

Modelo Computacional Imperativo

Algoritmos y Estructuras de Datos I

Martín Ariel Domínguez

Modelo computacional funcional

- ¿Cómo computa un valor un lenguaje como Haskell?
- El modelo funcional se basa en aplicar reglas de reducción sobre expresiones sintácticas.
- Ejemplo: queremos calcular $[1, 2] ++ [3, 4, 5]$

Modelo computacional funcional

- Recordemos que `[1, 2]` es una abreviación para `1 : 2 : []`
- La manera de computar un valor es mediante la aplicación de reglas de reducción.
- En este caso tenemos reglas definidas por *pattern matching* en la construcción de la lista, dadas por la definición de `++`.

Modelo computacional funcional

$(++) :: [a] \rightarrow [a] \rightarrow [a]$

$[] ++ ys = ys$

$(x : xs) ++ ys = x : (xs ++ ys)$


- Entonces, $(1:2:[]) ++ (3:4:5:[])$ reduce a:
 - $(1:2:[]) ++ (3:4:5:[])$
 - $1:((2:[])++(3:4:5:[]))$
 - $1:(2:([]++(3:4:5:[])))$
 - $1:(2:(3:4:5:[])) = [1,2,3,4,5]$
- No se pueden aplicar más reglas: llegamos al valor.

Modelo computacional imperativo

- **Expresiones** y **Sentencias** acceden a un estado común.
- Un **estado** es un conjunto de variables junto con su valor.

Expresiones

Sentencias




$\{x \mapsto 1, y \mapsto 3, w \mapsto 5$
 $, z \mapsto \text{True}, b \mapsto \text{False} \}$ **Estado**

Modelo computacional imperativo

- En nuestro modelo, las **expresiones** acceden al estado sólo para consultarlo (sin modificarlo).

Expresiones

Sentencias




$\{ x \mapsto 1, y \mapsto 3, w \mapsto 5, \\ z \mapsto \text{True}, b \mapsto \text{False} \}$

Modelo computacional imperativo

- Las **sentencias** pueden modificar el estado (cambiar el valor de alguna variable).

Expresiones

Sentencias




$\{ x \mapsto 1, y \mapsto 3, w \mapsto 4, \\ z \mapsto \text{True}, b \mapsto \text{True} \}$

Modelo computacional imperativo

- **Expresiones** y **Sentencias** acceden a un estado común
- Un estado es un conjunto de variables junto con su valor.

Expresiones

Sentencias



$\{ x \mapsto 1, y \mapsto 3, w \mapsto 4, \\ z \mapsto \text{True}, b \mapsto \text{True} \}$

Modelo computacional: ejemplo

Expresiones

¿Hay Jaque Mate?
¿Cuántos caballos
tengo?
¿Es posible mover el
alfil?



Sentencias

Mover Alfil en Diagonal 2
Casilleros

Mover El Peon 1
Casillero

Arriba Retroceder La
Torre

Evaluación de expresiones

Estado $\xrightarrow{\quad}$ $\{x \mapsto 1, y \mapsto 3, w \mapsto 4,$
 $\quad \quad \quad z \mapsto \text{True}, b \mapsto \text{True} \}$

Evaluar($x + w$, Estado) = 5

Evaluar($x * y$, Estado) = 3

Evaluar($x > 5$, Estado) = False

Evaluar($b \ \&\& \ x \geq 10$, Estado) = False

El valor de la expresiones depende del Estado.

Evaluación de Sentencias

skip

$x := E$

$S1 ; S2$

if $B1 \rightarrow S1$

...

