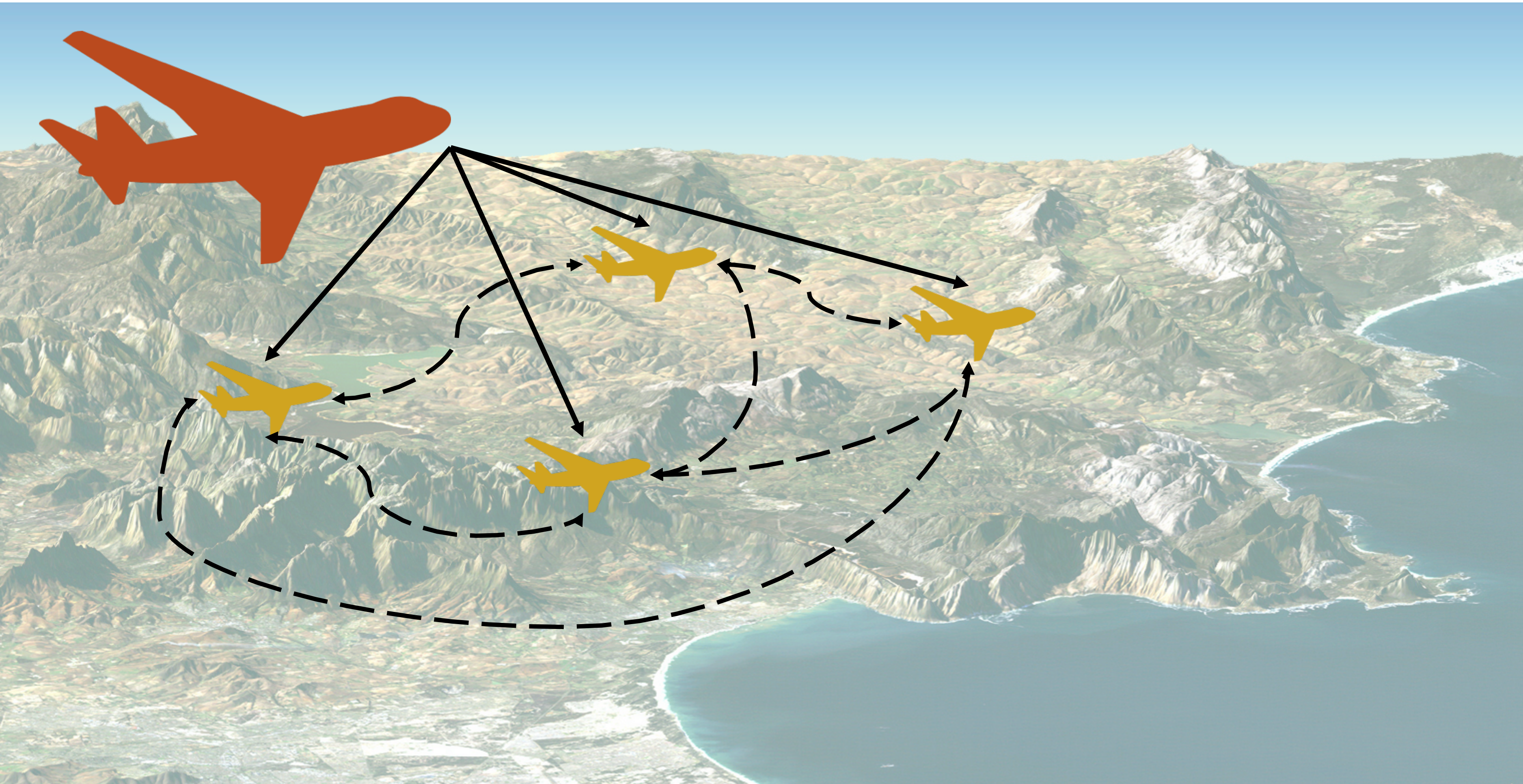
A DOMAIN-SPECIFIC LANGUAGE FOR REACTIVE CONTROL PROTOCOLS

OF AIRCRAFT ELECTRIC POWER SYSTEMS

Huan Xu | University of Maryland, College Park
Necmiye Ozay, Ufuk Topcu, Robert Rogersten, Richard M. Murray

May 8, 2014
HCSS Annapolis, MD

APPLICATION



- Deployable autonomous fleet
- Pilot sets high-level mission
- Goals automatically assigned
- Changing tasks, real-time coordination
- Account for environment/adversary

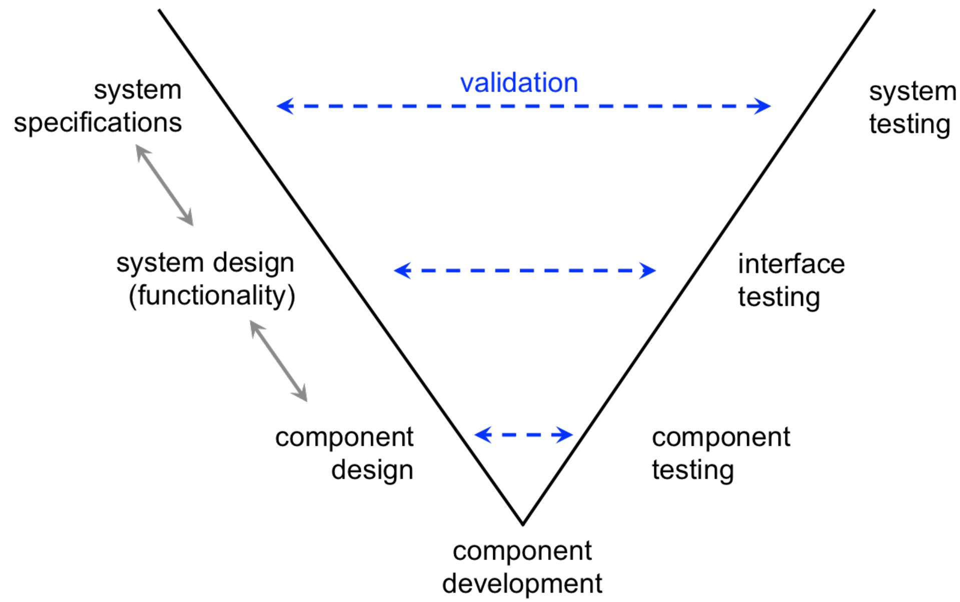
OBJECTIVES

How do we…

… design interconnected subsystems/agents working to achieve a common goal

… specify complex temporal tasks in a manner that allows rigorous proof of correctness

