

1 Máquina URM, comandos e programas URM

Definição¹: MÁQUINA URM (UNLIMITED REGISTER MACHINE)

Uma máquina URM é constituída por uma sucessão de registos² $\langle R_i \rangle_{i \in \mathbb{N}}$. Cada registo pode conter um valor $r \in \mathbb{N}_0$. É usual usar a seguinte notação:

- para cada $n \in \mathbb{N}$, r_n designa o número natural guardado no registo R_n
- uma configuração da máquina é uma sucessão de naturais $\langle r_i \rangle_{i \in \mathbb{N}}$

É possível escrever programas para as máquinas URM. Estes são sequências finitas de comandos URM. Estes comandos permitem alterar o conteúdo dos registos da máquina.

Definição: PROGRAMAS URM

Um programa URM é uma sequência finita³ $\langle p_i \rangle_{i \in 1..k}$ (com $k \in \mathbb{N}$) de comandos URM.

COMANDOS URM

- Comandos zero: para cada $n \geq 1$

$$Z(n)$$

é o comando que regista 0 em R_n e deixa inalterados os outros registos.

- Comandos sucessor: para cada $n \geq 1$

$$S(n)$$

é o comando incrementa de uma unidade o conteúdo de R_n e deixa inalterados os outros registos.

- Comandos salto: para cada $m \geq 1$, $n \geq 1$, $q \geq 1$

$$J(m, n, q)$$

é o comando que

- compara r_m com r_n deixando inalterados todos registos
- se $r_m = r_n$ a máquina passa a executar o q -ésimo comando do programa se existir e pára caso tal comando não exista
- se $r_m \neq r_n$ a máquina passa a executar o comando seguinte

¹Estas notas são baseadas em *Sintaxe e Semântica de Linguagens I*, J-F Costa, DMIST, 2000 e *Computability-an introduction to recursive function theory*, N. Cutland, Cambridge University Press, 1980.

²No que se segue, $\mathbb{N} = \mathbb{N}_0 \setminus \{0\}$

³No que se segue, para cada $k \in \mathbb{N}$, $1..k$ representa o conjunto $\{1, \dots, k\}$.

- Comandos transferência (ou cópia): para cada $m \geq 1, n \geq 1$

$$T(m, n)$$

é o comando que coloca o valor r_m no registo R_n , deixando os outros inalterados⁴.

Definição: ESPAÇO DE TRABALHO DE UM PROGRAMA

Se P um programa URM, representa-se por

$$\rho(P)$$

o maior $n \in \mathbb{N}$ tal que o registo R_n é referenciado no programa. Ao valor $\rho(P)$ chama-se espaço de trabalho de P .

Exemplo: Seguem-se alguns exemplos de programas URM (para facilitar a compreensão, é usual numerar os comandos de um programa embora tal não faça parte da definição). Faz-se ainda referência informal à noção de execução de um programa a partir de uma configuração da máquina. Esta assunto será abordado com mais detalhe na próxima secção.

1.

- 1 $J(3, 2, 5)$
- 2 $S(1)$
- 3 $S(3)$
- 4 $J(1, 1, 1)$

O espaço de trabalho deste programa é 3. Se este programa for executado

- a partir de uma configuração em que o conteúdo do registo R_1 é um natural x , o conteúdo do registo R_2 é um natural y e o conteúdo do registo R_3 é 0, então a execução do programa termina sempre e quando termina o conteúdo do registo R_1 é $x + y$, o conteúdo do registo R_3 é y e o conteúdo dos outros registos não se alterou;
- a partir de uma configuração em que o conteúdo dos registos R_2 e R_3 são iguais, a execução do programa termina sempre (só é executado o primeiro comando) e a configuração da máquina não é alterada;
- a partir de uma configuração em que o conteúdo do registo R_2 é menor que o conteúdo do registo R_3 , a execução do programa não termina.

2.