| $Bn \rightarrow Sn$

fi

do $B \rightarrow S$ od

Inacción

Asignación

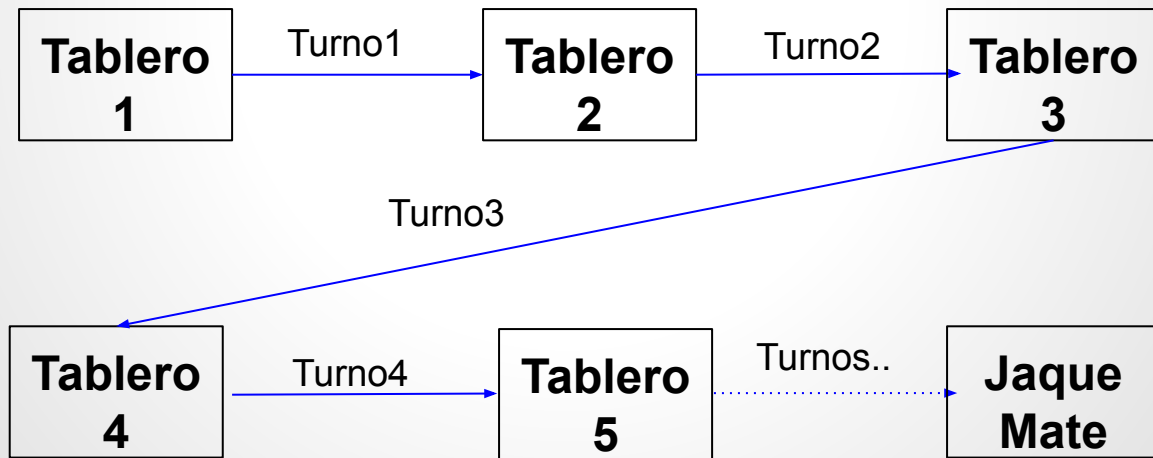
Secuencia

Condicional determinista

Ciclo de guarda única

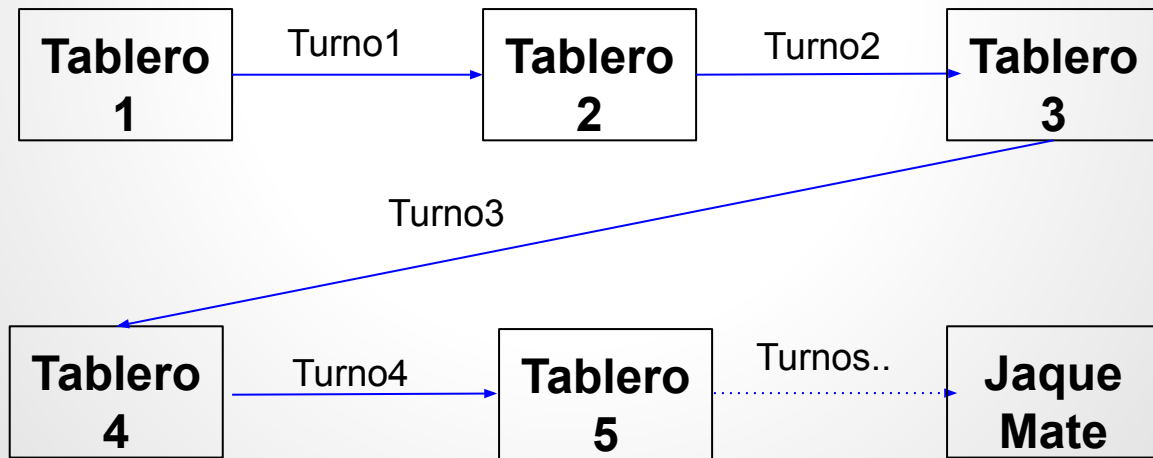
Sentencias: ejemplo

- En el ajedrez, tenemos un tablero inicial, y la “ejecución” de los movimientos se realiza por turnos (se mueve una ficha por turno).



