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

# Some Models of Computation

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

# Overview

- Post's Correspondence Problem (by Emil Post, 1946, [6])

# Overview

- Post's Correspondence Problem (by Emil Post, 1946, [6])
- Interaction Nets (by Yves Lafont, 1990, [4])

# Overview

- Post's Correspondence Problem (by Emil Post, 1946, [6])
- Interaction Nets (by Yves Lafont, 1990, [4])
    - Lambdascope (Vincent van Oostrom, 2003, [5])

# Overview

- Post's Correspondence Problem (by Emil Post, 1946, [6])
- Interaction Nets (by Yves Lafont, 1990, [4])
  - Lambdascope (Vincent van Oostrom, 2003, [5])
  - Lambdascope animation tool (Jan Rochel, 2010, [7])

# Overview

- Post's Correspondence Problem (by Emil Post, 1946, [6])
- Interaction Nets (by Yves Lafont, 1990, [4])
  - Lambdascope (Vincent van Oostrom, 2003, [5])
  - Lambdascope animation tool (Jan Rochel, 2010, [7])
- Compare computational power of models of computation

# Overview

- Post's Correspondence Problem (by Emil Post, 1946, [6])
- Interaction Nets (by Yves Lafont, 1990, [4])
  - Lambdascope (Vincent van Oostrom, 2003, [5])
  - Lambdascope animation tool (Jan Rochel, 2010, [7])
- Compare computational power of models of computation
- Fractran (by John Horton Conway, 1987, [2])

# Post’s Correspondence Problem (PCP)

Emil Leon Post:

- ”A Variant of a Recursively Unsolvable Problem”
  Bulletin of the American Mathematical Society, 1946.

# Post's Correspondence Problem (PCP)

Emil Leon Post:

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

Instance of PCP:

$I = \{\langle g_1, g'_1\rangle, \ldots, \langle g_k, g'_k\rangle\}$, where $k \geq 1$, $g_i, g'_i \in \Sigma^+$ for $i \in \{1, \ldots, k\}$.

# Post's Correspondence Problem (PCP)

Emil Leon Post:

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

Instance of PCP:

$I = \{\langle g_1, g_1' \rangle, \ldots, \langle g_k, g_k' \rangle\}$, where $k \geq 1$, $g_i, g_i' \in \Sigma^+$ for $i \in \{1, \ldots, k\}$.

Question: Is $I$ solvable?
Do there exist $n \geq 1$, and $i_1, \ldots, i_n \in \{1, \ldots, k\}$ such that:

$$g_{i_1} g_{i_2} \ldots g_{i_n} = g'_{i_1} g'_{i_2} \ldots g'_{i_n} \quad ?$$

# Post's Correspondence Problem (PCP)

Emil Leon Post:

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

Instance of PCP:

$I = \{\langle g_1, g_1' \rangle, \ldots, \langle g_k, g_k' \rangle\}$, where $k \geq 1$, $g_i, g_i' \in \Sigma^+$ for $i \in \{1, \ldots, k\}$.

Question: Is $I$ solvable?
Do there exist $n \geq 1$, and $i_1, \ldots, i_n \in \{1, \ldots, k\}$ such that:

$$g_{i_1} g_{i_2} \ldots g_{i_n} = g'_{i_1} g'_{i_2} \ldots g'_{i_n} \quad ?$$

**Theorem**

*Codings of solvable instances of PCP:*

$$\Big\{ \Big\langle \overbrace{\{\langle g_1, g_1' \rangle, \ldots, \langle g_k, g_k' \rangle \mid k \geq 1, g_i, g_i' \in \Sigma^+\}}^{\textit{PCP instance } I} \Big\rangle \mid I \textit{ is solvable} \Big\}$$

*form a set that is recursively enumerable, but not recursive.*

# 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 $\triangleq$ gates,
  - edges $\triangleq$ wires
- agents as computation entities:
  - interaction rules specify behavior

