Molecular Programming: How Software Engineers Will Lead in the Age of Nanotechnology

Robyn Lutz, Titus Klinge, Jack Lutz, Jim Lathrop

Department of Computer Science
Iowa State University
Ames, IA USA
LAMP: LAb for Molecular Programming
http://web.cs.iastate.edu/~lamp/

ASE 2015 Tutorial
Lincoln, NE USA
November 10, 2015

Outline: Introduction to Molecular Programming

- **Overview: what it is, recent advances, potential applications**
- Introduction to its key modeling paradigms
- The challenge of making molecular programmed systems robust
- Why software engineering?
- Example: software engineering for DNA nanopliers

Molecular programming

Molecular programming creates and programs a wide variety of synthetic nanosystems that autonomously assemble themselves from molecular components.

Programming is done by carefully choosing the molecular strands & their concentrations such that they will achieve the desired shape, structure, function or dynamic behavior.

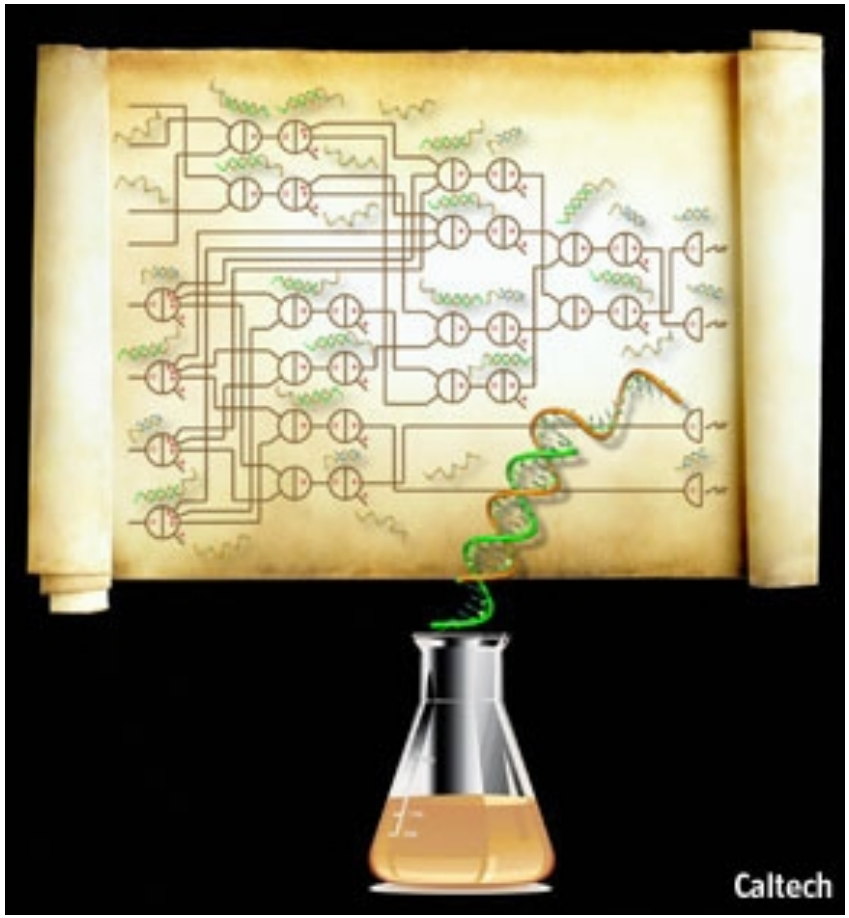
We are especially interested in the **requirements and design** for these programmed systems.

What does it mean to specify and validate requirements on programmable DNA self-assemblies?

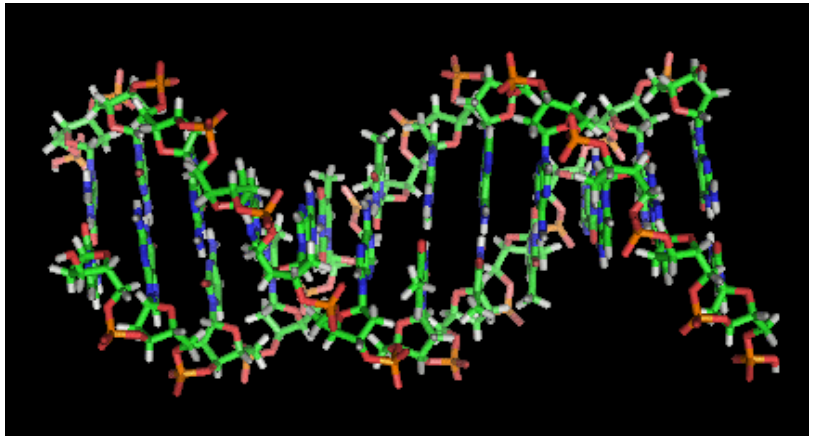
How can we verify formally that a self-assembly satisfies its requirements?

Molecular programming

"*Computing with soup*
Molecular computing: DNA is sometimes called the software of life. Now it is being used to build computers that can run inside cells." - The Economist, 3/3/2012



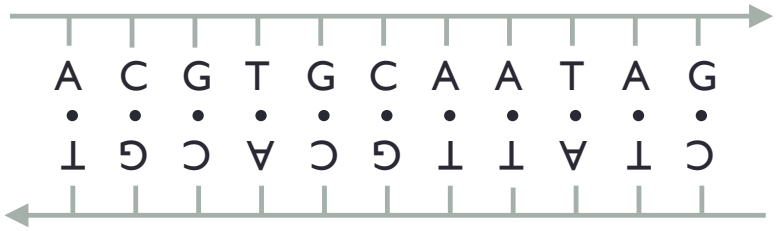
Background



~2 nm (2 billionth of a meter)

DNA double helix

DNA is programmable,
cheap, easy to synthesize
& *bio-compatible*



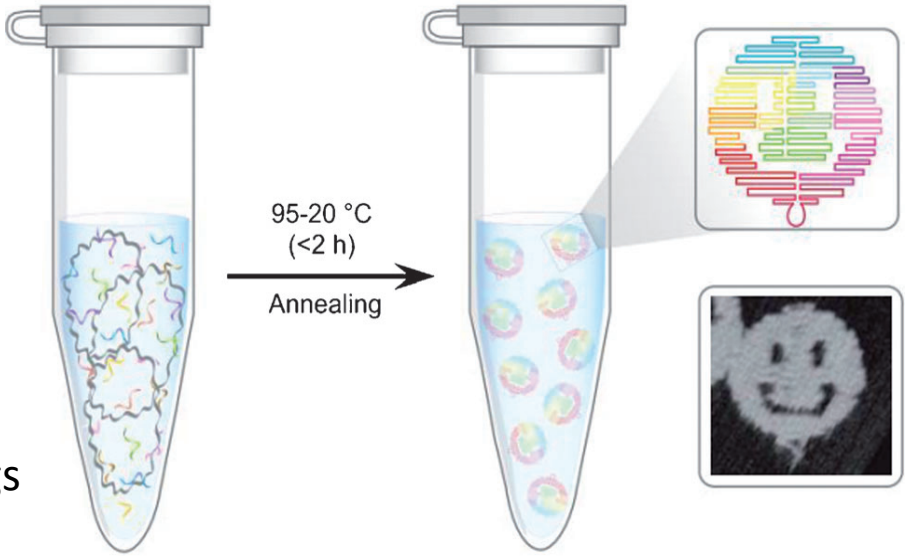
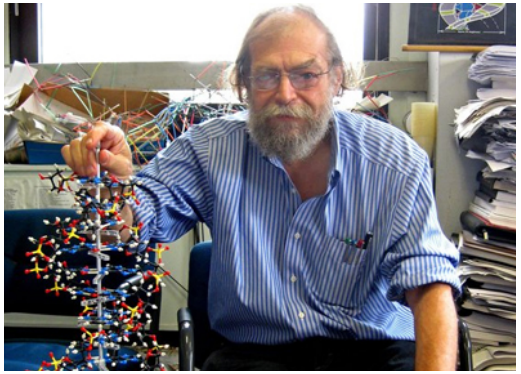


Figure taken from Rajendran, A., Endo, M. & Sugiyama, H. *Angew Chem Int Ed* (2011) DOI: 10.1002/anie.201102113.

On the order of 6 x 10^{10} devices simultaneously deployed.

(For comparison, 5 x 10^{10} internet of things connections est. by 2020 [Cisco].)

Background

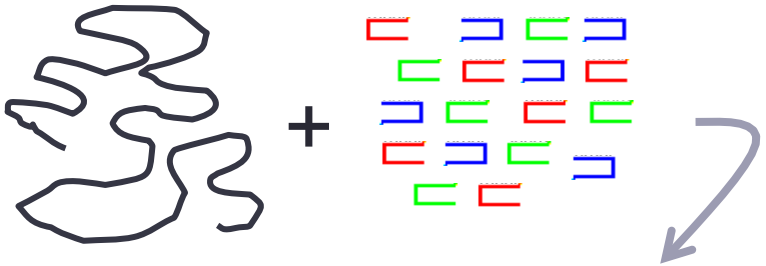


Seeman used the information-processing capabilities of DNA to program short strands of DNA to assemble themselves into specified structures and devices. (1986)

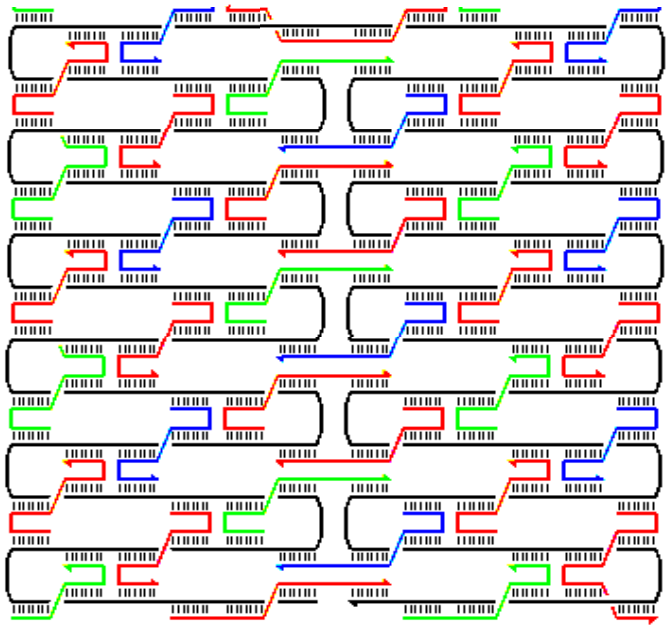
Winfree showed that self-assembly is Turing universal, i.e., that any computation can be simulated by self-assembly (1998). This implies that self- assembly can be algorithmically directed, and that extremely complex shapes and behaviors can be realized by self- assembly.