… synthesize a correct-by-construction control protocol that satisfies the specifications



Practical Impacts

- Design for certification/ verification
  - Problems discovered early
  - Less costly to fix
  - Faster development
  - Easier integration

OUTLINE

- *Application*
- *Objectives/Benefits*
- **Aircraft Electric Power System**
  - Problems
  - Reactive Control Synthesis
- **Domain-Specific Language**
  - Everything goes “under the hood”
- **Limitations**
- **Future Directions**

AIRCRAFT ELECTRIC POWER SYSTEM

Hydraulic, Pneumatic, Electric

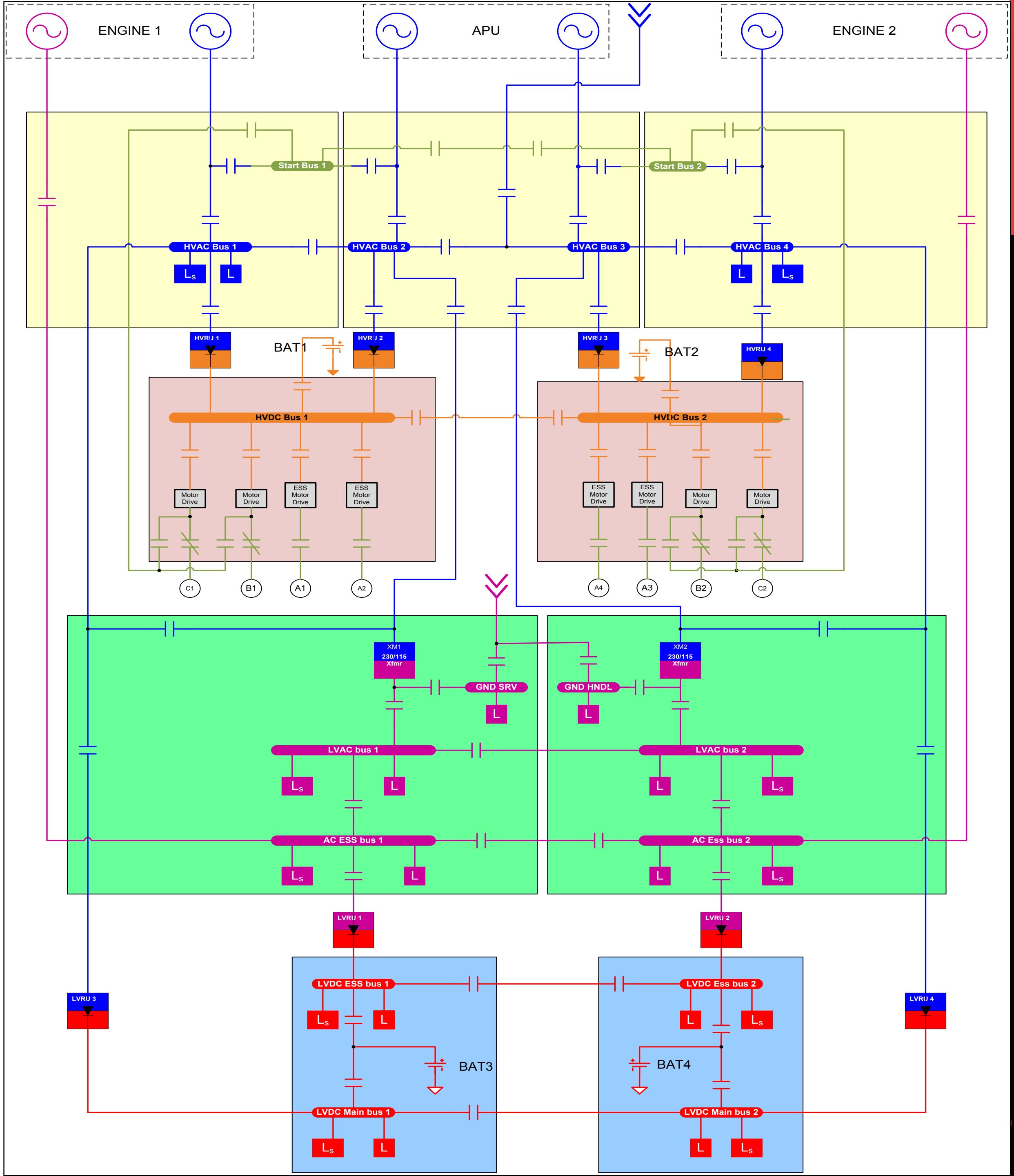
Fault-tolerant, reliable, autonomous

Systematic methods for design based

- formal specifications
- verification and validation of complex systems

Increasing complexity

- VMS systems designed for verification
- Need structure to allow verification tools to be applied
- Synthesizing “correct-by-construction” design protocols



AIRCRAFT ELECTRIC POWER SYSTEM

WHAT ARE SOME CONTROL/ LOGIC SYNTHESIS PROBLEMS?

Generation: Continuous controller to regulate the output voltage around a nominal value.

Distribution: Logic to reroute the power according to flight phases or fault conditions.

Load management: Logic to shed unimportant loads when failures in generation.

Fault detection: Logic to detect faults based on sensor measurements.

Cockpit interaction: Logic to coordinate controllers to accommodate pilot requests.

