

# Lecture 5: Three More Models

## Models of Computation

<https://clegra.github.io/moc/moc.html>

Clemens Grabmayer

Ph.D. Program, Advanced Courses Period  
Gran Sasso Science Institute  
L'Aquila, Italy

July 11, 2025

# Course overview

Monday, July 7 10.30 – 12.30	Tuesday, July 8 10.30 – 12.30	Wednesday, July 9 10.30 – 12.30	Thursday, July 10 10.30 – 12.30	Friday, July 11
<i>intro</i>	<i>classic models</i>			<i>additional models</i>
<b>Introduction to Computability</b>	<b>Machine Models</b>	<b>Recursive Functions</b>	<b>Lambda Calculus</b>	
computation and decision problems, from logic to computability, overview of models of computation relevance of MoCs	Post Machines, typical features, Turing's analysis of human computers, Turing machines, basic recursion theory	primitive recursive functions, Gödel–Herbrand recursive functions, partial recursive funct's, partial recursive = Turing-computable, Church's Thesis	$\lambda$ -terms, $\beta$ -reduction, $\lambda$ -definable functions, partial recursive = $\lambda$ -definable = Turing computable	
	<i>imperative programming</i>	<i>algebraic programming</i>	<i>functional programming</i>	
				14.30 – 16.30
				<b>Three more Models of Computation</b>
				Post's Correspondence Problem, Interaction-Nets, Fractran
				comparing computational power

# Some Models of Computation

machine model	mathematical model	sort
Turing machine Post machine register machine	Combinatory Logic $\lambda$ -calculus Herbrand–Gödel recursive functions partial-recursive/ $\mu$ -recursive functions Post canonical system (tag system) Post's Correspondence Problem Markov algorithms Lindenmayer systems	<i>classical</i>
	Fractran	<i>less well known</i>
cellular automata neural networks	term rewrite systems interaction nets logic-based models of computation concurrency and process algebra $\zeta$ -calculus evolutionary programming/genetic algorithms	<i>modern</i>
	abstract state machines	
	hypercomputation	<i>speculative</i>
	quantum computing bio-computing reversible computing	<i>physics-/biology- inspired</i>

# Overview

- ▶ **Post's Correspondence Problem** (by **Emil Post**, 1946, [5])

# Overview

- ▶ Post's Correspondence Problem (by Emil Post, 1946, [5])
- ▶ Compare computational power of models of computation

# Overview

- ▶ Post's Correspondence Problem (by [Emil Post](#), 1946, [5])
- ▶ Compare computational power of models of computation
- ▶ Interaction Nets (by [Yves Lafont](#), 1990, [4])

# Overview

- ▶ [Post's Correspondence Problem](#) (by [Emil Post](#), 1946, [[5](#)])
- ▶ Compare computational power of models of computation
- ▶ [Interaction Nets](#) (by [Yves Lafont](#), 1990, [[4](#)])
- ▶ [Fractran](#) (by [John Horton Conway](#), 1987, [[2](#)])

# Post's Correspondence Problem

Emil Leon Post:

- ▶ article:
  - ▶ "A Variant of a Recursively Unsolvable Problem"  
Bulletin of the American Mathematical Society, 1946.



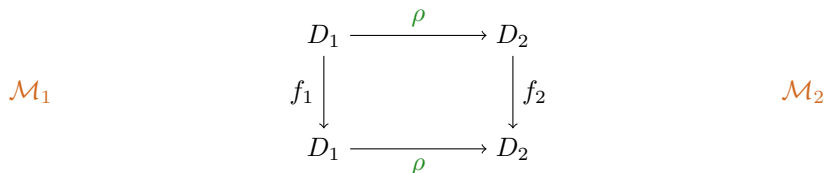
# Comparing computational power via encodings

- ▶ Simulation of models of computation  $\mathcal{M}_1 = \langle D_1, \mathcal{F}_1 \rangle$ ,  $\mathcal{M}_2 = \langle D_2, \mathcal{F}_2 \rangle$ :

# Comparing computational power via encodings

► Simulation of functions:

function  $f_2$  *simulates* function  $f_1$  via *encoding*  $\rho$  if:

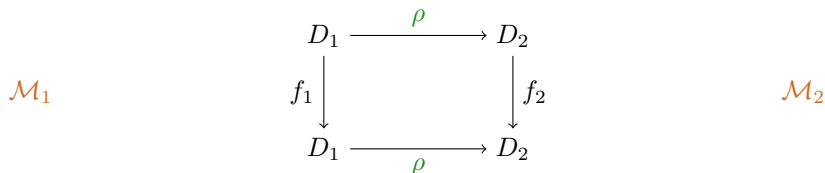


► Simulation of models of computation  $\mathcal{M}_1 = \langle D_1, \mathcal{F}_1 \rangle$ ,  $\mathcal{M}_2 = \langle D_2, \mathcal{F}_2 \rangle$ :

# Comparing computational power via encodings

► Simulation of functions:

function  $f_2$  *simulates* function  $f_1$  via *encoding*  $\rho$  if:

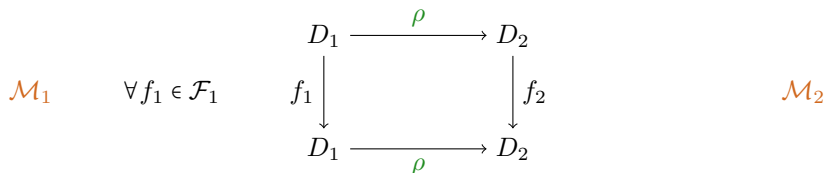


- Simulation of models of computation  $\mathcal{M}_1 = \langle D_1, \mathcal{F}_1 \rangle$ ,  $\mathcal{M}_2 = \langle D_2, \mathcal{F}_2 \rangle$ :  
 $\mathcal{M}_2$  *can simulate*  $\mathcal{M}_1$  via  $\rho$  ( $\mathcal{M}_1 \precsim_{\rho} \mathcal{M}_2$ ), if:

# Comparing computational power via encodings

► Simulation of functions:

function  $f_2$  *simulates* function  $f_1$  via *encoding*  $\rho$  if:

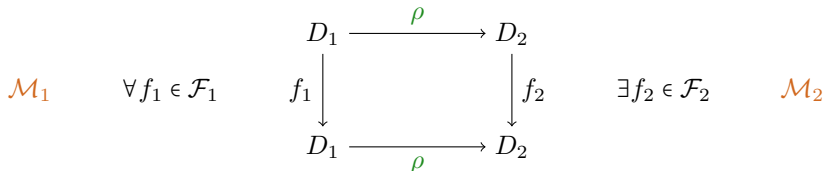


- Simulation of models of computation  $\mathcal{M}_1 = \langle D_1, \mathcal{F}_1 \rangle$ ,  $\mathcal{M}_2 = \langle D_2, \mathcal{F}_2 \rangle$ :  
 $\mathcal{M}_2$  *can simulate*  $\mathcal{M}_1$  via  $\rho$  ( $\mathcal{M}_1 \precsim_{\rho} \mathcal{M}_2$ ), if:

# Comparing computational power via encodings

► Simulation of functions:

function  $f_2$  *simulates* function  $f_1$  via *encoding*  $\rho$  if:

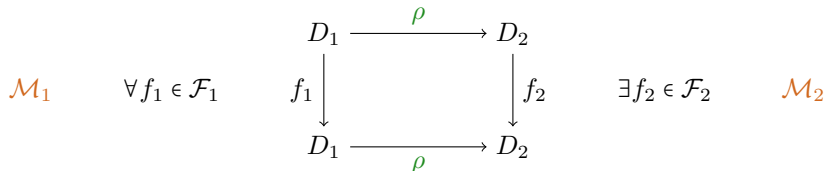


- Simulation of models of computation  $\mathcal{M}_1 = \langle D_1, \mathcal{F}_1 \rangle$ ,  $\mathcal{M}_2 = \langle D_2, \mathcal{F}_2 \rangle$ :  
 $\mathcal{M}_2$  *can simulate*  $\mathcal{M}_1$  via  $\rho$  ( $\mathcal{M}_1 \precsim_{\rho} \mathcal{M}_2$ ), if:

# Comparing computational power via encodings

► Simulation of functions:

function  $f_2$  *simulates* function  $f_1$  via *encoding*  $\rho$  if:



► Simulation of models of computation  $\mathcal{M}_1 = \langle D_1, \mathcal{F}_1 \rangle$ ,  $\mathcal{M}_2 = \langle D_2, \mathcal{F}_2 \rangle$ :  
 $\mathcal{M}_2$  *can simulate*  $\mathcal{M}_1$  via  $\rho$  ( $\mathcal{M}_1 \precsim_\rho \mathcal{M}_2$ ), if:

$$\forall f_1 \in \mathcal{F}_1 \exists f_2 \in \mathcal{F}_2 (f_2 \text{ simulates } f_1 \text{ via } \rho)$$

# Weak requirements on encodings (Boker/Dershowitz)

Traditional requirements on encodings are:

- ▶ *informally computable/effective/mechanizable in principle*
- ▶ *computable* with respect to a specific model (Turing machine, ...)

# Weak requirements on encodings (Boker/Dershowitz)

Traditional requirements on encodings are:

- ▶ *informally computable/effective/mechanizable in principle*
- ▶ *computable* with respect to a specific model (Turing machine, ...)

Boker & Dershowitz [1]: want a ‘**robust definition that does not itself depend on the notion of computability**’, and therefore suggest as encodings:



# Weak requirements on encodings (Boker/Dershowitz)

Traditional requirements on encodings are:

- ▶ *informally computable/effective/mechanizable in principle*
- ▶ *computable* with respect to a specific model (Turing machine, ...)

Boker & Dershowitz [1]: want a ‘*robust definition that does not itself depend on the notion of computability*’, and therefore suggest as encodings:

- injective* functions
- bijective* functions

# Weak requirements on encodings (Boker/Dershowitz)

Traditional requirements on encodings are:

- ▶ *informally computable/effective/mechanizable in principle*
- ▶ *computable* with respect to a specific model (Turing machine, ...)

Boker & Dershowitz [1]: want a ‘**robust definition that does not itself depend on the notion of computability**’, and therefore suggest as encodings:

- injective* functions
- bijjective* functions

Definition (**power subsumption** pre-order [Boker/Dershowitz 2006 [1]])

- $\mathcal{M}_1 \lesssim \mathcal{M}_2$  if: there is an **injective**  $\rho$  such that  $\mathcal{M}_1 \lesssim_{\rho} \mathcal{M}_2$
- $\mathcal{M}_1 \lesssim_{\text{bijjective}} \mathcal{M}_2$  if: there is a **bijjective**  $\rho$  such that  $\mathcal{M}_1 \lesssim_{\rho} \mathcal{M}_2$

# Anomalies for decision models

However, we found anomalies of these definitions.

$\mathcal{M} = \langle D, \mathcal{F} \rangle$  is a *decision model* if  $\{0, 1\} \subseteq D$ ,  $\forall f \in \mathcal{F} (f[D] \subseteq \{0, 1\})$ .

# Anomalies for decision models

However, we found anomalies of these definitions.

$\mathcal{M} = \langle D, \mathcal{F} \rangle$  is a *decision model* if  $\{0, 1\} \subseteq D$ ,  $\forall f \in \mathcal{F} (f[D] \subseteq \{0, 1\})$ .

Theorem (Endrullis/G/Hendriks, [3])

Let  $\Sigma$  and  $\Gamma$  with  $\{0, 1\} \subseteq \Sigma, \Gamma$  be alphabets.

Then for every countable decision model  $\mathcal{M} = \langle \Sigma^*, \mathcal{F} \rangle$ , it holds:

$$\mathcal{M} \lesssim \text{DFA}(\Gamma) \qquad \mathcal{M} \lesssim_{\text{bijective}} \text{DFA}(\Gamma)$$

# Anomalies for decision models

However, we found anomalies of these definitions.

$\mathcal{M} = \langle D, \mathcal{F} \rangle$  is a *decision model* if  $\{0, 1\} \subseteq D$ ,  $\forall f \in \mathcal{F} (f[D] \subseteq \{0, 1\})$ .

Theorem (Endrullis/G/Hendriks, [3])

Let  $\Sigma$  and  $\Gamma$  with  $\{0, 1\} \subseteq \Sigma, \Gamma$  be alphabets.

