

Recursion and complexity

(4th lecture)

Isabel Oitavem

CMA and DM, FCT-UNL

This work was partially supported by the Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) through the project UID/MAT/00297/2013 (Centro de Matemática e Aplicações).



FCT Fundação para a Ciência e a Tecnologia
MINISTÉRIO DA EDUCAÇÃO E CIÊNCIA



Models of computation: Nondeterministic Turing machines

A **deterministic Turing machine (TM)** with k tapes is a four-tuple

$$M = \langle Q, \Sigma, \delta, q_0 \rangle$$

where

Q is a finite set of states;

Σ is the tape alphabet;

δ is the transition function,

$$\delta : Q \times \Sigma^k \rightarrow Q \times \Sigma^k \times \{L, N, R\}^k;$$

$q_0 \in Q$ is the initial state.

Models of computation: Nondeterministic Turing machines

A **nondeterministic Turing machine (NTM)** with k tapes is a five-tuple

$$M = \langle Q; \Sigma, \delta, q_0, F \rangle$$

where

Q is a finite set of states;

Σ is the tape alphabet;

δ is the transition function,

$$\delta : Q \times \Sigma^k \rightarrow \mathcal{P}(Q \times \Sigma^{k-1} \times \{L, N, R\}^k);$$

$q_0 \in Q$ is the initial state;

F is the set of accepting final states.

Models of computation: Nondeterministic Turing machines

An **input w is accepted** by a nondeterministic machine M if, and only if, there exists a computation of M on w ending in an accepting configuration.

Models of computation: Nondeterministic Turing machines

An **input w is accepted** by a nondeterministic machine M if, and only if, there exists a computation of M on w ending in an accepting configuration.

Or alternatively, we define a bottom-up labeling of the computation tree (or part of it) of M on w by the following rules:

- ▶ the accepting leaves are labeled 1;
- ▶ any node is labeled 1 if at least one of its sons is labeled 1.

The machine accepts w if, and only if, the root is labeled 1.

Models of computation: Alternating Turing machines

A **nondeterministic Turing machine (NTM)** with k tapes is a five-tuple

$$M = \langle Q; \Sigma, \delta, q_0, F \rangle$$

where

Q is a finite set of states;

Σ is the tape alphabet;

δ is the transition function,

$$\delta : Q \times \Sigma^k \rightarrow \mathcal{P}(Q \times \Sigma^{k-1} \times \{L, N, R\}^k);$$

$q_0 \in Q$ is the initial state;

F is the set of accepting final states.

Models of computation: Alternating Turing machines

A **alternating Turing machine (ATM)** with k tapes is a five-tuple

$$M = \langle Q; \Sigma, \delta, q_0, g \rangle$$

where

Q is a finite set of states;

Σ is the tape alphabet;

δ is the transition function,

$$\delta : Q \times \Sigma^k \rightarrow \mathcal{P}(Q \times \Sigma^{k-1} \times \{L, N, R\}^k);$$

$q_0 \in Q$ is the initial state;

$g : Q \rightarrow \{\vee, \wedge, \text{acc}, \text{rej}\}$.

Models of computation: Alternating Turing machines

Given a tree in which internal nodes are either existential ($\vee$) or universal ($\wedge$), we consider the following labeling procedure

- ▶ the accepting leaves are labeled 1;
- ▶ any existential node is labeled 1 if at least one of its sons has been labeled 1;
- ▶ any universal node is labeled 1 if all its sons are labeled 1.

The machine **accepts the input** if and only if the root of the computation tree is labeled 1

Implicit recursion-theoretic approach: **FPspace**

A function f (over $\mathbb{W}$) is **computable in polynomial space** if, and only if, f is bitwise computable by an **alternating Turing machine** in polynomial time, and the length of the outputs of f is polynomial in the length of the inputs.

FPtime and FPspace: models of computation

- ▶ Model of computation
 - ▶ **FPtime**: Deterministic TM;
 - ▶ **FPspace**: Alternating TM.
- ▶ Resource constraint: **polynomial time**.