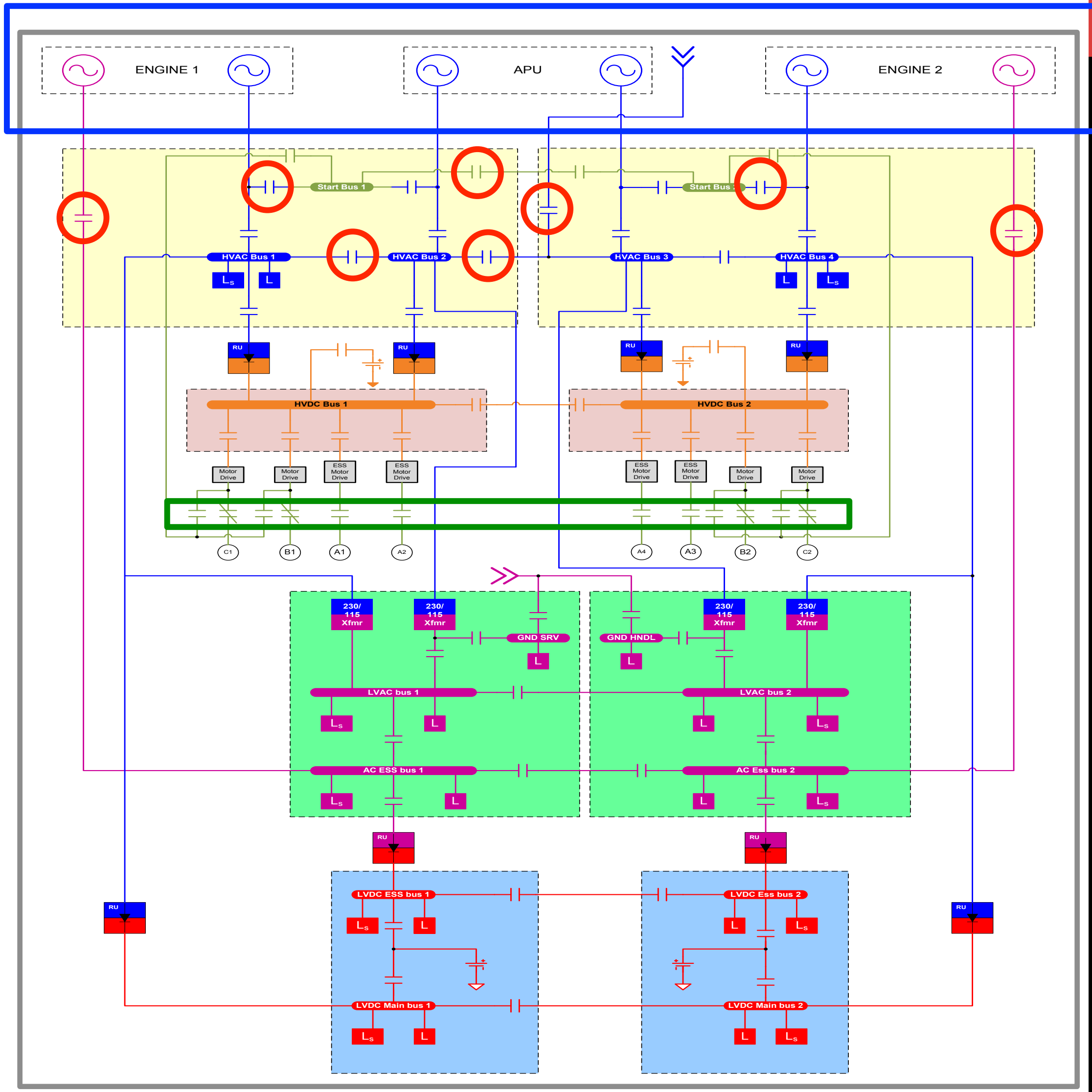
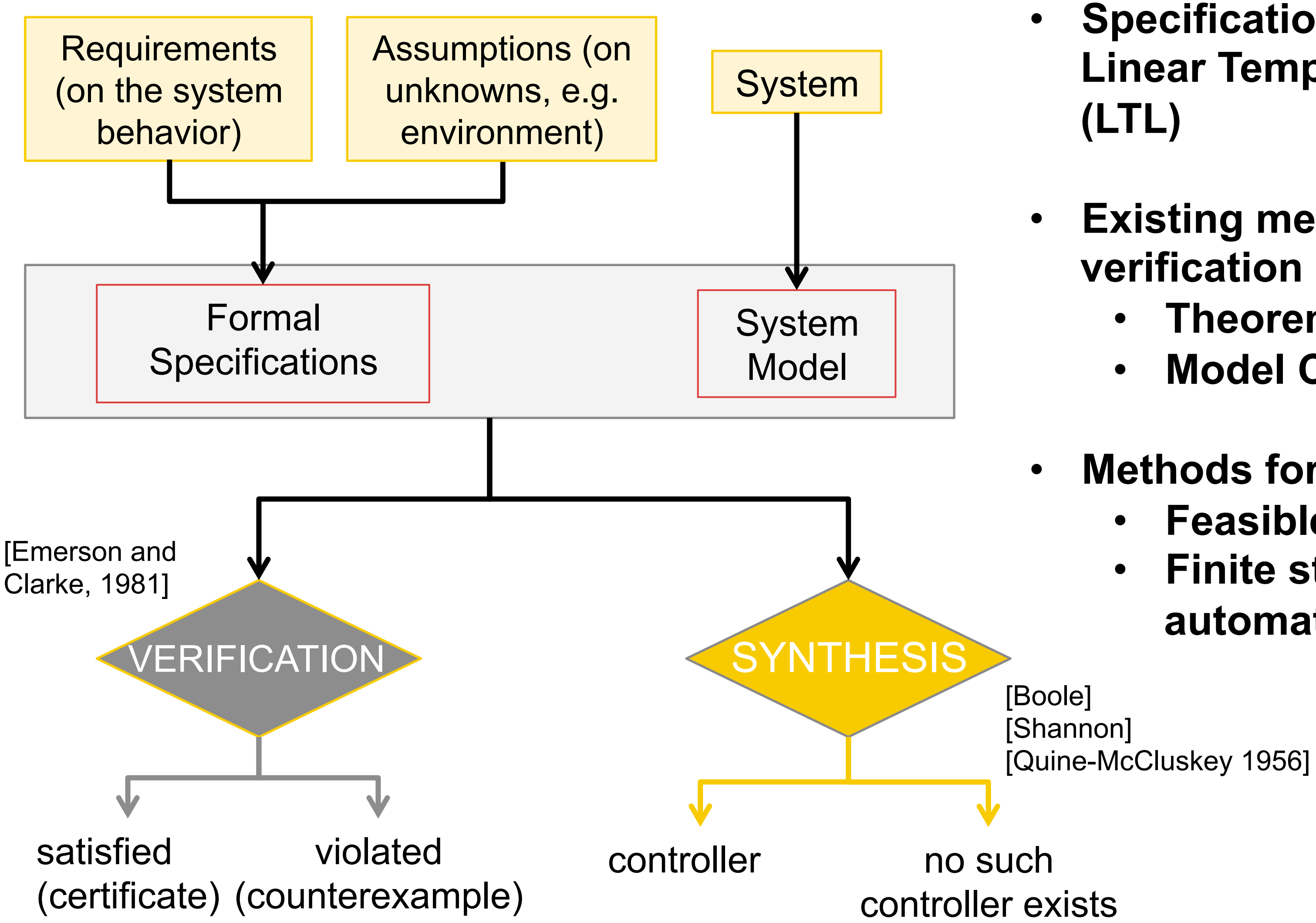


