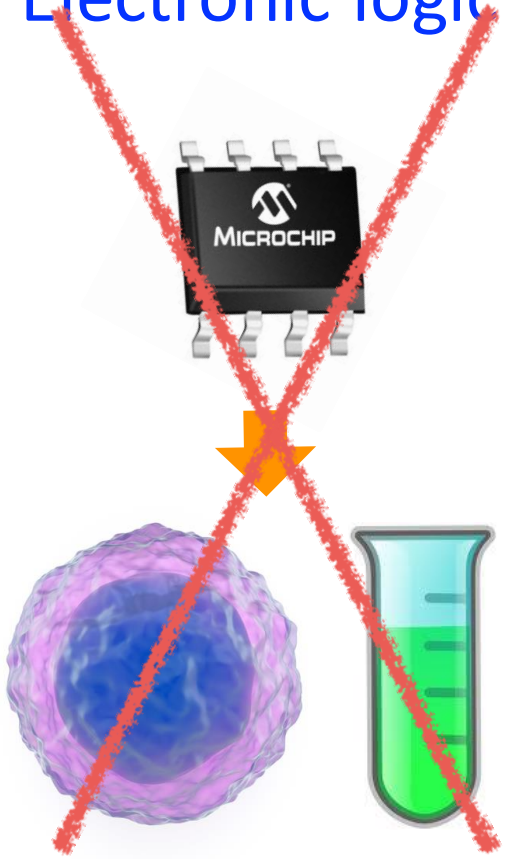
CRN++
Molecular Programming Language

Marko Vasic, David Soloveichik and Sarfraz Khurshid

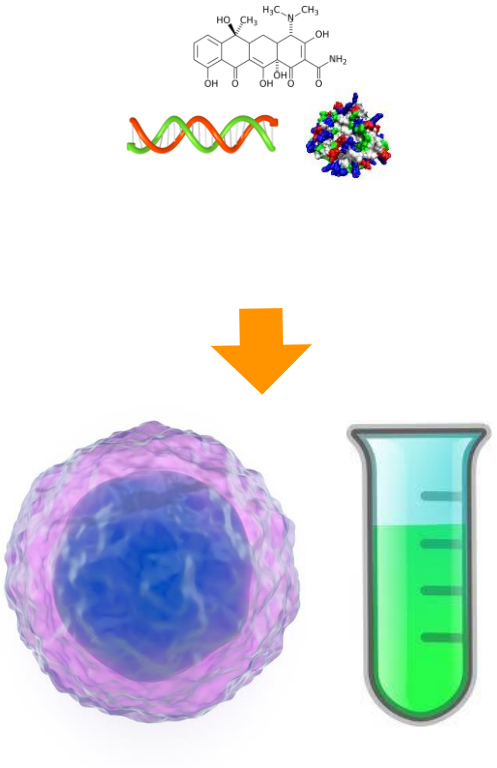
Arm Research Summit, Austin, USA
September 16, 2019

Goal: Embed logic into living and synthetic biochemical systems

Electronic logic



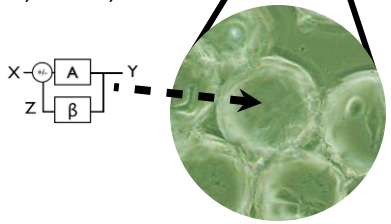
Chemical logic



Applications of Smart Molecular Systems

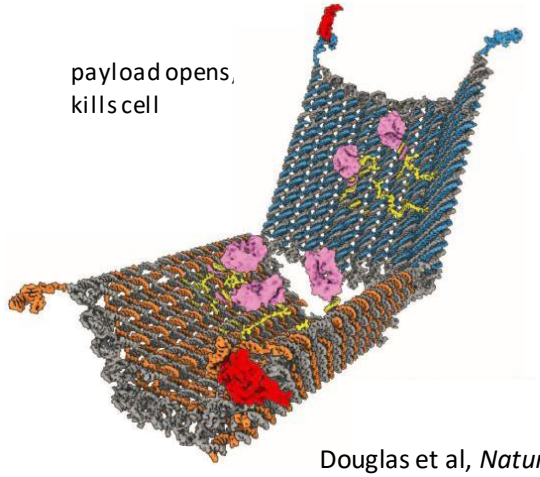
Cells as factories (bioreactors)

