

PROGRAMS AS FIRST-CLASS CITIZENS IN A BIOLOGY-MOTIVATED MODEL OF COMPUTATION

Lars Hartmann

Neil D. Jones

Jakob Grue Simonsen

+ Visualization by Søren Bjerregaard Vrist

(All now or recently at the University of Copenhagen)

Christopher Strachey lecture, Oxford (May 1, 2012)

Sources:

- ▶ Conference **CS2BIO** Computer Science to Biology (ENTCS proceedings June 2010)
- ▶ Journal **Scientific Annals of Computer Science** (2011, Vol. XXI)
- ▶ **Festschrift for Carolyn Talcott** (November 2011, Springer Festschrift Series, LNCS vol. 7000)
- ▶ Article accepted to appear in **Philosophical Transactions A** of the Royal Society

MY EARLY INSPIRATIONS

- ▶ Turing machines, studied since 1962
- ▶ A compiler for ALGOL 60, written in 1962-65

There were many unclear semantic points!

But what was a semantics???

- ▶ The IFIP Working Conference on **Formal Language Description Languages for Computer Programming** 1964, including
- ▶ **“Towards a formal semantics”** by Christopher Strachey
- ▶ **Denotational semantics**, e.g., from Joe Stoy’s 1977 book

Denotational Semantics: The Scott-Strachey Approach to Programming Language Semantics

ALAN TURING STARTED THE BALL ROLLING (IN 1936)

1. A convincing analysis of the nature of computation
2. A very early model of computation (MOC)
3. The first programmers' manual
4. Undecidability of the halting problem
5. The Universal Turing machine (a self-interpreter)
6. Contributor to the **“Confluence of ideas”**: that
all sensible models of computation are equivalent, e.g.,
 - ▶ Turing machine
 - ▶ Lambda calculus
 - ▶ Recursive function definitions
 - ▶ String rewrite systems

75 YEARS OF MODELS OF COMPUTATION (just a few)

Lambda calculus	Church 1936
Turing machine	1936
von Neumann architecture	1945
Finite automata	Rabin and Scott
Counter machine	Lambek and Minsky
Random access machine (RAM)	Cook and Reckhow
Random access stored program (RASP)	Elgot and Robinson
Cellular automaton, LIFE,..	von Neumann, Conway, Wolfram
Abstract state machine	Gurevich et al
Text register machine	Moss
Blob model	2010

SOME DIMENSIONS OF OF MOCS

- ▶ “Reasonable” machines (van Emde Boas, Ugo dal Lago)
PTIME is the same on Turing machine and λ -calculus
- ▶ General problem-solving
- ▶ Programmability
- ▶ Binding times, finiteness and uniformity
- ▶ Turing completeness
- ▶ Universal machine / self-interpreter

The Blob MOC:

- ▶ Originally motivated by biological computing, which has
- ▶ enormous potential (price, concurrency, automation, ...)
- ▶ A different set of MOC dimensions; may give some insight.

A VISIT TO STANFORD RESEARCH INSTITUTE IN 2008

SRI is doing quality work **to model biological systems** using Maude, a **term rewriting system implementation**. But... **my reaction after a 2 month visit**: where are the programs?

- ▶ Many, at the simulation level, i.e., Maude programs to simulate biological phenomena.
- ▶ But I could see **no programs at the biological level**.

In biomolecular computation models **it's hard to see anything like a program that realises or directs a computational process**.

- ▶ Many examples: given a problem, researchers **cleverly devise a biomolecular system** that can solve this particular problem
- ▶ The **algorithm being implemented** is hidden in the details of the system's construction, hard to see.

EXISTING CONNECTIONS BETWEEN BIOLOGY AND COMPUTATION

Turing completeness results for biomolecular computation:

- ▶ Cardelli, Chapman, Danos, Reif, Shapiro, Wolfram,...
- ▶ Net effect: any computable function can be computed, **in some sense**, by various biological mechanisms.
- ▶ **Not completely compelling** from a programming perspective.
(Gödel numbers, 2-counter machine simulation, ...)
- ▶ Our aim: a computation model where
 - “**program**” is **clearly visible and natural**, and
 - **Turing completeness is not artificial or accidental or horribly inefficient**, but belongs naturally to biomolecular computation

TWO DIFFERENT MEANINGS OF THE WORD “MODEL”

1. **Natural sciences** are **analytic**: a “model” describes an already-existing reality.