Figure courtesy of Rich Poisson, UTAS. Adapted from Honeywell Patent US 7,439,634 B2

FORMAL METHODS FOR VERIFICATION AND SYNTHESIS

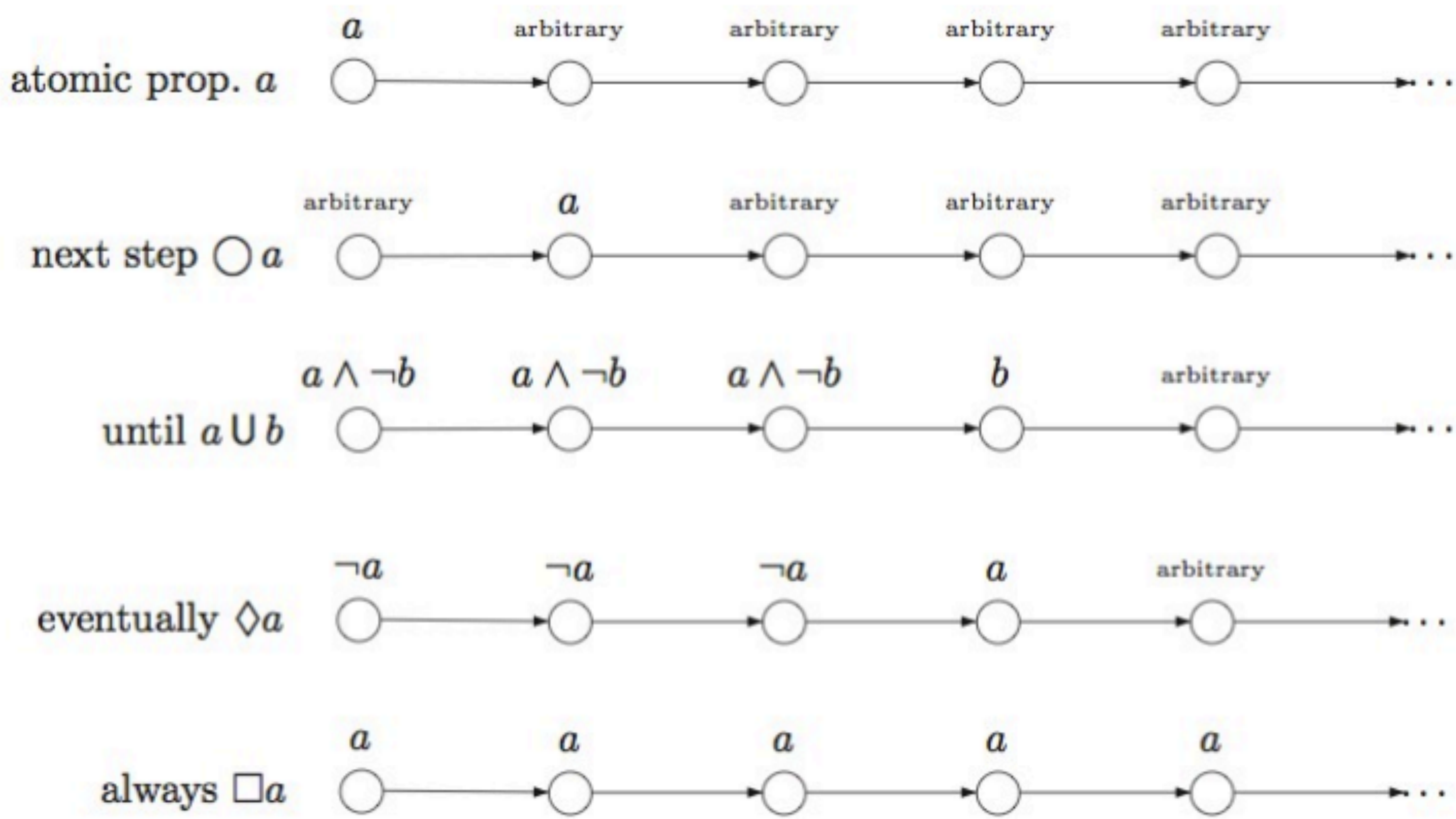


- **Specification using Linear Temporal Logic (LTL)**
- **Existing methods for verification**
  - **Theorem Proving**
  - **Model Checking**
- **Methods for Synthesis**
  - **Feasible paths**
  - **Finite state automata**

TEMPORAL LOGIC

- **“Temporal” refers to underlying nature of time** (A. Prior, 1950s)
  - Linear
  - Branching
- **Two key operators**
  - <> eventually – property satisfied at some point in future
  - [] always – property satisfied now and forever in future
- **Linear Temporal Logic (LTL)**
  - Introduced in 1970s (A. Pnueli)
  - Large collection of tools for specification, design, analysis
- **Other temporal logics**
  - CTL – Computation Tree Logic
  - TCTL – Timed CTL
  - MTL – Metric Temporal Logic (timed LTL)
  - TLA – temporal logic of actions (Leslie Lamport)
  - μ-calculus – “least fixed point” operator

LINEAR TEMPORAL LOGIC



GENERAL PROBLEM DESCRIPTION

Given a system model and LTL specification ϕ, design a controller to ensure that any system execution will satisfy ϕ.

$$\begin{aligned} s(t+1) &= As(t) + Bu(t) + Ed(t)) \\ u(t) &\in U \\ d(t) &\in D \end{aligned}$$

$$s \in \mathbb{R}^n, U \subseteq \mathbb{R}^m, D \subseteq \mathbb{R}^p$$

$$\varphi = \Big(\underbrace{\psi^e_{init}}_{\text{assumptions on initial condition}} \wedge \underbrace{\Box\psi^e_s \wedge \bigwedge_{i \in I_f} \Box\Diamond\psi^e_{f,i}}_{\text{assumptions on environment}} \Big) \implies \Big(\underbrace{\psi^s_{init} \wedge \Box\psi^s_s \wedge \bigwedge_{i \in I_g} \Box\Diamond\psi^s_{g,i}}_{\text{desired behavior}} \Big)$$

assume-guarantee

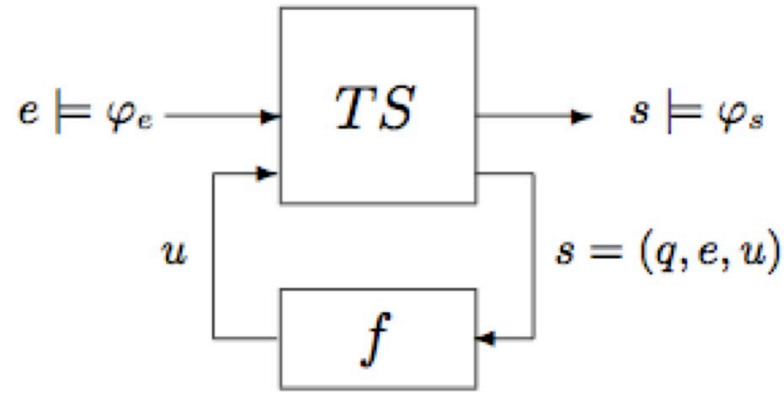
REACTIVE (OPEN) SYNTHESIS

Given:

Open transition system

$$TS = (Q, I, \mathcal{A}_{uc}, \mathcal{A}_c, R_{nom})$$

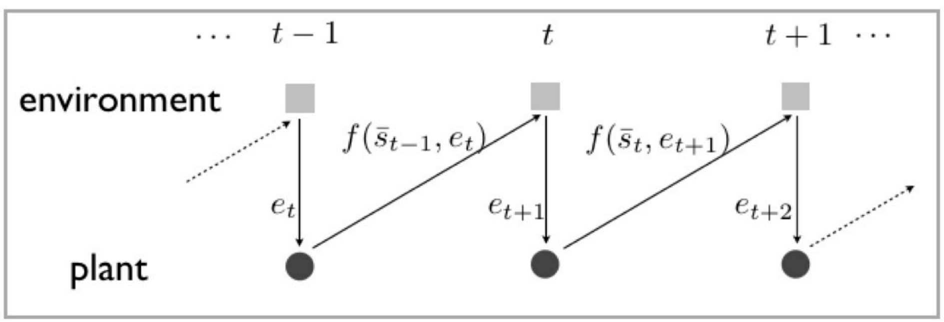
- Q finite set of states,
- $I \subseteq Q$ set of initial states,
- $\mathcal{A}_{uc}$ set of uncontrollable input actions
- $\mathcal{A}_c$ set of controllable input actions
- $R_{nom} \subseteq Q \times \mathcal{A} \times Q$ transition relation



Assume-guarantee type temporal logic specification

$$\varphi = \varphi_e \rightarrow \varphi_s$$

Compute: A strategy $f : (q_0, e_0, u_0, \cdots, q_{i-1}, e_{i-1}, u_{i-1}, q_i, e_i) \mapsto u_i$ with $(q_i, e_i, u_i, q_{i+1}) \in R_{nom}, \ \forall i \geq 0$ such that any controlled execution satisfies the specification.



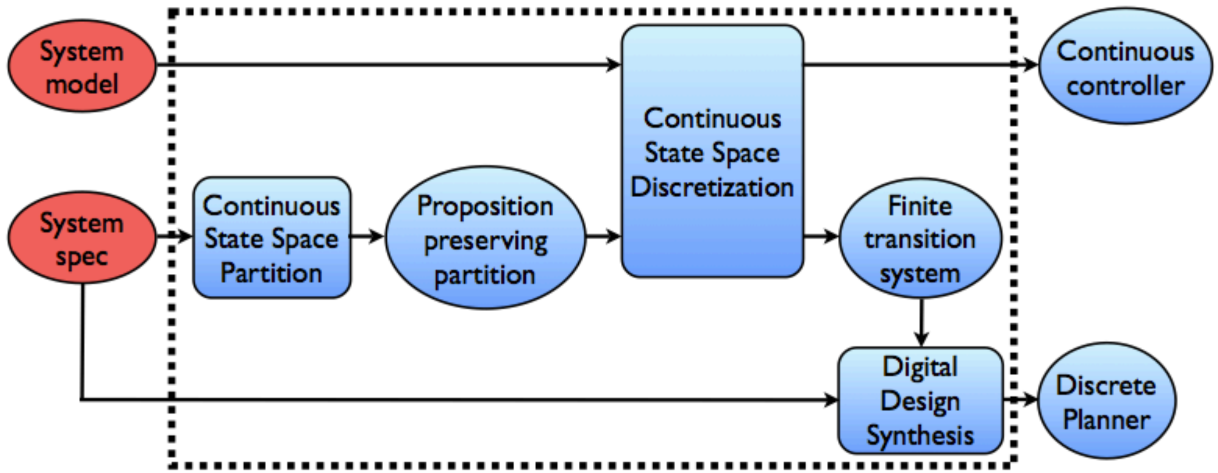
TEMPORAL LOGIC PLANNING TOOLBOX (TULIP)

[Wongpiromsarn, et al. HSCC2011]

http://tulip-control.sourceforge.net

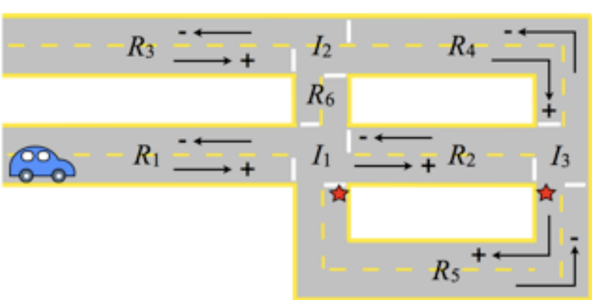
Python Toolbox

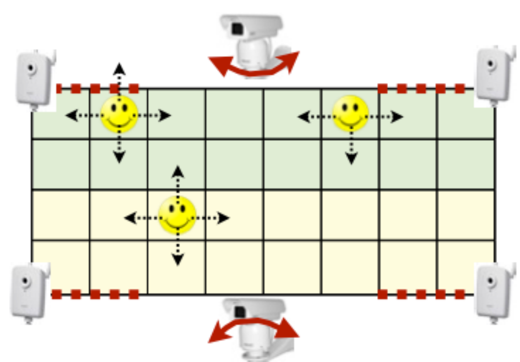
- GR(1), LTL specs
- Nonlinear dynamics
- Supports discretization via MPT
- Control protocol designed using JTLV
- Receding horizon compatible

