Stochastic Networked Embedded Systems

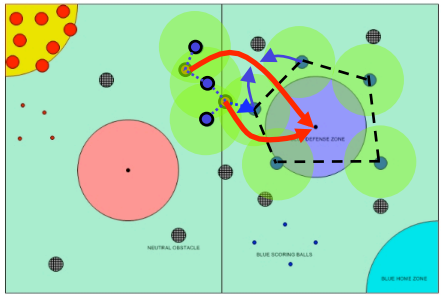
Specification and Analysis

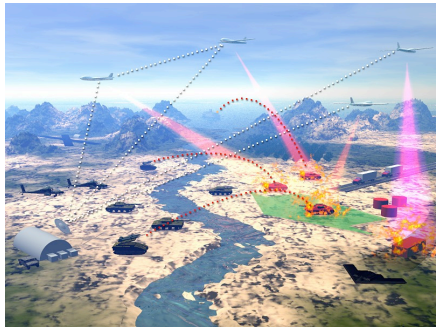
Eric Klavins
Electrical Engineering
University of Washington
Seattle, WA

AFOSR MURI: V&V
With Caltech (Murray, Doyle, Chandy and Parillo)

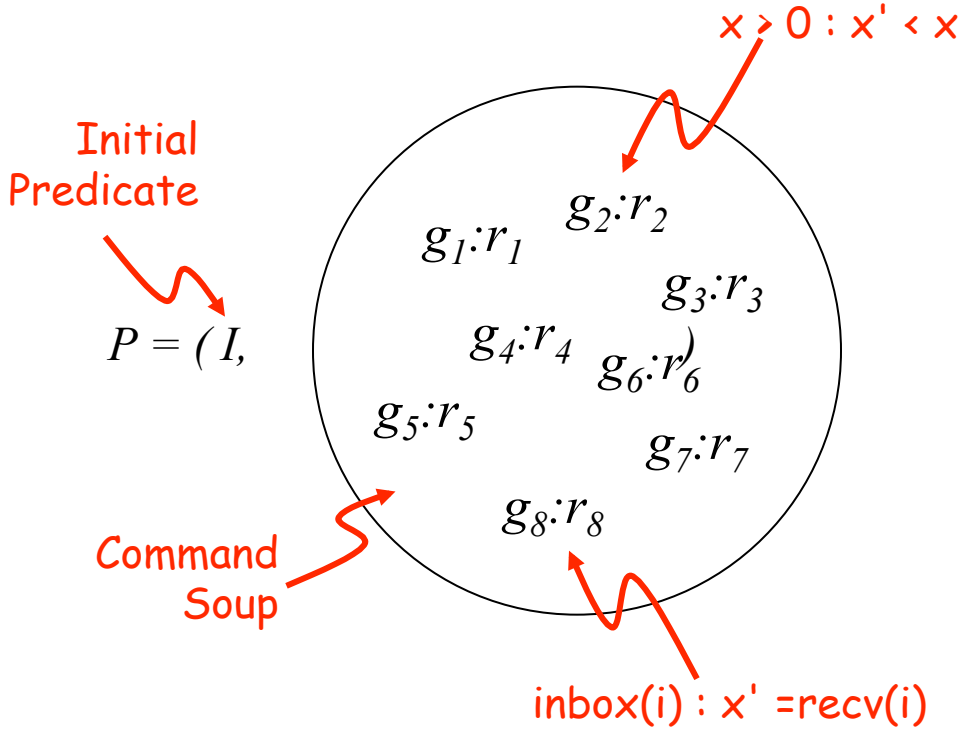
Outline

- Concurrency
  - Graph Grammars (Nondet, embedded)
  - Guarded Command Programs
  - Guarded Command Programs with Rates
- Markov Processes from GCPWR
- Composition
- Linearly Disabled Markov Processes
- Robustification





Concurrency

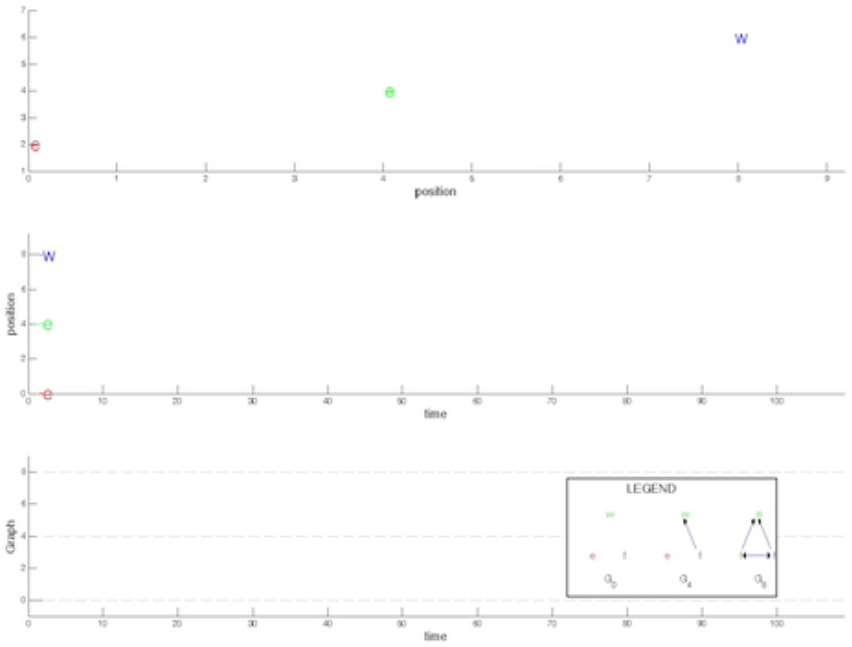


In CS, typically:

- Not real time
- No continuous state
- Performance not covered
- Limited stochastic tools

Concurrency and Control

$$\mathbf{u}_i(\gamma) = \begin{cases} 1 & \text{if } i.mode = \texttt{east} \\ -1 & \text{if } i.mode = \texttt{west} \\ \sum_{j \in N(i)} \mathbf{x}_j - \mathbf{x}_i & \text{if } i.mode = \texttt{follow} \end{cases}$$



Rule r_1 (Join)

Precondition:

$||x_i - x_j|| < 3$,

e *w*

Effect:

w ⟵ *f*

Rule r_2 (Close Cycle)

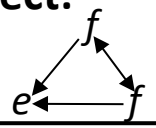
Precondition:

$||x_i - x_j|| < 5 \wedge ||x_k - x_j|| < 5 \wedge ||x_i - x_k|| < 5$

e

w ⟵ *f*

Effect:



Tools: Lyapunov + Hybrid Systems + Concurrency + Temportal Logic

Toyota Research

J. M. McNew, E. Klavins and M. Egerstedt. **Solving Coverage Problems with Embedded Graph Grammars**. *Hybrid Systems: Computation and Control*. 2007.