Rothemund introduced DNA origami, a very general method for using short DNA "staples" to cause a long, single-strand DNA "scaffold" (usually the genome of one specific virus) to fold itself into a desired shape.

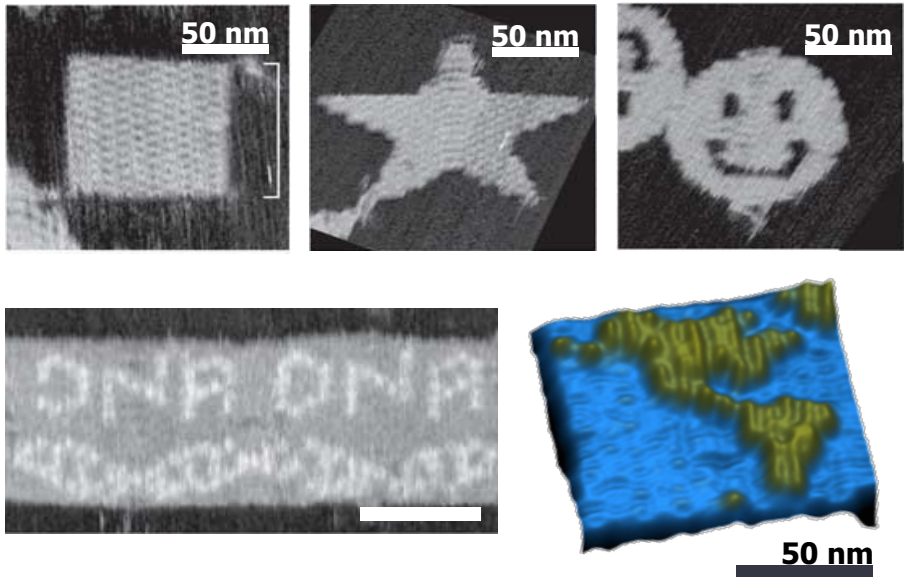
DNA origami



Many short "staple" strands fold a long "scaffold" strand into a desired shape.
(Different staple strands on different positions – regardless of color)



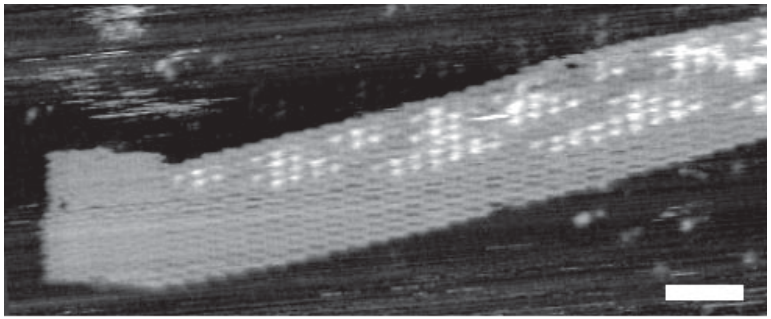
→ Arbitrary shapes and patterns

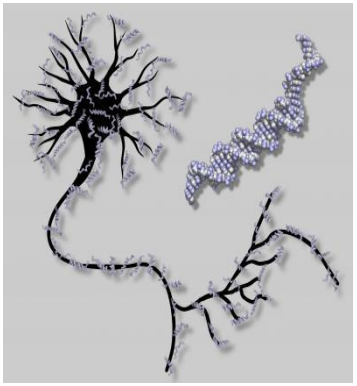


Slide from S. Woo

Rothemund, P.W. K. Folding DNA to create nanoscale shapes and patterns. *Nature* **440**, 297-302 (2006).

Computing with DNA





Qian, L., Winfree, E. & Bruck, J. *Nature* **475**, 368-372 (201

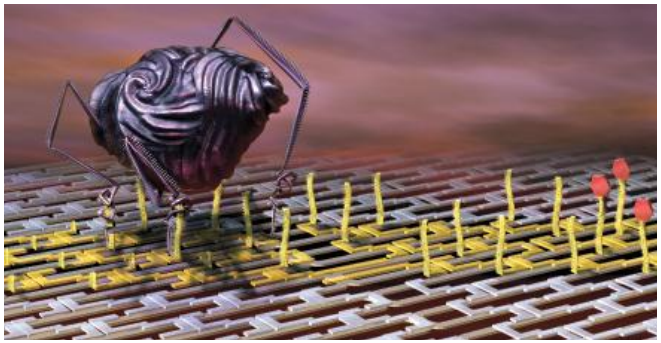
Neural network simulation

Binary Counter

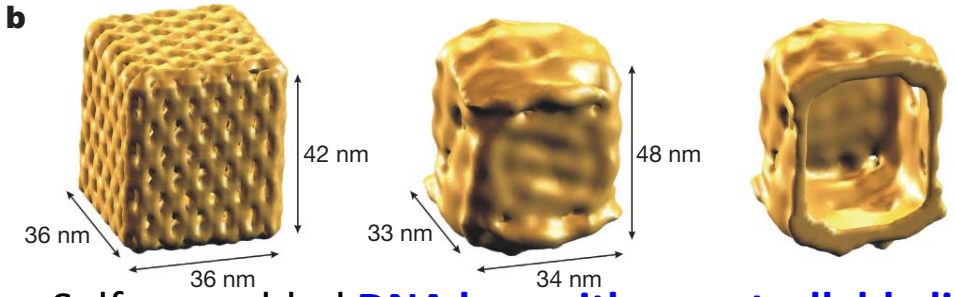
Barish, Schulman, Rothemund, and Winfree, PNAS 2009

Molecular computer: 130 DNA-strand calculates the floor of the **Square Root** of four-bit binary numbers (Qian and Winfree, 2011) Science Photo Library/Alamy.

Simulate digital signals with **DNA logic circuits** (AND, OR, NOT): Low (High)concentration/no fluorescence; Universal "seesaw" gate implements AND, OR by varying concentration of threshold gate (Qian and Winfree, 2011)



Molecular robots guided by prescriptive landscapes, Lund, et al., 2010. Image: *Science News*, 2010.



Self-assembled **DNA box with a controllable lid**, E. S. Andersen, et al., 2009.

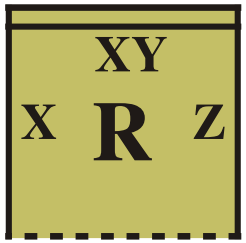
Many of the applications will be safety-critical

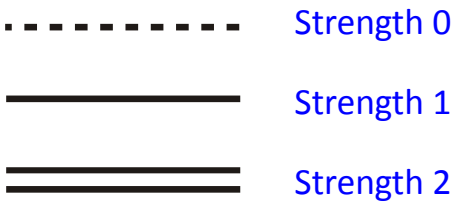
Outline: Introduction to Molecular Programming

- Overview: what it is, recent advances, potential applications
- **Introduction to its key modeling paradigms**
- The challenge of making molecular programmed systems robust
- Why software engineering?
- Example: software engineering for DNA nanopliers

DNA Tile Self Assembly: Winfree Ph.D. thesis, 1998

Extension of Wang tiling, 1961
Refined in Paul Rothemund's Ph.D. thesis, 2001



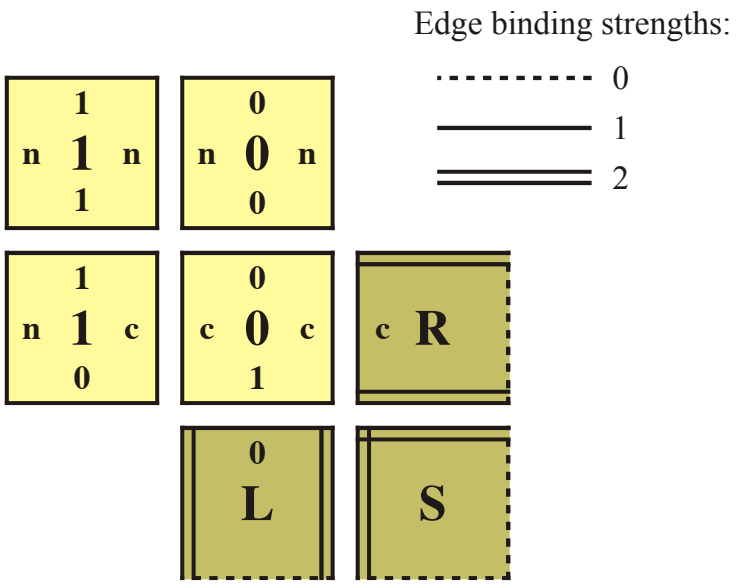


- Tile = unit square
- Each side has *glue* of certain *kind* and *strength* (0, 1, or 2).
- If tiles abut with matching kinds of glue, then they bind with this glue's strength.
- Tiles may have labels.
- Tiles cannot be rotated.

- Finitely many tile types
- Infinitely many of each type available
- Assembly starts from a *seed tile* (or *seed* assembly).
- A tile can attach to existing assembly if it binds with total strength at least 2 (the "temperature").