Then for every countable decision model  $\mathcal{M} = \langle \Sigma^*, \mathcal{F} \rangle$ , it holds:

$$\mathcal{M} \preceq \text{DFA}(\Gamma) \quad \mathcal{M} \preceq_{\text{bijective}} \text{DFA}(\Gamma)$$

$\text{TMD}(\Sigma)$ : class of Turing machine deciders with input alphabet  $\Sigma$

Anomaly (example)

$$\text{TMD}(\Sigma) \preceq_{\text{bijective}} \text{DFA}(\Gamma)$$

# Anomalies for decision models

However, we found anomalies of these definitions.

$\mathcal{M} = \langle D, \mathcal{F} \rangle$  is a *decision model* if  $\{0, 1\} \subseteq D$ ,  $\forall f \in \mathcal{F} (f[D] \subseteq \{0, 1\})$ .

**Theorem (Endrullis/G/Hendriks, [3])**

Let  $\Sigma$  and  $\Gamma$  with  $\{0, 1\} \subseteq \Sigma, \Gamma$  be alphabets.

Then for every countable decision model  $\mathcal{M} = \langle \Sigma^*, \mathcal{F} \rangle$ , it holds:

$$\mathcal{M} \preceq \text{DFA}(\Gamma) \quad \mathcal{M} \preceq_{\text{bijective}} \text{DFA}(\Gamma)$$

$\text{TMD}(\Sigma)$ : class of Turing machine deciders with input alphabet  $\Sigma$

**Anomaly (example)**

$$\text{TMD}(\Sigma) \preceq_{\text{bijective}} \text{DFA}(\Gamma)$$

These anomalies for **decision models** and **bijective encodings**:

- ▶ depend on **uncomputable encodings**

# Anomalies for decision models

However, we found anomalies of these definitions.

$\mathcal{M} = \langle D, \mathcal{F} \rangle$  is a *decision model* if  $\{0, 1\} \subseteq D$ ,  $\forall f \in \mathcal{F} (f[D] \subseteq \{0, 1\})$ .

**Theorem (Endrullis/G/Hendriks, [3])**

Let  $\Sigma$  and  $\Gamma$  with  $\{0, 1\} \subseteq \Sigma, \Gamma$  be alphabets.

Then for every countable decision model  $\mathcal{M} = \langle \Sigma^*, \mathcal{F} \rangle$ , it holds:

$$\mathcal{M} \preceq \text{DFA}(\Gamma) \quad \mathcal{M} \preceq_{\text{bijective}} \text{DFA}(\Gamma)$$

$\text{TMD}(\Sigma)$ : class of Turing machine deciders with input alphabet  $\Sigma$

**Anomaly (example)**

$$\text{TMD}(\Sigma) \preceq_{\text{bijective}} \text{DFA}(\Gamma)$$

These anomalies for **decision models** and **bijective encodings**:

- ▶ depend on **uncomputable encodings**
- ▶ can be extended to **some** moc's with unbounded output domain

# Anomalies for decision models

However, we found anomalies of these definitions.

$\mathcal{M} = \langle D, \mathcal{F} \rangle$  is a **decision model** if  $\{0, 1\} \subseteq D$ ,  $\forall f \in \mathcal{F} (f[D] \subseteq \{0, 1\})$ .

**Theorem (Endrullis/G/Hendriks, [3])**

Let  $\Sigma$  and  $\Gamma$  with  $\{0, 1\} \subseteq \Sigma, \Gamma$  be alphabets.

Then for **every countable decision model**  $\mathcal{M} = \langle \Sigma^*, \mathcal{F} \rangle$ , it holds:

$$\mathcal{M} \preceq \text{DFA}(\Gamma) \quad \mathcal{M} \preceq_{\text{bijective}} \text{DFA}(\Gamma)$$

$\text{TMD}(\Sigma)$ : class of Turing machine deciders with input alphabet  $\Sigma$

**Anomaly (example)**

$$\text{TMD}(\Sigma) \preceq_{\text{bijective}} \text{DFA}(\Gamma)$$

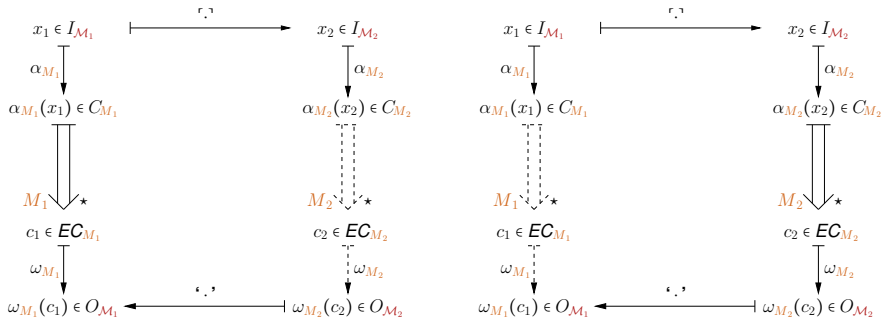
These anomalies for **decision models** and **bijective encodings**:

- ▶ depend on **uncomputable encodings**
- ▶ can be extended to **some** moc's with unbounded output domain
- ▶ but **do not extend** to **all** moc's



# Simulations between models of computation

models  $M_1 \in \mathcal{M}_1$  and  $M_2 \in \mathcal{M}_2$  **simulate each other** with respect to **computable** coding  $\ulcorner \cdot \urcorner : I_{\mathcal{M}_1} \rightarrow I_{\mathcal{M}_2}$  and decoding  $\lceil \cdot \rceil : O_{\mathcal{M}_2} \rightarrow O_{\mathcal{M}_1}$  if:



(defines a **Galois connection**)

# Models of computation, viewed abstractly

A(n abstractly viewed) model of computation (MoC) is a class  $\mathcal{M}$  of machines/systems/. . . such that every  $M \in \mathcal{M}$  it holds:

- ▷  $M$  has a countable set  $I_M$  of input objects, and a countable set  $O_M$  of output objects that are specific to the MoC  $\mathcal{M}$ ;

# Models of computation, viewed abstractly

A(n abstractly viewed) model of computation (MoC) is a class  $\mathcal{M}$  of machines/systems/. . . such that every  $M \in \mathcal{M}$  it holds:

- ▷  $M$  has a countable set  $I_{\mathcal{M}}$  of input objects, and a countable set  $O_{\mathcal{M}}$  of output objects that are specific to the MoC  $\mathcal{M}$ ;
- ▷  $M$  has a set  $C_M$  of configurations of  $M$ , which contains the subset  $EC_M \subseteq C_M$  of end-configurations of  $M$ ;

# Models of computation, viewed abstractly

A(n abstractly viewed) model of computation (MoC) is a class  $\mathcal{M}$  of machines/systems/. . . such that every  $M \in \mathcal{M}$  it holds:

- ▷  $M$  has a countable set  $I_{\mathcal{M}}$  of input objects, and a countable set  $O_{\mathcal{M}}$  of output objects that are specific to the MoC  $\mathcal{M}$ ;
- ▷  $M$  has a set  $C_M$  of configurations of  $M$ , which contains the subset  $EC_M \subseteq C_M$  of end-configurations of  $M$ ;
- ▷  $M$  has an injective input function  $\alpha_M : I_{\mathcal{M}} \rightarrow C_M$ , which maps input objects of  $M$  to configurations of  $M$ ;  $\alpha_M$  is computable;

# Models of computation, viewed abstractly

A(n abstractly viewed) model of computation (MoC) is a class  $\mathcal{M}$  of machines/systems/. . . such that every  $M \in \mathcal{M}$  it holds:

- ▷  $M$  has a countable set  $I_{\mathcal{M}}$  of input objects, and a countable set  $O_{\mathcal{M}}$  of output objects that are specific to the MoC  $\mathcal{M}$ ;
- ▷  $M$  has a set  $C_M$  of configurations of  $M$ , which contains the subset  $EC_M \subseteq C_M$  of end-configurations of  $M$ ;
- ▷  $M$  has an injective input function  $\alpha_M : I_{\mathcal{M}} \rightarrow C_M$ , which maps input objects of  $M$  to configurations of  $M$ ;  $\alpha_M$  is computable;
- ▷  $M$  defines a one-step computation relation  $\Rightarrow_M$  on the set  $C_M$ ; the transitive closure of  $\Rightarrow_M$  is designated by  $\Rightarrow_M^*$ ;

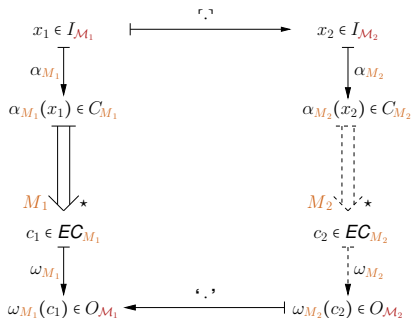
# Models of computation, viewed abstractly