S. Waydo, J. Hauser and R. Bailey, E. Klavins and R. Murray, **UAV as a Reliable Wingman: A Flight Demonstration**, *IEEE Transactions on Control Systems Technology*, Jul. 2007, Vol. 15, No. 4, pp. 680--688.

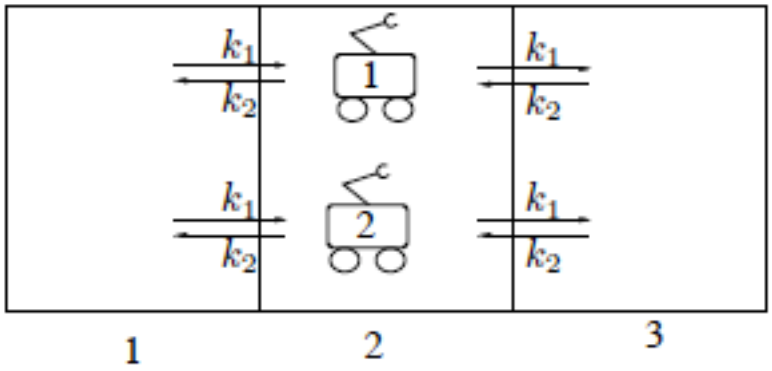
CCL: A Guarded Command Programming Language

```
program send_msg (i) :={

  needs mode, x, toRobot, msg;

  (mode = "idle") & (rand(1000) = 1) :{
    toRobot := (i + rand( N - 1 ) + 1) % N,

    send([to := toRobot , from := i, x := x,
    ack := false]),

  };
};
```


E. Klavins, **A Language for Modeling and Programming Cooperative Control Systems**, *Proceedings of the International Conference on Robotics and Automation*. May 2004, pp. 3403- 3410.

E. Klavins and R. Murray, **Distributed Algorithms for Cooperative Control**, *IEEE Pervasive Computing*. 2004, Vol. 3, No. 1, pp. 56--65.

Guarded Command Programs with Rates



General syntax:

```
guard : rule : rate
```

E.g.

```
pos(i) = 1 : pos(i)'=pos(i)+1 : 2.5
```

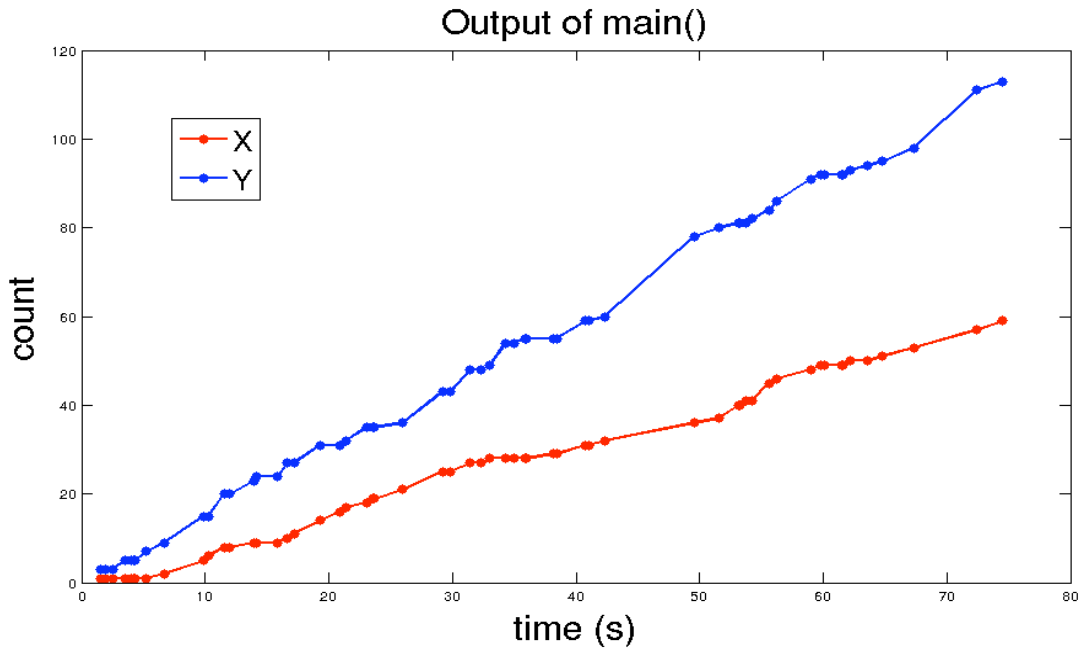
The rate is the probability that this rule with execute in time [t,t+dt) given that its guard is true at time t.

N. Napp and E. Klavins, **Robust by Composition: Programs for Multi-Robot Systems,** submitted to ICRA 2010.

The `rate` guard in CCL

```
program drunkard( k ) := {
        X:=0;
        (rate k):{ X++ };
  };

program main() := updateDT() + drunkard( 1.0 ) + drunkard( 2.0 );
```

Needed by `rate`



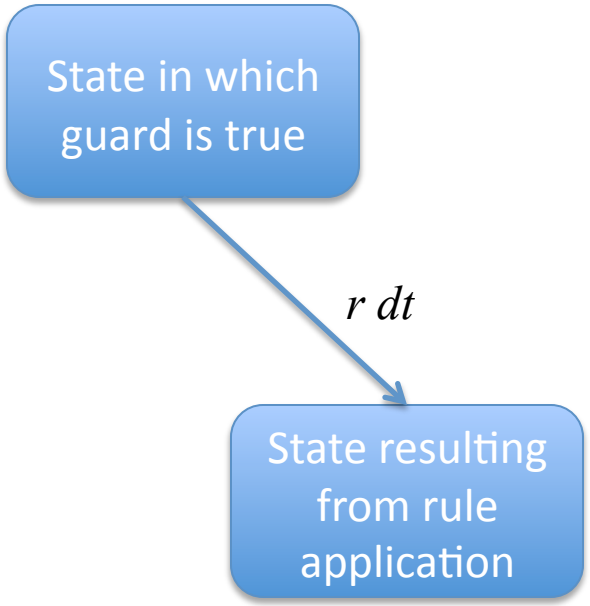
The Markov Process of a GCPWR

Each guarded command program corresponds to a infinitesimal generator matrix $\mathbf{Q}(\Psi)$.

$$\mathbf{A}_{i,j}(\psi) = r, \text{ if } (i,j) \in a \wedge i \in g \text{ and } 0 \text{ otherwise.}$$

$$\mathbf{A}(\Psi) = \sum_{\psi \in \Psi} \mathbf{A}(\psi)$$

$$\mathbf{Q}(\Psi) = \mathbf{A}(\Psi) - \mathrm{diag}(\mathbf{A}(\Psi)\mathbf{1})$$



Semantics of Program Composition

Composition:

$$\mathbf{A}(\Psi_1 \cup \Psi_2) = \mathbf{A}(\Psi_1) + \mathbf{A}(\Psi_2)$$
$$\mathbf{Q}(\Psi_1 \cup \Psi_2) = \mathbf{Q}(\Psi_1) + \mathbf{Q}(\Psi_2)$$