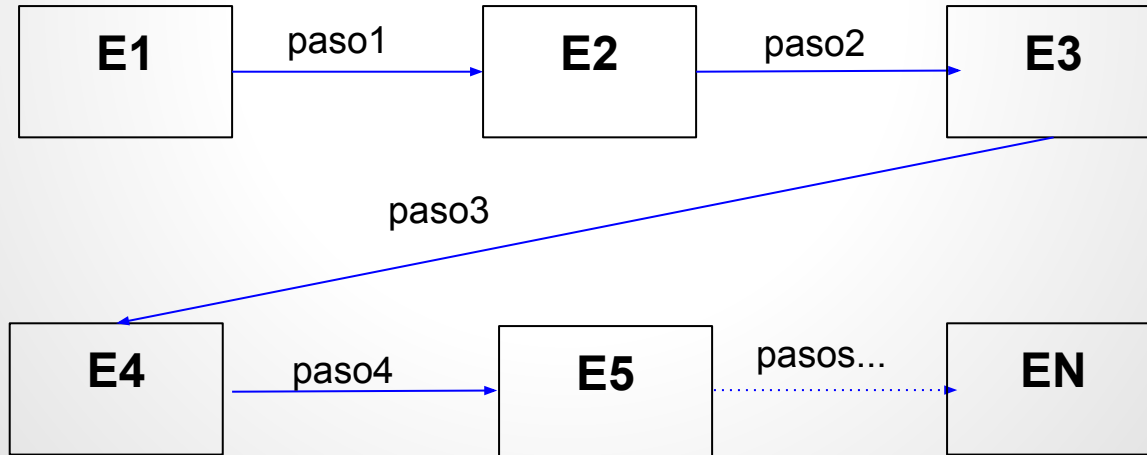
Sentencias: ejemplo

- Comenzamos con un tablero inicial, ejecutamos varios movimientos, y llegamos a un tablero final (jaque mate).



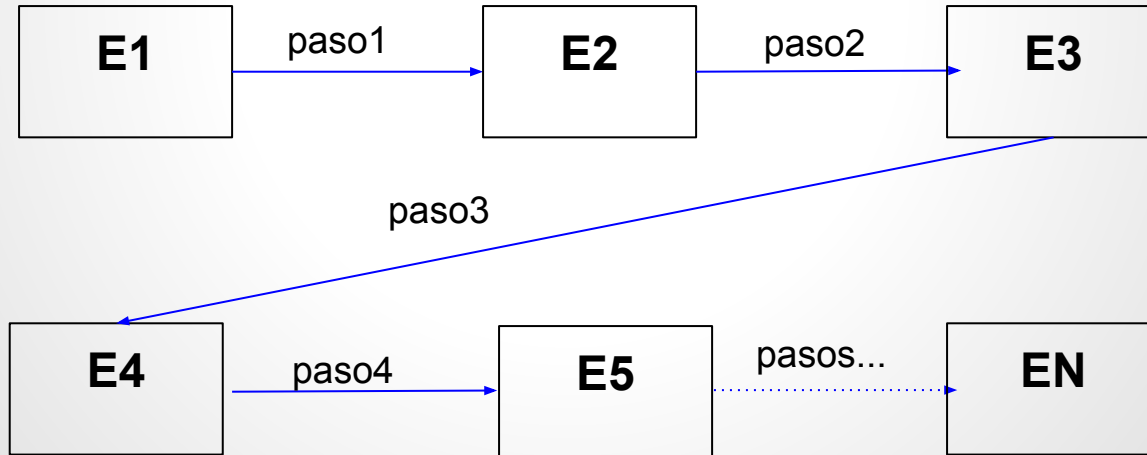
Sentencias: ejemplo

- Partiendo del estado inicial, pasando por estados intermedios, hasta el estado final.



Sentencias: ejemplo

- En nuestro modelo secuencial, las sentencias se ejecutan paso a paso.



Inacción



- La sentencia skip finaliza en único paso, sin modificar el estado.