A **good model** describes the real world well, e.g., is usable to predict the outcome of not-yet-performed experiments.

2. **Computer science and engineering** are **synthetic**: Given a problem specification, a goal is build a computer program or a hardware device to solve it. (cf. “model checking”)

A **good model** satisfies the problem specification.

A small insight:

- ▶ The “confluence of ideas” had **analytic** overtones, suggesting that **computability is a natural phenomenon**.
- ▶ Turing’s work (machine design, programming) was **synthetic**.

A DIFFERENCE IN PERSPECTIVE

- ▶ Natural science is **analytic**: how does nature really work?
- ▶ Computer science is **synthetic**: build programmable systems.

Therefore, **this research project**:

design a biology-like computing model **with programs**.

The blob model aims to fill a gap,

- ▶ to establish a biologically feasible framework in which
- ▶ **programs are first-class citizens.**

THE BIOLOGICAL WORLD IS NOT HARDWARE!

Need to **re-examine** programming language assumptions.

Computers have **programmer-friendly conveniences**, e.g.,

- ▶ A large **address space** of randomly accessible data
- ▶ **Pointers** to data, perhaps at a great “distance” from the current program or data
- ▶ **address arithmetic, index registers,...**
- ▶ **Unbounded fan-in**: many pointers to the same data item

None of these is biologically plausible!

Workarounds are needed

if we want to do biological programming.

FOR BIOCHEMICAL PLAUSIBILITY

- ▶ **There is no action at a distance** all effects achieved via **chains of local interactions**. Biological analog: **signaling**.
- ▶ **There are no pointers to data** (addresses, links, list pointers): To be acted on, a data value must be **physically adjacent** to an actuator. Biological analog: **chemical bond** between program and data.
- ▶ **A “yes”** \exists available resources to tap, i.e., energy to change the program control point, or to add data bonds.
Biological analogs: **ATP, oxygen, Brownian movement**.

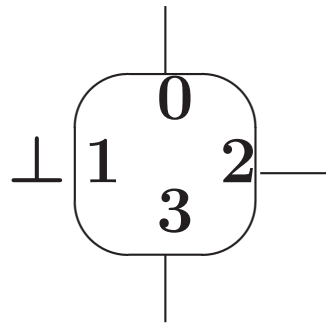
THE BLOB MODEL

Simplified view of a molecule and chemical interactions

(like Cardelli, Danos, Lanève,...)

Blobs are in a biological “soup” and are connected by **symmetrical bonds** linking their bond sites.

Picture of a blob: (Bond sites 0, 2 and 3 are bound, and 1 is unbound)



A blob has **4 bond sites** and **8 cargo bits** (boolean values).

Here: Bond sites 0, 2 and 3 are bound, and 1 is unbound.
(Cargo bits not shown)

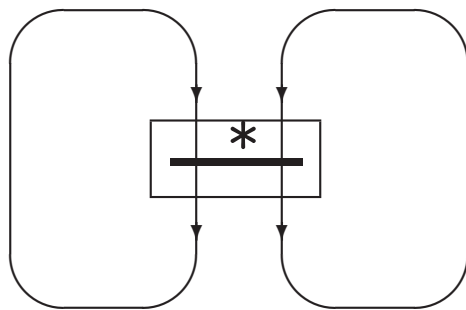
KEEPING THE FOCUS


How to structure a biologically feasible model of computation?


- ▶ Idea: keep current **program cursor** and **data cursor** always close to a focus point where all actions occur.
- ▶ How? Continually shift **both program and data**, to keep the active bits near the focus.

Running program p : computing $\llbracket p \rrbracket (d)$

Program p Data d

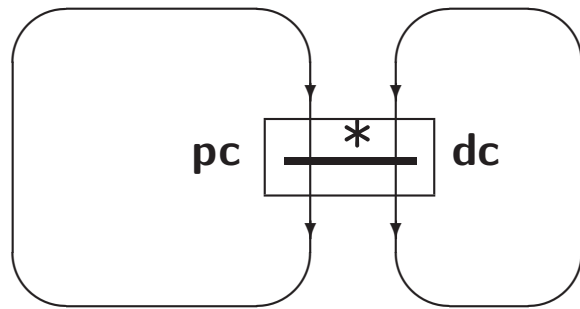


 = Focus point for control and data
(connects the PC and the DC)

 = **program-to-data bond: “the bug”**

WHAT HAPPENS AT THE PROGRAM-TO-DATA BOND ?