Scaling:

$$\mathbf{A}(\alpha\Psi) = \alpha\mathbf{A}(\Psi)$$
$$\mathbf{Q}(\alpha\Psi) = \alpha\mathbf{Q}(\Psi)$$

Reverse:

$$\mathbf{A}(\Psi^R) = \mathbf{A}(\Psi)^T$$
$$\mathbf{Q}(\Psi^R) = \mathbf{A}(\Psi)^T - \mathrm{diag}(\mathbf{A}(\Psi)^T\mathbf{1})$$

Expressibility

Without Rates

$\Phi_1 \cup \Phi_1$: Arbitrary fair interleavings may result in incorrect behaviors

With Rates

$a\Phi_1 \cup b\Phi_2$: Can balance the relative rates at which programs run

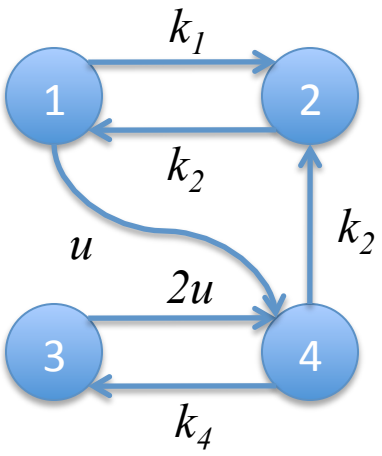
The Control Problem

$\Phi_{\text{plant}} \cup u\Phi_{\text{control}}$: Find u(t) so that some desired performance statistics result

Robustification

$\Phi_1 \cup \varepsilon\Phi_2$: Add a small amount of P2 to make P1 more robust

Linearly Disabled Markov Processes



$$u \in [0, 1]$$

Some uncertainty in Q

$$Q = Q_0 + uQ_1$$

$$y = g(x)$$

Some performance measure

Controllable Region

$$Q_{u=0} = Q_0$$

$$Q_{u=1} = Q_0 + Q_1$$

$$[y_{u=0}, y_{u=1}]$$

Control Problem

Robustly stabilize E[y*]

PI Control

Given a guarded command program with rates

$$\Phi_{\text{plant}} \cup u\Phi_{\text{control}}$$

Define

$$\dot{x} = f(\mathbf{q}, x) = \gamma(y(\mathbf{q}) - y^*)$$

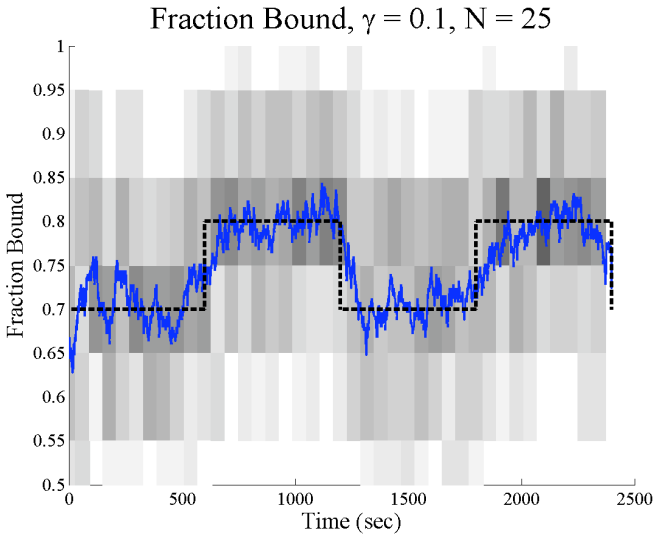
$$h(x) = \begin{cases} 0, & x \leq 0 \\ x, & 0 < x \leq 1 \\ 1, & 1 < x. \end{cases}$$

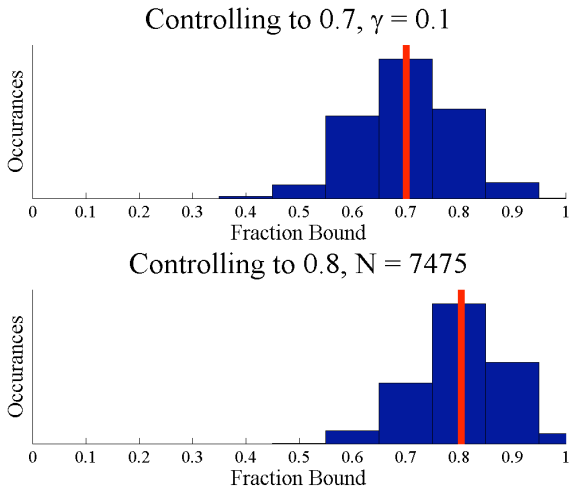
$$u = h(x)$$

Theorem: If y* is in the controllable region, then E[y] -> y*. The variance is tunable via gamma.

Interpretation: Can balance the rates at which two guarded command programs run so that a desired performance level is guaranteed.

