Computación y Estructuras Discretas III

Andrés A. Aristizábal P.
aaaristizabal@icesi.edu.co
Ángela Villota
apvillota@icesi.edu.co

Departamento de Computación y Sistemas Inteligentes
UNIVERSIDAD ICESI

2024-1

Agenda

1. **Turing Machines**
   - Topic presentation
   - Exercises

Agenda

1. **Turing Machines**
   - Topic presentation
   - Exercises

Turing Machines

What else can an MT be used for?

What else can an MT be used for?

Since Turing machines have the ability to transform input strings, erase or overwrite symbols, they can also be used as mechanisms to compute functions.

What else can an MT be used for?

Since Turing machines have the ability to transform input strings, erase or overwrite symbols, they can also be used as mechanisms to compute functions.

How can we formally define a Turing machine as a function calculator?

Turing Machines

What else can an MT be used for?

Since Turing machines have the ability to transform input strings, erase or overwrite symbols, they can also be used as mechanisms to compute functions.

How can we formally define a Turing machine as a function calculator?

Definición

A Turing machine $M = (Q, q_0, q_f, \Sigma, \Gamma, b\!\!\!/, \delta)$ *computes a function* $h : \Sigma^* \rightarrow \Gamma^*$ *if for every input* $w \in \Sigma^*$ *we have*

Turing Machines

What else can an MT be used for?

Since Turing machines have the ability to transform input strings, erase or overwrite symbols, they can also be used as mechanisms to compute functions.

How can we formally define a Turing machine as a function calculator?

Definición

A Turing machine $M = (Q, q_0, q_f, \Sigma, \Gamma, b, \delta)$ computes a function $h : \Sigma^* \to \Gamma^*$ if for every input $w \in \Sigma^*$ we have

$$q_0 w \overset{*}{\vdash} q_f v, \text{ whenever } v = h(w).$$

Turing Machines

What else can an MT be used for?

Since Turing machines have the ability to transform input strings, erase or overwrite symbols, they can also be used as mechanisms to compute functions.

How can we formally define a Turing machine as a function calculator?

Definición

A Turing machine $M = (Q, q_0, q_f, \Sigma, \Gamma, b\!\!\!/, \delta)$ computes a function $h : \Sigma^* \to \Gamma^*$ if for every input $w \in \Sigma^*$ we have

$$q_0 w \overset{*}{\vdash} q_f v, \text{ whenever } v = h(w).$$

- *If there exists a Turing machine that computes the function h, then h is said to be Turing-computable.*

Turing Machines

What else can an MT be used for?

Since Turing machines have the ability to transform input strings, erase or overwrite symbols, they can also be used as mechanisms to compute functions.

How can we formally define a Turing machine as a function calculator?

Definición

A Turing machine $M = (Q, q_0, q_f, \Sigma, \Gamma, b\!\!\!/, \delta)$ *computes a function* $h : \Sigma^* \to \Gamma^*$ *if for every input* $w \in \Sigma^*$ *we have*

$$q_0 w \overset{*}{\vdash} q_f v, \text{ whenever } v = h(w).$$

- *If there exists a Turing machine that computes the function h, then h is said to be Turing-computable.*
- *The Turing machine model used here is standard, but there are no accepting states.*

Turing Machines

What else can an MT be used for?

Since Turing machines have the ability to transform input strings, erase or overwrite symbols, they can also be used as mechanisms to compute functions.

How can we formally define a Turing machine as a function calculator?

Definición

A Turing machine $M = (Q, q_0, q_f, \Sigma, \Gamma, b\!\!\!/, \delta)$ *computes a function* $h : \Sigma^* \to \Gamma^*$ *if for every input* $w \in \Sigma^*$ *we have*

$$q_0 w \overset{*}{\vdash} q_f v, \text{ whenever } v = h(w).$$

- *If there exists a Turing machine that computes the function h, then h is said to be Turing-computable.*
- *The Turing machine model used here is standard, but there are no accepting states.*
- *The state* q_f*, called the final state, is used to terminate the input processing and produce the output.*

Definición

- *Transitions from the final state q_f are not allowed.*

Turing Machines

Definición

- *Transitions from the final state q_f are not allowed.*
- *It is worth noting that M must terminate processing the string in configuration $q_f v$, meaning the control unit must be scanning the first symbol of the output v.*

