Speaking the Language of Molecules

Luca Cardelli
Microsoft Research

BioBITs, Torino 2010-05-06
http://lucacardelli.name
Outline

• Molecular Structures
  o Getting smaller
  o Self–assembly

• Molecular Languages
  o Natural languages: proteins, genes, membranes
  o Modeling languages (systems biology)
  o Executable languages (nano–engineering)

• Molecular Compilation
  o Intermediate Languages
  o Analysis Tools and Techniques
  o Nano–programming workflow
Molecular Structures
Smaller and Smaller

First working transistor

First integrated circuit

50 years later

25nm NAND flash
Intel&Micron, Jan. 2010. ~50atoms.

Single molecule transistor

Molecules on a chip

~10 Moore’s Law cycles left!

Building The Smallest Things

- How do we build structures that are by definition smaller than your tools?
- Basic answer: you can’t. Structures (and tools) should build themselves!
- By *programmed self-assembly*.

www.youtube.com/watch?v=Ey7Emmddf7Y
Molecular IKEA

• Nature can self-assemble. Can we?

• “Dear IKEA, please send me a chest of drawers that assembles itself.”

• We need a magical material where the pieces are pre-programmed to fit into each other.

• At the molecular scale many such materials exist...

Programmed Self–Assembly

Proteins

DNA/RNA

Membranes
Molecular Languages
- natural languages -
Abstract Machines of Biochemistry

Gene Machine
Nucleotides

Regulation

Protein Machine
Aminoacids

Make proteins
Send signals
Hold receptors, host reactions
Enact fusion, fission

Membrane Machine
Phospholipids

Confinement, Storage
Bulk Transport

Metabolism, Propulsion
Signaling, Transport

Confine genome and regulators
Direct construction
Hold receptors, host reactions
Enact fusion, fission

Surface and Extracellular Features

Data Structures of Biochemistry

- Gene Machine
- Protein Machine
- Membrane Machine

- Strings
- Records
- Hierarchical Multisets
- Trees

Glycan Machine

Nucleotides, Amino Acids, Phospholipids, Glycans, Machine, Sugars, Gene, Protein, Membrane, Records, Hierarchical Multisets, Trees.
Languages of Biochemistry

Gene Machine

Gene Regulatory Networks

Transport Networks

Biochemical Networks

Protein Machine

Membrane Machine

x

y

A

B

C

P

Q
The Protein Machine

On/Off switches

Protein

Binding Sites

Inaccessible

Switching accessible switches
- May cause other switches and binding sites to become (in)accessible.

Binding accessible sites
- May cause other switches and binding sites to become (in)accessible.

cf. BioCalculus [Kitano&Nagasaki], κ-calculus [Danos&Laneve]
Molecular Interaction Maps (Kohn/Kitano)

CellDesigner

Regulation of a gene influences transcription. The regulatory region has precise DNA sequences meant for binding regulators.

Transcription produces molecules (RNA or, through RNA, proteins) that bind to regulatory region of other genes (or that are end-products).

Human (and mammalian) Genome Size
- 3Gbp (Giga base pairs) 750MB @ 4bp/Byte (CD)
- Non-repetitive: 1Gbp 250MB
- In genes: 320Mbp 80MB
- Coding: 160Mbp 40MB
- Protein-coding genes: 30,000–40,000

M.Genitalium (smallest true organism)
- 580,073bp 145KB (eBook)

E.Coli (bacteria): 4Mbp 1MB (floppy)

Yeast (eukarya): 12Mbp 3MB (MP3 song)

Wheat 17Gbp 4.25GB (DVD)
Function of a Regulatory Region

The Membrane Machine

Molecular transport and transformation through dynamic compartment fusion and fission.

Voet, Voet & Pratt
Fundamentals of Biochemistry
Wiley 1999. Ch10 Fig 10–22.
... in 3D

Controlled by surface proteins
Integration

Gene Machine

Slow
Digital

Fast
Analog

Turing Completeness

Protein Machine

Small

Membrane Machine

Big

Digital

Analog

Fast
Molecular Languages
- modeling languages -
From Instructions to Programs

• We have seen the instruction sets:
  o Proteins – complexation, phosphorilation
  o Genes – activation, inhibition
  o Membranes – fusion, fission

• How do we combine them into programs?
  o I.e., into models (quantitative programs)

• How do we study their semantics?
  o I.e., their kinetics (quantitative semantics)
Chemistry

• Chemical reactions
  o $A + B \rightarrow_r C + D$ (a program)

• Ordinary Differential Equations
  o $\frac{d[A]}{dt} = -r[A][B]$ … (a semantics)

• Rich analytical techniques based on Calculus

• But prone to combinatorial explosion
  o Due to the peculiarities of protein interactions
High(er)–Level Languages

- **Protein Networks**
  - Process Algebra (stochastic π-calculus etc.)
    - Priami, Regev–Shapiro, etc.
  - Graph Rewriting (kappa, BioNetGen etc.)
    - Danos–Laneve, Fontana & al., etc.