Example



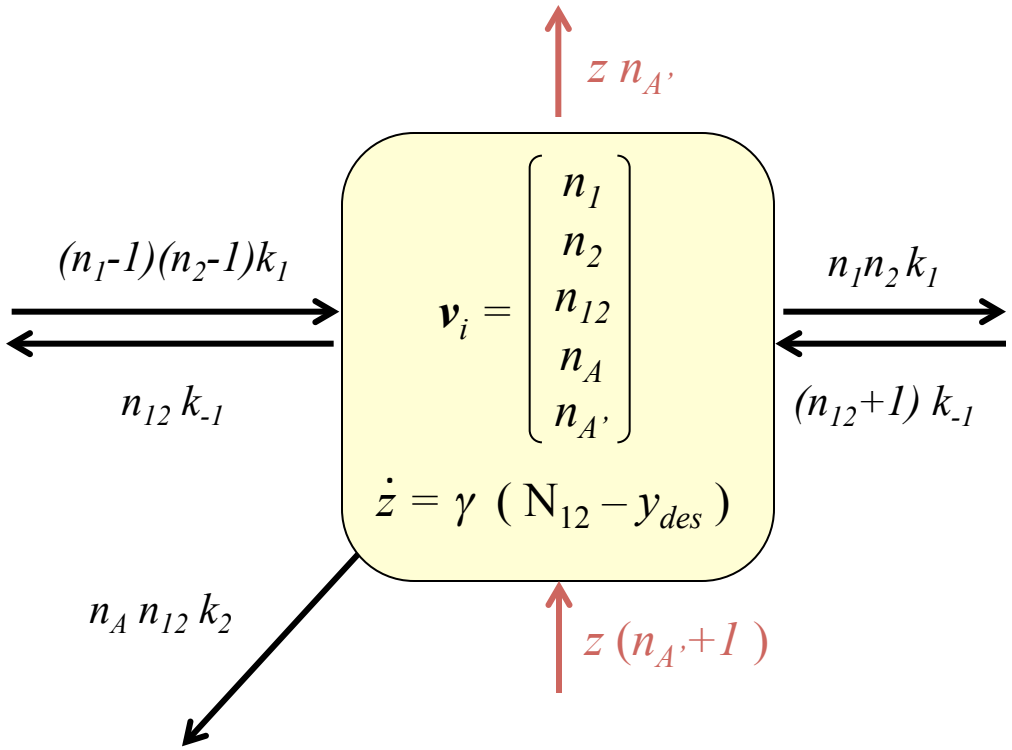


$$P_1 + P_2 \underset{k_{-1}}{\overset{k_1}{\rightleftharpoons}} P_{12}$$

$$P_{12} + A \xrightarrow{k_2} P_1 + P_2 + A'$$

$$A' \xrightarrow{uk_3} A$$

$$u = h(z), \quad \dot{z} = \gamma(N_{P_{12}} - y^*)$$



Expressibility

Without Rates

$\Phi_1 \cup \Phi_1$: Arbitrary fair interleavings may result in incorrect behaviors

With Rates

$a\Phi_1 \cup b\Phi_2$: Can balance the relative rates at which programs run

The Control Problem

$\Phi_{\text{plant}} \cup u\Phi_{\text{control}}$: Find u(t) so that some desired performance statistics result

Robustification

$\Phi_1 \cup \varepsilon\Phi_2$: Add a small amount of P2 to make P1 more robust

Robustification

$$\Phi_1 \cup \varepsilon\Phi_2$$

A high performance program describes how the system should work when everything is functioning normally.

A safe, but low performance program works in more situations, but is slow.

Robustification

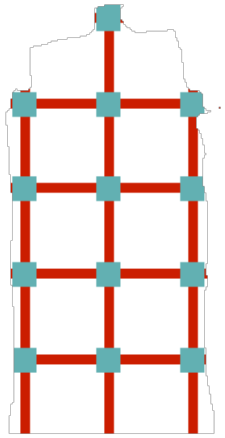
$$\Phi_1 \cup \varepsilon\Phi_2$$

Theorem 5. *Given two* Q*-matrices* A *and* B *with the same, unique recurrent state* s^**, then for any* $\delta > 0$ $\exists$ $\varepsilon > 0$ *such that for* $C = A + \varepsilon B$

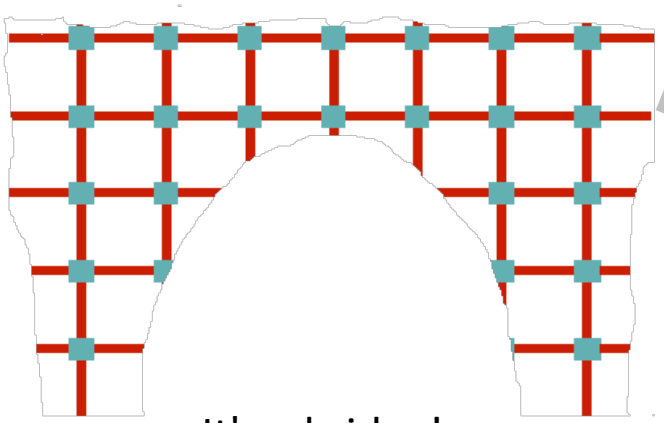
1) $|\lambda_2(A) - \lambda_2(C)| < \delta$,
2) $\kappa_C(\{s^*\}^C, s^*) \geq \kappa_B(\{s^*\}^C, s^*)$,
3) $\pi_C = \pi_A$.

Increased connectivity means alternative paths around fault modes

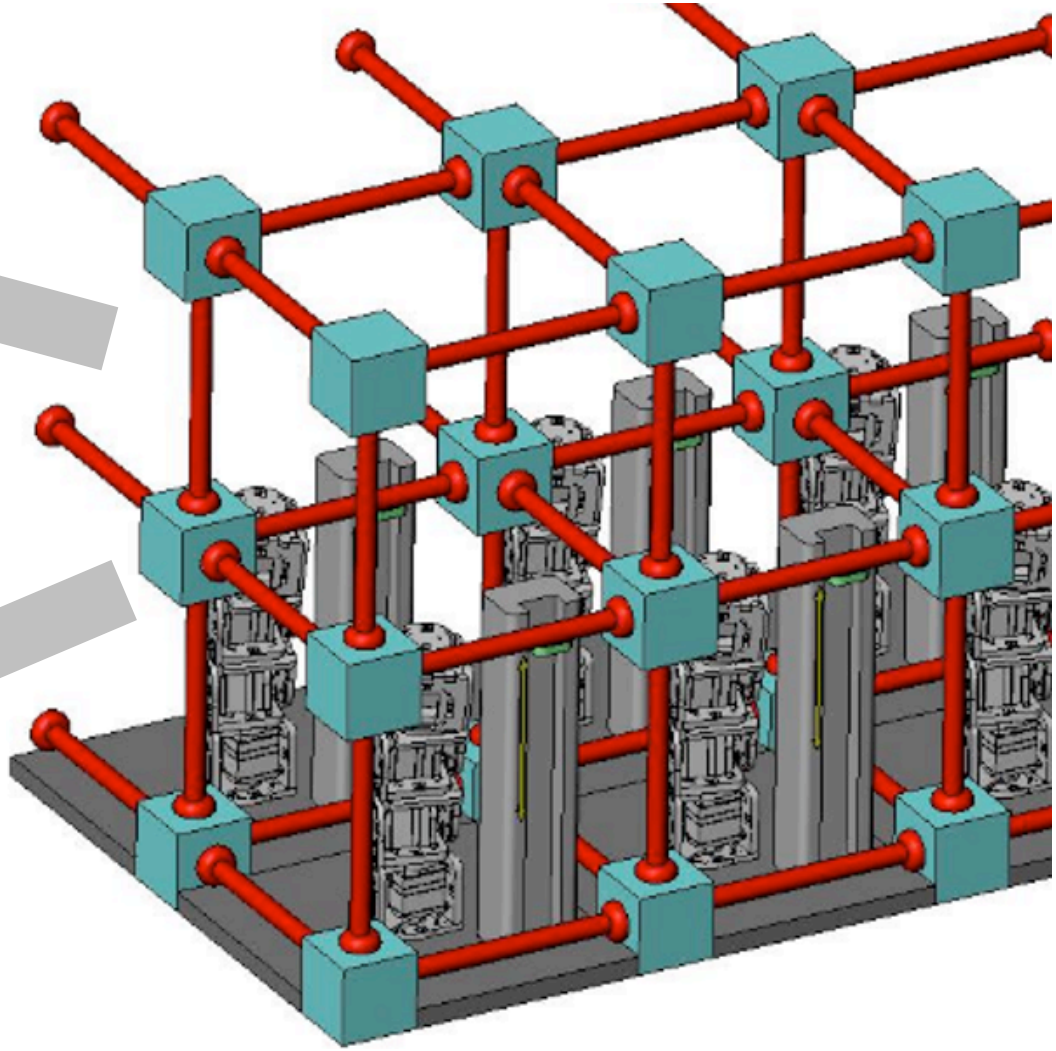
•Stochastic Factory Floor (SFF)