Khalil, Collins, *Nat Rev Gen* 2010



"chemical controller" within cells would dramatically increase production

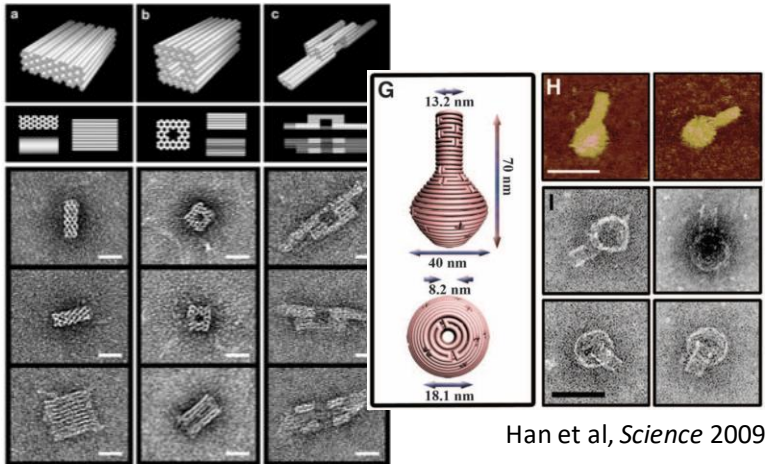
Smart drugs for health



Douglas et al, *Nature* 2012

target cells for killing by detecting particular combination of indicators

Manufacturing complex nanostructures and materials by self-assembly



Han et al, *Science* 2009

Douglas et al, *Nature* 2009

controlling assembly with local computational rules

“Hello World” of molecular programming

$$A \rightarrow Z1 + Y$$
$$B \rightarrow Z2 + Y$$
$$Z1 + Z2 \rightarrow K$$
$$Y + K \rightarrow W$$

- Computes max function: Y = max(A, B)
- Abstract: doesn’t say what the chemical species (hardware) are
- Hard to reason, can we make it easier?

How is computation in chemistry defined?

CRN:

$$A + B + F \rightarrow A + B + C$$

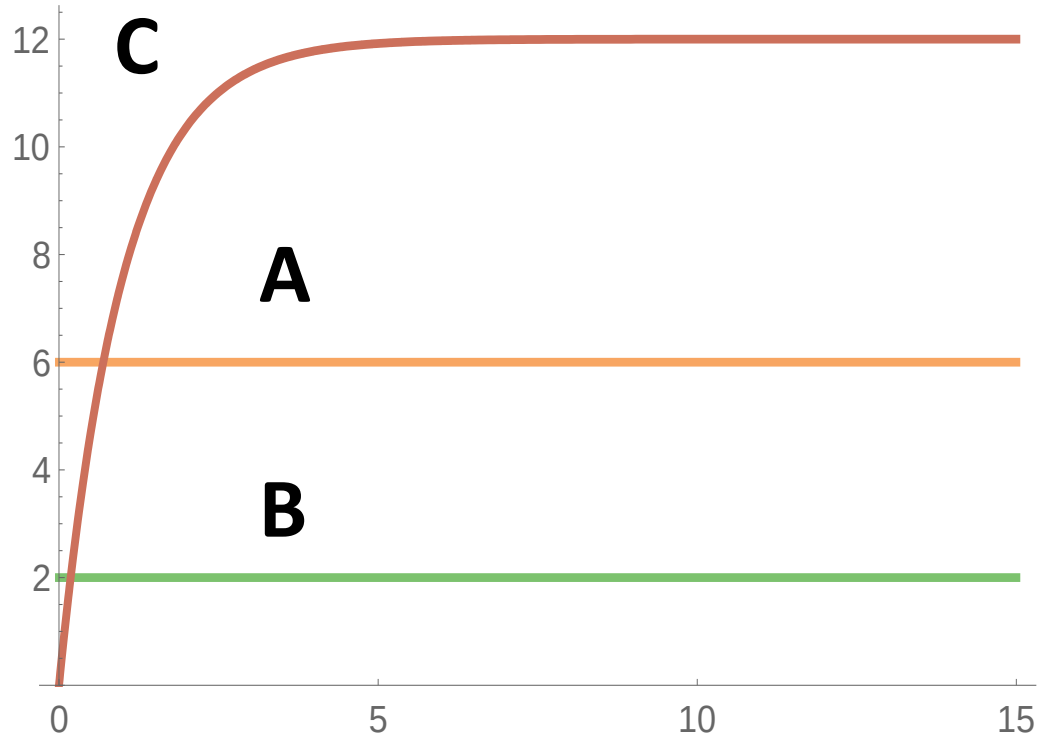
$$C + F \rightarrow W$$

ODE: $\frac{dC}{dt} = A \cdot B \cdot F - C \cdot F$

Equilibrium: $C = A \cdot B$

Multiplication!

Species concentrations



Challenges of molecular computation

- Analog computation: all reactions are always occurring.
  - How to detect and extract operations that are human understandable?
  - How do we model sequential execution that we are used to?
  - How do we model conditional execution (branching)?
- Convergence to solution: only at equilibrium correct solution is reached.
  - We need to compute in finite time $\rightarrow$ error is inevitable.
  - How do we control error?

CRN++ modules to implement basic commands

- CRN++ implements basic modules that are intuitive and understandable.

Load:

$$A \rightarrow A + B$$
$$B \rightarrow W$$

Add:

$$A \rightarrow A + C$$
$$B \rightarrow B + C$$
$$C \rightarrow W$$

Sub:

$$A \rightarrow A + C$$
$$B \rightarrow B + H$$
$$C \rightarrow W$$
$$C + H \rightarrow W$$