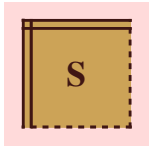
Tiling Slides from Jack

Tile assembly model example

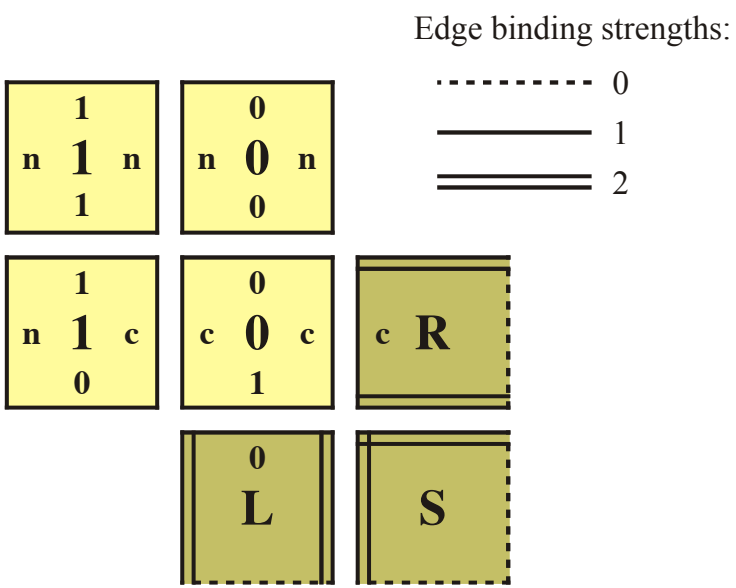


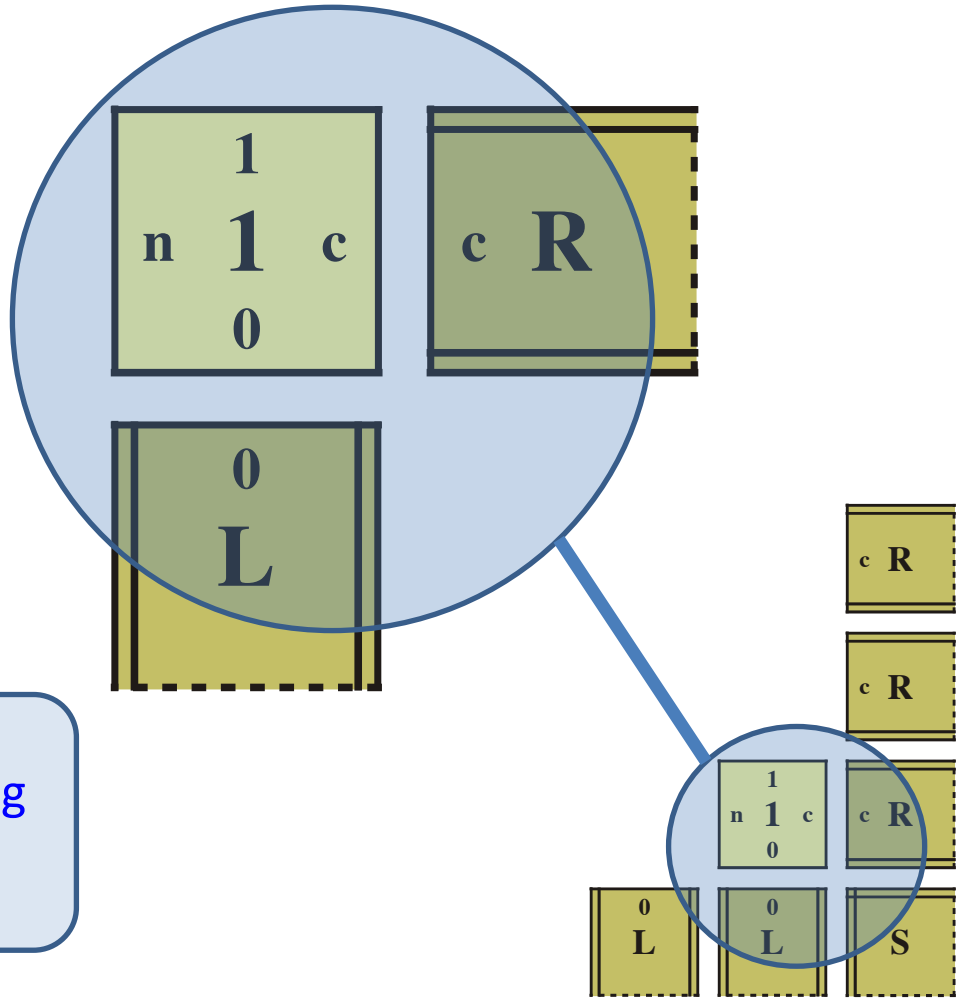
"library" of available tiles

seed tile



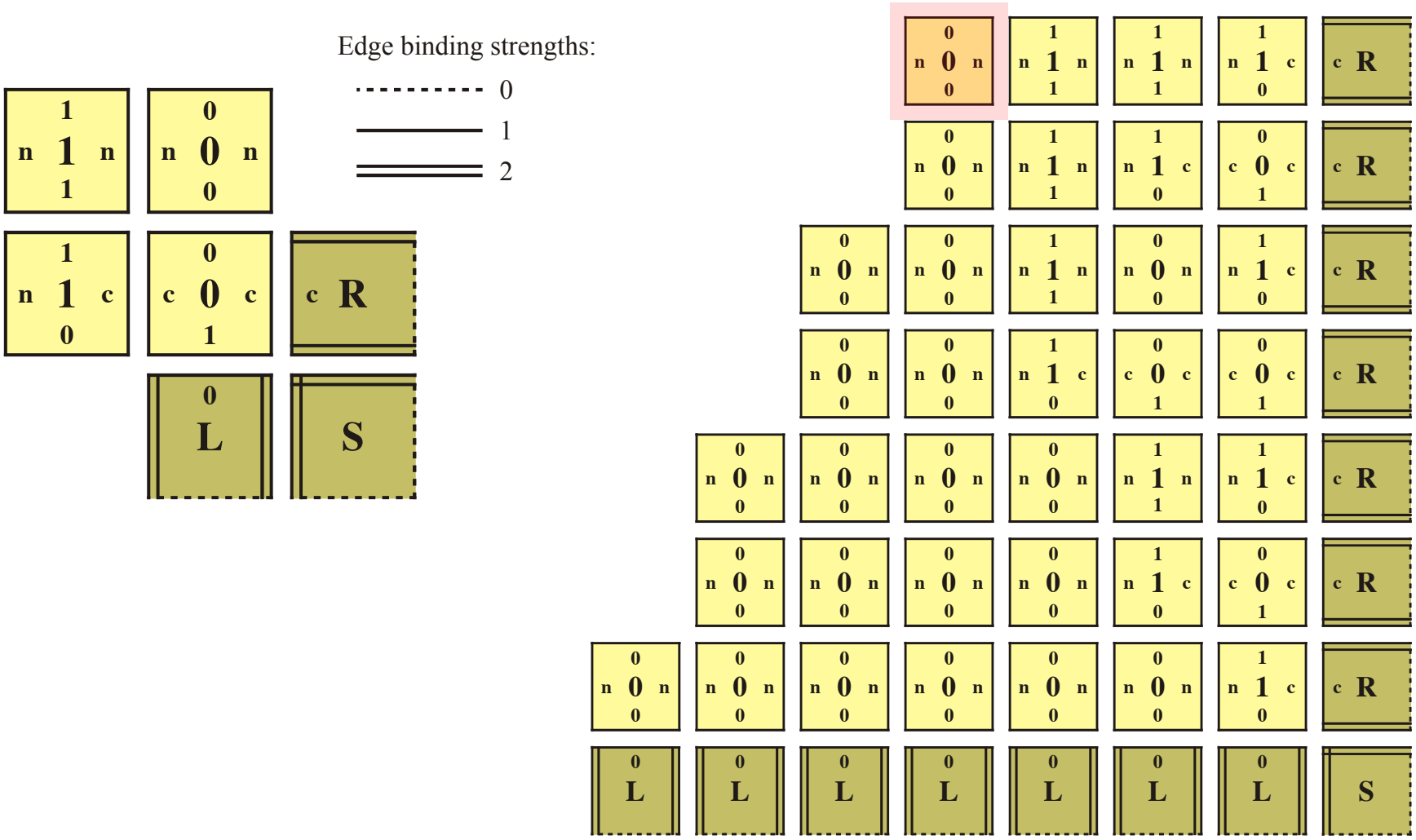
Tile assembly model example



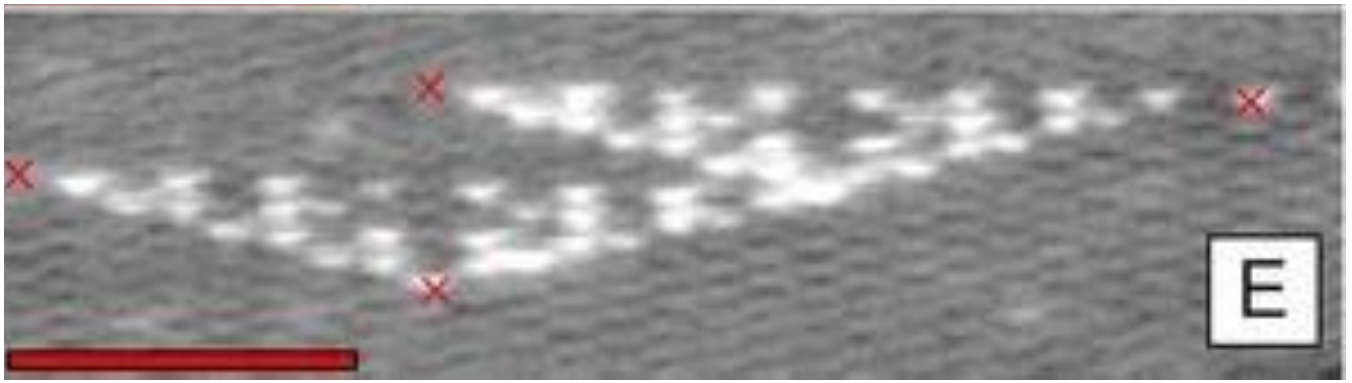


Cooperation is key to computing with tile-assembly model.

Tile assembly model example



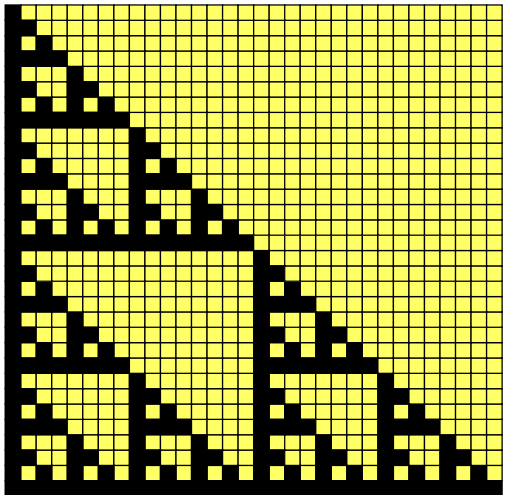
Computing with DNA



Rothemund, Papdakis, Winfree, Algorithmic self-assembly of DNA Sierpinski triangles, 2004

the Tile Assembly Model is computationally universal, i.e., can simulate any Turing machine (Winfree, ’98)