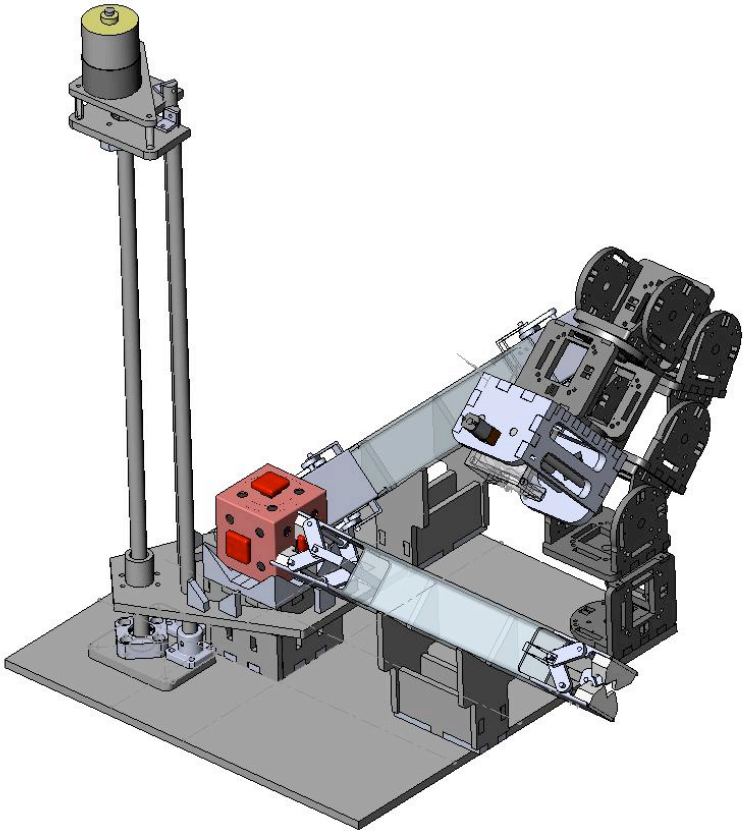
It's a skyscraper!



It's a bridge!



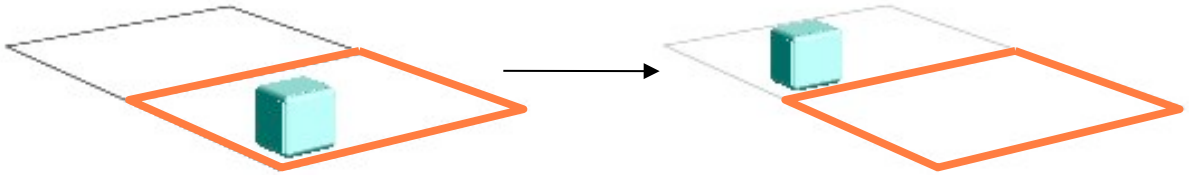
Current Hardware



NSF: EFRI Reconfigurable Systems
With MIT (Rus), U. Penn (Yim) and Cornell (Lipson)

Programmed in CCL

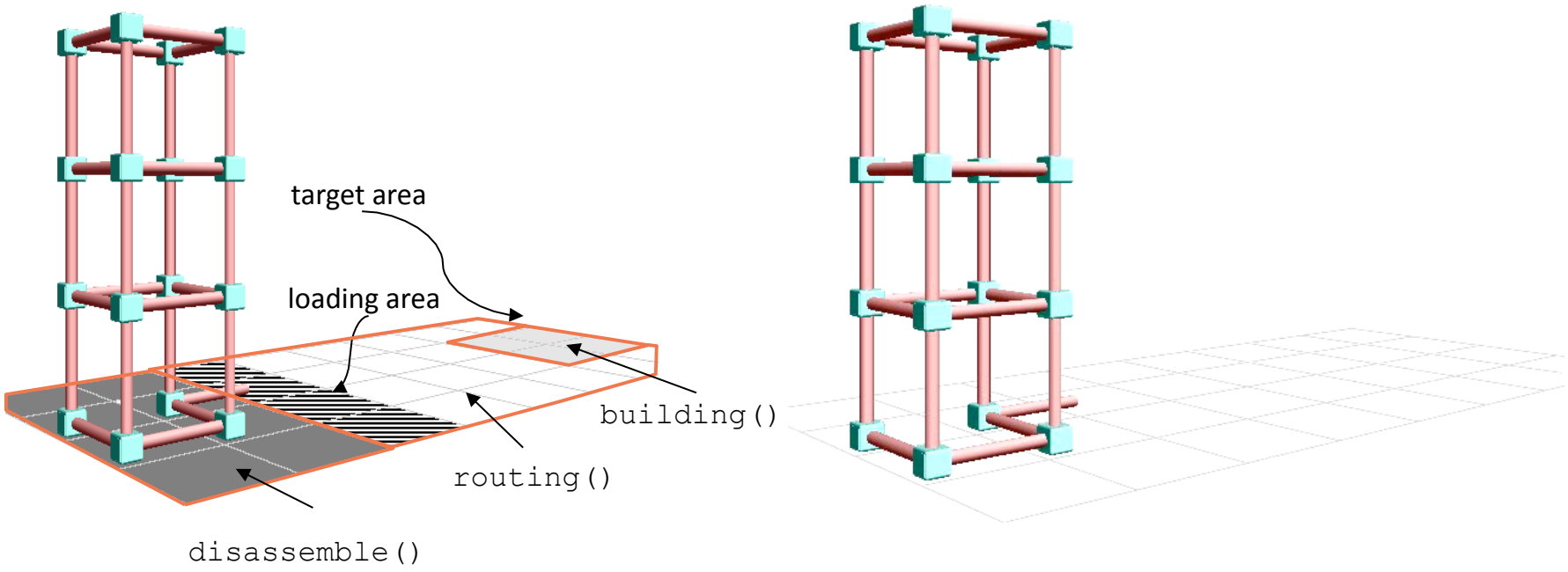
```
(rate k)&(checkFilled THIS NODE ) & (checkEmpty NORTH NODE) : {
     moveNode NORTH
};
```



```
(rate k)&(checkFilled THIS TRUSS_X ) & (checkEmpty THIS TRUSS_Y) : {
     moveTruss TRUSS_X THIS  TRUSS_Y
};
```