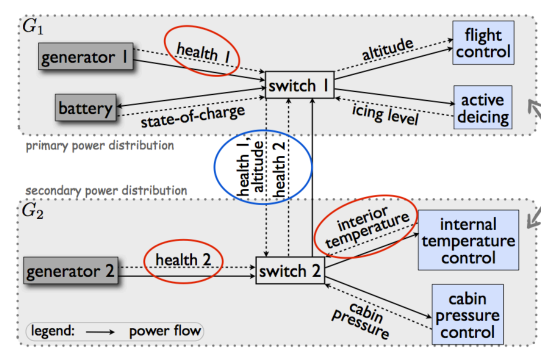


Past Applications of TuLiP

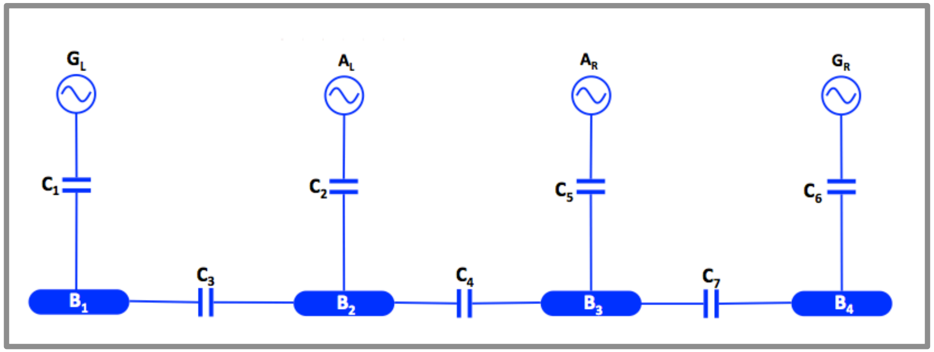
- Autonomous vehicles - traffic planner (intersections and roads, with other vehicles)
- Distributed camera networks - cooperating cameras to track people in region
- Electric power transfer - fault-tolerant control of generator + switches + loads







PROBLEM FORMULATION

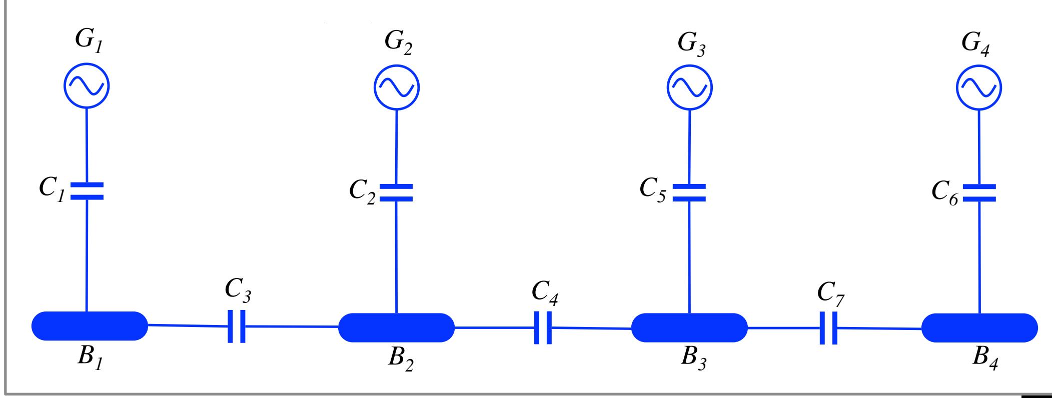


1. No AC bus shall be simultaneously powered by more than one AC source.
2. The aircraft electric power system shall provide power with the following characteristics: 115 +/- 5 V (amplitude) and 400 Hz (frequency) for AC loads and 28 +/-2V for DC loads.
3. Buses shall be according to the priority tables.
4. AC buses shall not be unpowered for more than 50ms.
5. The failure probability must be less than 10^{-9} for the duration of a mission.

Given a candidate topology and text-based requirements, build *a controller* that would reconfigure the system (via closing and opening contactors) by *sensing* and *reacting to* the faults and changes in system status in a way to ensure that the requirements are met.

[Xu, Topcu, Murray, TCNS2014, *UR*]

SPECIFICATIONS



Graph $G = (V,E)$

- $V = \{v_1, \ldots, v_n\}$ (generators, buses)
- $E = \{c_1, \ldots, c_m\}$ (contactors)

Adjacency Matrix A_{ij}

Environment Variables G_1-G_4

System Variables C_1-C_7, B_1-B_4

Environment Assumption φ_e

$$\square\{\bigvee_{i=1}^{4} G_i = 1\}$$

System Model (Live Paths) φ_s

$$\square\{((G_1 = 1) \wedge (C_1 = 1)) \rightarrow (B_1 = 1)\}$$

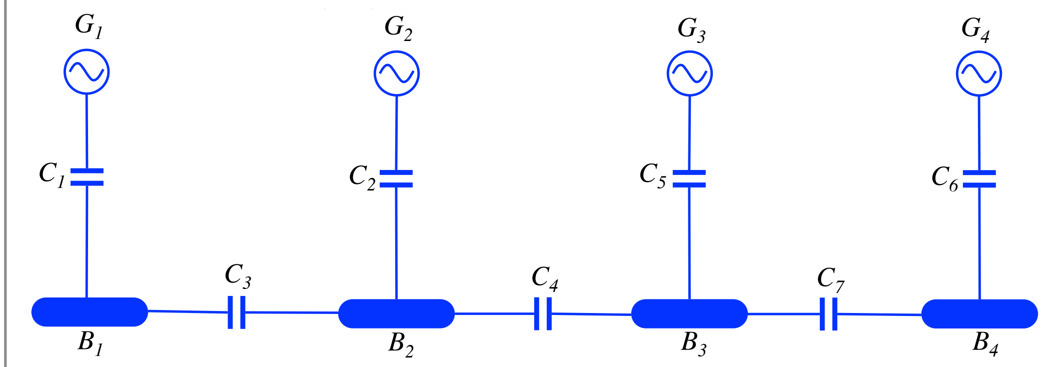
$$\square\{((G_2 = 1) \wedge (C_2 = 1) \wedge (B_2 = 1) \wedge (C_3 = 1)) \rightarrow (B_1 = 1)\}$$

$$\square\{((G_3 = 1) \wedge (C_5 = 1) \wedge (B_3 = 1) \wedge (C_4 = 1) \wedge (B_2 = 1) \wedge (C_3 = 1)) \rightarrow (B_1 = 1)\}$$