Definición

- *Transitions from the final state q_f are not allowed.*
- *It is worth noting that M must terminate processing the string in configuration $q_f v$, meaning the control unit must be scanning the first symbol of the output v.*
- *If M does not halt on a particular input w, the function h is not defined on w. In other words, the domain of h consists only of those strings for which M halts.*

Definición

- *Transitions from the final state q_f are not allowed.*
- *It is worth noting that M must terminate processing the string in configuration $q_f v$, meaning the control unit must be scanning the first symbol of the output v.*
- *If M does not halt on a particular input w, the function h is not defined on w. In other words, the domain of h consists only of those strings for which M halts.*
- *It is common to write $h(w)! \downarrow$ to indicate that h is defined on w (i.e., h converges on w) and $h(w)! \uparrow$ to indicate that h is not defined on w (i.e., h diverges on w). If the function h is defined for every input w (i.e., if M halts for every w), h is said to be a total function.*

Definición

- *Transitions from the final state q_f are not allowed.*
- *It is worth noting that M must terminate processing the string in configuration $q_f v$, meaning the control unit must be scanning the first symbol of the output v.*
- *If M does not halt on a particular input w, the function h is not defined on w. In other words, the domain of h consists only of those strings for which M halts.*
- *It is common to write $h(w)! \downarrow$ to indicate that h is defined on w (i.e., h converges on w) and $h(w)! \uparrow$ to indicate that h is not defined on w (i.e., h diverges on w). If the function h is defined for every input w (i.e., if M halts for every w), h is said to be a total function.*
- *A function undefined for some inputs is called a partial function.*

Example

Design a Turing machine M that computes the remainder of the division of n by 3*, for any natural number* $n \geq 1$ *written in unary notation (n is written as a sequence of n ones).*

Example

Design a Turing machine M that computes the remainder of the division of n by 3*, for any natural number* $n \geq 1$ *written in unary notation (n is written as a sequence of n ones).*

Solution

- *The possible remainders of division by* 3 *are* 0, 1*, and* 2*, so only* 3 *states, besides* q_f*, are needed to compute this function.*

Example

Design a Turing machine M that computes the remainder of the division of n by 3, for any natural number $n \geq 1$ written in unary notation (n is written as a sequence of n ones).

Solution

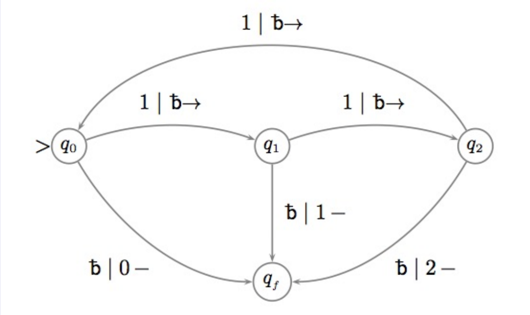
- *The possible remainders of division by 3 are 0, 1, and 2, so only 3 states, besides q_f, are needed to compute this function.*
- *M scans from left to right the input sequence, erasing the ones and alternatingly transitioning between states q_0 (representing remainder 0), q_1 (remainder 1), and q_2 (remainder 2).*

Solution

- *This is the transition diagram of M :*

Solution

- *This is the transition diagram of M* :



Can the notion of a Turing-computable function be extended?

Can the notion of a Turing-computable function be extended?

Yes indeed, the notion of a Turing-computable function can be easily extended to functions of several arguments. Specifically, a function h of k arguments is Turing-computable if for every k-tuple $(w_1, w_2, \ldots, w_k)$ we have

Can the notion of a Turing-computable function be extended?

Yes indeed, the notion of a Turing-computable function can be easily extended to functions of several arguments. Specifically, a function h of k arguments is Turing-computable if for every k-tuple $(w_1, w_2, ..., w_k)$ we have

$$q_0 w_1 \not{b} w_2 \not{b} \cdots \not{b} w_k \not{b} \overset{*}{\vdash} q_f v, \text{ whenever } v = h(w_1, w_2, ..., w_k).$$

Can the notion of a Turing-computable function be extended?

Yes indeed, the notion of a Turing-computable function can be easily extended to functions of several arguments. Specifically, a function h of k arguments is Turing-computable if for every k-tuple $(w_1, w_2, ..., w_k)$ we have

$$q_0 w_1 \not{b} w_2 \not{b} \cdots \not{b} w_k \not{b} \overset{*}{\vdash} q_f v, \text{ whenever } v = h(w_1, w_2, ..., w_k).$$

- Note that to write the input on the tape, the k arguments $w_1, w_2, \ldots, w_k$ are separated by the blank symbol $\not{b}$.