Program p Data d – each with its cursor



□ = Focus point for control and data
(connects the PC and the DC)

* = **program-to-data bond**

An instruction at pc can ...

- ▶ **Move** the data cursor along a bond
- ▶ **Branch**: is a data cursor's bond empty or not ?
- ▶ **Branch**: is a data cargo bit 1 or 0 ?
- ▶ **Insert** a new blob at a bond
- ▶ **Swap**: interchange some bonds
- ▶ **Fan-in**: merge control from two predecessor instructions

A MOVIE IS WORTH $\text{DURATION} \times \text{FRAMERATE} \times 1000$
WORDS

(Circle.avi)

AIMS OF THE BLOB MODEL

A model of computation that is

- ▶ **biochemically plausible**: semantics by chemical-like reaction rules;
- ▶ **programmable** (a bit like low-level computer machine code);
- ▶ **uniform**: new “hardware” not needed to solve new problems;
- ▶ **stored-program**: programs = data;
programs are **executable** and **compilable** and **interpretable**
- ▶ **universal**: all computable functions can be computed
- ▶ **Turing complete** in a strong sense: \exists a universal algorithm
(able to execute any program, asymptotically efficient)

OTHER COMPUTATIONAL FRAMEWORKS

Circuits, BDDs, finite automata: Nonuniform, Turing incomplete!

Turing machine:

- ▶ **Pro Visible program;** complete; universal machine exists
- ▶ **Con Asymptotically slow:** universal machine takes time $O(n^2)$ to simulate a program running in time $O(n)$

Other program-based models: Post, Minsky, LISP, RAM, RASP...

Complex, biologically implausible

Cellular automata: von Neumann, LIFE, Wolfram,...

- ▶ **Pro:** Can simulate a Turing machine
- ▶ **Con:** Complex, **biologically implausible** (synchronisation!)
- ▶ Program = start cell pattern? global transition function?
- ▶ There seems to be no natural universal cellular automaton.

PROGRAM BLOBS AND DATA BLOBS

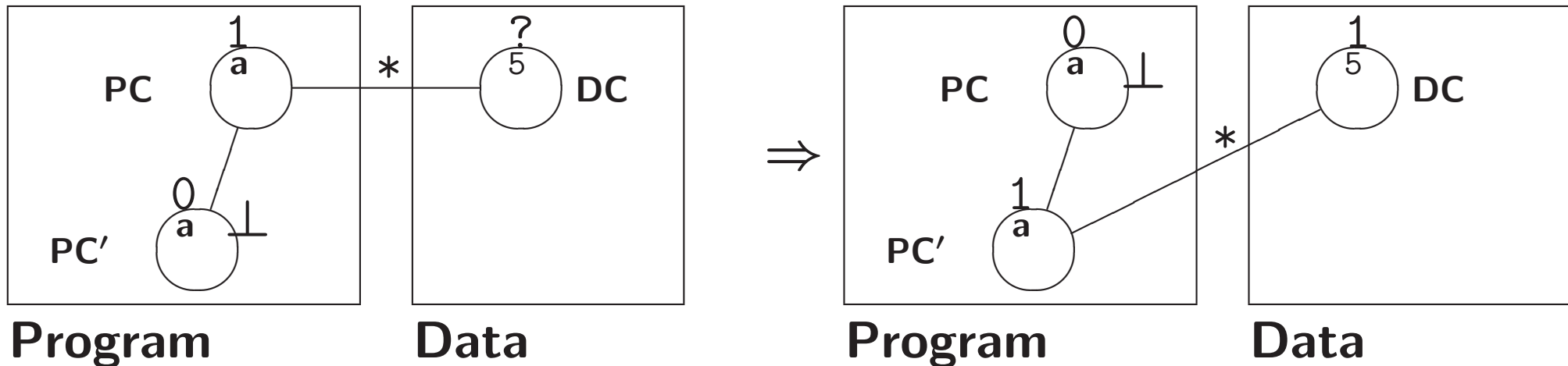
- ▶ A program p is (by definition) a connected assembly of blobs.
- ▶ The data space d is (also) a connected assembly of blobs.