A(n abstractly viewed) model of computation (MoC) is a class  $\mathcal{M}$  of machines/systems/. . . such that every  $M \in \mathcal{M}$  it holds:

- ▷  $M$  has a countable set  $I_{\mathcal{M}}$  of input objects, and a countable set  $O_{\mathcal{M}}$  of output objects that are specific to the MoC  $\mathcal{M}$ ;
- ▷  $M$  has a set  $C_M$  of configurations of  $M$ , which contains the subset  $EC_M \subseteq C_M$  of end-configurations of  $M$ ;
- ▷  $M$  has an injective input function  $\alpha_M : I_{\mathcal{M}} \rightarrow C_M$ , which maps input objects of  $M$  to configurations of  $M$ ;  $\alpha_M$  is computable;
- ▷  $M$  defines a one-step computation relation  $\Rightarrow_M$  on the set  $C_M$ ; the transitive closure of  $\Rightarrow_M$  is designated by  $\Rightarrow_M^*$ ;
- ▷  $M$  has a partial output function  $\omega_M : EC_M \rightarrow O_{\mathcal{M}}$ , which maps some end-configurations of  $M$  to output objects of  $M$ ;  $\omega_M$  is computable, and membership of end-configurations in  $\text{dom}(\omega_M)$  is decidable.

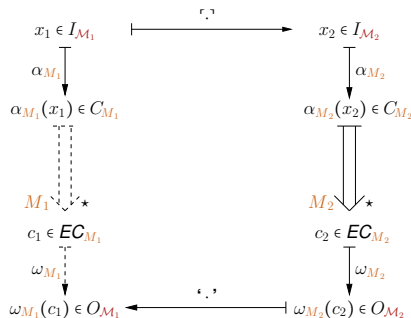
# Simulations between models of computation

models  $M_1 \in \mathcal{M}_1$  and  $M_2 \in \mathcal{M}_2$  **simulate each other** with respect to coding  $\lceil \cdot \rceil : I_{\mathcal{M}_1} \rightarrow I_{\mathcal{M}_2}$  and decoding  $\lfloor \cdot \rfloor : O_{\mathcal{M}_2} \rightarrow O_{\mathcal{M}_1}$  if:



# Simulations between models of computation

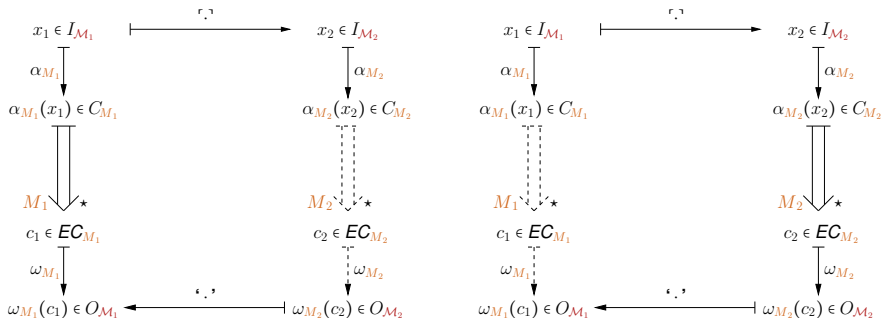
models  $M_1 \in \mathcal{M}_1$  and  $M_2 \in \mathcal{M}_2$  **simulate each other** with respect to coding  $\lceil \cdot \rceil : I_{\mathcal{M}_1} \rightarrow I_{\mathcal{M}_2}$  and decoding  $\lfloor \cdot \rfloor : O_{\mathcal{M}_2} \rightarrow O_{\mathcal{M}_1}$  if:





# Simulations between models of computation

models  $M_1 \in \mathcal{M}_1$  and  $M_2 \in \mathcal{M}_2$  **simulate each other** with respect to coding  $\lceil \cdot \rceil : I_{\mathcal{M}_1} \rightarrow I_{\mathcal{M}_2}$  and decoding  $\lfloor \cdot \rfloor : O_{\mathcal{M}_2} \rightarrow O_{\mathcal{M}_1}$  if:



(defines a **Galois connection**)

# Comparing Computational Power of MoC's

## Definition

Let  $\mathcal{M}_1$  and  $\mathcal{M}_2$  be MoC's.