Asignación

Estado
anterior

$(x \mapsto 1, y \mapsto 3, w \mapsto -1, z \mapsto 15, b \mapsto \text{False})$

Transformación

$x := e$

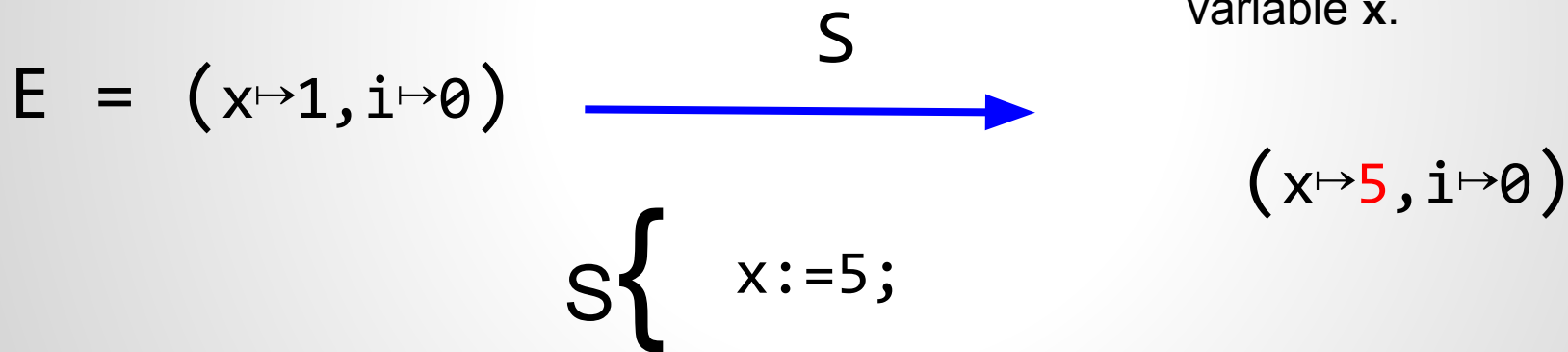
$\text{eval}(e, \text{estado}) = v$

Estado
posterior

$(x \mapsto v, y \mapsto 3, w \mapsto -1, z \mapsto 15, b \mapsto \text{False})$

La asignación $x := e$, modificando el estado únicamente en la variable x , reemplazando su valor anterior por el valor de la expresión e en el estado inicial.

Asignación - Ejemplo



Luego de ejecutar s la única modificación es el valor asociado a la variable x .

Condicional

Ejecutar if se resuelve:

if $B_1 \rightarrow S_1$
□ $B_2 \rightarrow S_2$
 . . .
□ $B_n \rightarrow S_n$
fi

E

E

Luego ejecutar(S_i)



Si sabemos que : S_i es el primer programa tal
que la guarda B_i cumple
 $\text{Evaluar}(B_i, E) = \text{True}$.

Condicional - Ejemplo

$E = (x \mapsto 1, i \mapsto 2)$

S



Como la segunda guarda es verdadera en el estado E . Debo ejecutar S' .

Luego de ejecutar S' el estado queda en

```
S { if (i < 1) ->  
      i := 2;  
    □ (i ≥ 1) ->  
      i := 5; } S'  
fi
```

$(x \mapsto 1, i \mapsto 5)$

Condicional

- Difiere del condicional de otros lenguajes como C, donde si no existe ninguna guarda verdadera, el `if` equivale a `skip`.
- Difiere del condicional del teórico. Elegimos la versión determinista para simplificar la traducción a C.

Ciclo

- El ciclo sirve para ejecutar sentencias en forma repetitiva.

```
do NO (llegué a la última fila o hay otra ficha) ->  
    mover la torre 1 casillero arriba ;
```

```
od
```

- Evidentemente necesita varios pasos para completarse, salvo que al comienzo no se cumpla la guarda (justo arranqué con la ficha en la última fila).

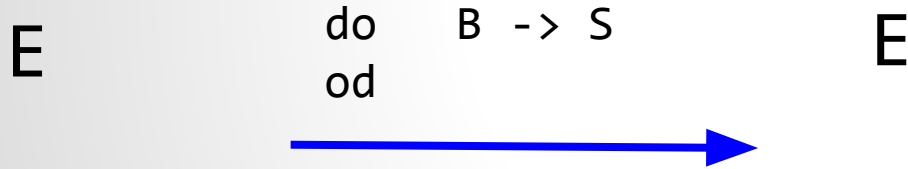
Ciclo

- Puede no terminar nunca!

```
do el rey vive ->  
    mover alguna ficha  
od
```

