Lecture 5: Three More Models

Models of Computation

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
intro	*classic models*			*additional models*
Introduction to Computability	**Machine Models**	**Recursive Functions**	**Lambda Calculus**	
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	
	imperative programming	*algebraic programming*	*functional programming*	
				14.30 – 16.30
				Three more Models of Computation
				Post's Correspondence Problem, Interaction-Nets, Fractran
				comparing computational power

Some Models of Computation

<table>
<tr><th>machine model</th><th>mathematical model</th><th>sort</th></tr>
<tr><td rowspan="2">Turing machine
Post machine
register machine</td><td>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</td><td rowspan="2">classical</td></tr>
<tr><td></td></tr>
<tr><td></td><td>Fractran</td><td>less well known</td></tr>
<tr><td rowspan="2">cellular automata
neural networks</td><td>term rewrite systems
interaction nets
logic-based models of computation
concurrency and process algebra
ς-calculus
evolutionary programming/genetic algorithms</td><td rowspan="2">modern</td></tr>
<tr><td colspan="2">abstract state machines</td></tr>
<tr><td colspan="2">hypercomputation</td><td>speculative</td></tr>
<tr><td colspan="2">quantum computing
bio-computing
reversible computing</td><td>physics-/biology-inspired</td></tr>
</table>

Overview

- Compare computational power of models of computation

Overview

- Compare computational power of models of computation
- Post's Correspondence Problem (by Emil Post, 1946, [4])

Overview

- Compare computational power of models of computation
- Post's Correspondence Problem (by Emil Post, 1946, [4])
- Interaction Nets (by Yves Lafont, 1990, [3])

Overview

- Compare computational power of models of computation
- Post's Correspondence Problem (by Emil Post, 1946, [4])
- Interaction Nets (by Yves Lafont, 1990, [3])
- Fractran (by John Horton Conway, 1987, [1])

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;

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;
- M defines a one-step computation relation $\mapsto_M$ on the set C_M; the transitive closure of $\mapsto_M$ is designated by $\mapsto_M^*$;

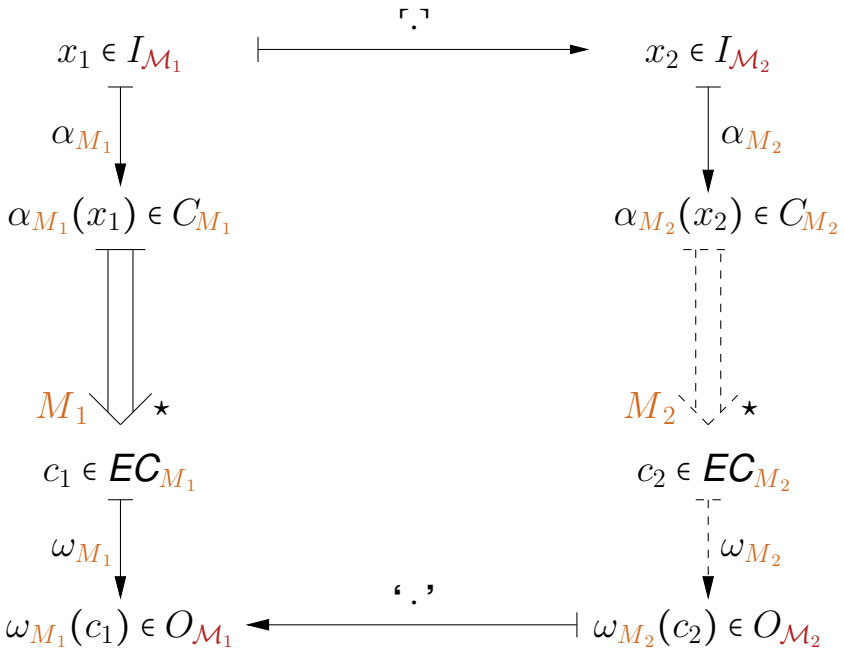
Models of computation, viewed abstractly

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;
- M defines a one-step computation relation $\mapsto_M$ on the set C_M; the transitive closure of $\mapsto_M$ is designated by $\mapsto_M^*$;
- 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;