…

$$\square\{\neg(live\ path) \rightarrow (B_1 = 0)\}$$

SPECIFICATIONS



No paralleling AC sources φ_s

$$\forall\ G_i, G_j\ \square\neg\{\bigwedge_{i \in paths(i,j)} C_i = 0\}$$

Disconnect unhealthy generators φ_s

Intent $\tilde{C}$

$$\square\{\bigwedge_{C_j \in edges(G)} (G_i = 0) \rightarrow (\tilde{C}_j = 0)\}$$

Essential buses never unpowered for more than X time φ_s

$$\square\{(B_i = 0) \rightarrow (\bigcirc x_{B_i} = x_{B_i} + \delta)\}$$

$$\square\{(B_i = 1) \rightarrow (\bigcirc x_{B_i} = 0\}$$

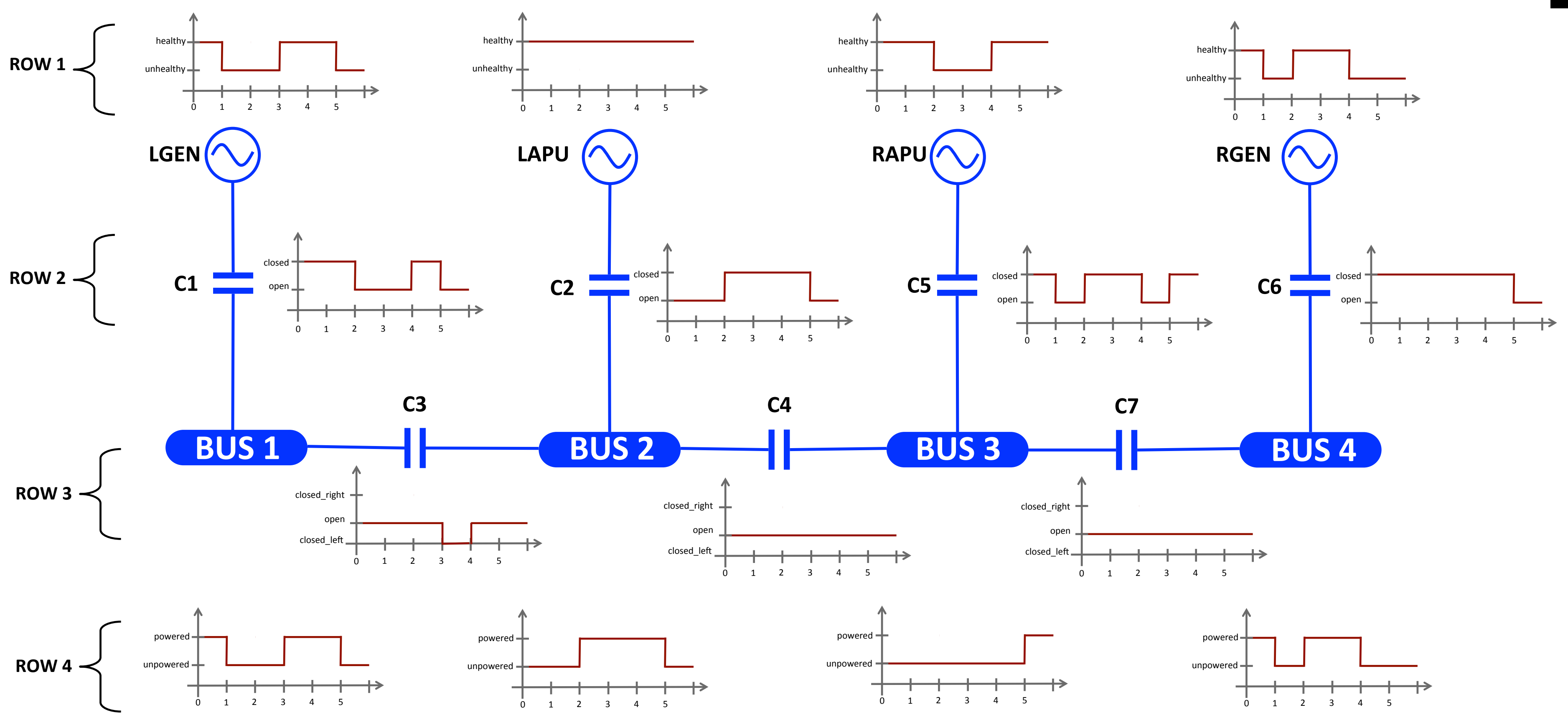
$$\square\{x_{B_i} \leq X\}$$

SYNTHESIS RESULTS

Formal Spec in LTL

$$\varphi_e \rightarrow \varphi_s$$

For one simulation trace…



DOMAIN-SPECIFIC LANGUAGES

General Purpose Language
- C
- Java
- Python
- UML

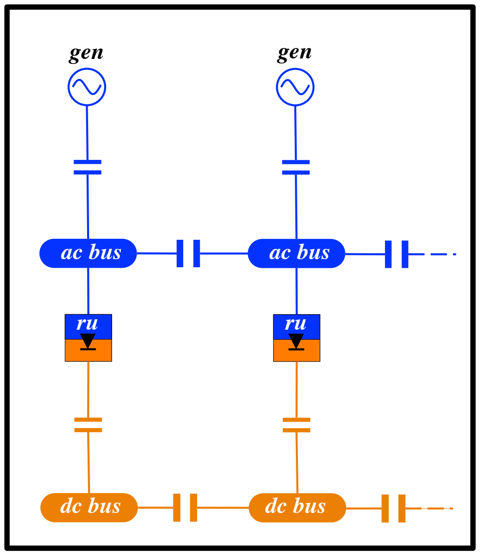
Domain-Specific Language
- HTML
- GraphViz
- Mathematica
- SQL

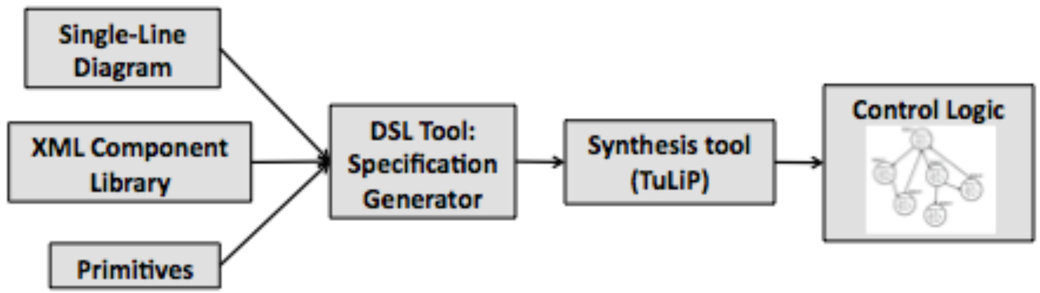
- **Text-based specifications are ambiguous**
- **Formal languages**
  - Difficult to learn
  - Tedious to write