- **Gene Networks**
  - Synchronous Boolean networks
    - Stewart Kauffman, etc.
  - Asynchronous Boolean networks
    - René Thomas, etc.

- **Membrane Networks**
  - Membrane Computing
    - Gheorghe Păun, etc.
  - Brane Calculi
    - Luca Cardelli, etc.
Reactions vs. Reagents

Says what “A” does.

r: A + B →_{k1} C + D
s: C + D →_{k2} A + B

Says what “A” is.

A = !_{r_{k1}}; C
C = ?_{s_{k2}}; A
B = ?_{r_{k1}}; D
D = !_{s_{k2}}; B

The same “math model”

CTMC

Reaction oriented

1 line per reaction

Interaction oriented

1 line per agent

Does A become C or D?

A becomes C not D!
Formal Connections

These diagrams commute via appropriate maps.

L. Cardelli: “On Process Rate Semantics” (TCS)
L. Cardelli: “A Process Algebra Master Equation” (QEST’07)
Execution?

• Chemistry is not easily executable
  o Please Mr Chemist, execute me these reactions that I just made up

• Similarly, the molecular languages seen so fare are **descriptive** (modeling) languages

• How can we actually **execute** molecular languages? With real molecules?
Molecular Languages
- executable languages -
Nanoscale Control Systems

- **Sensing**
  - Reacting to forces
  - Binding to molecules

- **Actuating**
  - Releasing molecules
  - Producing forces

- **Constructing**
  - Chassis
  - Growth

- **Computing**
  - Signal Processing
  - Decision Making

Nucleic Acids can do all this.
And interface to **biology**.
DNA

GC Base Pair
Guanine–Cytosine

TA Base Pair
Thymine–Adenine

Sequence of Base Pairs (GACT alphabet)

Interactive DNA Tutorial
(http://www.biosciences.bham.ac.uk/labs/minchin/tutorials/dna.html)
Robust, and *Long*

- **DNA in each human cell:**
  - 3 billion base pairs
  - 2 meters long, 2nm thick
  - folded into a 6µm ball
  - 750 MegaBytes

- **A huge amount for a cell**
  - Every time a cell replicates it has to copy *2 meters of DNA* reliably.
  - To get a feeling for the scale disparity, compute:

- **DNA in human body**
  - 10 trillion cells
  - 133 Astronomical Units long
  - 7.5 OctaBytes

- **DNA in human population**
  - 20 million light years long

---

DNA wrapping into chromosomes

Andromeda Galaxy
2.5 million light years away
Zipping Along

- DNA can support structural and computational complexity.

DNA replication in *real time*

- In Humans: 50 nucleotides/second
  - Whole genome in a few hours (with parallel processing)
- In Bacteria: 1000 nucleotides/second
  - (higher error rate)

DNA transcription in *real time*

- RNA polymerase II:
  - 15–30 bases/second

Drew Berry
http://www.wehi.edu.au/wehi-tv
**Sensing**

**Aptamers**: natural or artificially evolved DNA molecules that stick to other molecules (highly selectively).

Adenine riboswitch aptamer

Constructing

Crosslinking

Chengde Mao, Purdue

Andrew Turberfield, Oxford
Actuating

DNA tweezers

Bernard Yurke, Boise State

DNA walkers
Computing

• Sensors and Actuators at the 'edge' of the system
  o They can use disparate technologies and phenomena

• Computation in the 'kernel' of the system

• Compositionality in the kernel
  o The components should use uniform inputs and outputs
  o The components should be ‘computationally complete’
“Embedded” Computing

• Using bacterial machinery (e.g.) as the hardware. Using embedded gene networks as the software.

• MIT Registry of Standard Biological Parts

• GenoCAD
  o Meaningful sequences [Cai et al.]

• GEC
  o [Pedersen & Phillips]
“Autonomous” Computing

• Mix & go
  o All (or most) parts are synthesized
  o No manual cycling (cf. early DNA computing)
  o In some cases, all parts are made of DNA (no enzyme/proteins)

• Self-assembled and self-powered
  o Can run on its own (e.g. environmental sensing)
  o Or be embedded into organisms, but running ‘separately’
Curing

A doctor in each cell

Fig. 1 Medicine in 2050: “Doctor in a Cell”
Autonomous DNA Computing
Why Compute with DNA?