- ① The computational power of  $\mathcal{M}_1$  is subsumed by that of  $\mathcal{M}_2$ , denoted symbolically by  $\mathcal{M}_1 \leq \mathcal{M}_2$ , if:

( $\exists$  a pair  $\langle \ulcorner \cdot \urcorner, \lceil \cdot \rceil \rangle$  of computable encoding and decoding functions  $\ulcorner \cdot \urcorner : I_{\mathcal{M}_1} \rightarrow I_{\mathcal{M}_2}$  and  $\lceil \cdot \rceil : O_{\mathcal{M}_2} \rightarrow O_{\mathcal{M}_1}$

$(\forall M_1 \in \mathcal{M}_1) (\exists M_2 \in \mathcal{M}_2)$

$[M_1 \text{ and } M_2 \text{ simulate each other w.r.t. } \langle \ulcorner \cdot \urcorner, \lceil \cdot \rceil \rangle]$ .

# Comparing Computational Power of MoC's

## Definition

Let  $\mathcal{M}_1$  and  $\mathcal{M}_2$  be MoC's.

- 1 The computational power of  $\mathcal{M}_1$  is subsumed by that of  $\mathcal{M}_2$ , denoted symbolically by  $\mathcal{M}_1 \leq \mathcal{M}_2$ , if:

( $\exists$  a pair  $\langle \ulcorner \cdot \urcorner, \lceil \cdot \rceil \rangle$  of computable encoding and decoding functions  $\ulcorner \cdot \urcorner : I_{\mathcal{M}_1} \rightarrow I_{\mathcal{M}_2}$  and  $\lceil \cdot \rceil : O_{\mathcal{M}_2} \rightarrow O_{\mathcal{M}_1}$

$(\forall M_1 \in \mathcal{M}_1) (\exists M_2 \in \mathcal{M}_2)$

$[M_1 \text{ and } M_2 \text{ simulate each other w.r.t. } \langle \ulcorner \cdot \urcorner, \lceil \cdot \rceil \rangle]$ ).

- 2 The computational power of  $\mathcal{M}_1$  is equivalent to that of  $\mathcal{M}_2$ , denoted by  $\mathcal{M}_1 \sim \mathcal{M}_2$ , if both  $\mathcal{M}_1 \leq \mathcal{M}_2$  and  $\mathcal{M}_2 \leq \mathcal{M}_1$  hold.

# Comparing Computational Power of MoC's

## Theorem

For all models  $\mathcal{M}_1$  and  $\mathcal{M}_2$ , and encoding and decoding functions  $\lceil \cdot \rceil : I_{\mathcal{M}_1} \rightarrow I_{\mathcal{M}_2}$  and  $\lceil \cdot \rceil : O_{\mathcal{M}_2} \rightarrow O_{\mathcal{M}_1}$  it holds:

$$\mathcal{M}_1 \leq_{(\lceil \cdot \rceil, \lceil \cdot \rceil)} \mathcal{M}_2 \implies \mathcal{F}(\mathcal{M}_1) \subseteq \{ \lceil \cdot \rceil \circ f \circ \lceil \cdot \rceil \mid f \in \mathcal{F}(\mathcal{M}_2) \}.$$

# Turing completeness and equivalence

By  $\mathcal{TM}(\Sigma)$  we mean the model of Turing machines over input alphabet  $\Sigma$ .

## Definition

Let  $\mathcal{M}$  a model of computation.

$\mathcal{M}$  is **Turing-complete** if  $\mathcal{TM}(\Sigma) \leq \mathcal{M}$  for some alphabet  $\Sigma$  with  $\Sigma \neq \emptyset$ .

$\mathcal{M}$  is **Turing-equivalent** if  $\mathcal{M} \sim \mathcal{TM}(\Sigma)$  for some alphabet  $\Sigma \neq \emptyset$ .