Ciclo - Caso 1

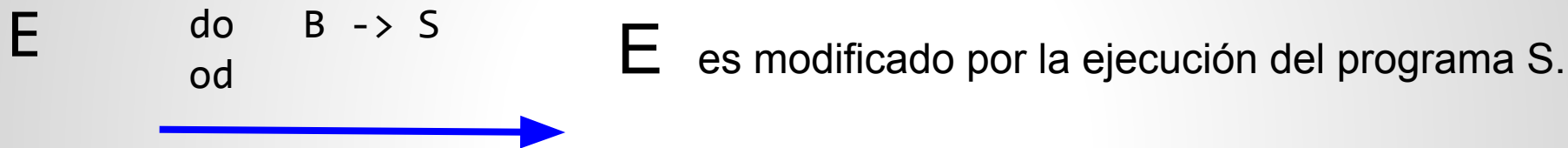
Ejecutar do se resuelve:



Si sabemos que : $\text{evaluar}(B, E) = \text{False}$

Si la guarda es Falsa, es igual que un skip, no modifica el estado y termina el ciclo. Se sigue ejecutando la siguiente instrucción siguiente, en caso de haber.

Ciclo - Caso 2



Si sabemos que : $\text{eval}(B, E) = \text{True}$

Luego de ejecutar S , se vuelve a evaluar B .

Ciclo - Caso 2 - Ejemplo

$$E = (x \mapsto 1, i \mapsto 0) \xrightarrow{S}$$

Como la guarda ($i < 5$) es verdadera en el estado E . Debo ejecutar S' .

Luego de ejecutar S' el estado queda en

$$S \left\{ \begin{array}{l} \text{do} \quad (i < 5) \rightarrow \\ \quad \quad x := x + 1; \\ \quad \quad i := i + 1; \\ \text{od} \end{array} \right\} S'$$

$$(x \mapsto 2, i \mapsto 1)$$

A posteriori se evalúa nuevamente la guarda.

Concatenar programas

- Una secuencia $S1;S2$ es una concatenación de sentencias.
- Cada S_i puede requerir más de un paso (turno) para completarse.
- Hacer un paso en $S1;S2$ es hacer un paso en $S1$. Luego tenemos dos casos:
 - Caso 1: Si $S1$ finaliza en ese paso, entonces en el siguiente paso continúo con $S2$, usando el estado en el que terminó $S1$.
 - Caso 2: Si $S1$ necesita más pasos para completarse, en el siguiente paso continúo ejecutando lo que le falta.

Secuencia - Caso 1

Si sabemos que $s1$ termina en un paso:



Entonces:



Secuencia - Caso 1 - Ejemplo

$(x \mapsto 1, y \mapsto 3)$

$S1; S2$



Ejecuté S1 y modifiqué el estado a:

$(x \mapsto 3, y \mapsto 3)$

Debo ejecutar luego

$S1 \left\{ \begin{array}{l} x := 3; \\ y := 2; \\ x = x + y; \end{array} \right.$

$S2 \left\{ \begin{array}{l} y := 2; \\ x = x + y; \end{array} \right.$

Secuencia - Caso 2

Si sabemos que S1 **NO** termina en un paso:

E1 $\xrightarrow{S1}$ Termina en un estado E2 y resta por ejecutar **S1'**, luego de un paso de **S1**.

Entonces:

E1 $\xrightarrow{S1; S2}$ Me queda por ejecutar **S1'** ; **S2** utilizando el estado E2

Secuencia - Caso 2 - Ejemplo

$(x \mapsto 1, y \mapsto 3)$ $\xrightarrow{S1; S2}$

$S1 \left\{ \begin{array}{l} x := 3; \\ y := 2; \end{array} \right\} S1'$

$S2 \left\{ \begin{array}{l} x = x + y; \end{array} \right.$

Ejecuté la primer
instrucción de $S1'$ y
modifiqué el estado a:

$(x \mapsto 3, y \mapsto 3)$

Debo ejecutar luego, $S1'$
concatenado con $S2$

$S1'; S2 \left\{ \begin{array}{l} y := 2; \\ x = x + y; \end{array} \right.$