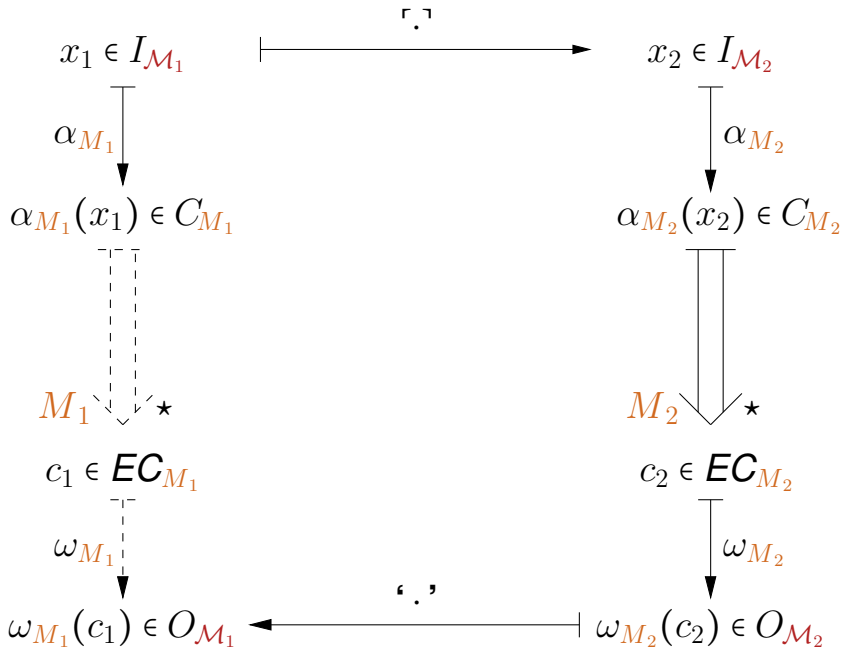
Simulations between models of computation

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



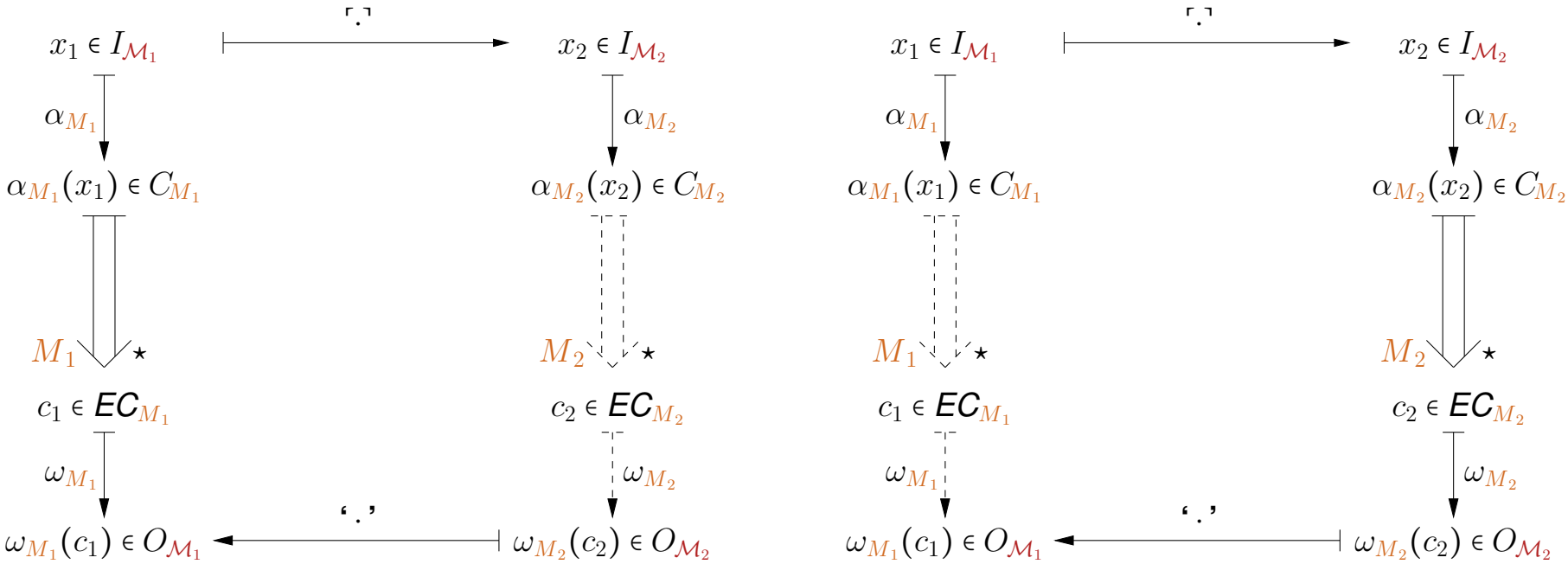
Simulations between models of computation