DTM

$$\begin{array}{c} c_0 \\ | \\ c_1 \\ | \\ c_2 \\ \vdots \end{array}$$

NTM and **ATM**

$$\begin{array}{ccccccc} & & c_\epsilon & & & & \\ & & \wedge & & & & \\ & c_0 & & c_1 & & & \\ & \wedge & & \wedge & & & \\ c_{00} & c_{01} & & c_{10} & c_{11} & & \\ \vdots & \vdots & & \vdots & \vdots & & \end{array}$$

Implicit recursion-theoretic approach

FPtime = $[\mathcal{SI}; \mathbf{SC}, \mathbf{SR}]$ (Bellantoni-Cook 1992)

FPspace = $[\mathcal{SI}; \mathbf{SC}, ?]$

SR (Input-sorted recursion over $\mathbb{W}$):

$$f(\epsilon, \bar{x}; \bar{y}) = g(\epsilon, \bar{x}; \bar{y})$$

$$f(z0, \bar{x}; \bar{y}) = h(z0, \bar{x}; \bar{y}, f(z, \bar{x}; \bar{y}))$$

$$f(z1, \bar{x}; \bar{y}) = h(z1, \bar{x}; \bar{y}, f(z, \bar{x}; \bar{y}))$$

Example: $f(11)$ leads to $h(11, h(1, g(\epsilon)))$

h

|

h

|

g

Implicit recursion-theoretic approach

FPtime = $[\mathcal{SI}; \mathbf{SC}, \mathbf{SR}]$ (Bellantoni-Cook 1992)

FPspace = $[\mathcal{SI}; \mathbf{SC}, ?]$

SR (Input-sorted recursion over $\mathbb{W}$):

$$f(\epsilon, \bar{x}; \bar{y}) = g(\epsilon, \bar{x}; \bar{y})$$

$$f(z0, \bar{x}; \bar{y}) = h(z0, \bar{x}; \bar{y}, f(z, \bar{x}; \bar{y}))$$

$$f(z1, \bar{x}; \bar{y}) = h(z1, \bar{x}; \bar{y}, f(z, \bar{x}; \bar{y}))$$

Example: $f(11)$ leads to $h(11, h(1, g(\epsilon)))$

h
|
 h
|
 g

SR reproduces the sequential structure of deterministic computations.

Implicit recursion-theoretic approach

FPtime = $[\mathcal{SI}; \mathbf{SC}, \mathbf{SR}]$ (Bellantoni-Cook 1992)

FPspace = $[\mathcal{SI}; \mathbf{SC}, \mathbf{STR}]$ (O. 2008)

SR (Input-sorted recursion over $\mathbb{W}$):

$$f(\epsilon, \bar{x}; \bar{y}) = g(\epsilon, \bar{x}; \bar{y})$$

$$f(z0, \bar{x}; \bar{y}) = h(z0, \bar{x}; \bar{y}, f(z, \bar{x}; \bar{y}))$$

$$f(z1, \bar{x}; \bar{y}) = h(z1, \bar{x}; \bar{y}, f(z, \bar{x}; \bar{y}))$$

STR is defined analogously to **SR**, but

- ▶ double the recursive call
- ▶ distinguish the recursive calls from each other via a pointer p .

Implicit recursion-theoretic approach

FPtime = $[\mathcal{SI}; \mathbf{SC}, \mathbf{SR}]$ (Bellantoni-Cook 1992)

FPspace = $[\mathcal{SI}; \mathbf{SC}, \mathbf{STR}]$ (O. 2008)

SR (Input-sorted recursion over $\mathbb{W}$):

$$f(\epsilon, \bar{x}; \bar{y}) = g(\epsilon, \bar{x}; \bar{y})$$

$$f(z0, \bar{x}; \bar{y}) = h(z0, \bar{x}; \bar{y}, f(z, \bar{x}; \bar{y}))$$

$$f(z1, \bar{x}; \bar{y}) = h(z1, \bar{x}; \bar{y}, f(z, \bar{x}; \bar{y}))$$

STR:

$$f(\epsilon, p, \bar{x}; \bar{y}) = g(\epsilon, p, \bar{x}; \bar{y})$$

$$f(z0, p, \bar{x}; \bar{y}) = h(z0, p, \bar{x}; \bar{y}, f(z, p0, \bar{x}; \bar{y}), f(z, p1, \bar{x}; \bar{y}))$$

$$f(z1, p, \bar{x}; \bar{y}) = h(z1, p, \bar{x}; \bar{y}, f(z, p0, \bar{x}; \bar{y}), f(z, p1, \bar{x}; \bar{y}))$$

Implicit recursion-theoretic approach

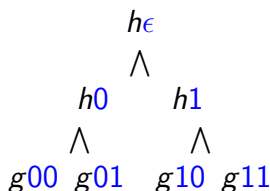
STR: $f(\epsilon, p, \bar{x}; \bar{y}) = g(\epsilon, p, \bar{x}; \bar{y})$

$$f(z0, p, \bar{x}; \bar{y}) = h(z0, p, \bar{x}; \bar{y}, f(z, p0, \bar{x}; \bar{y}), f(z, p1, \bar{x}; \bar{y}))$$

$$f(z1, p, \bar{x}; \bar{y}) = h(z1, p, \bar{x}; \bar{y}, f(z, p0, \bar{x}; \bar{y}), f(z, p1, \bar{x}; \bar{y}))$$

Example:

$f(11, \epsilon)$ leads to $h(\epsilon, h(0, g(00), g(01)), h(1, g(10), g(11)))$.