Mul:

$$A + B \rightarrow A + B + C$$
$$C \rightarrow W$$

Div:

$$A \rightarrow A + C$$
$$B + C \rightarrow B$$

…others

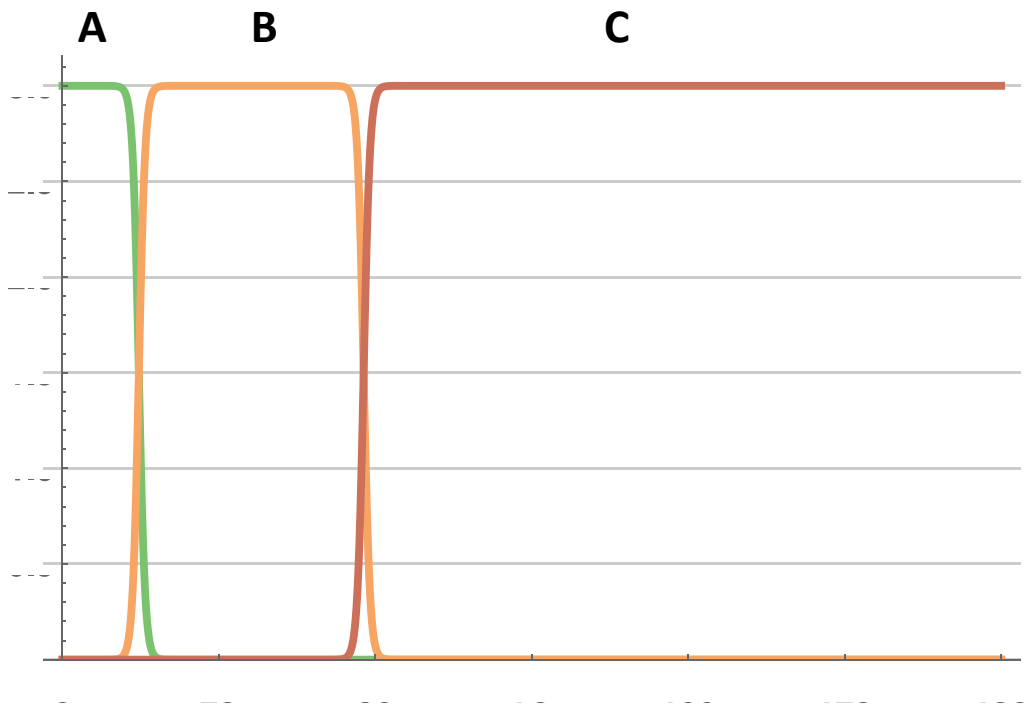
Buisman et al. 2009

- Modules must respect strict properties to enable compositions.
- CRN++ is modular and allows for including additional modules.
  - Modules need to follow the principles of composability and convergence speed

How do we enable sequential execution?

- CRN++ provides step statement for ordering execution
- It relies on chemical oscillator underneath (analog to clock signal)

```
CRN = {
    conc[a, 3],
    step[{
        rxn[a, b, 1]
    }],
    step[{
        rxn[b, c, 1]
    }]
};
```



How do we model conditional execution?

- CRN++ supports cmp instruction that enables branching.

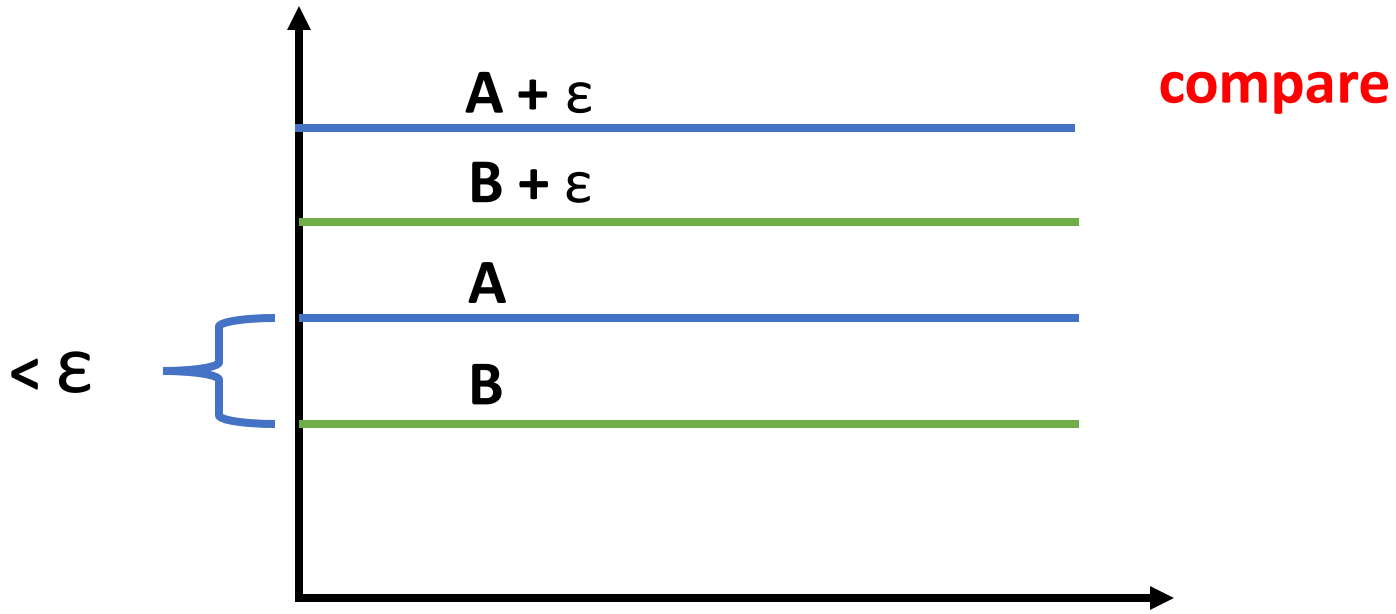
```
CRN = {
   step[{
      ……..
      cmp[a, b]
   }],
   step[{
      ifGT[{ ... }],
      ifLT[{ ... }]
   }]
};
```

