Nano Tasks

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

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

- **Constructing**
  - By spontaneous self-assembly
  - Catalyzed by stimuli

- **Computing**
  - All that under ‘program control’
  - Analog: Signal Filtering, Amplification
  - Digital: Logical gates

- **Nucleic Acids (DNA/RNA)**
  - Probably the only materials that can perform all these functions.
  - Technology relatively well developed.
  - Can interface to biological entities.
Compositionality

- Sensors and Actuators at the 'edge' of the system
  - They can use disparate kinds of inputs (sensors) and outputs (actuators)

- The 'kernel' of the system computes
  - Must use uniform inputs and outputs

- Compositionality in the kernel
  - Supporting 'arbitrary' computing complexity
  - The output of each computing component must be the same kind of 'signal' as the input
    - If the inputs are voltages, the outputs must be voltages
    - If the inputs are proteins, the outputs must be proteins
    - If the outputs are photons the inputs must be photons
    - If the inputs are DNA, the outputs must be DNA

- What should our nano-signals be?
What does DNA Compute?

● **Electronics has electrons**
  - All electrons are the same
  - All you can do is see if you have few (‘False’) or lots (‘True’) of electrons
  - Hence Boolean logic is at the basis of digital circuit design
  - Symbolic and numeric computation has to be encoded above that
  - But mostly we want to compute with symbols and numbers, not with Booleans

● **DNA computing has symbols (DNA words)**
  - DNA words are not all the same
  - Symbolic computation can be done *directly*
  - We can also directly use molecular concurrency

● **Process Algebra as the ‘Boolean Algebra’ of DNA Computing**
  - What are the ‘gates’ of symbolic concurrent computation?
  - That’s what Process Algebra is about
  - (Process Algebra comes from the theory of concurrent systems)
Implementing "Arbitrary" Computing Functions
DNA Compilation

Separating Circuit Design from Gate Design

High level languages (TBD)

Low level languages

Circuit Design Space

DNA
DNA Compilation

Separating Circuit Design from Gate Design

High level languages (TBD)

Low level languages

Discrete Chemistry

Interacting Automata

Boolean Networks

Finite State Automata

Petri Nets

Circuit Design

(e.g. half-adders from Boolean gates)

Gate Design Space

DNA
DNA Compilation

Separating Circuit Design from Gate Design

DNA

- Strand Displacement
- Strand Algebra
- Interacting Automata
- Boolean Networks
- Finite State Automata
- Petri Nets
- Discrete Chemistry
- High level languages (TBD)
- Low level languages

Circuit Design
(e.g. half-adders from Boolean gates)

Gate Design
(e.g. Boolean gates from transistors)

Seesaw Gates


Verification of DNA gate implementation

Higher-level languages
DNA Compilation

Separating Circuit Design from Gate Design

- Higher-level languages
  - Discrete Chemistry
  - Interacting Automata
  - Boolean Networks
  - Finite State Automata
  - Petri Nets

- Low level languages
  - Circuit Design
    - (e.g. half-adders from Boolean gates)
  - Gate Design
    - (e.g. Boolean gates from transistors)

- Other DNA Mechanisms
  - Strand Displacement
  - Strand Algebra

- DNA
DNA Compilation

Separating Circuit Design from Gate Design

Rest of the talk: bottom up
Toehold Mediated Strand Displacement
Strand Displacement Reaction

Irreversible
Transducer Gate

$G_b, G_t$ (gate backbone and trigger) form the transducer.

Any history segment that is not determined by the gate structure is said to be ‘generic’ (can be anything).

Any gate segment that is not a non-history segment of an input or output signal is taken to be ‘fresh’ (globally unique for the gate), to avoid possible interferences.
Strand Displacement
Intermediate Language
Syntax

A. Syntax of DNA molecules $D$

Upper strand with sequence complementary to $S$

```
S
```<s>

Molecule with segments $G_1, ..., G_K$

```
G_1: G_2: ...: G_K
```

Parallel molecules $D_1, ..., D_K$

```
D_1 | D_2 | ... | D_K
```

Molecules $D$ with private domains $N_1, ..., N_K$

```
(N_1, ..., N_K) \_ D
new (N_1, ..., N_K) \_ D
```
Strand Displacement Simulation Tool

1 Transducer gate $x.y$ (3 initial species)

directive sample 30.0 1000
new xt@1.0,1.0
new yt@1.0,1.0
{ 1000 * <xt x^:xb> | 1000 * <yt a> | 1000 * <yt y> | 1000 * <xt a> | 1000 * <yt a> }

x | x.y → y

x | x.y → y

Luca Cardelli 2009-10-12
Strand Algebra
Strand Algebra

\[ P ::= x : [x_1, \ldots, x_n] \cdot [y_1, \ldots, y_m] : 0 : P \mid P : P^* \quad n \geq 1, \quad m \geq 0 \]

- \( x \) is a \textit{signal}
- \([x_1, \ldots, x_n] \cdot [y_1, \ldots, y_m]\) is a \textit{gate}
- 0 is an \textit{inert solution}
- \( P \mid P \) is \textit{parallel composition} of signals and gates
- \( P^* \) is a \textit{population (multiset)} of signals and gates

\textbf{Reaction Rule}

\[ x_1 \mid \ldots \mid x_n \mid [x_1, \ldots, x_n] \cdot [y_1, \ldots, y_m] \rightarrow y_1 \mid \ldots \mid y_m \]

Equivalent to (stochastic) place-transition Petri Nets.
Compiling Strand Algebra to DNA

\[ P ::= x : [x_1, \ldots, x_n].[y_1, \ldots, y_m] : 0 : P \mid P : P^* \quad n \geq 1, \ m \geq 0 \]

- \( \text{compile}(x) = \)

- \( \text{compile}([x_1, \ldots, x_n].[y_1, \ldots, y_m]) = \)

- \( \text{compile}(0) = \) empty solution

- \( \text{compile}(P \mid P') = \text{mix} (\text{compile}(P), \text{compile}(P')) \)

- \( \text{compile}(P^*) = \text{population} (\text{compile}(P)) \)
Computational Abstractions
This encoding is *compositional*, and can encode any Boolean network:
- multi-stage networks can be assembled (*combinatorial logic*)
- network loops are allowed (*sequential logic*)
Petri Nets to Strand Algebra

Transitions as Gates
Place markings as Signals

\[
([p_1,p_2].[p_3,p_4])^* | p_1 | p_1 | p_4
\]
Experiments
Sequence Design

**Input**

NUPACK nucleic acid package

**Output**

NUPACK nucleic acid package

Design

*Sequence designs*

- Average percentage of correct nucleotides: 99.1%
- Average number of incorrect nucleotides: 0.475
- GG content: 74.5%
- Sequence: GGGUCG+UCUA+GCC+GGC+GCG+UGG+GUCU+GGCGCGC+GGCGCGC
Conclusions
Conclusion

- **Nucleic Acids**
  - Programmable matter

- **DNA Strand Displacement**
  - A computational mechanism at the molecular level

- **DNA Compilation**
  - High-level languages (Boolean Networks, Petri Nets, Interacting Automata)
  - Intermediate languages (Strand Algebra, Strand Displacement Language).
  - Sequence generation.

- **Tools**
  - Thermodynamic analysis.
  - Simulation.
  - Verification/Optimization (not yet).