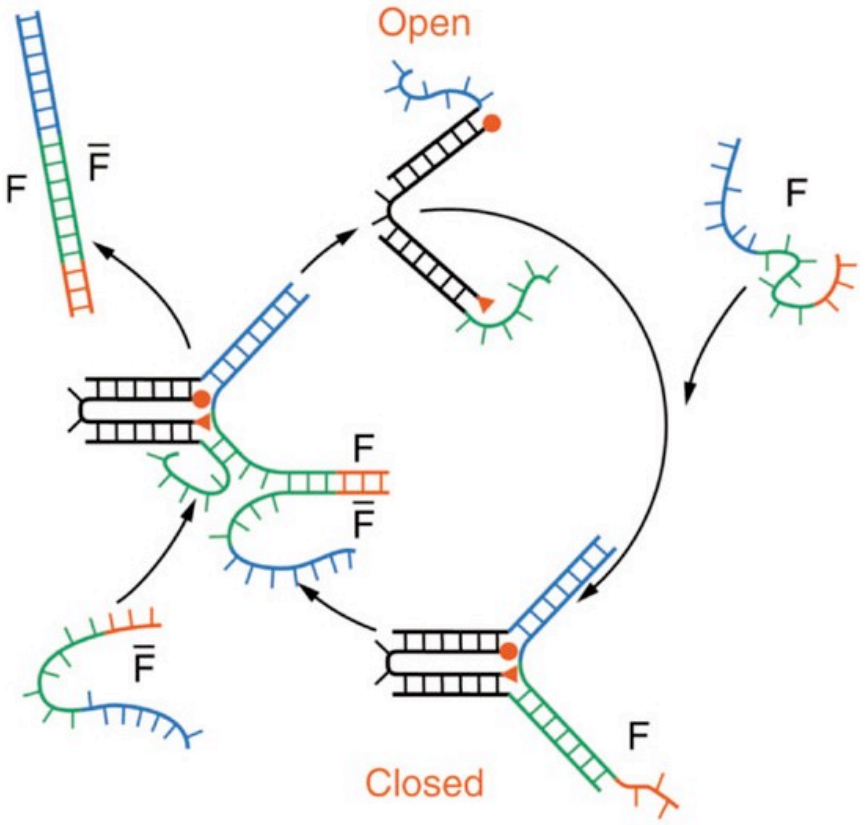
the Tile Assembly Model is intrinsically universal, i.e., can simulate the behavior of any other tile assembly system, including itself (Doty, Lutz, Patitz, Schweller, Summers, and Woods ’12)



Dynamic DNA Strand Displacement

Use toehold-mediated branch migration and strand displacement to create self-assembling devices with dynamic behaviors
[Yurke, 2000; Soloveichik, 2008; Zhang & Seeling, 2011; Qian & Winfree, 2011; Qian, Winfree & Bruck, 2011]

Dynamic strand displacement is Turing universal but only if exactly one stack molecule is in the solution
[Qian, Soloveichik & Winfree, 2011]



Yurke, B. et al. A DNA-fuelled molecular machine made of DNA. *Nature,* 2000.

Chemical Reaction Networks (CRNs)

- Serve as a programming language to specify the desired behavior for a computational subsystem implemented in DNA
- A set of reactions (rules) describe the system's probabilistic behavior
- Molecular species react with one another according to the programmed rules: Reactants -> Products
- Compiling CRNs into DNA strand-displacement reactions is automated, both in silico and in vitro [Chen et al., 2013].

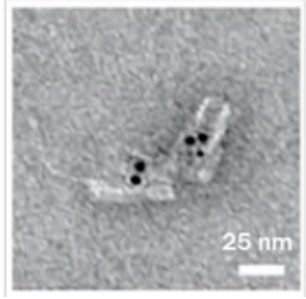
Outline: Introduction to Molecular Programming

- Overview: what it is, recent advances, potential applications
- Introduction to its key modeling paradigms
- **The challenge of making molecular programmed systems robust**
- Why software engineering?
- Example: software engineering for DNA nanopliers

Challenge: Safety-critical programs must be verifiably robust

DNA nanorobot to deliver targeted drug therapeutics:
http://wyss.harvard.edu/viewpage/326/

March 18, 2015: DNA Nanobots Set To Seek and Destroy Cancer Cells In Human Trial



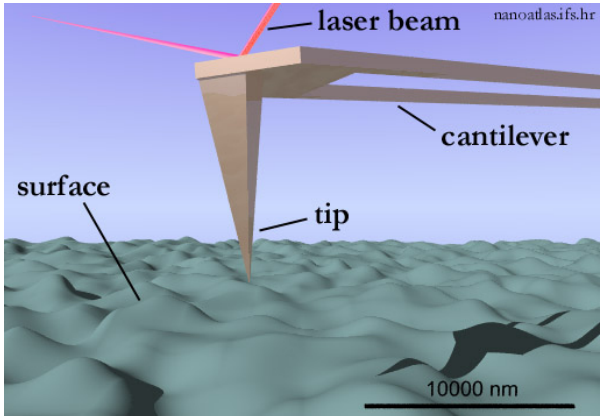
http://www.iflscience.com/health-and-medicine/dna-nanobots-will-seek-and-destroy-cancer-cells

http://nextbigfuture.com/2015/03/ido-bachelet-dna-nanobots-summary-with.html

Programming languages

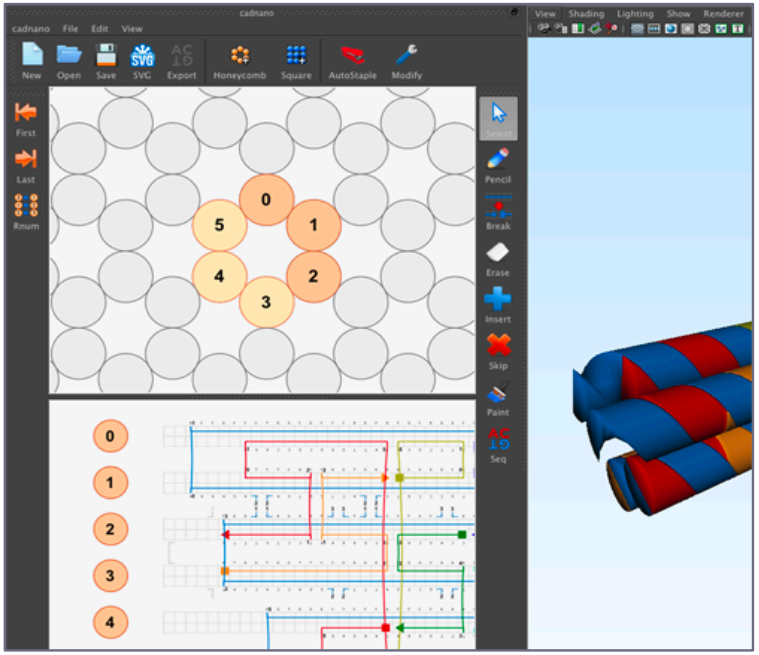
A. Phillips and L. Cardelli, "A programming language for composable DNA circuits," *Journal of the Royal Society Interface*, vol. 6, pp. S419–S436, 2009.

Verification tools



nanoatlas.ifs.hr

Design tools



CaDNAno.org

S. Woo

Computational model checking

Marta Kwiatkowska, Gethin Norman and David Parker. PRISM 4.0: Verification of Probabilistic Real-time Systems. In Proc. 23rd International Conference on Computer Aided Verification (CAV'11), volume 6806 of LNCS, pages 585-591, Springer, 2011.

Outline: Introduction to Molecular Programming

- Overview: what it is, recent advances, potential applications
- Introduction to its key modeling paradigms
- The challenge of making molecular programmed systems robust
- **Why software engineering?**
- Example: software engineering for DNA nanopliers

What Software Engineering contributes

Requirements engineering, design modeling, and design verification techniques empower us to address the challenge of programming systems that are verifiably robust.

Claims:

- Requirements analysis and verification can help us design a robust, embeddable, and scalable molecular device
- Model-based simulation, probabilistic model checking, and formal analysis each play unique, essential roles

How will software engineers lead?

You have techniques to help answer some challenges molecular programming faces:

- Requirements: are these realizable goals? (role of RE)
- Design: does it meets the goals? (role of design modeling & verification)
- Verification: can we find the bugs early? reduce test time? help set parameter values? (role of verification)
- Safety-critical systems: can we give some assurance that certain bad things won't/are unlikely to happen? (role of safety analysis)

Outline: Introduction to Molecular Programming

- Overview: what it is, recent advances, potential applications
- Introduction to its key modeling paradigms
- The challenge of making molecular programmed systems robust
- Why software engineering?
- **Example: software engineering for DNA origami pliers to detect presence of a target molecule**

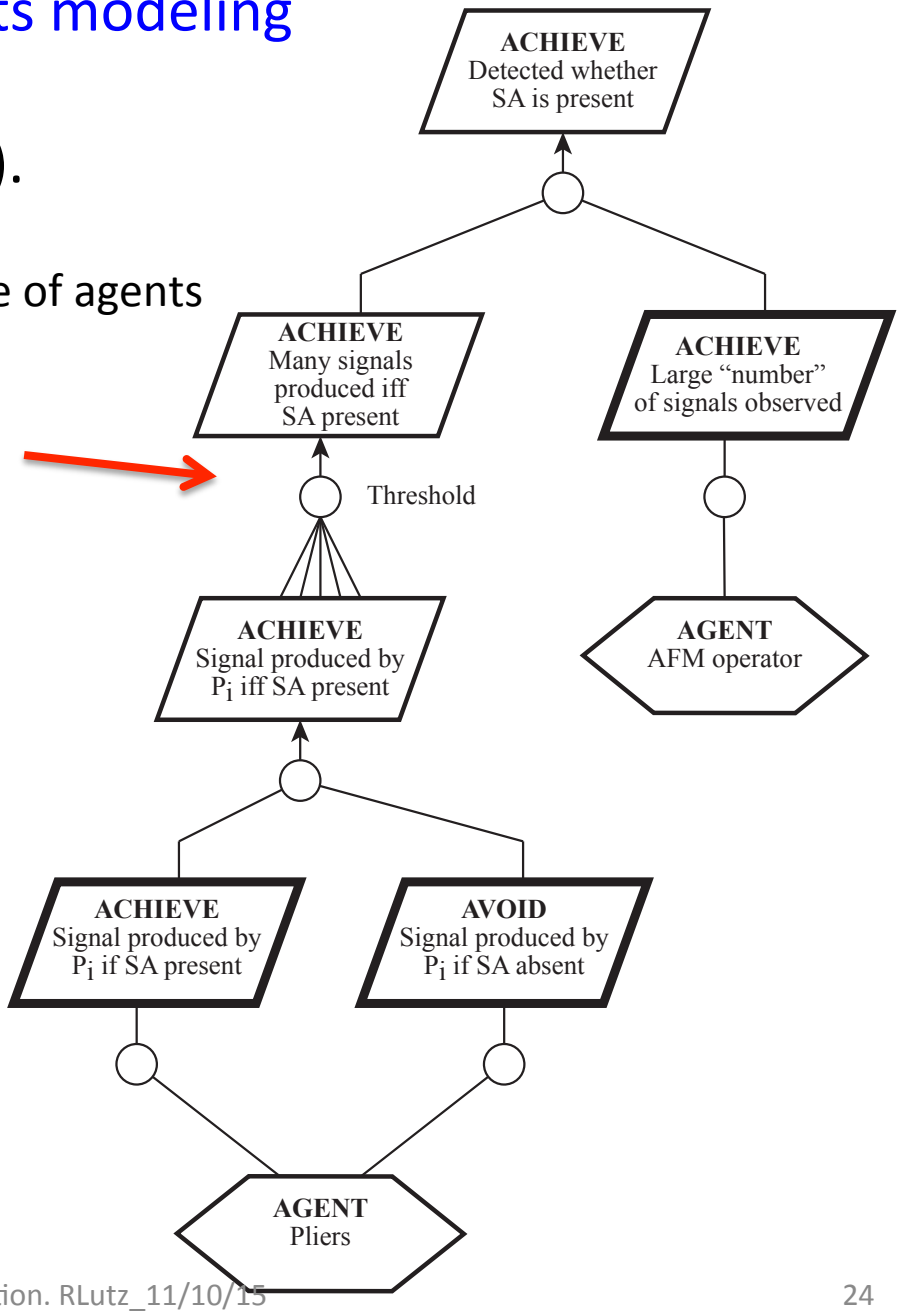
A. Kuzuya, Y. Sakai, T. Yamazaki, Y. Xu, and M. Komiyama, "Nanomechanical DNA origami 'single-molecule beacons' directly imaged by atomic force microscopy," *Nat Commun*, vol. 2, 2011/08/23/online.