Comparison

Only if a > b

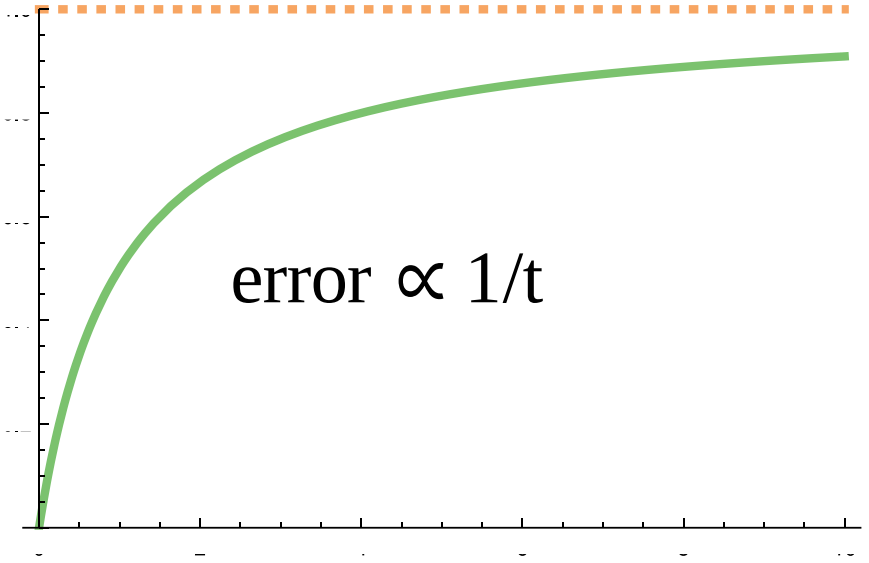
Epsilon Equality

- Signal analog (ε equality, ε = 0.5)
- GT, GE, EQ, LE, LT blocks supported

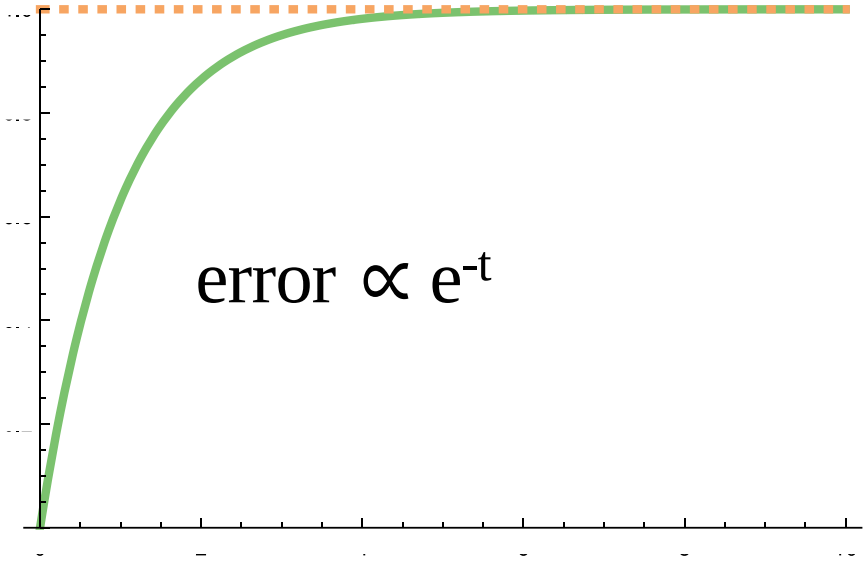


Reducing error through faster convergence

Linear convergence



Exponential convergence



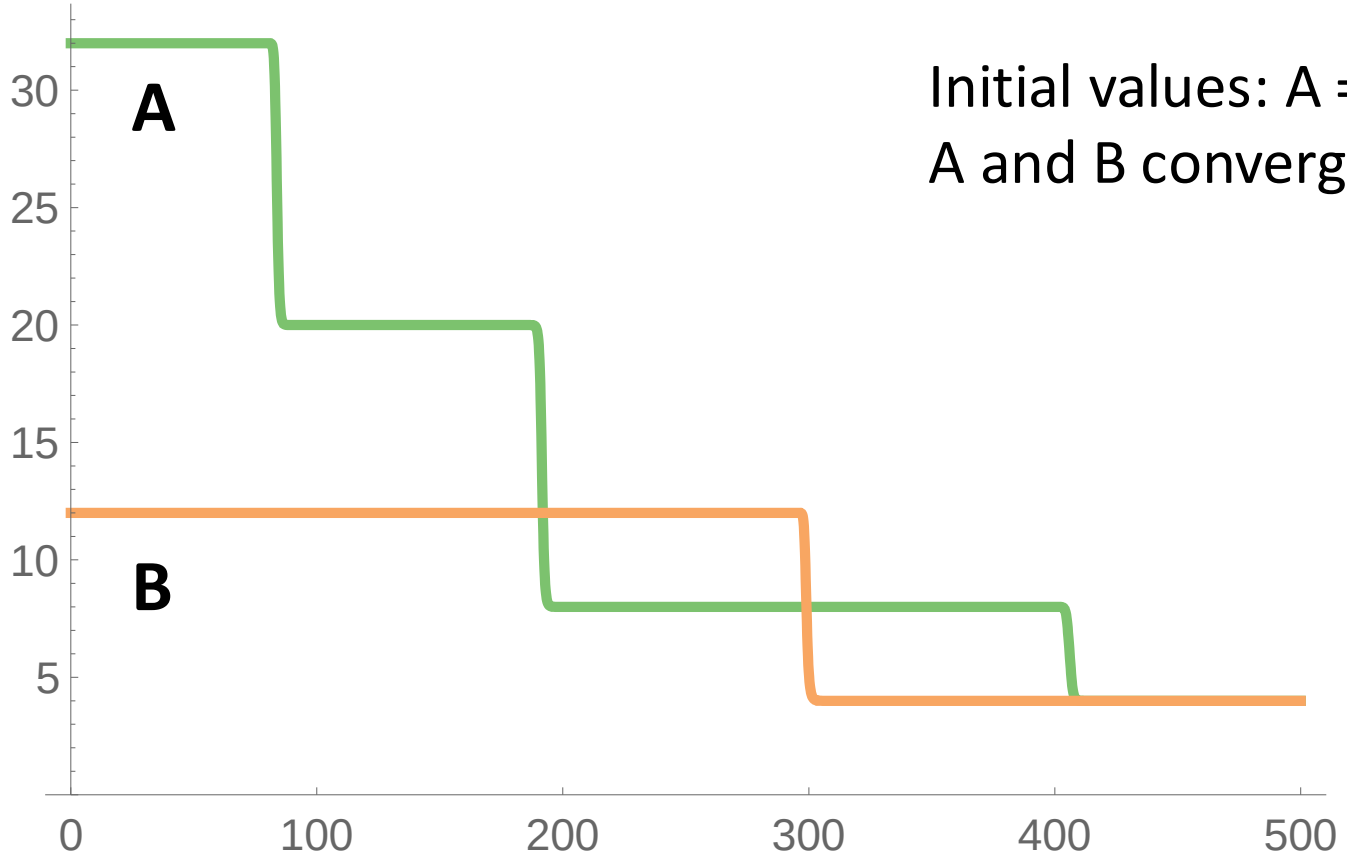
Putting It All Together: GCD

- Compute Greatest Common Divisor of two numbers

```
1:  procedure gcd(a, b)
2:    while a ≠ b do
3:      if a > b then