At any moment during execution, i.e., computation of $[[p]](d)$:

- ▶ The **program cursor** (PC) is in p .
- ▶ The **data cursor** (DC) is in d .
- ▶ There is a bond $*$ (“the bug”) between the PC and the DC, at bond sites 0.

EXAMPLE INSTRUCTION: SCG 1 5

(SET CARGO BIT 5 TO 1)



► “The bug” * has moved:

- before execution, it connected PC with DC.
- After: it connects successor PC' with DC.

► Control: **activation bits** 0, 1 have been swapped.

Instruction syntax: the 8-bit string 11001101 is grouped as

$\underbrace{1}_a \underbrace{100}_{SCG} \underbrace{1}_v \underbrace{101}_c$

ANOTHER ANIMATION: APPEND

(Append.avi)

MORE ABOUT INSTRUCTIONS:

(one per blob)

Instruction form: (8 control bits and 4 bonds)

opcode parameters (bond0, bond1, bond2, bond3)

Why exactly 4 bonds?

- ▶ Predecessor (1 bond); true and false successors (2 bonds);
- ▶ 1 bond to link the program cursor and the data cursor.

It's almost a von Neumann machine code, but...

- ▶ A bond is a **two-way link between two adjacent blobs**.
- ▶ A bond is not an address.
- ▶ There is no address space as in conventional computer (and hence: no address decoding hardware).
- ▶ Also: no registers (though cargo bits can be used).

Language M is as powerful as L (write $L \leq M$) if

$$\forall p \in L\text{-programs} \exists q \in M\text{-programs} (\llbracket p \rrbracket^L = \llbracket q \rrbracket^M)$$

L and M are languages (biological, programming, whatever).

Aim: show that an interesting M is Turing complete.

Usual way: **reduce** an already Turing complete language, e.g.,

- ▶ from: $L =$ two-counter machines 2CM. **very, very slow!**
- ▶ to: $M =$ a biomolecular system of the sort being studied.
- ▶ The technical trick: show **how to construct**
 - **from** any 2CM program,
 - a biomolecular M -system **that simulates** the given 2CM.

TURING'S WAY: INTERPRETATION

Turing completeness is usually shown by **simulation**, e.,g.,

- ▶ for any 2CM program you can build a biomolecular system

But the biomolecular system is usually built by hand.

The effect: **hand computation** of the \exists quantifier in

$$\forall p \exists q ([p]^L = [q]^M)$$

In contrast, Turing's original "Universal machine" (UM) works by **interpretation**, where \exists is realised by machine.

- ▶ The UM can execute **any** TM program, if coded on the UM's tape along with its input data.
- ▶ Our research follows Turing's line, in a biological context: It does **simulation by general interpretation**, and not by **one-problem-at-a-time** constructions.

PROGRAM EXECUTION BY INTERPRETATION

$$\llbracket \text{interpreter} \rrbracket (\text{program}, \text{data}_{in}) = \llbracket \text{program} \rrbracket (\text{data}_{in})$$

- ▶ Now program is a **passive data object**: both program and data_{in} are data for the **interpreter**.
- ▶ program is now executed by **running the interpreter program**.
- ▶ This **self-interpretation** is useful in practice.
- ▶ **Turing's original "Universal machine"** was a self-interpreter.

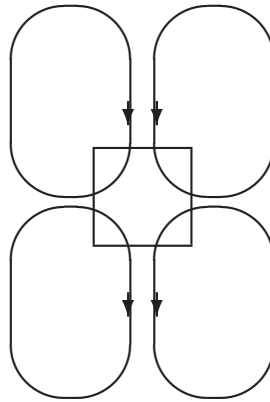
A "blob universal machine"

We have programmed a self-interpreter for the blob formalism – analogous to Turing's original universal machine.

This gives: Turing-completeness in a new biological framework.