We did goal-oriented requirements modeling and analysis of the DNA pliers (following van Lamsweerde, 2009).

~ a nanomole of agents

Goal is typically achieved when "enough" of the individual agents achieve their assigned subgoals: introduced a *threshold node*

(doesn't depend on probability something *might* happen, as it certainly will, but on the number of times that it *does*)

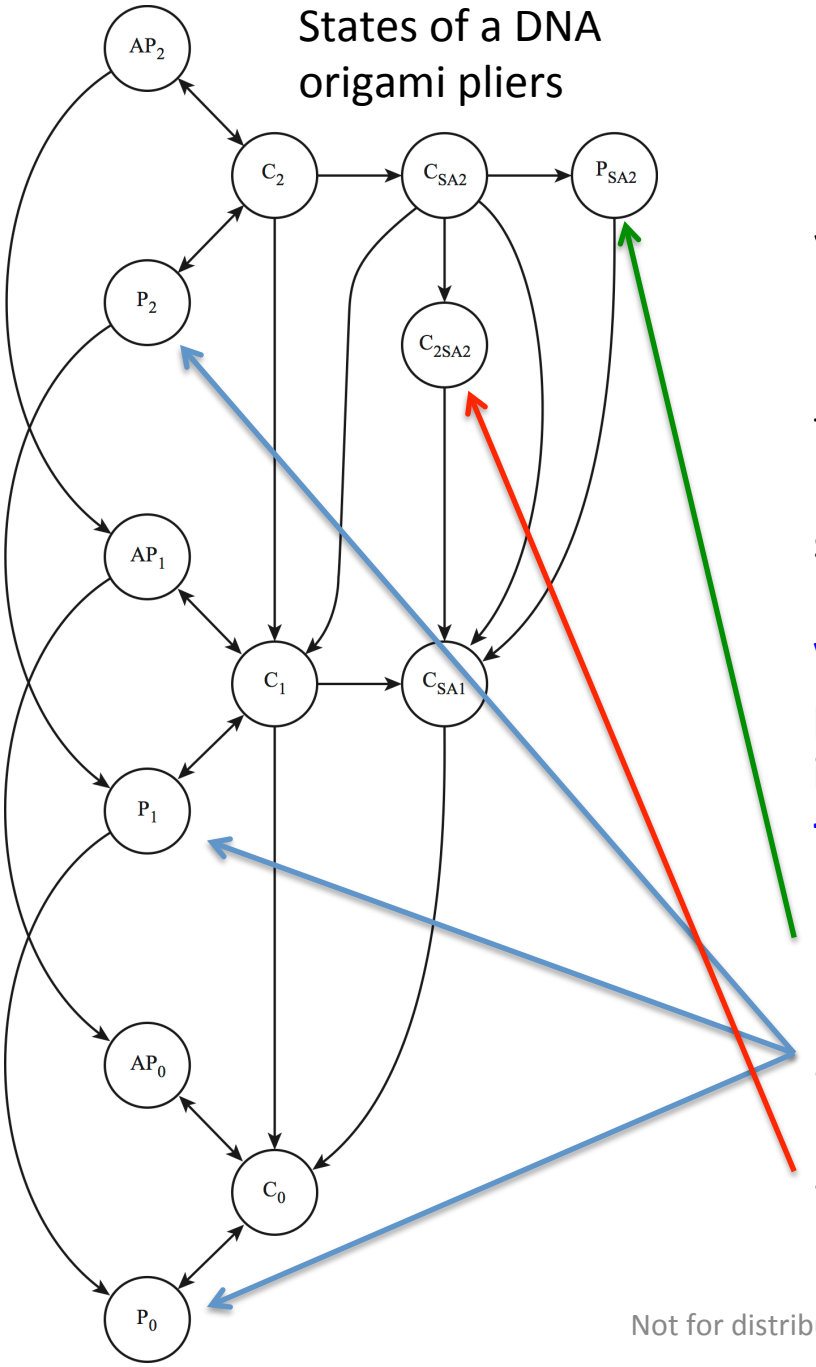


Analysis of the DNA pliers identified a failure mode
[Lutz et al., ICSE 2012]

1. Identify possible obstacles to meeting requirement (subgoal)
2. Assess likelihood and criticality
3. Resolve by adding or modifying a requirement, agent assignment, domain assumption, etc. [van Lamsweerde & Letier, 2000, van Lamsweerde 2010]

---

1. Identified: a *failure mode* : one target molecule binds to left jaw, another to right jaw, prevents jaws from closing.
2. Assessed: feasible and of concern
3. Resolved: added new requirement (if left ligand of a pliers binds to a SA molecule, right ligand should not bind to a different SA molecule, etc.).



Does the DNA system do what you want?

Modeled, using the software PRISM tool, the behavior of pliers in solution with SA molecules or unset strands. (Unset strands strip the jaws' lining and open the jaws.)

Verified by model checking that derived properties are satisfied in the model-- if the ratio of targets to detectors isn't too low or too high:

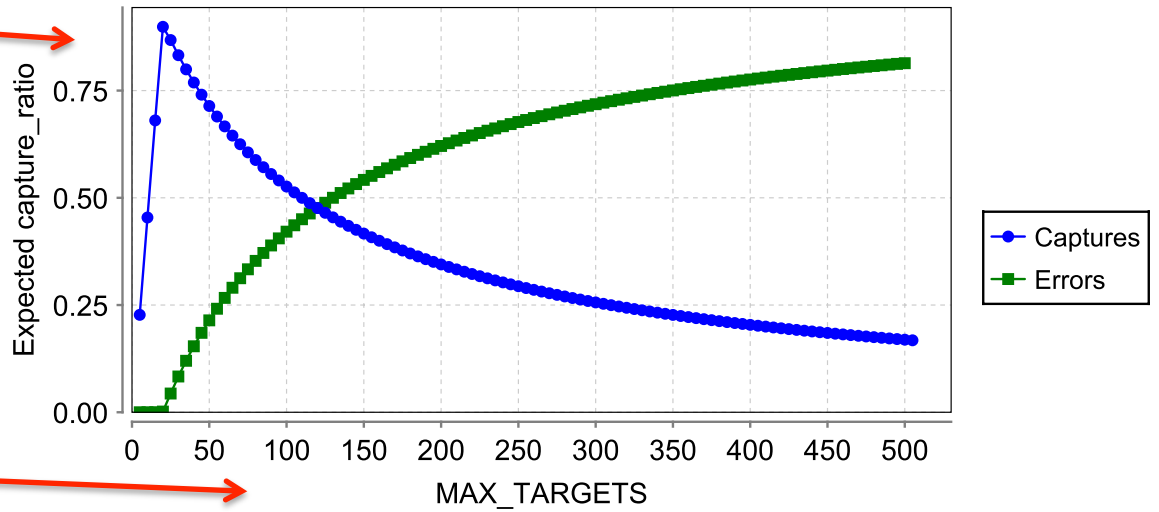
1. Achieve capture when the target molecule (SA) is present
2. Avoid false positives when SA is not present
3. Avoid error state (one SA in each jaw)

Simulation of target capture

Using PRISM, a probabilistic model checker [Lakin et al., 2012]

Y-axis:
Ratio of closed pliers (indicating capture) to open or antiparallel pliers

X-axis:
Number of target molecules in system



As concentration of target molecule is increased, the expected value of the capture ratio decreases.

Extending software engineering techniques to molecular programming

Extended standard requirements analysis method with a threshold gate to handle goals where computation by a very large number of instances of a nanodevice is mediated by a threshold

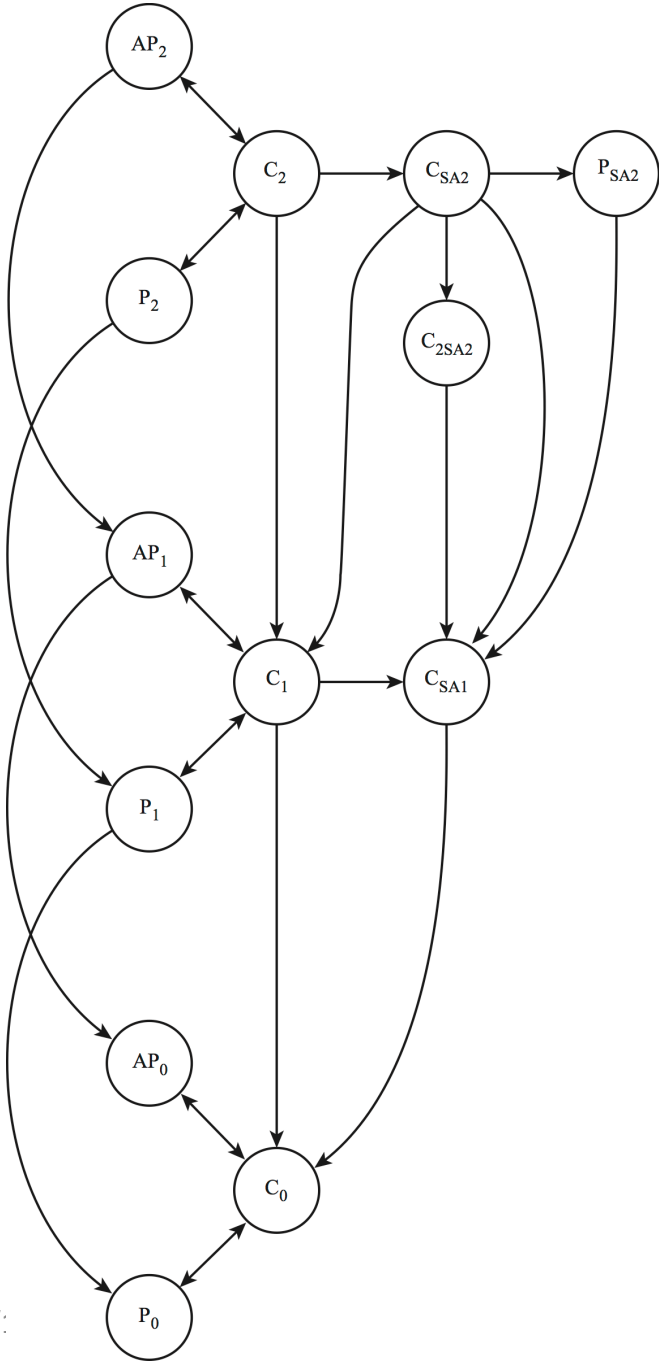
Used goal-oriented requirements analysis to uncover failure modes and associated missing requirements in the nanopliers [Lutz et al, ICSE 2012; RE 2012]

Product family of nanopliers
[Lutz et al., RE 2012]

- We later studied other nanopliers and realized that they all shared many common features, with a few key variations among them
- 3 products:
  - Simple pliers
  - G-zipper pliers
  - Unzipper pliers
- Can we leverage the similarities by representing them as a product family?
- Does this support reuse in modeling and analysis?