4:        a ← a − b
5:      else
6:        b ← b − a
7:      end if
8:    end while
9:    return a
10: end procedure
```

```
GCD= {
    step[{
        ld[a, atmp],
        ld[b, btmp],
        cmp[a, b]
    }],
    step[{
        ifGT[{ sub[atmp, btmp, a] }],
        ifLT[{ sub[btmp, atmp, b] }]
    }]
};
```

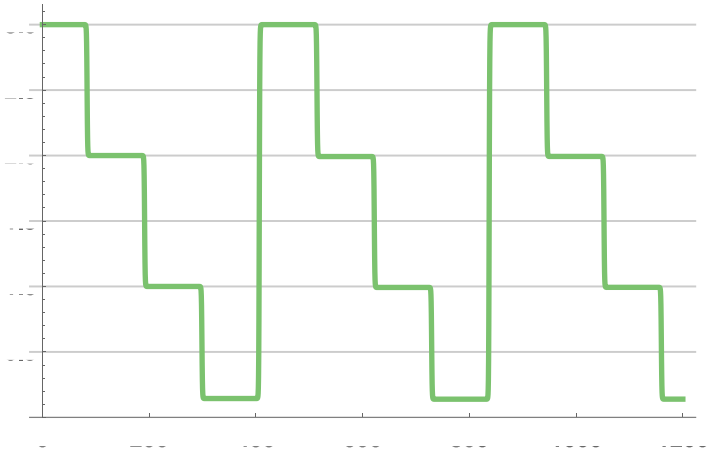
GCD Simulation Results



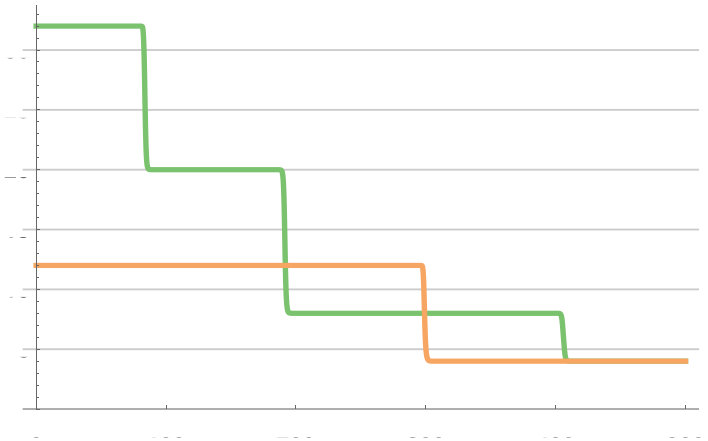
Initial values: A = 32, B = 12
A and B converge to GCD(32, 12) = 4.