SELF-INTERPRETATION IN THE BLOB WORLD

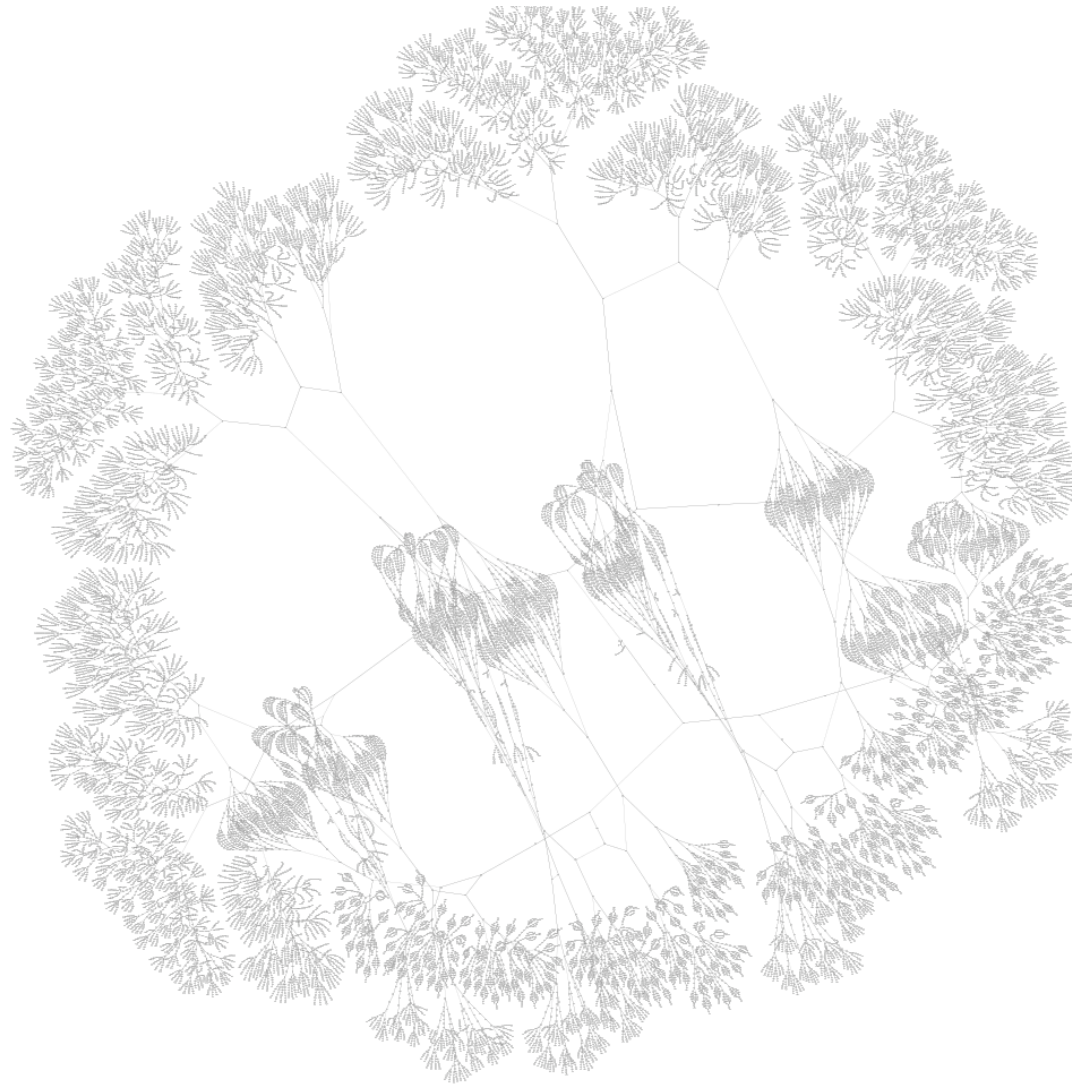
Interpreter and its data



Program p Data d

Picture of the computation: $[[\text{interpreter}]](p, d)$

BIRDS-EYE VIEW OF A SELF-INTERPRETER



(Not shown: Each 'finger' along the periphery has a connection to the main control in the center)

A “BLOB UNIVERSAL MACHINE”

We have developed a self-interpreter for the blob formalism – analogous to Turing’s original universal machine.

This gives: Turing-completeness in a new biological framework.

Blob programs do not have to be encoded!

Self-interpretation without asymptotic slowdown.

- ▶ the blob model has higher connectivity than the TM;
- ▶ faster self-interpretation than original universal machine.