Can the notion of a Turing-computable function be extended?

Yes indeed, the notion of a Turing-computable function can be easily extended to functions of several arguments. Specifically, a function h of k arguments is Turing-computable if for every k-tuple $(w_1, w_2, ..., w_k)$ we have

$$q_0 w_1 \not{b} w_2 \not{b} \cdots \not{b} w_k \not{b} \overset{*}{\vdash} q_f v, \text{ whenever } v = h(w_1, w_2, ..., w_k).$$

- Note that to write the input on the tape, the k arguments $w_1, w_2, \ldots, w_k$ are separated by the blank symbol $\not{b}$.
- Just as before, transitions from the final state q_f are not allowed.

Can the notion of a Turing-computable function be extended?

Yes indeed, the notion of a Turing-computable function can be easily extended to functions of several arguments. Specifically, a function h of k arguments is Turing-computable if for every k-tuple $(w_1, w_2, ..., w_k)$ we have

$$q_0 w_1 \not{b} w_2 \not{b} \cdots \not{b} w_k \not{b} \overset{*}{\vdash} q_f v, \text{ whenever } v = h(w_1, w_2, ..., w_k).$$

- Note that to write the input on the tape, the k arguments $w_1, w_2, \ldots, w_k$ are separated by the blank symbol $\not{b}$.
- Just as before, transitions from the final state q_f are not allowed.
- This definition covers both cases of total functions (defined for all inputs) and partial functions (undefined for some inputs).

Turing Machines

Example

Design a Turing machine M that computes the addition function, in unary notation. This function takes two arguments, $h(n, m) = n + m$*, where* $n, m \geq 1$*. With input* $1^n b 1^m$*, M should produce as output* 1^{n+m}*. The sequences of ones,* 1^n *and* 1^m*, represent the natural numbers n and m, respectively.*

Example

Design a Turing machine M that computes the addition function, in unary notation. This function takes two arguments, $h(n, m) = n + m$, where $n, m \geq 1$. With input $1^n b 1^m$, M should produce as output 1^{n+m}. The sequences of ones, 1^n and 1^m, represent the natural numbers n and m, respectively.

Solution

- *M will transform $1^n b 1^m$ into 1^{n+m} by shifting the string 1^m one cell to the left.*

Turing Machines

Example

Design a Turing machine M that computes the addition function, in unary notation. This function takes two arguments, $h(n, m) = n + m$, where $n, m \geq 1$. With input $1^n \flat 1^m$, M should produce as output 1^{n+m}. The sequences of ones, 1^n and 1^m, represent the natural numbers n and m, respectively.

Solution

- *M will transform $1^n \flat 1^m$ into 1^{n+m} by shifting the string 1^m one cell to the left.*
- *The separator $\flat$ is overwritten by 1, and the last 1 of 1^m is overwritten by the blank symbol.*

Example

Design a Turing machine M that computes the addition function, in unary notation. This function takes two arguments, $h(n, m) = n + m$, where $n, m \geq 1$. With input $1^n b 1^m$, M should produce as output 1^{n+m}. The sequences of ones, 1^n and 1^m, represent the natural numbers n and m, respectively.

Solution

- *M will transform $1^n b 1^m$ into 1^{n+m} by shifting the string 1^m one cell to the left.*
- *The separator b is overwritten by 1, and the last 1 of 1^m is overwritten by the blank symbol.*
- *The transition diagram of the Turing machine implementing this idea is:*

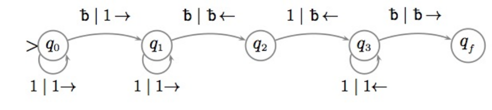
Turing Machines

Example

Design a Turing machine M that computes the addition function, in unary notation. This function takes two arguments, $h(n, m) = n + m$, where $n, m \geq 1$. With input $1^n \not{b} 1^m$, M should produce as output 1^{n+m}. The sequences of ones, 1^n and 1^m, represent the natural numbers n and m, respectively.

Solution

- *M will transform $1^n \not{b} 1^m$ into 1^{n+m} by shifting the string 1^m one cell to the left.*
- *The separator $\not{b}$ is overwritten by 1, and the last 1 of 1^m is overwritten by the blank symbol.*
- *The transition diagram of the Turing machine implementing this idea is:*



Agenda

1. **Turing Machines**
   - Topic presentation
   - Exercises