CRN++ Programs

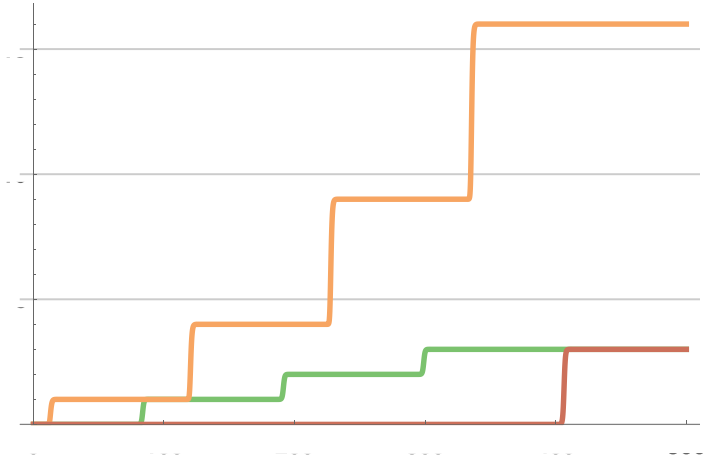
Counter



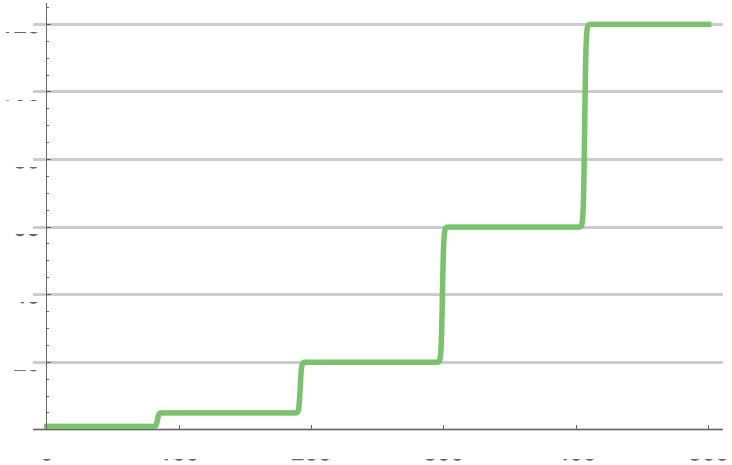
GCD



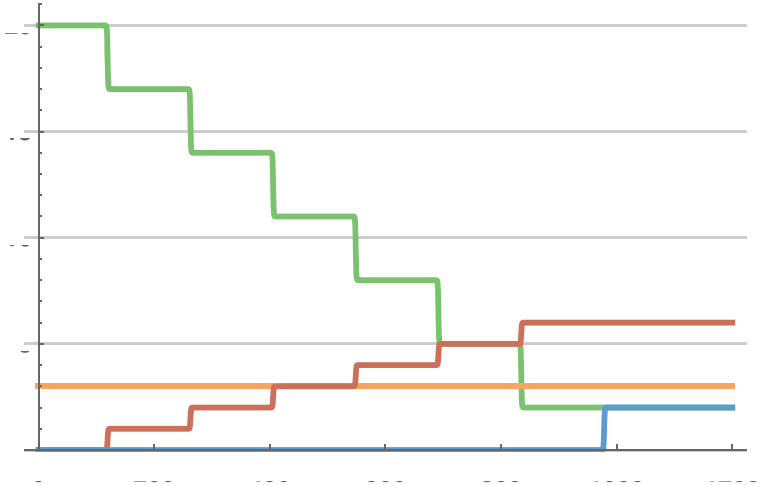
Integer square root



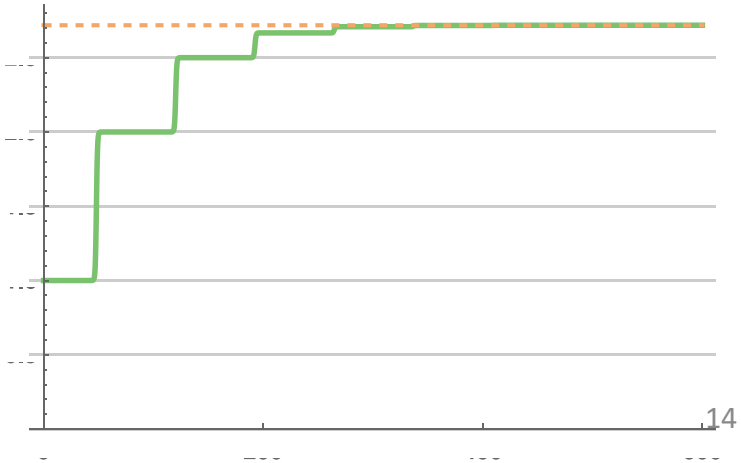
Factorial



Integer Division



Euler Number Approximation



Error Analysis

- Error inevitable
- Feedback during simulation (expected vs actual value)

```
GCD= {
   conc[a, 32], conc[b, 12],
   step[{
      ld[a, atmp],
      ld[b, btmp],
      cmp[a, b]
   }],
   step[{
      ifGT[{ sub[atmp, btmp, a] }],
      ifLT[{ sub[btmp, atmp, b] }]
   }]
};
```