# Interaction Nets

Yves Lafont (1990) [4] ([link](#) [pdf](#)) proposed:

- ▶ a programming language with a simple graph rewriting semantics

# Interaction Nets

Yves Lafont (1990) [4] ([link](#) [pdf](#)) proposed:

- ▶ a programming language with a simple graph rewriting semantics

An interaction net is specified by:

- ▶ a set of agents
- ▶ a set of interaction rules

# Interaction Nets

Yves Lafont (1990) [4] ([link pdf](#)) proposed:

- ▶ a programming language with a simple graph rewriting semantics

An interaction net is specified by:

- ▶ a set of agents
- ▶ a set of interaction rules

Analogy with:

- ▶ electric circuits:
  - ▶ agents  $\hat{=}$  gates,
  - ▶ edges  $\hat{=}$  wires



# Interaction Nets

Yves Lafont (1990) [4] ([link pdf](#)) proposed:

- ▶ a programming language with a simple graph rewriting semantics

An interaction net is specified by:

- ▶ a set of agents
- ▶ a set of interaction rules

Analogy with:

- ▶ electric circuits:
  - ▶ agents  $\hat{=}$  gates,
  - ▶ edges  $\hat{=}$  wires
- ▶ agents as computation entities:
  - ▶ interaction rules specify behavior

# Fractran

John Horton Conway:

- ▶ article:
  - ▶ **FRACTRAN:**  
A Simple Universal Programming Language for Arithmetic
- ▶ talk video:
  - ▶ "Fractran: A Ridiculous Logical Language"

# Summary

- ▶ [Post's Correspondence Problem](#) (by [Emil Post](#), 1946, [[5](#)])
- ▶ Compare computational power of models of computation
- ▶ [Interaction Nets](#) (by [Yves Lafont](#), 1990, [[4](#)])
- ▶ [Fractran](#) (by [John Horton Conway](#), 1987, [[2](#)])

# Some Models of Computation

machine model	mathematical model	sort
Turing machine Post machine register machine	Combinatory Logic $\lambda$ -calculus Herbrand–Gödel recursive functions partial-recursive/ $\mu$ -recursive functions Post canonical system (tag system) Post's Correspondence Problem Markov algorithms Lindenmayer systems	<i>classical</i>
	Fractran	<i>less well known</i>
cellular automata neural networks	term rewrite systems interaction nets logic-based models of computation concurrency and process algebra $\zeta$ -calculus evolutionary programming/genetic algorithms	<i>modern</i>
	abstract state machines	
	hypercomputation	<i>speculative</i>
	quantum computing bio-computing reversible computing	<i>physics-/biology- inspired</i>

# Course overview

Monday, July 7 10.30 – 12.30	Tuesday, July 8 10.30 – 12.30	Wednesday, July 9 10.30 – 12.30	Thursday, July 10 10.30 – 12.30	Friday, July 11
<i>intro</i>	<i>classic models</i>			<i>additional models</i>
<b>Introduction to Computability</b>	<b>Machine Models</b>	<b>Recursive Functions</b>	<b>Lambda Calculus</b>	
computation and decision problems, from logic to computability, overview of models of computation relevance of MoCs	Post Machines, typical features, Turing's analysis of human computers, Turing machines, basic recursion theory	primitive recursive functions, Gödel–Herbrand recursive functions, partial recursive funct's, partial recursive = Turing-computable, Church's Thesis	$\lambda$ -terms, $\beta$ -reduction, $\lambda$ -definable functions, partial recursive = $\lambda$ -definable = Turing computable	
	<i>imperative programming</i>	<i>algebraic programming</i>	<i>functional programming</i>	
				14.30 – 16.30
				<b>Three more Models of Computation</b>
				Post's Correspondence Problem, Interaction-Nets, Fractran
				comparing computational power

# References I



Udi Boker and Nachum Dershowitz.

Comparing computational power.

*Logic Journal of the IGPL*, 14(5):633–647, 10 2006.



John Horton Conway.

FRACTRAN: A Simple Universal Programming Language for Arithmetic.

58(2):345–363, April 1936.



Jörg Endrullis, Clemens Grabmayer, and Dimitri Hendriks.

Regularity-Preserving but not Reflecting Encodings.

In *Proceedings of the 30th Annual ACM/IEEE Symposium on Logic in Computer Science 2015 (Kyoto, Japan, July 6–10, 2015)*, pages 535–546, July 2015.



Yves Lafont.

Interaction Nets.

*Proceedings of POPL'90*, pages 95–108, 1990.

# References II



[Emil Leon Post.](#)

A Variant of a Recursively Unsolvable Problem.

*Bulletin of the American Mathematical Society*, 52:264–268,  
1946.