# 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 $\mathcal{M}$;

# 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 $\alpha_M : I_{\mathcal{M}} \to C_M$, 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 $\alpha_M : I_{\mathcal{M}} \to C_M$, which maps input objects of $M$ to configurations of $M$; $\alpha_M$ is computable;
- $M$ defines a one-step computation relation $\Rrightarrow_M$ on the set $C_M$; the transitive closure of $\Rrightarrow_M$ is designated by $\Rrightarrow_M^*$;

# 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 $\alpha_M : I_{\mathcal{M}} \to C_M$, which maps input objects of $M$ to configurations of $M$; $\alpha_M$ is computable;
- $M$ defines a one-step computation relation $\Rrightarrow_M$ on the set $C_M$; the transitive closure of $\Rrightarrow_M$ is designated by $\Rrightarrow_M^*$;
- $M$ has a partial output function $\omega_M : EC_M \rightharpoonup O_{\mathcal{M}}$, which maps some end-configurations of $M$ to output objects of $M$; $\omega_M$ is computable, and membership of end-configurations in $dom(\omega_M)$ 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 computable coding $\ulcorner\cdot\urcorner : I_{\mathcal{M}_1} \to I_{\mathcal{M}_2}$ and decoding ‘$\cdot$’ $: O_{\mathcal{M}_2} \to O_{\mathcal{M}_1}$ if:

$$
\begin{tikzcd}
x_1 \in I_{\mathcal{M}_1} \arrow[r, mapsto, "\ulcorner\cdot\urcorner"] \arrow[d, mapsto, "\alpha_{M_1}"'] & x_2 \in I_{\mathcal{M}_2} \arrow[d, mapsto, "\alpha_{M_2}"] \\
\alpha_{M_1}(x_1) \in C_{M_1} \arrow[d, Rightarrow, "M_1"', "\star"] & \alpha_{M_2}(x_2) \in C_{M_2} \arrow[d, Rightarrow, dashed, "M_2"', "\star"] \\
c_1 \in \mathit{EC}_{M_1} \arrow[d, mapsto, "\omega_{M_1}"'] & c_2 \in \mathit{EC}_{M_2} \arrow[d, mapsto, dashed, "\omega_{M_2}"] \\
\omega_{M_1}(c_1) \in O_{\mathcal{M}_1} & \omega_{M_2}(c_2) \in O_{\mathcal{M}_2} \arrow[l, mapsto, "‘\cdot’"']
\end{tikzcd}
\qquad
\begin{tikzcd}
x_1 \in I_{\mathcal{M}_1} \arrow[r, mapsto, "\ulcorner\cdot\urcorner"] \arrow[d, mapsto, "\alpha_{M_1}"'] & x_2 \in I_{\mathcal{M}_2} \arrow[d, mapsto, "\alpha_{M_2}"] \\
\alpha_{M_1}(x_1) \in C_{M_1} \arrow[d, Rightarrow, dashed, "M_1"', "\star"] & \alpha_{M_2}(x_2) \in C_{M_2} \arrow[d, Rightarrow, "M_2"', "\star"] \\
c_1 \in \mathit{EC}_{M_1} \arrow[d, mapsto, dashed, "\omega_{M_1}"'] & c_2 \in \mathit{EC}_{M_2} \arrow[d, mapsto, "\omega_{M_2}"] \\
\omega_{M_1}(c_1) \in O_{\mathcal{M}_1} & \omega_{M_2}(c_2) \in O_{\mathcal{M}_2} \arrow[l, mapsto, "‘\cdot’"']
\end{tikzcd}
$$

(defines a Galois connection)

# 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:

$\mathcal{M}_1$

$$\begin{array}{ccc} D_1 & \xrightarrow{\rho} & D_2 \\ {\scriptstyle f_1}\downarrow & & \downarrow{\scriptstyle f_2} \\ D_1 & \xrightarrow[\rho]{} & D_2 \end{array}$$

$\mathcal{M}_2$

- 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:

$\mathcal{M}_1$

$$\begin{array}{ccc} D_1 & \xrightarrow{\rho} & D_2 \\ {\scriptstyle f_1}\downarrow & & \downarrow{\scriptstyle f_2} \\ D_1 & \xrightarrow[\rho]{} & D_2 \end{array}$$

$\mathcal{M}_2$

- 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 $\rho$ ($\mathcal{M}_1 \lesssim_\rho \mathcal{M}_2$), if:

# Comparing computational power via encodings

- Simulation of functions:
  function $f_2$ *simulates* function $f_1$ via *encoding* $\rho$ if:

$$\mathcal{M}_1 \qquad \forall f_1 \in \mathcal{F}_1 \qquad \begin{array}{ccc} D_1 & \xrightarrow{\rho} & D_2 \\ {\scriptstyle f_1}\downarrow & & \downarrow{\scriptstyle f_2} \\ D_1 & \xrightarrow[\rho]{} & D_2 \end{array} \qquad \mathcal{M}_2$$

- 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 $\rho$ ($\mathcal{M}_1 \lesssim_\rho \mathcal{M}_2$), if:

# Comparing computational power via encodings

- Simulation of functions:
  function $f_2$ *simulates* function $f_1$ via *encoding* $\rho$ if:

$$\mathcal{M}_1 \qquad \forall f_1 \in \mathcal{F}_1 \qquad \begin{array}{ccc} D_1 & \xrightarrow{\ \rho\ } & D_2 \\ {\scriptstyle f_1}\downarrow & & \downarrow{\scriptstyle f_2} \\ D_1 & \xrightarrow[\ \rho\ ]{} & D_2 \end{array} \qquad \exists f_2 \in \mathcal{F}_2 \qquad \mathcal{M}_2$$

- 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 $\rho$ ($\mathcal{M}_1 \lesssim_\rho \mathcal{M}_2$), if:

# Comparing computational power via encodings

- Simulation of functions:
  function $f_2$ *simulates* function $f_1$ via *encoding* $\rho$ if:

$$\mathcal{M}_1 \qquad \forall f_1 \in \mathcal{F}_1 \qquad \begin{array}{ccc} D_1 & \xrightarrow{\rho} & D_2 \\ {\scriptstyle f_1}\downarrow & & \downarrow{\scriptstyle f_2} \\ D_1 & \xrightarrow[\rho]{} & D_2 \end{array} \qquad \exists f_2 \in \mathcal{F}_2 \qquad \mathcal{M}_2$$

- 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 $\rho$ ($\mathcal{M}_1 \lesssim_\rho \mathcal{M}_2$), if:

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

# 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:

(i) *injective* functions

(ii) *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:

(i) *injective* functions

(ii) *bijective* functions

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

(i) $\mathcal{M}_1 \lesssim \mathcal{M}_2$ if: there is an injective $\rho$ such that $\mathcal{M}_1 \lesssim_\rho \mathcal{M}_2$

(ii) $\mathcal{M}_1 \lesssim_{\text{bijective}} \mathcal{M}_2$ if: there is a bijective $\rho$ 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 $\{0,1\} \subseteq D, \quad \forall f \in \mathcal{F} \, (f[D] \subseteq \{0,1\})$.

# 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:

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

# 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:

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

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

### Anomaly (example)

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

# 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:

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

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

**Anomaly (example)**

$$\mathrm{TMD}(\Sigma) \lesssim_{\mathrm{bijective}} \mathrm{DFA}(\Gamma)$$

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 $\{0,1\} \subseteq D, \quad \forall f \in \mathcal{F}\,(f[D] \subseteq \{0,1\})$.