Modeling nanodevices

- Memoryless, stochastic, asynchronous process
- Modeled behavior of one nanoplier as continuous time Markov chain (CTMC) in PRISM model
- Each transition is a chemical reaction
- Rate of transitions determined by concentrations of reactants
- Product family reuse:
  Shares common subgraph



Properties derived from requirements

- Specified in PRISM's continuous stochastic logic language (CSL), using PRISM's reward-based properties
- Example of common properties:

```
P=? [ F<=t "target_detected" ]
```

"What is the probability of target being detected within time t?"

- Example of variable (device-specific) properties:

```
R{"errors"}=? [ S ]
```

"What is the expected number of errors at equilibrium?" (errors are instances of "avoid" failure mode)

Extending software engineering techniques to molecular programming

Handling similar nanoplier devices as a product family made the requirements more complete; may save modeling effort [RE 2012]

Morning Break in Commons Area, 10:00-10:30

Software engineering for molecular programming

Design modeling and verification of a molecular watchdog timer

A molecular watchdog timer

Many envisioned DNA nanosystem applications will be safety-critical (ex. bio-sensors, drug delivery)

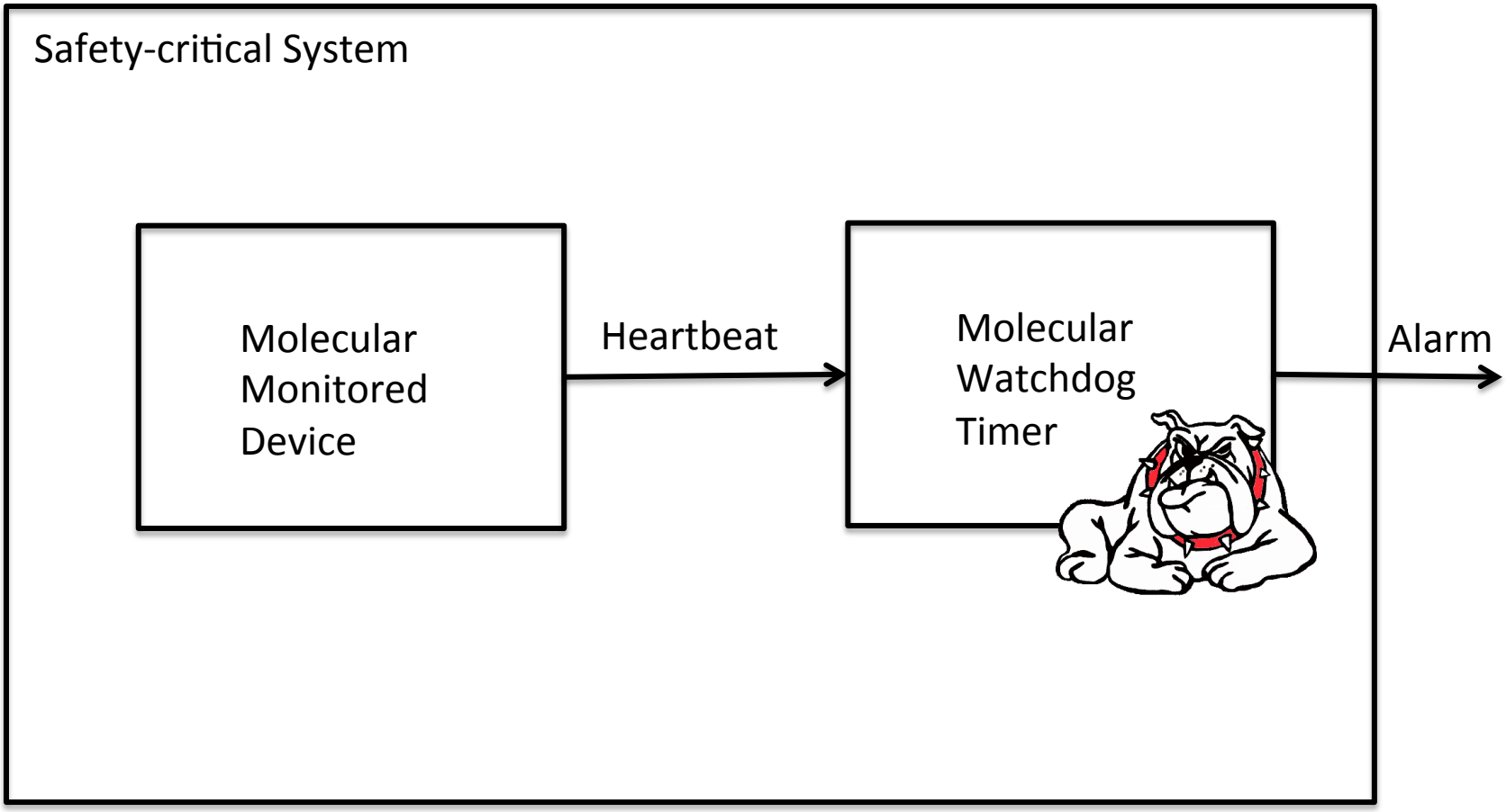
Problem:

Monitoring for when an expected event occurs in a DNA nanosystem is hard. Monitoring for the absence of an expected event is even harder.

Solution we proposed:

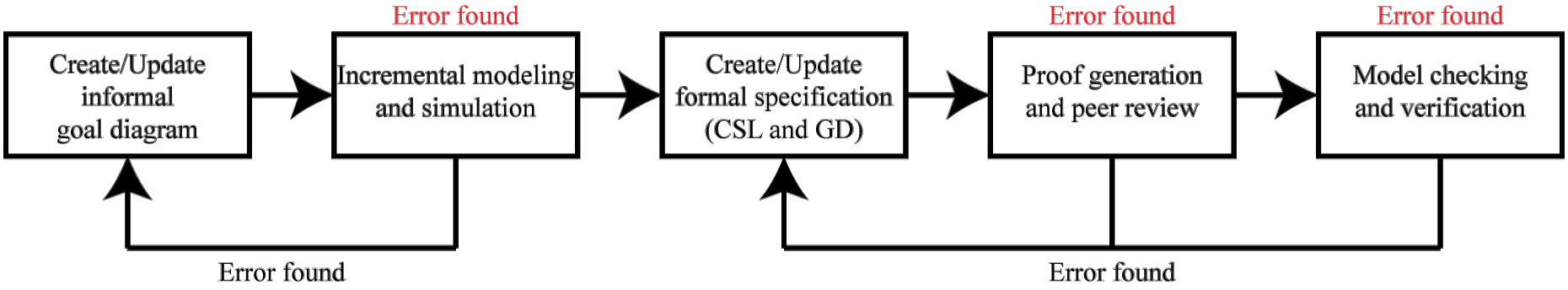
A molecular watchdog timer

Design of a programmed molecular system



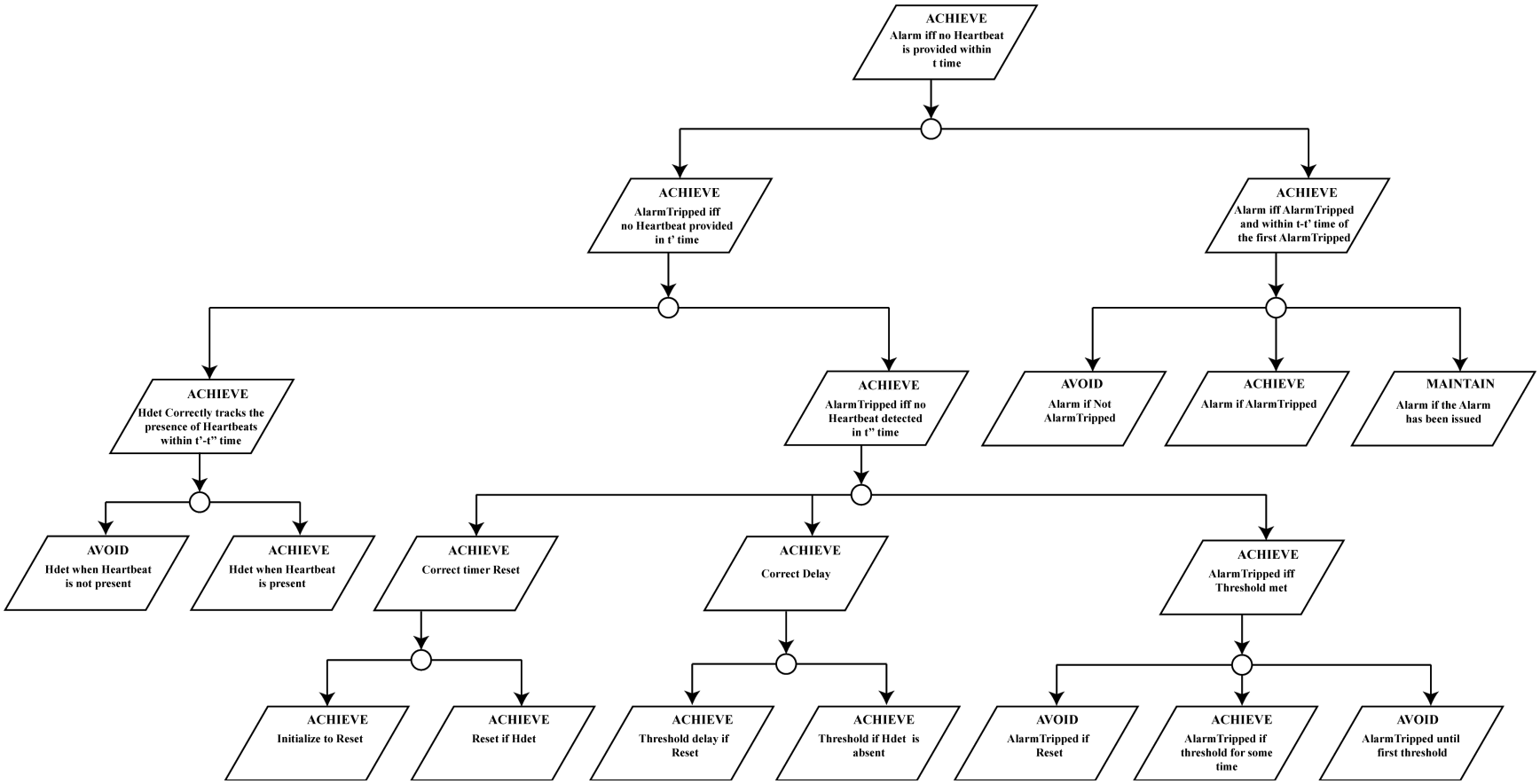
Images: wikipedia, recantha.co.uk

Approach


Error found
Error found
Error found
Create/Update informal goal diagram
Incremental modeling and simulation
Create/Update formal specification (CSL and GD)
Proof generation and peer review
Model checking and verification
Error found
Error found

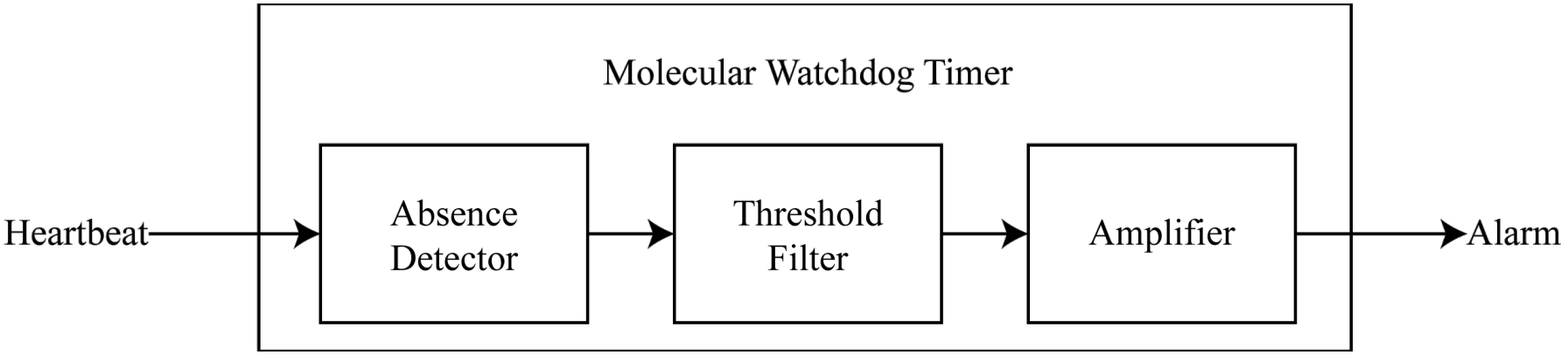
Goal Diagram

[van Lamsweerde ‘09]

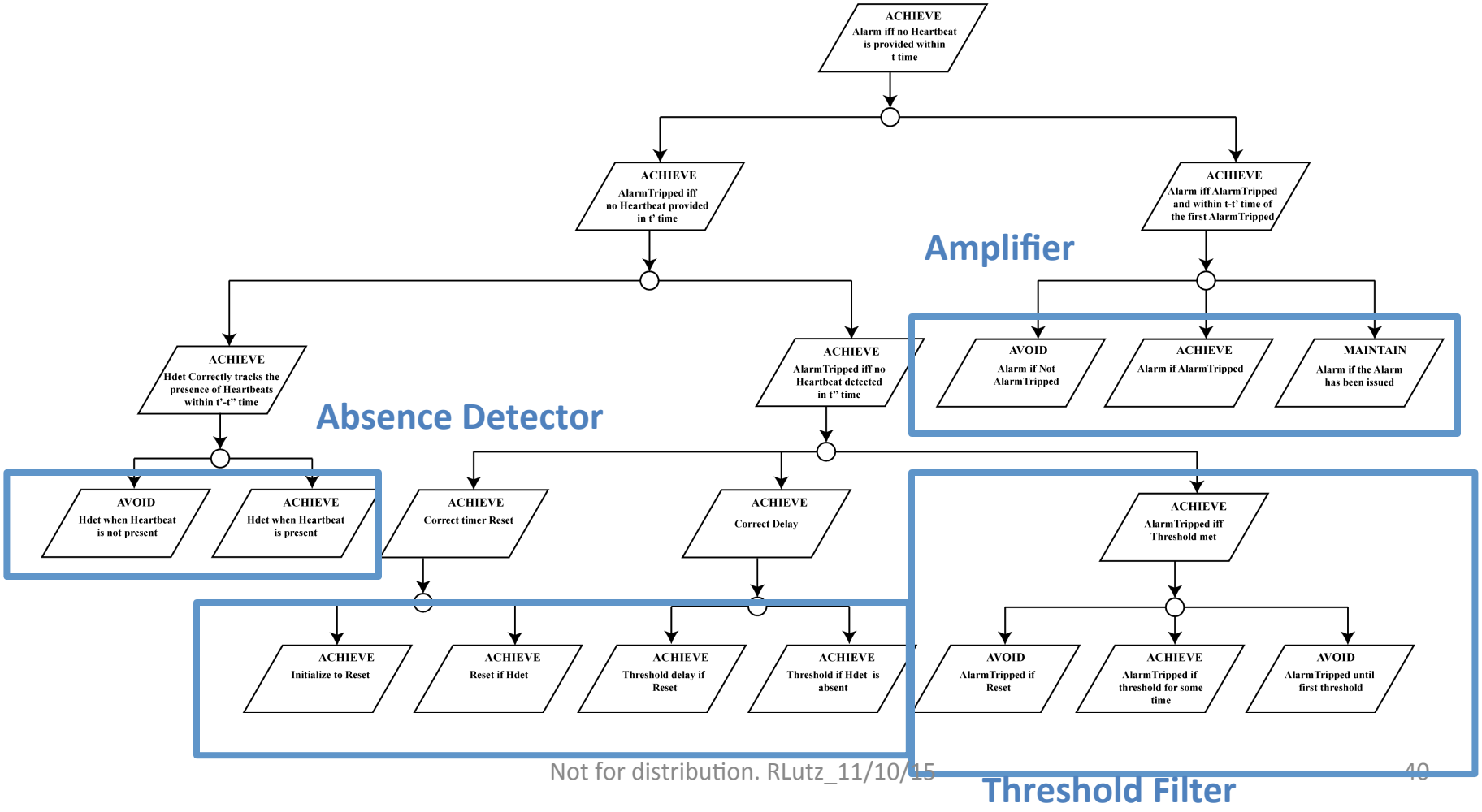


Molecular Watchdog Timer (MWT) Design

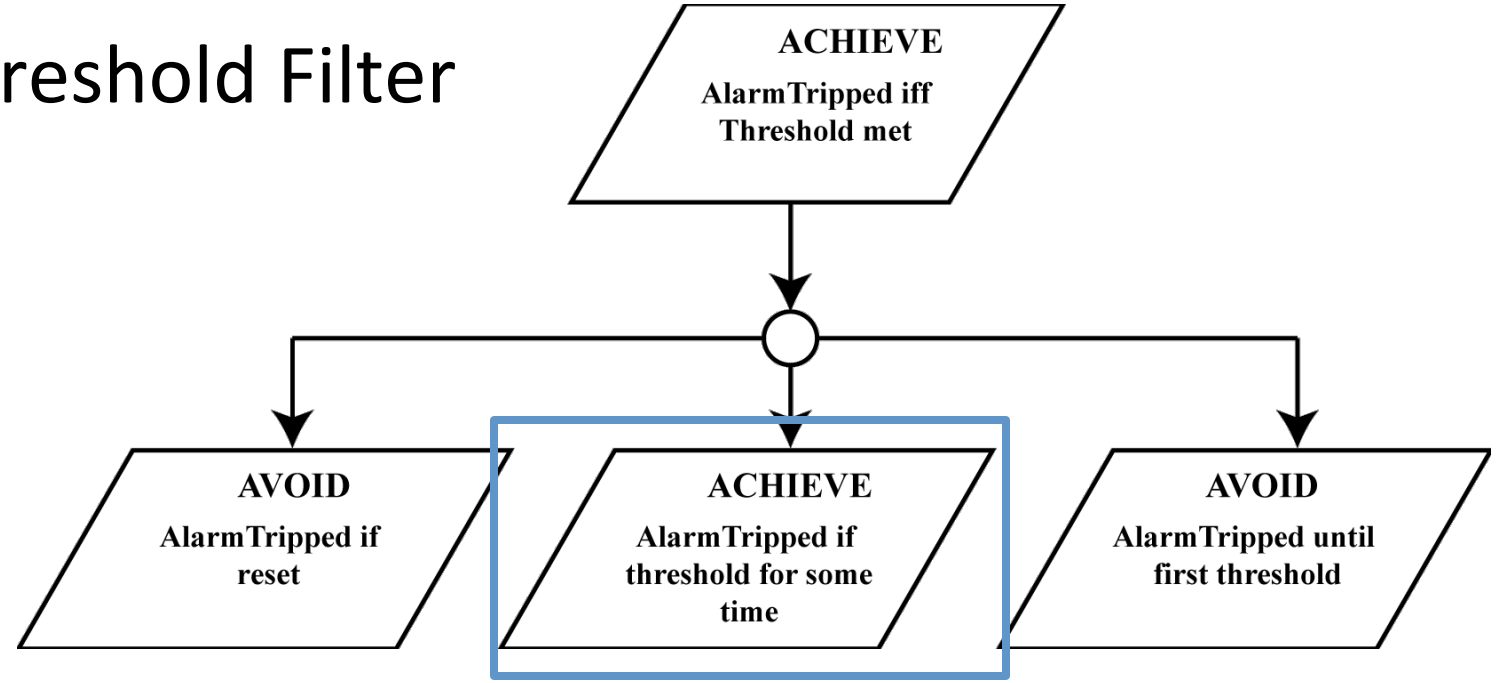
- 3 components
  - Absence Detector
  - Threshold Filter
  - Amplifier



Goals assigned to agents



Threshold Filter



ACHIEVE: AlarmTripped if threshold for some time

$$\boldsymbol{\mathcal{P}}_{\geq 1} \Box \left[Th_H \implies \boldsymbol{\mathcal{P}}_{\geq 1-\eta_4} \Diamond_{\leq w_{th}} \left(A_{trip} \vee \neg Th_H \right) \right]$$

Automated Verification

- Model Checking to verify formal goals.
  - Used SimBiology to generate a skeleton of our model
  - Expanded the model in PRISM [Kwiatkowska et al., 2011]
  - Used PRISM to test different parameter values until the properties were satisfied.