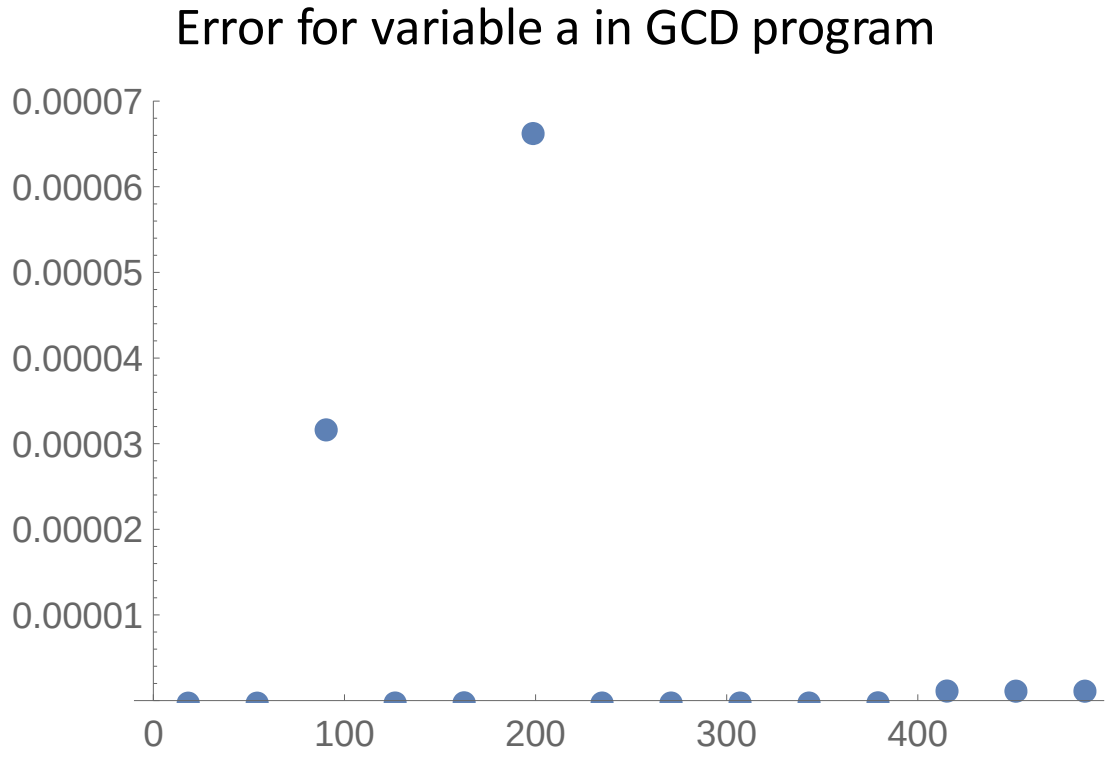
Expected: **a=32, b=12, atmp=0, btmp=0**
Actual: a=32, b=12, atmp=0, btmp=0

Expected: a=32, b=12, **atmp=32**, **btmp=12**
Actual: a=32.05, b=12.01, atmp=31.99, btmp=0.001

Expected: **a=20**, b=12, atmp=32, btmp=12
Actual: a=20.011, b=12.007, atmp=31.98, btmp=0.02