**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:

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

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

**Anomaly (example)**

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

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:

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

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

## Anomaly (example)

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

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 coding $\ulcorner\cdot\urcorner : I_{\mathcal{M}_1} \to I_{\mathcal{M}_2}$ and decoding $‘\cdot’ : O_{\mathcal{M}_2} \to O_{\mathcal{M}_1}$ if:

$$
\begin{tikzcd}
x_1 \in I_{\mathcal{M}_1} \arrow[r, mapsto, "\ulcorner\cdot\urcorner"] \arrow[d, mapsto, "\alpha_{M_1}"'] & x_2 \in I_{\mathcal{M}_2} \arrow[d, mapsto, "\alpha_{M_2}"] \\
\alpha_{M_1}(x_1) \in C_{M_1} \arrow[d, Rightarrow, "M_1"', "\star"] & \alpha_{M_2}(x_2) \in C_{M_2} \arrow[d, Rightarrow, dashed, "M_2"', "\star"] \\
c_1 \in \mathit{EC}_{M_1} \arrow[d, mapsto, "\omega_{M_1}"'] & c_2 \in \mathit{EC}_{M_2} \arrow[d, mapsto, dashed, "\omega_{M_2}"] \\
\omega_{M_1}(c_1) \in O_{\mathcal{M}_1} & \omega_{M_2}(c_2) \in O_{\mathcal{M}_2} \arrow[l, mapsto, "‘\cdot’"']
\end{tikzcd}
$$

# 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 $\ulcorner\cdot\urcorner : I_{\mathcal{M}_1} \to I_{\mathcal{M}_2}$ and decoding $‘\cdot’ : O_{\mathcal{M}_2} \to O_{\mathcal{M}_1}$ if:

$$
\begin{tikzcd}
x_1 \in I_{\mathcal{M}_1} \arrow[rr, mapsto, "\ulcorner\cdot\urcorner"] \arrow[d, mapsto, "\alpha_{M_1}"'] & & x_2 \in I_{\mathcal{M}_2} \arrow[d, mapsto, "\alpha_{M_2}"] \\
\alpha_{M_1}(x_1) \in C_{M_1} \arrow[d, Rightarrow, dashed, "{}_{M_1}\Downarrow^{\star}"'] & & \alpha_{M_2}(x_2) \in C_{M_2} \arrow[d, Rightarrow, "{}_{M_2}\Downarrow^{\star}"] \\
c_1 \in \mathit{EC}_{M_1} \arrow[d, mapsto, dashed, "\omega_{M_1}"'] & & c_2 \in \mathit{EC}_{M_2} \arrow[d, mapsto, "\omega_{M_2}"] \\
\omega_{M_1}(c_1) \in O_{\mathcal{M}_1} & & \omega_{M_2}(c_2) \in O_{\mathcal{M}_2} \arrow[ll, mapsto, "‘\cdot’"']
\end{tikzcd}
$$

# 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 $\ulcorner \cdot \urcorner : I_{\mathcal{M}_1} \to I_{\mathcal{M}_2}$ and decoding $`\cdot' : O_{\mathcal{M}_2} \to O_{\mathcal{M}_1}$ if:

$$
\begin{tikzcd}
x_1 \in I_{\mathcal{M}_1} \arrow[r, mapsto, "\ulcorner \cdot \urcorner"] \arrow[d, mapsto, "\alpha_{M_1}"'] & x_2 \in I_{\mathcal{M}_2} \arrow[d, mapsto, "\alpha_{M_2}"] \\
\alpha_{M_1}(x_1) \in C_{M_1} \arrow[d, Rightarrow, "M_1"', "\star"] & \alpha_{M_2}(x_2) \in C_{M_2} \arrow[d, Rightarrow, dashed, "M_2"', "\star"] \\
c_1 \in \mathit{EC}_{M_1} \arrow[d, mapsto, "\omega_{M_1}"'] & c_2 \in \mathit{EC}_{M_2} \arrow[d, mapsto, dashed, "\omega_{M_2}"] \\
\omega_{M_1}(c_1) \in O_{\mathcal{M}_1} & \omega_{M_2}(c_2) \in O_{\mathcal{M}_2} \arrow[l, mapsto, "`\cdot'"']
\end{tikzcd}
\qquad
\begin{tikzcd}
x_1 \in I_{\mathcal{M}_1} \arrow[r, mapsto, "\ulcorner \cdot \urcorner"] \arrow[d, mapsto, "\alpha_{M_1}"'] & x_2 \in I_{\mathcal{M}_2} \arrow[d, mapsto, "\alpha_{M_2}"] \\
\alpha_{M_1}(x_1) \in C_{M_1} \arrow[d, Rightarrow, dashed, "M_1"', "\star"] & \alpha_{M_2}(x_2) \in C_{M_2} \arrow[d, Rightarrow, "M_2"', "\star"] \\
c_1 \in \mathit{EC}_{M_1} \arrow[d, mapsto, dashed, "\omega_{M_1}"'] & c_2 \in \mathit{EC}_{M_2} \arrow[d, mapsto, "\omega_{M_2}"] \\
\omega_{M_1}(c_1) \in O_{\mathcal{M}_1} & \omega_{M_2}(c_2) \in O_{\mathcal{M}_2} \arrow[l, mapsto, "`\cdot'"']
\end{tikzcd}
$$

(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 $\langle \ulcorner\cdot\urcorner, `\cdot' \rangle$ of computable encoding and decoding functions $\ulcorner\cdot\urcorner : I_{\mathcal{M}_1} \to I_{\mathcal{M}_2}$ and $`\cdot' : O_{\mathcal{M}_2} \to O_{\mathcal{M}_1}$

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

   $$\Big[\, M_1 \text{ and } M_2 \text{ simulate each other w.r.t. } \langle \ulcorner\cdot\urcorner, `\cdot' \rangle \,\Big].$$

# 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 $\langle \ulcorner\cdot\urcorner, `\cdot' \rangle$ of computable encoding and decoding functions $\ulcorner\cdot\urcorner : I_{\mathcal{M}_1} \to I_{\mathcal{M}_2}$ and $`\cdot' : O_{\mathcal{M}_2} \to O_{\mathcal{M}_1}$

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

   $$\big[\, M_1 \text{ and } M_2 \text{ simulate each other w.r.t. } \langle \ulcorner\cdot\urcorner, `\cdot' \rangle \,\big].$$

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 $\ulcorner\cdot\urcorner : I_{\mathcal{M}_1} \to I_{\mathcal{M}_2}$ and $`\cdot' : O_{\mathcal{M}_2} \to O_{\mathcal{M}_1}$ it holds:*

$$\mathcal{M}_1 \leq_{\langle \ulcorner\cdot\urcorner, `\cdot' \rangle} \mathcal{M}_2 \implies \mathcal{F}(\mathcal{M}_1) \subseteq \left\{ `\cdot' \circ f \circ \ulcorner\cdot\urcorner \;\middle|\; f \in \mathcal{F}(\mathcal{M}_2) \right\}.$$

# 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 $\Sigma \neq \varnothing$.

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

# 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, [6])
- Interaction Nets (by Yves Lafont, 1990, [4])
- Compare computational power of models of computation
- Fractran (by John Horton Conway, 1987, [2])

# Some Models of Computation

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

# Course overview

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


- Vincent van Oostrom, Kees-Jan van de Looij, and Marijn Zwitserlood.
  Lambdascope.
  Extended Abstract, Workshop ALPS, Kyoto, April 10th 2004, 2004.
  http://www.phil.uu.nl/~oostrom/publication/pdf/lambdascope.pdf.

- Emil Leon Post.
  A Variant of a Recursively Unsolvable Problem.
  *Bulletin of the American Mathematical Society*, 52:264–268, 1946.

# References III


Jan Rochel.
graph-rewriting-lambdascope: Lambdascope, an optimal evaluator of the lambda calculus.
Haskell package on Hackage, `https://hackage.haskell.org/package/graph-rewriting-lambdascope`, 2010.
Lambdascope interaction-net animation tool.