models $M_1 \in \mathcal{M}_1$ and $M_2 \in \mathcal{M}_2$ simulate each other with respect to coding $\ulcorner\cdot\urcorner : I_{\mathcal{M}_1} \to I_{\mathcal{M}_2}$ and decoding $\text{`}\cdot\text{'} : O_{\mathcal{M}_2} \to O_{\mathcal{M}_1}$ 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 $\ulcorner\cdot\urcorner : I_{\mathcal{M}_1} \to I_{\mathcal{M}_2}$ and decoding $\text{`}\cdot\text{'} : O_{\mathcal{M}_2} \to O_{\mathcal{M}_1}$ 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 $\langle \ulcorner \cdot \urcorner, \text{`} \cdot \text{'} \rangle$ of encoding and decoding functions
   $\ulcorner \cdot \urcorner : I_{\mathcal{M}_1} \to I_{\mathcal{M}_2}$ and $\text{`} \cdot \text{'} : O_{\mathcal{M}_2} \to O_{\mathcal{M}_1}$
   $(\forall M_1 \in \mathcal{M}_1)\,(\exists M_2 \in \mathcal{M}_2)$
   $\big[\, M_1$ and M_2 simulate each other w.r.t. $\langle \ulcorner \cdot \urcorner, \text{`} \cdot \text{'} \rangle \,\big]$.

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 $\langle \ulcorner\cdot\urcorner, \text{`}\cdot\text{'} \rangle$ of encoding and decoding functions
   $\ulcorner\cdot\urcorner : I_{\mathcal{M}_1} \to I_{\mathcal{M}_2}$ and $\text{`}\cdot\text{'} : O_{\mathcal{M}_2} \to O_{\mathcal{M}_1}$
   $(\forall M_1 \in \mathcal{M}_1)\,(\exists M_2 \in \mathcal{M}_2)$
   $\big[\, M_1$ and M_2 simulate each other w.r.t. $\langle \ulcorner\cdot\urcorner, \text{`}\cdot\text{'} \rangle \,\big]$.

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 $\lq\cdot\rq : O_{\mathcal{M}_2} \to O_{\mathcal{M}_1}$ it holds:

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

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 $\mathcal{TM}(\Sigma) \leq \mathcal{M}$ for some alphabet Σ with $\Sigma \neq \varnothing$.

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

Post's Correspondence Problem

Interaction Nets

Yves Lafont (1990) [3] proposed a programming language:

- a simple graph rewriting semantics,
-
-

Fractran

Some Models of Computation

<table>
<tr><th>machine model</th><th>mathematical model</th><th>sort</th></tr>
<tr><td rowspan="2">Turing machine
Post machine
register machine</td><td>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</td><td rowspan="2">classical</td></tr>
<tr><td></td></tr>
<tr><td></td><td>Fractran</td><td>less well known</td></tr>
<tr><td rowspan="2">cellular automata
neural networks</td><td>term rewrite systems
interaction nets
logic-based models of computation
concurrency and process algebra
ς-calculus
evolutionary programming/genetic algorithms</td><td rowspan="2">modern</td></tr>
<tr><td colspan="2">abstract state machines</td></tr>
<tr><td colspan="2">hypercomputation</td><td>speculative</td></tr>
<tr><td colspan="2">quantum computing
bio-computing
reversible computing</td><td>physics-/biology-inspired</td></tr>
</table>

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
intro	*classic models*			*additional models*
Introduction to Computability	**Machine Models**	**Recursive Functions**	**Lambda Calculus**	
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	
	imperative programming	*algebraic programming*	*functional programming*	
				14.30 – 16.30
				Three more Models of Computation
				Post's Correspondence Problem, Interaction-Nets, Fractran
				comparing computational power

References I

John Horton Conway.
FRACTRAN: A Simple Universal Programming Language for Arithmetic.
58(2):345–363, April 1936.

Maribel Fernández.
Models of Computation (An Introduction to Computability Theory).
Springer, Dordrecht Heidelberg London New York, 2009.

Yves Lafont.
Interaction Nets.
Proceedings of POPL'90, pages 95–108, 1990.

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