- 1 $J(1, 4, 10)$
- 2 $S(3)$
- 3 $J(1, 3, 7)$

⁴Este comando não é necessariamente primitivo, pois pode ser substituído por $< Z(n)J(m, n, 5)S(n)J(1, 1, 2) >$

- 4 $S(2)$
- 5 $S(3)$
- 6 $J(1, 1, 3)$
- 7 $T(2, 1)$

O espaço de trabalho deste programa é 4. Se este programa for executado

- a partir de uma configuração em que o conteúdo do registo R_1 é um natural x e o conteúdo dos registos R_2 , R_3 e R_4 são 0, então a execução do programa termina sempre e quando termina o conteúdo do registo R_1 é 0 se $x = 0$ e é $x - 1$ se $x \neq 0$, o conteúdo do registo R_2 é igual ao do registo R_1 , o conteúdo do registo R_3 é 0 se $x = 0$ e é x se $x \neq 0$, e o conteúdo dos outros registos não se alterou;
- a partir de uma configuração em que o conteúdo dos registos R_1 e R_4 são iguais, a execução do programa termina sempre (só é executado o primeiro comando) e a configuração da máquina não é alterada;
- a partir de uma configuração em que o conteúdo do registo R_2 é menor que o conteúdo do registo R_3 , a execução do programa não termina.

Observação: De um modo informal, pode dizer-se que existem duas formas de um programa $P = \langle p_i \rangle_{i \in 1..k}$ terminar a sua execução: (i) execução de um comando de salto $J(m, n, q)$ com $q > k$ numa situação em que os conteúdos dos registos R_n e R_m são iguais ou (ii) execução do último comando (p_k) no caso de este ser um comando zero, sucessor, transferência ou um comando salto $J(m, n, q)$ com $q \leq k$ numa situação em que os conteúdos dos registos R_n e R_m são diferentes. Este assunto será apresentado do modo mais preciso na secção seguinte.

Definição: PROGRAMA URM NORMALIZADO

O programa $P = \langle p_i \rangle_{i \in 1..k}$ diz-se normalizado sse, para cada $i \in 1..k$, se $p_i = J(m, n, q)$ então $q \leq k + 1$.

Proposição: Qualquer programa $P = \langle p_i \rangle_{i \in 1..k}$ pode ser transformado num programa normalizado P' equivalente, no sentido em que, se P e P' começarem a ser executados a partir de uma dada configuração (inicial) da máquina, então a execução de P termina sse a de P' termina e quando terminam a sua execução, a configuração (final) da máquina é a mesma nos dois casos.

Prova: Basta substituir em P cada comando $J(m, n, q)$ com $q > k + 1$ por $J(m, n, k + 1)$.

Exemplo:

- O programa apresentado em 1. do Exemplo anterior é um programa normalizado.

- O programa apresentado em 2. do Exemplo anterior não é um programa normalizado. Normalizando este programa, obtém-se o programa:

```

1  $J(1, 4, 8)$ 
2  $S(3)$ 
3  $J(1, 3, 7)$ 
4  $S(2)$ 
5  $S(3)$ 
6  $J(1, 1, 3)$ 
7  $T(2, 1)$ 

```

Observação: EMULADOR URM E FICHAS ELECTRÓNICAS

Pode ser utilizada uma sintaxe diferente para os comandos URM acima mencionados. No âmbito do emulador da máquina URM disponibilizado através da página [www](#) desta disciplina e no âmbito do sistema de fichas electrónicas, a sintaxe dos comandos acima referidos é a seguinte:

- Comandos zero: $ZERO[n]$
- Comandos sucessor: $SUCC[n]$
- Comandos salto: $JUMP[m, n, q]$
- Comandos transferência ou cópia: $COPY[m, n]$

Ainda neste âmbito considera-se também o comando seguinte.

- Comando de paragem:

$HALT[]$

é o comando que faz terminar a execução do programa.

Note-se que o comando $HALT[]$ permite que para terminar a execução de um programa se possa deixar de utilizar comandos salto $JUMP[m, n, q]$, com q maior que o número de comandos do programa. Assumindo que se trabalha apenas com programas normalizados, nestas situações basta introduzir, na posição q , o comando $HALT[]$. É fácil perceber que para cada programa P escrito com recurso ao comando $HALT[]$ existe um programa P' equivalente (no sentido atrás referido) escrito sem recurso ao comando $HALT[]$ e vice-versa.

No âmbito do emulador URM referido e nas fichas electrónicas que os alunos devem realizar, existe ainda a possibilidade de utilizar *oráculos* num programa. Os oráculos podem ser vistos como módulos que calculam determinadas funções. Através da página [www](#) da disciplina pode encontrar-se a lista de oráculos disponíveis. Como exemplo refere-se aqui o caso do oráculo $SUM[n, m, q]$. Este oráculo, ao ser executado, coloca no registo R_q o valor da soma dos conteúdos de R_n e R_m . Note-se que a execução deste oráculo só vai modificar (eventualmente) o conteúdo do registo R_q .