ACHIEVE: AlarmTripped if threshold for some time

$$\boldsymbol{\mathcal{P}}_{\geq 1} \Box \left[Th_H \implies \boldsymbol{\mathcal{P}}_{\geq 1-\eta_4} \Diamond_{\leq w_{th}} \left(A_{trip} \vee \neg Th_H \right) \right]$$

Errors discovered through automated verification
[Ellis, et al., ASE 2014]

- Absence Detector Agent cannot achieve assigned goal
- Missing initialization case
- Missing domain property
- Goal too strict
- Goals coupled too tightly

Extending software engineering techniques to molecular programming

- Designed and verified the model for a watchdog timer, a device commonly used to monitor the health of a safety critical system [ASE 2014]

Lunch in the Atrium, 12:30-1:30

en.wikipedia.org

Wrapping up & moving forward

Wikipedia commons

Wikipedia commons

SE ⟷ MP

What good is software engineering for a molecular programmed system?

A better way to make a new thing in DNA!

- Provided a process for incremental goal refinement/obstacle analysis
- Identified infeasible and fragile requirements given the domain
- Indicated other requirements as feasible in this domain
- Model-based predictions of reliable performance
- Identified incorrect domain assumptions
- Model-based assurance that some failure modes are unlikely
- Yielded requirements for more robust MWT
- Reduced number of experiments needed (< time / < cost)

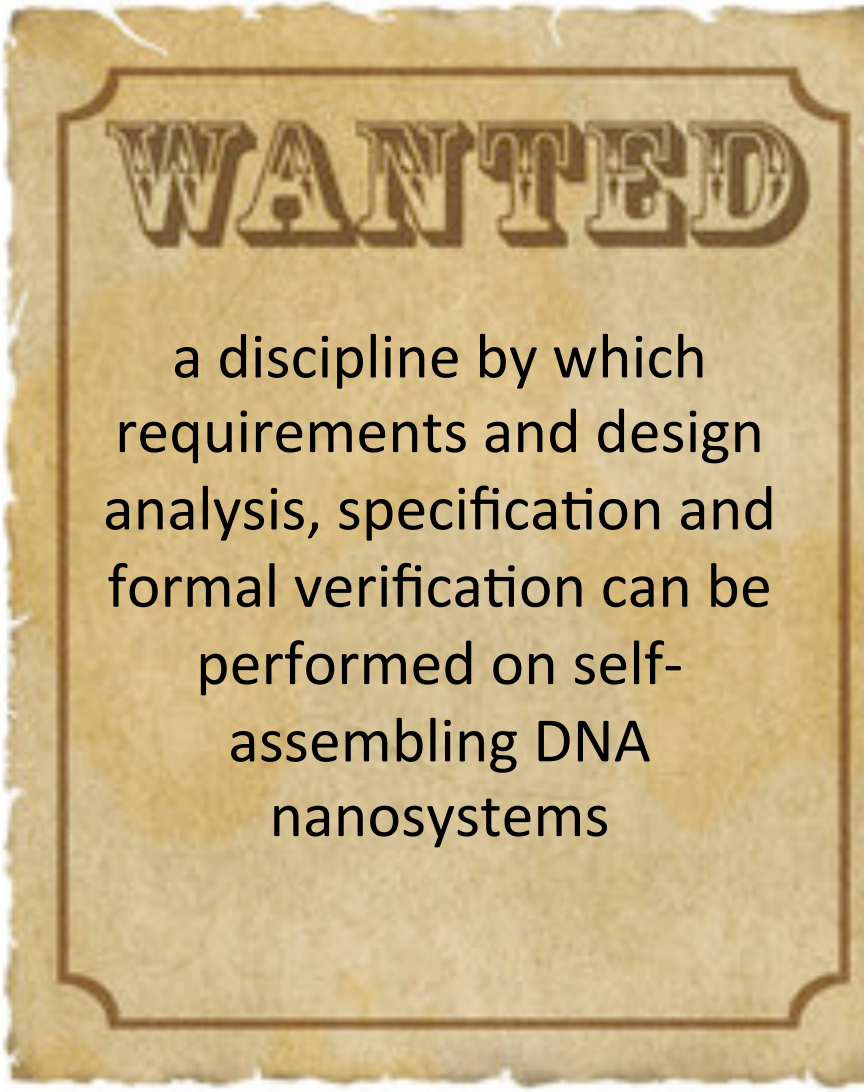
What good is molecular programming for software engineering?

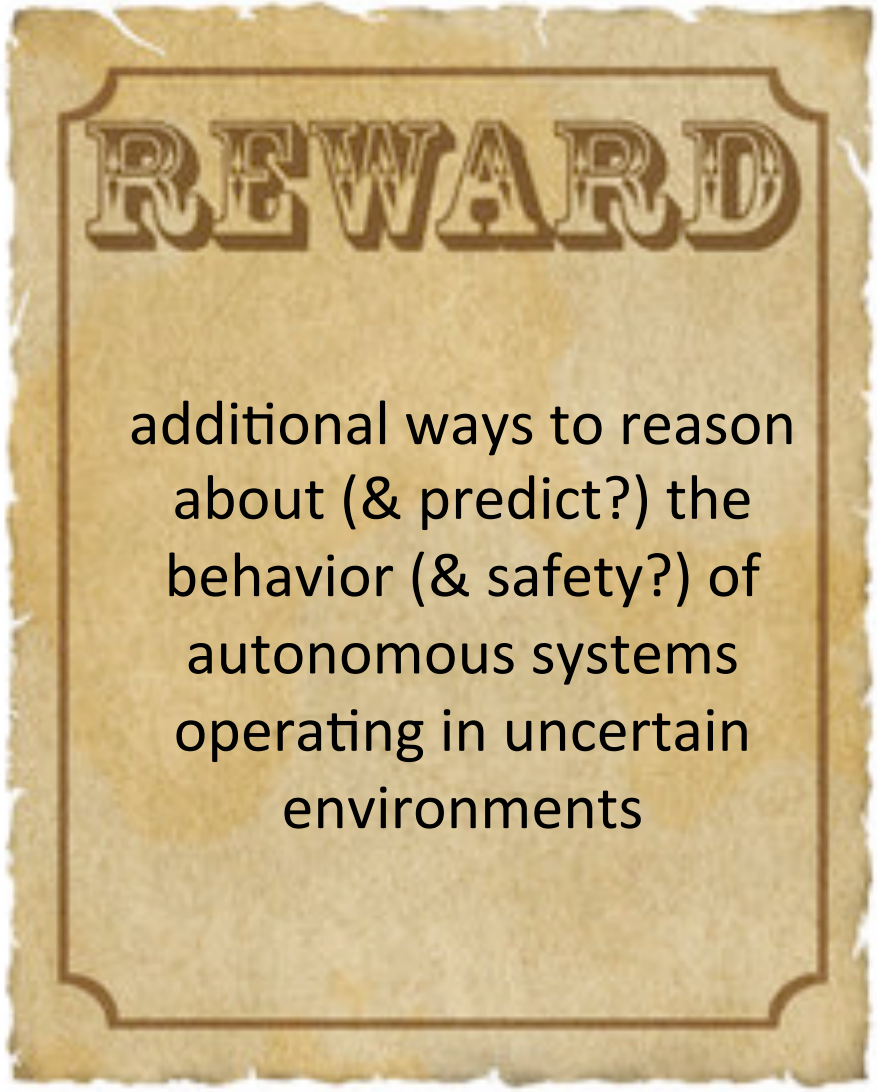
Better techniques for some troublesome domains!

- Shared interest in failures caused by inadequate environmental assumptions
- How to determine the validity/safety/performance of decisions based on "mobiquitous" sensors in a partially understood environment
- Needed focus on autonomous, failure-prone agents
- Better understanding of how computation occurs in nature

Your software engineering techniques offer solutions to challenges molecular programming faces

- Autonomous self-assembly
- Modular and composable
- Distributed and mobile (a nanomole of agents)
- Highly concurrent and asynchronous
- High failure rate of individual agents
- Soft goals (prefer lower cost for chemicals)
- Safety-critical applications (~soon)
- Embedded applications (in our bodies)
- Cross-disciplinary by nature


WANTED
a discipline by which requirements and design analysis, specification and formal verification can be performed on self-assembling DNA nanosystems


REWARD
additional ways to reason about (& predict?) the behavior (& safety?) of autonomous systems operating in uncertain environments

More information

R. Lutz, J. Lutz, J. Lathrop, T. Klinge, E. Henderson, D. Mathur, and D. Abo Sheasha. Engineering and verifying requirements for programmable self-assembling nanomachines. In *Proc. 34th International Conference on Software Engineeri*ng (Zurich, Switzerland, June 2-9, 2012).

R. Lutz, J. Lutz, J. Lathrop, T. Klinge, D. Mathur, D. Stull, T. Bergquist, and E. Henderson. Requirements analysis for a product family of DNA nanodevices. In *Proc. 20th IEEE International Requirements Engineering Conference* (Chicago, IL, Sept. 24-28, 2012).

S. Ellis, E. Henderson, T. Klinge, J. Lathrop, J. Lutz, R. Lutz, D. Mathur, and A. Miner. Automated Requirements Analysis for a Molecular Watchdog Timer. In *Proc. 29th IEEE/ACM International Conference on Automated Software Engineering (*Västerås, Sweden, Sept. 15-19, 2014). IFIP TC2 Manfred Paul Award "for Excellence in Software: Theory and Practice."

T. Tun, R. Lutz, B. Nakayama, Y. Yu, D. Mathur, and B. Nuseibeh. The role of environmental assumptions in failures of DNA nanosystems. In *Proc. Workshop on Complex Faults and Failures in Large Software Systems (COUFLESS),* co-located with ICSE 2015 Firenze, Italy, May 24, 2015).

This work was supported in part by National Science Foundation grants 1143830, 1247051 and 1545028.

Thanks!