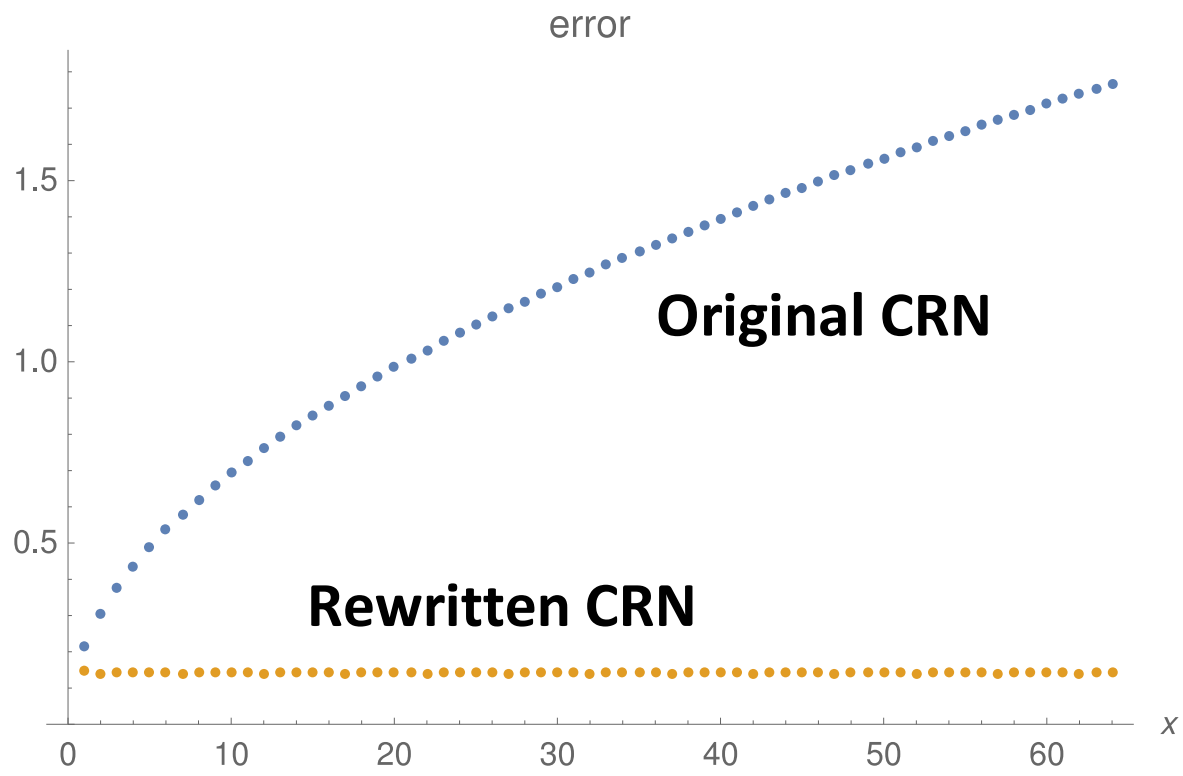
Error Analysis

- Plot showing difference between expected and actual value
- Insight into sources of error



Program Refactoring

- Rewriting CRN++ programs can lead to solutions with smaller error
- Rewriting subtraction.
- Error now constant!



Conclusions and Future Work

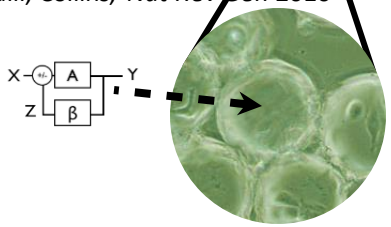
- Designed an intuitive language for programming chemistry
- Future work:
  - How do we understand algorithms that happen in living cells and map them back to programming languages such as CRN++
  - How do we translate CRN++ languages to chemical reactions biologists are familiar with, such as transcriptional networks, protein-protein networks, etc.
- CRN++ open sourced: https://github.com/marko-vasic/crnPlusPlus
- Contact: vasic@utexas.edu

Questions

Applications of Smart Molecular Systems

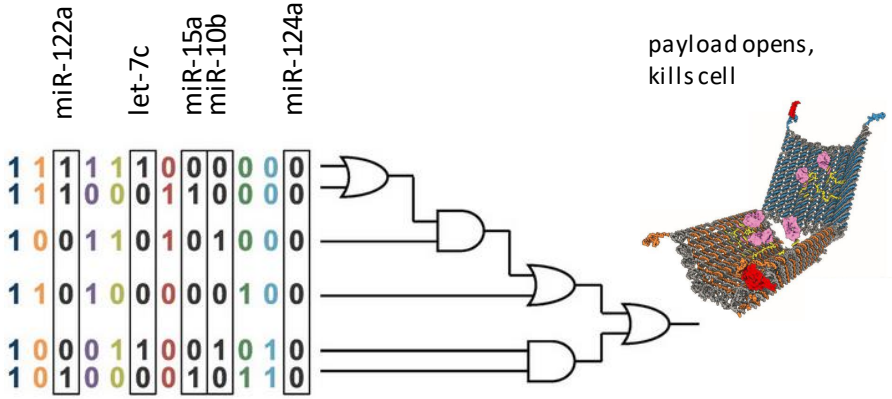
Cells as factories (bioreactors)

Khalil, Collins, *Nat Rev Gen* 2010



"chemical controller" within cells would dramatically increase production

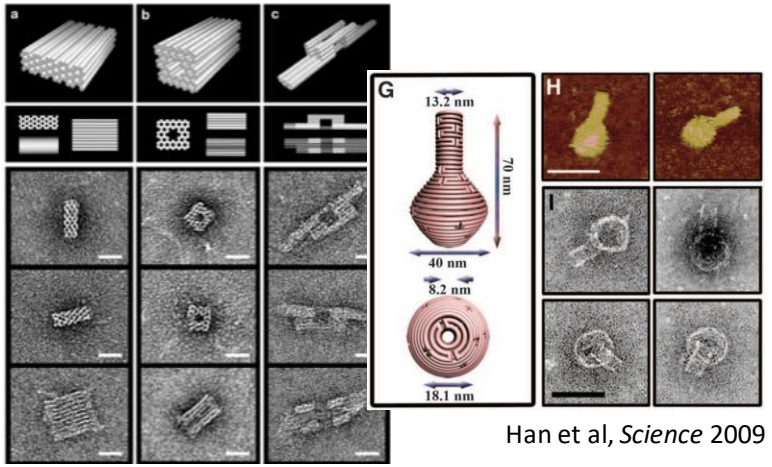
Smart drugs for health



Seelig et al, *Science* 2006 + Douglas et al, *Nature* 2012

target cells for killing by detecting particular combination of indicators

Manufacturing complex nanostructures and materials by self-assembly



Douglas et al, *Nature* 2009

Han et al, *Science* 2009

controlling assembly with local computational rules