• Non–goals
  o Not to solve NP–complete problems.
  o Not to replace electronics.
  o Not necessarily using genes or producing proteins.

• For general ‘molecular programming’
  o To precisely control the organization and dynamics of matter and information at the molecular level.
  o To interact algorithmically with biological entities.
  o The use of DNA is “accidental”: no genes involved.
  o In fact, no material of biological origin.
Domains

• Subsequences on a DNA strand are called domains. *Provided* they are “independent” of each other.

• I.e., differently named domains must not hybridize:
  o With each other
  o With each other’s complement
  o With subsequences of each other
  o With concatenations of other domains (or their complements)
  o Etc.

• Choosing domains (subsequences) that are suitably independent is a tricky issue that is still somewhat of an open problem (with a vast literature). But it can work in practice.
Short Domains

Reversible Hybridization
Long Domains

Irreversible Hybridization
Strand Displacement

“Toehold Mediated”
Strand Displacement

Toehold Binding
Strand Displacement

Branch Migration
Strand Displacement

Displacement
Strand Displacement

Irreversible release
Bad Match

t x z

t x y
Bad Match
Bad Match
Bad Match

Cannot proceed
Hence will undo
Two-Domain Architecture

• Signals: 1 toehold + 1 recognition region

• Gates: “top–nicked double strands” (or equivalently double strands with open toeholds)

Garbage collection “built into” the gates

Two-Domain DNA Strand Displacement

Luca Cardelli

Transducer $x \rightarrow y$
Transducer $x \rightarrow y$

$ta$ is a *private* signal (a different ‘a’ for each $xy$ pair)
Transducer $x \rightarrow y$
Transducer $x \rightarrow y$

Active waste
Transducer $x \rightarrow y$
Transducer $x \rightarrow y$

So far, a $tx$ signal has produced an $at$ cosignal. But we want signals as output, not cosignals.
Transducer $x \rightarrow y$
Transducer $x \rightarrow y$
Transducer $x \rightarrow y$
Transducer $x \rightarrow y$

Here is our output ty signal.

But we are not done yet:
1) We need to make the output irreversible.
2) We need to remove the garbage.
We can use (2) to achieve (1).
Transducer $x \rightarrow y$
Transducer $x \rightarrow y$
Transducer $x \rightarrow y$
Transducer $x \rightarrow y$
Transducer \( x \rightarrow y \)
Transducer $x \rightarrow y$

Done.

N.B. the gate is consumed: it is the energy source.
Reaction Graph for $x \rightarrow y$
General $n \times m$ Join–Fork

- Easily generalized to 2+ inputs (with 1+ collectors).
- Easily generalized to 2+ outputs.

Figure 9: 3-Join $J_{wxyz}$ | $tw$ | $tx$ | $ty$ → $tz$: initial state plus inputs $tw$, $tx$, $ty$. 
Strand Algebra

• An intermediate language for molecular computing

  o Signals: \( x \)
  o Gates: \([x_1,\ldots,x_n].[y_1,\ldots,y_m]\)
  o Parallel composition: | 
  o Populations: (...)*

\[
\begin{align*}
  x_1 & \mid \ldots \mid x_n \mid [x_1,\ldots,x_n].[y_1,\ldots,y_m] \rightarrow y_1 \mid \ldots \mid y_m
\end{align*}
\]
Petri Net Transitions

• Computing power equivalent to Petri Nets
  o Not Turing complete, but as good as chemistry itself.
  o The correspondence is not completely trivial: gates are consumed by activation, hence a persistent Petri net transition requires a stable population of gates.

• Hence, many other mechanisms are expressible
  o E.g. Boolean networks
Compilation Issues

- Intermediate Language
- Structural Language
- Gate Design
- Circuit Design

Intermediate Language #2

4-domain Signals
3-domain Signals
2-domain Signals

Boolean Networks
Petri Nets

Strand Algebra
Conclusions
Summary

• Molecular Structures
  o Hard to build… but they can build themselves!

• Molecular Languages
  o Concurrent, quantitative

• Molecular Compilation
  o Molecular architectures, verification, optimization

• Molecular Programming
  o In silico, in vitro, in vivo…
Acknowledgments

• Microsoft Research
  o Andrew Phillips

• Caltech
  o Winfree Lab

• U.Washington
  o Seelig Lab