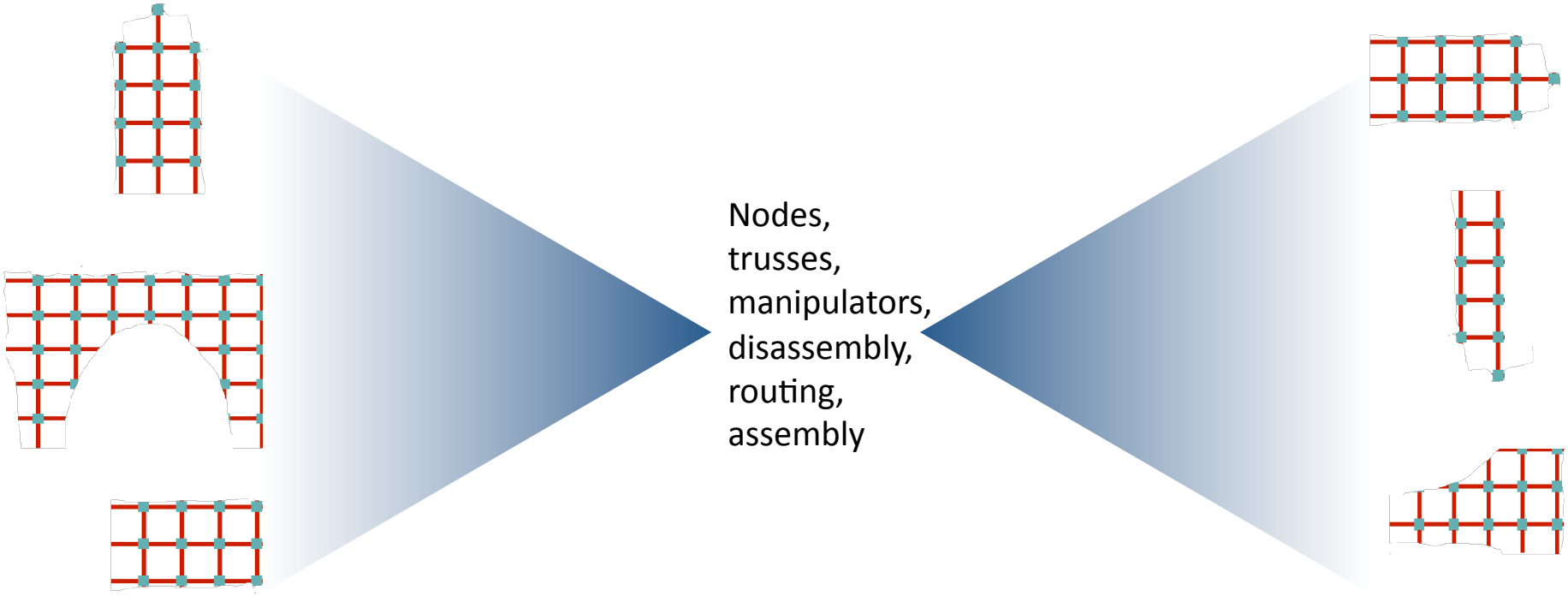
```
(rate k)&(checkFilled THIS TRUSS_X ) & (checkEmpty NORTH TRUSS_X) : {
     moveTruss TRUSS_X NORTH TRUSS_X
};
```

Reconfiguration Program

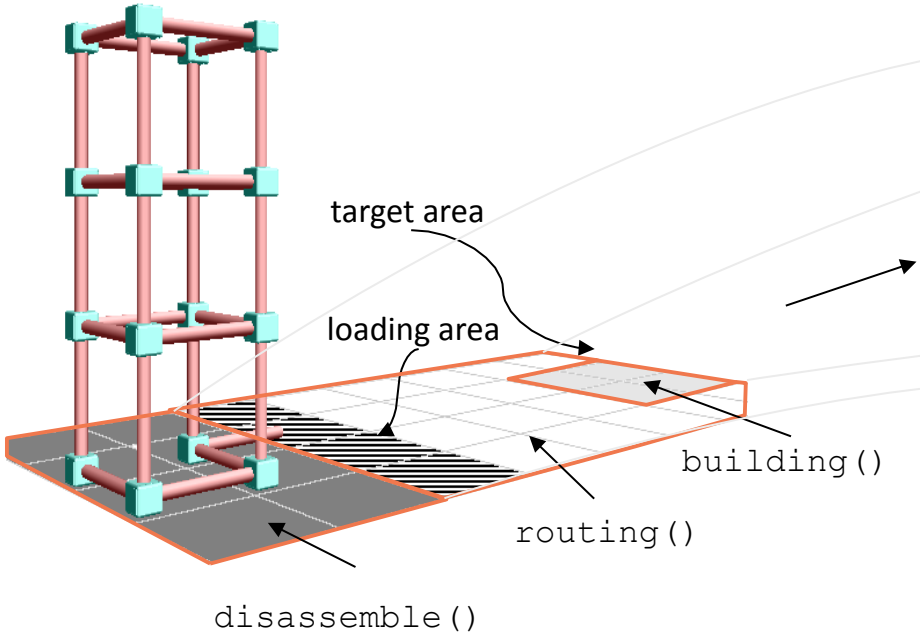


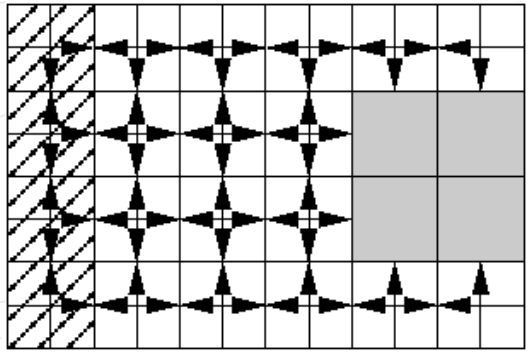

```
main():= disassemble() + routing() + building();
```

Bowtie Architecture



Routing Program



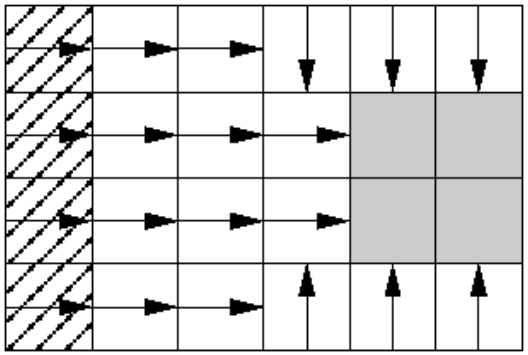


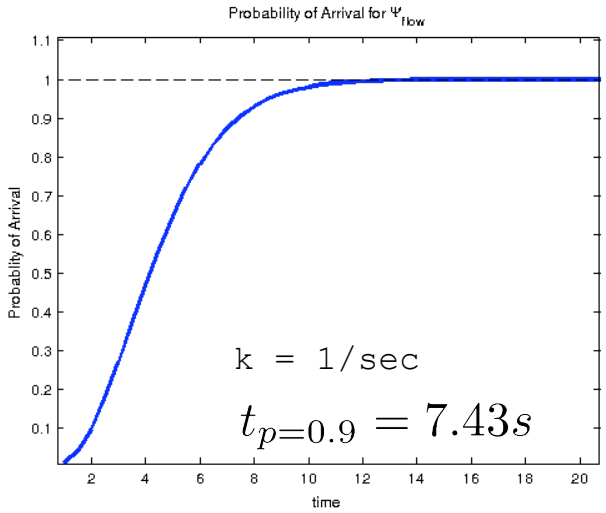
Quality of Routing Program:

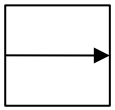
- Part appears in loading area at time 0
- Prob(target area at time t)

```
main():= disassemble() + routing() + building();
```

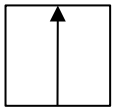
Flow Field: `flow(k)`



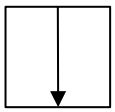




```
program east(k):= {
       (rate k ) & (checkFilled THIS NODE) & (checkEmpty EAST NODE):{
            moveNode EAST
       };};
```

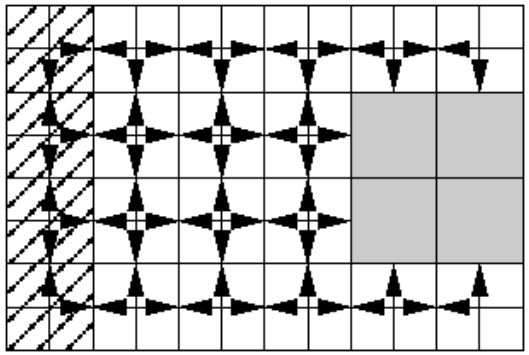


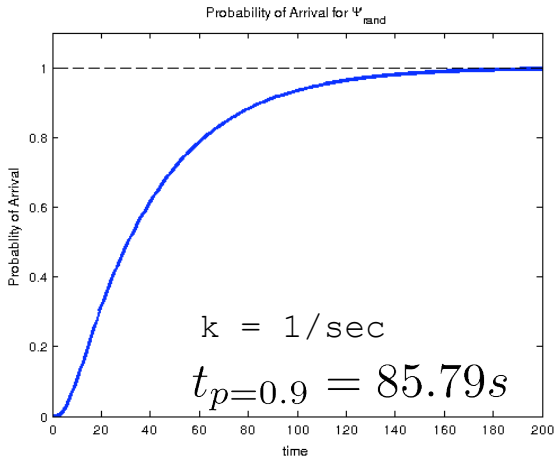
```
program north(k):= {
       (rate k ) & (checkFilled THIS NODE) & (checkEmpty NORTH NODE):{
            moveNode NORTH
       };};
```

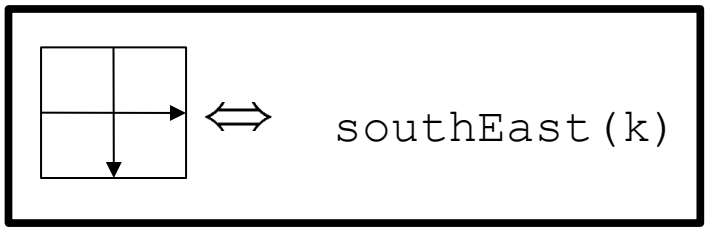


```
program south(k):= {
       (rate k ) & (checkFilled THIS NODE) & (checkEmpty SOUTH NODE):{
            moveNode SOUTH
       };};
```

Random Walk: `random(k)`



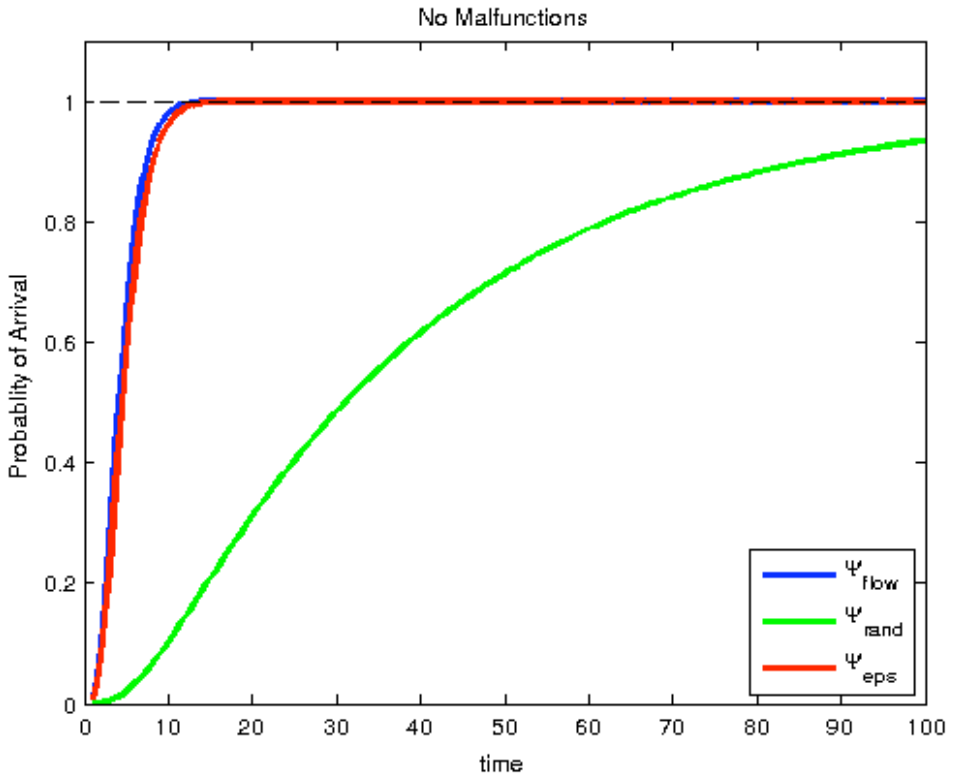





```
program southEast(k) := {
        (rate (k / 2)) & (checkFilled THIS NODE) & (checkEmpty EAST NODE):{
                moveNode EAST
        };
        (rate (k / 2)) & (checkFilled THIS NODE) & (checkEmpty SOUTH NODE):{
                moveNode SOUTH
        };
};
```

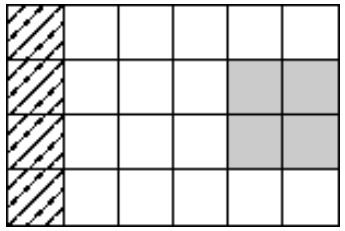
Robustifying Programs

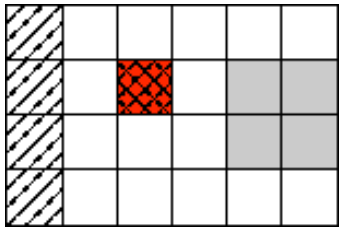
```
robust (k) := flow( 0.9*k )  + random (0.1 * k);
```