Por razões operacionais, os programas URM que podem ser executados no emulador da máquina URM e os programas que os alunos têm submeter através

do sistema de fichas electrónicas devem ter a seguinte sintaxe: cada programa é uma lista

$$\begin{aligned} &1 : cmd_1; \\ &2 : cmd_2; \\ &\dots \\ &k : cmd_k; \end{aligned}$$

onde, para cada $1 \leq j \leq k$, cmd_j é $ZERO[n]$, $SUCC[n]$, $COPY[n]$, $JUMP[m, n, q]$ com $q \leq k$, $HALT[]$ ou um dos oráculos disponíveis. Deste modo, neste contexto, é obrigatória a numeração (sequencial) dos comandos, devendo existir : entre o número e o comando e ; após cada comando (incluindo o último). Note-se que como os comandos salto permitidos são apenas do tipo $JUMP[m, n, q]$ com $q \leq k$, a terminação da execução através de comandos salto com $q > k$ tal como anteriormente acontecia já não é aqui possível, pelo que, como se referiu acima, situações desse tipo têm resolvidas com recurso ao comando $HALT[]$.

Existe ainda um outro requisito que deve ser observado: a *execução de um programa deve sempre terminar através da execução de um comando $HALT[]$* . Isto significa, em particular, que se anteriormente a execução de um programa terminava com a execução de um comando zero, sucessor, transferência ou um comando salto $J(m, n, q)$ com $q \leq k$ numa situação em que os conteúdos dos registos R_n e R_m eram diferentes, agora, para que a execução do programa possa terminar, há que acrescentar o comando $HALT[]$ imeditamente a seguir ao comando em causa.

Com estes requisitos o programa 1. apresentado no primeiro Exemplo deve ser escrito do seguinte modo

$$\begin{aligned} &1 : JUMP[3, 2, 5]; \\ &2 : SUCC[1]; \\ &3 : SUCC[3]; \\ &4 : JUMP[1, 1, 1]; \\ &5 : HALT[]; \end{aligned}$$

e o programa 2. deve ser escrito do seguinte modo

$$\begin{aligned} &1 : JUMP[1, 4, 8] \\ &2 : SUCC[3]; \\ &3 : JUMP[1, 3, 7] \\ &4 : SUCC[2] \\ &5 : SUCC[3] \\ &6 : JUMP[1, 1, 3]; \\ &7 : COPY[2, 1]; \\ &8 : HALT[]; \end{aligned}$$

2 Funções URM-computáveis

Antes de definir a noção de função URM-computável, há que introduzir a noção de computação de um programa a partir de uma configuração e a noção de

função (de aridade n) calculada por um programa.

Informalmente, a computação de um programa URM, $P = \langle p_i \rangle_{i \in 1..k}$, a partir de uma configuração r é uma sequência finita (computação finita)

$$\eta_1 q_1 \eta_2 q_2 \dots \eta_n q_n \eta_{n+1}$$

ou uma sequência infinita (computação infinita)

$$\eta_1 q_1 \eta_2 q_2 \dots \eta_n q_n \dots$$

de configurações e comandos que se inicia com a configuração r (i.e. $\eta_1 = r$), a configuração inicial, e prossegue até ser possível alternando os comandos de P que vão sendo executados, q_i , com as configurações resultantes dessa execução (a partir da configuração imediatamente anterior), η_{i+1} . No caso de ser uma sequência finita, a configuração η_{n+1} , que é a configuração resultante da execução do último comando a ser executado, diz-se configuração final.

A noção rigorosa de computação de um programa URM a partir de uma configuração é algo elaborada sendo necessário definir algumas noções preliminares auxiliares. No fim destas notas apresenta-se a definição rigorosa para a noção de computação.