Implicit recursion-theoretic approach

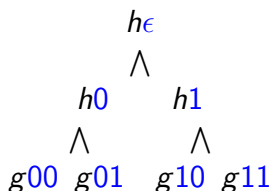
STR: $f(\epsilon, p, \bar{x}; \bar{y}) = g(\epsilon, p, \bar{x}; \bar{y})$

$f(z0, p, \bar{x}; \bar{y}) = h(z0, p, \bar{x}; \bar{y}, f(z, p0, \bar{x}; \bar{y}), f(z, p1, \bar{x}; \bar{y}))$

$f(z1, p, \bar{x}; \bar{y}) = h(z1, p, \bar{x}; \bar{y}, f(z, p0, \bar{x}; \bar{y}), f(z, p1, \bar{x}; \bar{y}))$

Example:

$f(11, \epsilon)$ leads to $h(\epsilon, h(0, g(00), g(01)), h(1, g(10), g(11)))$.



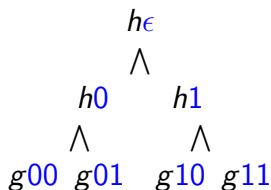
The mentioned input is the **pointer**, and it gives the **address from the root of the tree to the current node**.

STR trivially **extends SR**.

Implicit recursion-theoretic approach

Example:

$f(11, \epsilon)$ leads to $h(\epsilon, h(0, g(00), g(01)), h(1, g(10), g(11)))$.



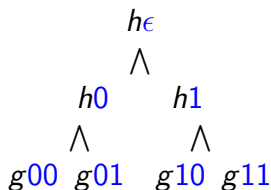
Bottom-up labeling:

(assuming that non-terminating configurations have two successor configurations)

Implicit recursion-theoretic approach

Example:

$f(11, \epsilon)$ leads to $h(\epsilon, h(0, g(00), g(01)), h(1, g(10), g(11)))$.



Bottom-up labeling:

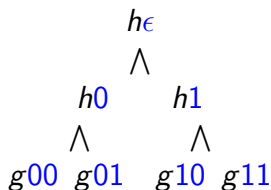
(assuming that non-terminating configurations have two successor configurations)

g and h execute the computation determined by the pointer and read the state of the last computed configuration:

Implicit recursion-theoretic approach

Example:

$f(11, \epsilon)$ leads to $h(\epsilon, h(0, g(00), g(01)), h(1, g(10), g(11)))$.



Bottom-up labeling:

(assuming that non-terminating configurations have two successor configurations)

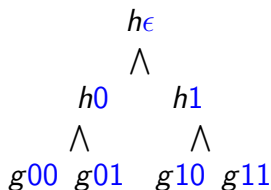
g and h execute the computation determined by the pointer and read the state of the last computed configuration:

- ▶ g returns 1 if it is an accepting state; 0 otherwise.

Implicit recursion-theoretic approach

Example:

$f(11, \epsilon)$ leads to $h(\epsilon, h(0, g(00), g(01)), h(1, g(10), g(11)))$.



Bottom-up labeling:

(assuming that non-terminating configurations have two successor configurations)

g and h execute the computation determined by the pointer and read the state of the last computed configuration:

- ▶ g returns 1 if it is an accepting state; 0 otherwise.
- ▶ h does $\vee$ or $\wedge$ of its last two inputs, depending on the read state.

FPtime = [$\mathcal{SI}$; **SC**, **SR**]
FPspace = [$\mathcal{SI}$; **SC**, **STR**]

(Bellantoni-Cook 1992)
(O. 2008)

FPtime = [$\mathcal{SI}$; **SC**, **SR**]

FPspace = [$\mathcal{SI}$; **SC**, **STR**]

(Bellantoni-Cook 1992)

(O. 2008)

Class	Model of Computation	time bound
FPtime	DTM	poly
NP	NTM	poly
FPspace	ATM	poly
PP	PTM	poly
BPP	PTM	poly + bounded error