Lecture 5: Three More Models

Models of Computation

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

Clemens Grabmayer

Department of Computer Science

GRAN SASSO SCIENCE INSTITUTE
SCHOOL OF ADVANCED STUDIES
Scuola Universitaria Superiore

L'Aquila, Italy

Teaching Mobility Program (PNRR-TNE DESK)
University of Novi Sad
Novi Sad, Serbia

March 2026

Course overview

🟩	🟩	🟩	🟩	🟪
intro	*classic models*			*additional models*
Introduction to Computability	**Machine Models**	**Recursive Functions**	**Lambda Calculus**	**Three more Models of Computation**
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	λ-terms, β-reduction, λ-definable functions, partial recursive = λ-definable = Turing computable	Post's Correspondence Problem, Interaction-Nets, Fractran
	imperative programming	*algebraic programming*	*functional programming*	

Some Models of Computation

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

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])

Emil Post

Emil Leon Post (1897–1954)

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:

$I = \big\{\langle g_1, g'_1\rangle, \ldots, \langle g_k, g'_k\rangle\big\}$, 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.

Yves Lafont

Yves Lafont

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

Comparing computational power via encodings

- Simulation of functions:
  function f_2 *simulates* function f_1 via *encoding* ρ if:

$$\mathcal{M}_1 \qquad \forall f_1 \in \mathcal{F}_1 \qquad \begin{tikzcd} D_1 \arrow[r, "\rho"] \arrow[d, "f_1"'] & D_2 \arrow[d, "f_2"] \\ D_1 \arrow[r, "\rho"'] & D_2 \end{tikzcd} \qquad \exists f_2 \in \mathcal{F}_2 \qquad \mathcal{M}_2$$

- 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 ρ ($\mathcal{M}_1 \lesssim_\rho \mathcal{M}_2$), 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, . . .)

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

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

Let Σ and Γ 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 Σ

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
- 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 $\ulcorner\cdot\urcorner : I_{\mathcal{M}_1} \to I_{\mathcal{M}_2}$ 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)

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; α_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 $\omega_M : EC_M \rightharpoonup O_{\mathcal{M}}$, which maps some end-configurations of M to output objects of M; ω_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 ‘ · ’ $: O_{\mathcal{M}_2} \to O_{\mathcal{M}_1}$ if:

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

(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 $`\cdot' : O_{\mathcal{M}_2} \to O_{\mathcal{M}_1}$

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

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

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 Σ.

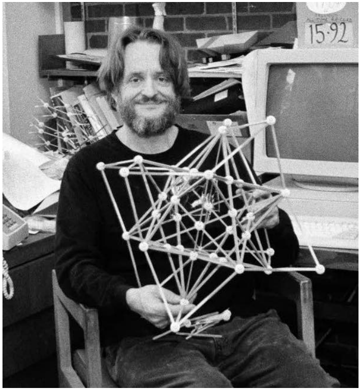
Definition

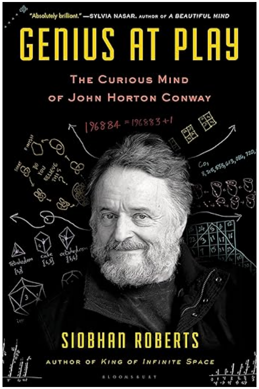
Let $\mathcal{M}$ a model of computation.

$\mathcal{M}$ is Turing-complete if $\mathcal{TM}(\Sigma) \le \mathcal{M}$ for some alphabet Σ with $\Sigma \neq \varnothing$.

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

John Horton Conway




"Absolutely brilliant." —SYLVIA NASAR, AUTHOR OF A BEAUTIFUL MIND
GENIUS AT PLAY
THE CURIOUS MIND OF JOHN HORTON CONWAY
196884 = 196883 + 1
SIOBHAN ROBERTS
AUTHOR OF KING OF INFINITE SPACE
BLOOMSBURY

John Horton Conway (1937–2020)

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

Course overview

intro	*classic models*			*additional models*
Introduction to Computability	**Machine Models**	**Recursive Functions**	**Lambda Calculus**	**Three more Models of Computation**
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	λ-terms, β-reduction, λ-definable functions, partial recursive = λ-definable = Turing computable	Post's Correspondence Problem, Interaction-Nets, Fractran
	imperative programming	*algebraic programming*	*functional programming*	

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.