Notação:

Seja P um programa URM e $r = \langle r_i \rangle_{i \in \mathbb{N}}$ uma configuração da máquina,

- usa-se $P(r)$ para representar a computação de P a partir da configuração inicial r
- $P(r) \downarrow$ significa que $P(r)$ é computação finita
- $P(r) \downarrow b$ significa que $P(r)$ é computação finita e na configuração final, o valor guardado no primeiro registo é b
- $P(r_1, \dots, r_i, \dots) \uparrow$ significa que $P(r)$ é computação infinita
- se, em particular, existe $k \in \mathbb{N}$ tal que $r_j = 0$ para cada $j > k$ usam-se as notações $P(r_1, \dots, r_k)$, $P(r_1, \dots, r_k) \downarrow$, $P(r_1, \dots, r_k) \uparrow$ e $P(r_1, \dots, r_k) \downarrow b$

Definição: FUNÇÃO DE ARIDADE n CALCULADA POR PROGRAMA

Seja $f : \mathbb{N}_0^n \rightarrow \mathbb{N}_0$, diz-se que o programa P calcula f sse

- $P(r_1, \dots, r_n) \uparrow$ sse $(r_1, \dots, r_n) \notin D_f$ e
- $P(r_1, \dots, r_n) \downarrow b$ sse $(r_1, \dots, r_n) \in D_f$ e $f(r_1, \dots, r_n) = b$

Exemplo:

- O programa apresentado em 1. do Exemplo anterior calcula a função unária $f : \mathbb{N}_0 \rightarrow \mathbb{N}_0$ tal que $f(x) = x$ e calcula função binária $g : \mathbb{N}_0 \times \mathbb{N}_0 \rightarrow \mathbb{N}_0$ tal que $g(x, y) = x + y$.
- O programa apresentado em 2. do Exemplo anterior calcula a função unária $f : \mathbb{N}_0 \rightarrow \mathbb{N}_0$ tal que $f(x) =$ “se $x = 0$ então 0 senão $x - 1$ ” e calcula a função binária $g : \mathbb{N}_0 \times \mathbb{N}_0 \rightarrow \mathbb{N}_0$ tal que $g(x, y) =$ “se $x = 0$ então 0 senão $x + y - 1$ ”.

Definição: FUNÇÃO COMPUTÁVEL PELA MÁQUINA URM

A função $f : \mathbb{N}_0^n \rightarrow \mathbb{N}_0$ é computável pela máquina URM sse existe um programa URM, P , que calcula f .

Notação:

- Representa-se por C a classe das funções computáveis pela máquina URM.
- Representa-se por C_n a classe das funções de aridade n que pertencem a C .

Observação: Dado um programa P e $n \in \mathbb{N}$ existe uma e uma só função de aridade n que é calculada por P . Esta função é representada por $f_P^{(n)}$.

Por outro lado, dada uma função computável, esta pode ser calculada/computada por mais de um programa.

3 Predicados e predicados decidíveis

Informalmente, um predicado n -ário, ou de aridade n , sobre \mathbb{N}_0 , com $n \in \mathbb{N}_0$, é uma asserção acerca de números naturais na qual são referenciados n naturais:

- $M_1 \equiv$ “ x é par ” (predicado unário ou de aridade 1)
- $M_2 \equiv$ “ x é maior que y ” (predicado binário ou de aridade 2)
- $M_3 \equiv$ “ z é o resto da divisão de x por y ” (predicado ternário ou de aridade 3)

Definição: PREDICADO n -ÁRIO SOBRE OS NATURAIS

Um predicado n -ário M , $n \in \mathbb{N}_0$, sobre os naturais é um subconjunto de \mathbb{N}_0^n , ou seja

$$M \subseteq \mathbb{N}_0^n.$$

Dado $(x_1, \dots, x_n) \in \mathbb{N}_0^n$, diz-se que o predicado é verdadeiro para (x_1, \dots, x_n) sse $(x_1, \dots, x_n) \in M$. Usa-se a notação $M(x_1, \dots, x_n)$ para representar este facto.

Definição: FUNÇÃO CARACTERÍSTICA DE UM PREDICADO

Seja M um predicado sobre os naturais. A função característica de M , ou característica de M , é a função

$$c_M : \mathbb{N}_0^n \rightarrow \mathbb{N}_0$$

tal que

- $c_M(x_1, \dots, x_n) = 1$ se $M(x_1, \dots, x_n)$ e
- $c_M(x_1, \dots, x_n) = 0$ caso contrário.

Definição: O predicado M é decidível sse $c_M \in C$.

