  $X$ ; copy  $\mathcal{A}$  to  $X$ ; C.E; goto  $q_{\ell+1}$
  - Si  $I_\ell =_{def} \text{if } X \text{ then goto } \ell' \text{ else goto } \ell''$  entonces  
 $q_\ell$ : compute  $X$  in  $\mathcal{A}$ ; C.E; if  $\mathcal{A}$  then goto  $q_{\ell'}$  else goto  $q_{\ell''}$
  - Si  $I_\ell =_{def} \text{goto } q_{\ell'}$  entonces  
 $q_\ell : \text{goto } q_{\ell'}$



# Ejemplo

MT

La máquina  $T_{\text{reverse}}$  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  $\llbracket p \rrbracket(x) = y$  entonces  $y \in L(T_p)$ .



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

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

$$(q, w_X, w_Y, w_{Z_1}, \dots, w_{Z_n}, w_A)$$

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

$$(q_1, (x^\ddagger, 0, \dots, 0)) \vdash^* (q_f, (w_X, y^\ddagger, \dots, w_{Z_n}, w_A))$$



# Equivalencia de nociones de computabilidad

## Proposición

*Los lenguajes WHILE , GOTO y LTURING son equivalentes a MT .*



# Equivalencia de nociones de computabilidad

## Proposición

*Los lenguajes WHILE , GOTO y LTURING son equivalentes a MT .*

## Demostración.

Hemos mostrado que:



# Equivalencia de nociones de computabilidad

## Proposición

*Los lenguajes WHILE , GOTO y LTURING son equivalentes a MT.*

## Demostración.

Hemos mostrado que:

- WHILE  $\Leftrightarrow$  GOTO



# Equivalencia de nociones de computabilidad

## Proposición

*Los lenguajes WHILE , GOTO y LTURING son equivalentes a MT.*

## Demostración.

Hemos mostrado que:

- WHILE  $\Leftrightarrow$  GOTO
- GOTO  $\Rightarrow$  MT



# Equivalencia de nociones de computabilidad

## Proposición

*Los lenguajes WHILE , GOTO y LTURING son equivalentes a MT .*

## Demostración.

Hemos mostrado que:

- WHILE  $\Leftrightarrow$  GOTO
- GOTO  $\Rightarrow$  MT
- LTURING  $\Rightarrow$  GOTO .



# Equivalencia de nociones de computabilidad

## Proposición

*Los lenguajes WHILE , GOTO y LTURING son equivalentes a MT.*

## Demostración.

Hemos mostrado que:

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