Why is asymptotic slowdown avoided?

**The time to interpret one blob instruction
is bounded by a constant c
(that may depend on the program being interpreted)**

SOME DESIRABLE PROPERTIES OF A MOC

- ▶ **Existence of programs**; and **general problem solving**: a natural path from an informal algorithm to an MOC program.
- ▶ **Turing completeness**.
- ▶ **Uniformity** and **strong finiteness**: one set of hardware is enough for all problems.
- ▶ **Physical realizability**, e.g., execution possible without action at a distance, e.g., data pointers.
- ▶ **Programs as data objects**: Readability for universal machine. Writeability for program generation, e.g., a compiler.
- ▶ **Plausible program running times**, e.g., polynomially related to programming languages, e.g., λ -calculus.

WHAT SEEMS JUST AROUND THE CORNER

- ▶ Programs are currently similar to classical **machine code**; this requires (too much) programmer skill. Possible solutions:
 - Devise an intermediate-level blob programming language (Christopher Strachey, we need you!)
 - Describe/constrain program behavior, data structures by **static program analysis**; or a **type system**.
 - **Program activation** should be possible: once a program is generated, **start executing it**. Needs “stored program” model (as in von Neumann architecture or RASP).
- ▶ **Needed**: bounds on time or energy to perform a **single program step**. A **cost model**, including code motion.
- ▶ **Concurrency** (programs perhaps generated dynamically by one master program, analogous to biological reproduction.)

WHAT HAS NOT YET BEEN DONE

- ▶ Promise of **tighter analogy between universality and self-reproduction.**
- ▶ A usable higher-level programming language
- ▶ Find a **true, biological** (not just “plausible”) implementation of the fixed set of reduction rules in vitro.
- ▶ **Computational complexity**, e.g., limitations imposed by a **3-dimensional blob-space.**

CONTRIBUTIONS OF THIS WORK

- ▶ Programmable **bio-level computation** where **programs = data**.
- ▶ Blob semantics by **abstract biochemical reaction rules**.
- ▶ All computable functions are blob-computable:
 - This can be done with **one fixed instruction set**
(i.e., a “machine language”)
 - **We don't need new rule sets** (biochemical architectures) to solve new problems; it's **enough to write new programs**.
- ▶ (Uniform) Turing-completeness
- ▶ Interpreters and compilers seem to make sense at biological level, may give useful operational and utilitarian tools.

References

- [1] **L.M. Adleman**. On constructing a molecular computer. In DIMACS, AMS, pages 1–21, 1996.
- [2] **L. Cardelli, G. Zavattaro**. Turing universality of the biochemical ground form. MSCS, 19, 2009.
- [3] **Paul Chapman**. Life universal computer. <http://www.igblan.free-online.co.uk/igblan/ca/>, 2002.
- [4] **V. Danos, J. Feret, W. Fontana,, J. Krivine**. Abstract interpretation of cellular signalling networks. Volume 4905 of VMCAI, Lecture Notes in Computer Science, pages 83–97, October 1970.
- [5] **V. Danos, C. Laneve**. Formal molecular biology. TCS, 325:69 – 110, 2004.
- [6] **M. Gardner**. Mathematical recreations. Scientific American, October 1970.
- [7] **M. Hagiya**. Designing chemical, biological systems. New Generation Comput., 26(3):295, 2008.
- [8] **L. Kari, G. Rozenberg**. The many facets of natural computing. Commun. ACM, 51(10):72–83, 2008.
- [9] **E. Shapiro**. Mechanical Turing machine: Blueprint for a biomolecular computer, Weizmann, 1999.
- [10] **E. Shapiro, Y. Benenson**. Bringing DNA computers to life. Scientific American, 294:44–51, 2006.
- [11] **C. Talcott**. Pathway logic. Volume 5016 of SFM, LNCS, pages 21–53, 2008.
- [12] **J. v. Neumann, A.W. Burks**. Theory of Self-Reproducing Automata. Univ. Illinois Press, 1966.
- [13] **E. Winfree**. Toward molecular programming with DNA. SIGOPS Oper. Syst. Rev., 42(2):1–1, 2008.
- [14] **S. Wolfram**. A New Kind of Science. 2002.

THANK YOU!

Questions?