Exemplo:

- O predicado unário sobre os naturais “ x é par ” é decidível pois o programa

- 1 $J(1, 2, 6)$
- 2 $S(2)$
- 3 $J(1, 2, 7)$
- 4 $S(2)$
- 5 $J(1, 1, 1)$
- 6 $S(3)$
- 7 $T(3, 1)$

calcula a função característica do predicado, ou seja, a função $c : \mathbb{N}_0 \rightarrow \mathbb{N}_0$ tal que $c(x) = 1$ se x é par e $c(x) = 0$ se x não é par.

- O predicado binário sobre os naturais “ x é maior que y ” é decidível pois o programa

- 1 $J(1, 3, 6)$
- 2 $J(2, 3, 5)$
- 3 $S(3)$
- 4 $J(1, 1, 1)$
- 5 $S(4)$
- 6 $T(4, 1)$

calcula a função característica do predicado, ou seja, a função $c : \mathbb{N}_0 \times \mathbb{N}_0 \rightarrow \mathbb{N}_0$ tal que $c(x, y) = 1$ se x é maior que y e $c(x, y) = 0$ se x não é maior que y .

4 Algumas noções necessárias na sequência

Definição: COMPOSIÇÃO SEQUENCIAL DE PROGRAMAS

Sendo $P = \langle p_i \rangle_{i \in 1..k}$ e $Q = \langle q_i \rangle_{i \in 1..r}$ dois programas normalizados, a composição sequencial de P e Q é o programa $S = \langle s_j \rangle_{j \in 1..k+r}$ tal que, para cada $j \in 1..k+r$,

- $s_j = p_j$ se $1 \leq j \leq k$
- $s_j = q_{j-k}$ se $k < j \leq k+r$ e q_{j-k} é $Z(n)$ ou $S(n)$
- $J(m, n, q+k)$ se $k < j \leq k+r$ e q_{j-k} é $J(m, n, q)$.

Na sequência, será relevante poder utilizar um programa P (normalizado) num contexto mais geral, isto é, no contexto de um programa maior, em que os argumentos não estão necessariamente nos primeiros registos e em que se pretende que o resultado do programa (se existir) não seja colocado necessariamente no primeiro registo. No que se segue

$$P[l_1, \dots, l_n \rightarrow l]$$

representa o programa

$$\begin{array}{c} T(l_1, 1) \\ \dots \\ T(l_n, n) \\ Z(n+1) \\ \dots \\ Z(\rho(P)) \\ P \\ T(1, l) \end{array}$$

que representa o programa que calcula a mesma função n -ária que P mas indo buscar os argumentos iniciais aos registos R_{l_1}, \dots, R_{l_n} e colocando o valor final no registo R_l .

5 Definição rigorosa para a noção de computação de um programa

Como se referiu anteriormente, a noção rigorosa de computação de um programa URM a partir de uma configuração é algo elaborada sendo necessário definir algumas noções preliminares auxiliares. Seguidamente apresenta-se a definição rigorosa para a noção de computação.

Notação:

Seja P um programa URM e $\langle r_i \rangle_{i \in \mathbb{N}}$ uma configuração da máquina,

- $[n \rightarrow R_j] \langle r_i \rangle_{i \in \mathbb{N}}$ com $n \in \mathbb{N}_0$ e $j \in \mathbb{N}$, representa a configuração que se obtém a partir de $\langle r_i \rangle_{i \in \mathbb{N}}$ colocando o valor n no registo R_j e deixando inalterados os outros registos

No que se segue \mathcal{P} designa o conjunto de todos os programas URM e \mathbf{R} o conjunto de todas as configurações. As noções auxiliares são as funções

- $ord : \mathcal{P} \times \mathbb{N} \times \mathbf{R} \rightarrow \mathbb{N}$ (ordem do próximo comando)

tal que

$$ord(\langle p_i \rangle_{i \in 1..k}, j, \langle r_i \rangle_{i \in \mathbb{N}}) = \begin{cases} j+1 & \text{se } 1 \leq j < k \text{ e} \\ & p_j \text{ é } Z(n), S(n), T(m, n) \text{ ou} \\ & \text{ou } J(m, n, q) \text{ com } r_m \neq r_n \\ q & \text{se } 1 \leq j \leq k \text{ e} \\ & p_j \text{ é } J(m, n, q) \text{ tal que} \\ & 1 \leq q \leq k \text{ e } r_m = r_n \\ \text{nd} & \text{nos outros casos} \end{cases}$$

Esta função indica qual é a ordem do comando que se vai executar após a execução do comando de ordem j (p_j) sobre a configuração $\langle r_i \rangle_{i \in \mathbb{N}}$.