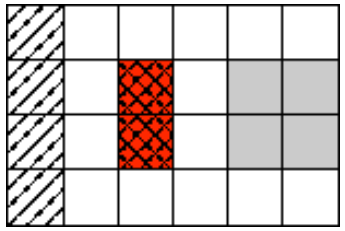


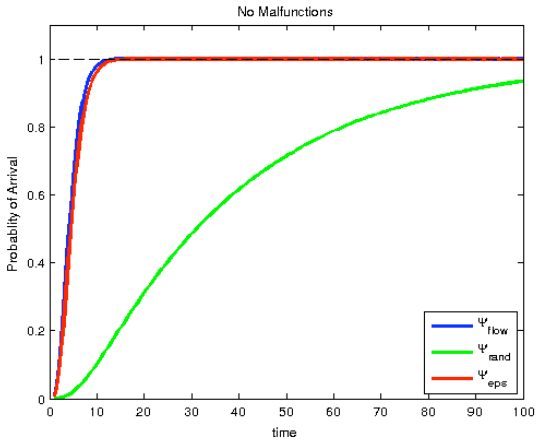
flow(1) random(1) robust(1)

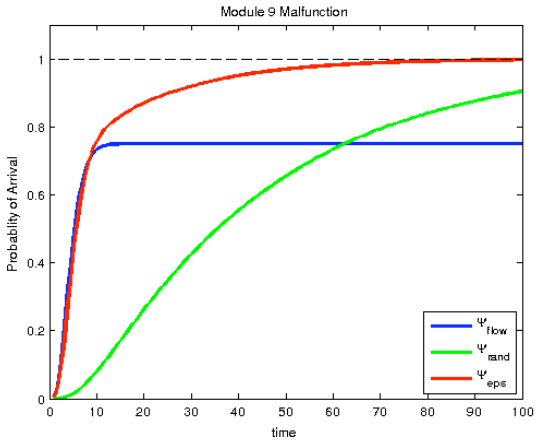
Robustness vs. Performance

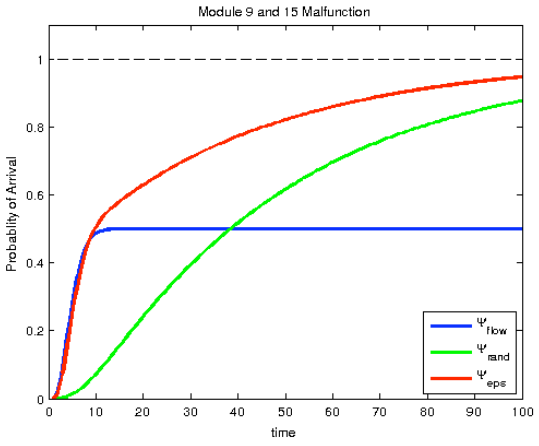












`flow(1)`

`random(1)`

`robust(1)`

Expressibility

Without Rates

$\Phi_1 \cup \Phi_1$: Arbitrary fair interleavings may result in incorrect behaviors

With Rates

$a\Phi_1 \cup b\Phi_2$: Can balance the relative rates at which programs run

The Control Problem

$\Phi_{\text{plant}} \cup u\Phi_{\text{control}}$: Find u(t) so that some desired performance statistics result

Robustification

$\Phi_1 \cup \varepsilon\Phi_2$: Add a small amount of P2 to make P1 more robust

Current Work

GCPWR

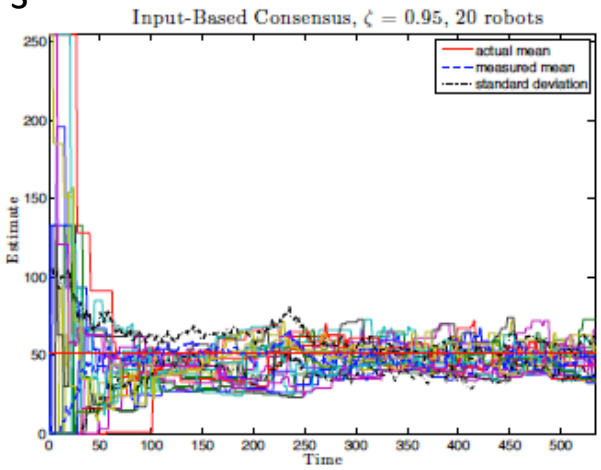
- Formalizing probabilistic failure models
- Synthesis of minimal robustification programs
- Automated reasoning about GCPWR

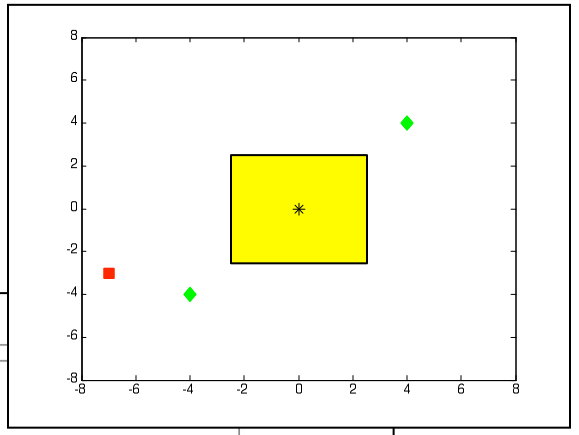
David Thorsley's Work

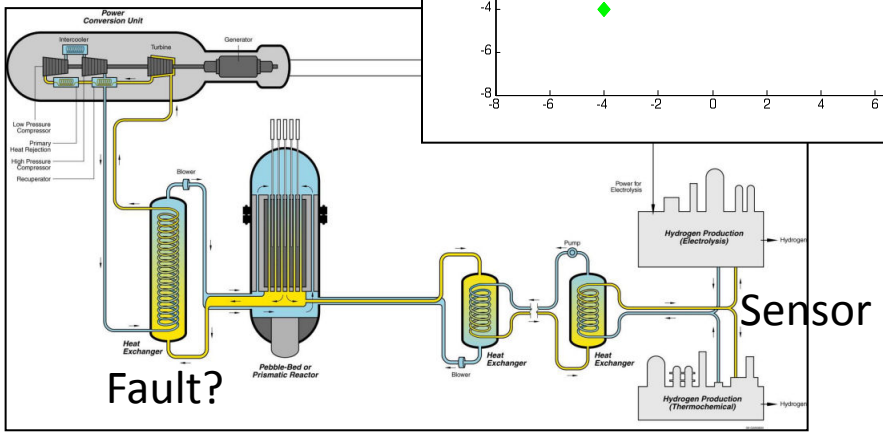
- Observation of internal state in guarded command programs
  - Diagnosis
  - Hypothesis Testing

Fayette Shaw's

- Distributed Task Allocation







Future Questions

- How can we use randomness to
    - improve system robustness?
    - make systems easier to verify?
    - yield safe, essentially deterministic behavior?
- How do we reason about stochastic software / control systems?
    - stochastic Lyapunov functions
    - local to global results for stochastic processes