- $conf : \mathcal{P} \times \mathbb{N} \times \mathbf{R} \rightarrow \mathbf{R}$ (próxima configuração)

tal que

$$conf(\langle p_i \rangle_{i \in 1..k}, j, \langle r_i \rangle_{i \in \mathbb{N}}) = \begin{cases} [0 \rightarrow R_n] \langle r_i \rangle_{i \in \mathbb{N}} & \text{se } 1 \leq j \leq k \\ & \text{e } p_j \text{ é } Z(n) \\ [r_n + 1 \rightarrow R_n] \langle r_i \rangle_{i \in \mathbb{N}} & \text{se } 1 \leq j \leq k \\ & \text{e } p_j \text{ é } S(n) \\ [r_m \rightarrow R_n] \langle r_i \rangle_{i \in \mathbb{N}} & \text{se } 1 \leq j \leq k \\ & \text{e } p_j \text{ é } T(m, n) \\ \langle r_i \rangle_{i \in \mathbb{N}} & \text{se } 1 \leq j \leq k \\ & \text{e } p_j \text{ é } J(m, n, q) \\ \text{nd} & \text{nos outros casos} \end{cases}$$

Esta função indica qual é a configuração que se obtém quando se executa o comando de ordem j (p_j) sobre a configuração $\langle r_i \rangle_{i \in \mathbb{N}}$.

- $\xi : \mathbb{N} \rightarrow \mathbb{N}$ e $\eta : \mathbb{N} \rightarrow \mathbf{R}$, funções mutuamente recursivas, que se definem no contexto de um programa P e uma configuração r da seguinte forma

$$\xi(j) = \begin{cases} 1 & \text{se } j = 1 \\ \text{ord}(P, \xi(j-1), \eta(j-1)) & \text{se } j > 1, \xi(j-1) \downarrow \\ & \text{e } \text{ord}(P, \xi(j-1), \eta(j-1)) \downarrow \\ \text{nd} & \text{nos outros casos} \end{cases}$$

$$\eta(j) = \begin{cases} r & \text{se } j = 1 \\ \text{conf}(P, \xi(j-1), \eta(j-1)) & \text{se } j > 1, \xi(j-1) \downarrow \text{ e} \\ & \text{conf}(P, \xi(j-1), \eta(j-1)) \downarrow \\ \text{nd} & \text{nos outros casos} \end{cases}$$

tendo em conta que $\xi(j-1) \downarrow$ e $\eta(j-1) \downarrow$ significam que $\xi(j-1)$ e $\eta(j-1)$ estão definidos. A função ξ indica, para cada j , qual é (se existir) a ordem do comando que vai ser executado após a execução do comando de ordem $\xi(j-1)$ sobre a configuração $\eta(j-1)$. A função η indica, para cada j , qual é (se existir) a configuração que se obtém quando se executa o comando de ordem $\xi(j-1)$ sobre a configuração $\eta(j-1)$.

Definição: COMPUTAÇÃO DE P A PARTIR DE UMA CONFIGURAÇÃO r

Seja $P = \langle p_i \rangle_{i \in 1..k}$ um programa URM e r uma configuração.

- Uma computação finita de P a partir de r é, se existir, a sequência alternada de configurações e comandos

$$\eta_1 q_1 \eta_2 q_2 \dots \eta_n q_n \eta_{n+1}$$

tal que

- $\eta_j = \eta(j)$ para cada $1 \leq j \leq n+1$
- $\xi(j) \downarrow$ para cada $1 \leq j \leq n$ e $\xi(n+1) \uparrow^5$
- $q_j = p_{\xi(j)}$ para cada $1 \leq j \leq n$.

A configuração η_1 é a configuração inicial e a configuração η_{n+1} é a configuração final.

⁵ou seja, $\xi(n+1)$ não está definido

- Uma computação infinita de P a partir de r é, se existir, a sequência alternada de configurações e comandos

$$\eta_1 q_1 \eta_2 q_2 \dots \eta_n q_n \dots$$

tal que

- $\eta_j = \eta(j)$ para cada $j \geq 1$
- $\xi(j) \downarrow$ para cada $j \geq 1$
- $q_j = p_{\xi(j)}$ para cada $j \geq 1$.

A configuração η_1 é a configuração inicial.