CPS
[An et al. 2011]
[Bhave et al. 2011]

PRIMITIVES

- **Single-line diagrams and synthesis tools don't "speak" the same language**
- **Idea: Use primitives to represent requirements**






```
<contactor>
      <failure>
         1e-3
      </failure>
      <opentime>
         15
      </opentime>
      <closetime>
         20
      </closetime>
</contactor>
```

```
disc_props['B530'] = '(g3=1) & (c35=1)'
disc_props['B800'] = '(b9=1) & (ru7=1) & (b5=1) & (b4=1) & (g0=1) & (c89=1) & (c57=1) & (c45=1) & (c04=1)'
disc_props['B801'] = '(ru6=1) & (b4=1) & (g0=1) & (c46=1) & (c04=1)'
disc_props['B810'] = '(b9=1) & (ru7=1) & (b5=1) & (b4=1) & (g1=1) & (c89=1) & (c57=1) & (c45=1) & (c14=1)'
disc_props['B811'] = '(ru6=1) & (b4=1) & (g1=1) & (c46=1) & (c14=1)'
disc_props['B820'] = '(b9=1) & (ru7=1) & (b5=1) & (g2=1) & (c89=1) & (c57=1) & (c25=1)'
disc_props['B821'] = '(b9=1) & (ru7=1) & (b5=1) & (b4=1) & (g2=1) & (c89=1) & (c57=1) & (c45=1) & (c24=1)'
disc_props['B822'] = '(ru6=1) & (b4=1) & (g2=1) & (c46=1) & (c24=1)'
disc_props['B823'] = '(ru6=1) & (b4=1) & (b5=1) & (g2=1) & (c46=1) & (c45=1) & (c25=1)'
disc_props['B830'] = '(b9=1) & (ru7=1) & (b5=1) & (g3=1) & (c89=1) & (c57=1) & (c35=1)'
disc_props['B831'] = '(ru6=1) & (b4=1) & (b5=1) & (g3=1) & (c46=1) & (c45=1) & (c35=1)'
disc_props['B900'] = '(b8=1) & (ru6=1) & (b4=1) & (g0=1) & (c89=1) & (c46=1) & (c04=1)'
disc_props['B901'] = '(ru7=1) & (b5=1) & (b4=1) & (g0=1) & (c57=1) & (c45=1) & (c04=1)'
disc_props['B910'] = '(b8=1) & (ru6=1) & (b4=1) & (g1=1) & (c89=1) & (c46=1) & (c14=1)'
disc_props['B911'] = '(ru7=1) & (b5=1) & (b4=1) & (g1=1) & (c57=1) & (c45=1) & (c14=1)'
disc_props['B920'] = '(b8=1) & (ru6=1) & (b4=1) & (g2=1) & (c89=1) & (c46=1) & (c24=1)'
disc_props['B921'] = '(b8=1) & (ru6=1) & (b4=1) & (b5=1) & (g2=1) & (c89=1) & (c46=1) & (c45=1) & (c25=1)'
disc_props['B922'] = '(ru7=1) & (b5=1) & (g2=1) & (c57=1) & (c25=1)'
disc_props['B923'] = '(ru7=1) & (b5=1) & (b4=1) & (g2=1) & (c57=1) & (c45=1) & (c24=1)'
disc_props['B930'] = '(b8=1) & (ru6=1) & (b4=1) & (b5=1) & (g3=1) & (c89=1) & (c46=1) & (c45=1) & (c35=1)'
disc_props['B931'] = '(ru7=1) & (b5=1) & (g3=1) & (c57=1) & (c35=1)'
assumptions += '&\n\t□((g0 + g1 + g2 + g3) >= 1)'
assumptions += '&\n\t□((ru6 + ru7) >= -1)'
```

[Xu, Ozay, Murray INCOSE *in prep*]

APPROACH

1. Input: Topology

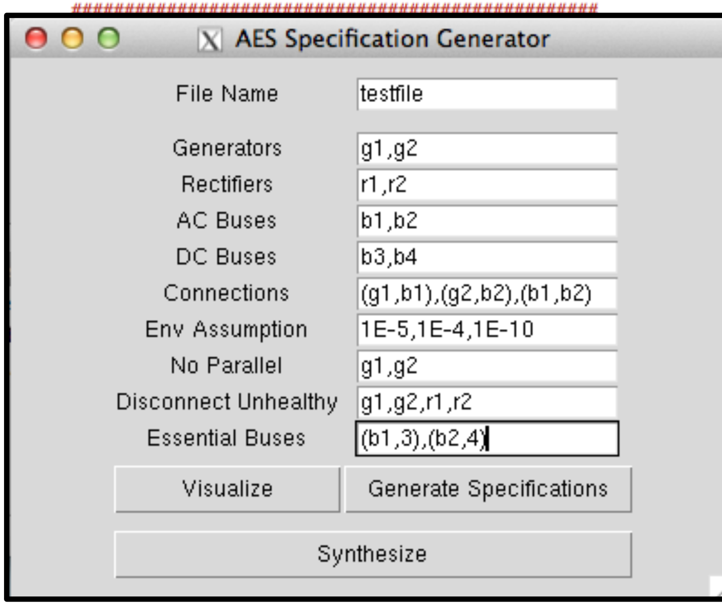
```
A = np.matrix([[0, 0, 0, 1, 0, 0, 0, 0, 0],
     [0, 0, 0, 1, 1, 0, 0, 0, 0],
     [0, 0, 0, 0, 1, 0, 0, 0, 0],
     [1, 1, 0, 0, 1, 1, 0, 0, 0],
     [0, 1, 1, 1, 0, 0, 1, 0, 0],
     [0, 0, 0, 1, 0, 0, 0, 1, 0],
     [0, 0, 0, 0, 1, 0, 0, 0, 1],
     [0, 0, 0, 0, 0, 1, 0, 0, 1],
     [0, 0, 0, 0, 0, 0, 1, 1, 0]])

#Node definitions
busac = [3,4]
busess = [3,4]
busdc = [7,8]
null = []
rus = [5,6]
gens = [0,1,2]

#Failure Probabilities
genfail = 3 #10^-x, where x=3
rufail = 3
safety = 6
```

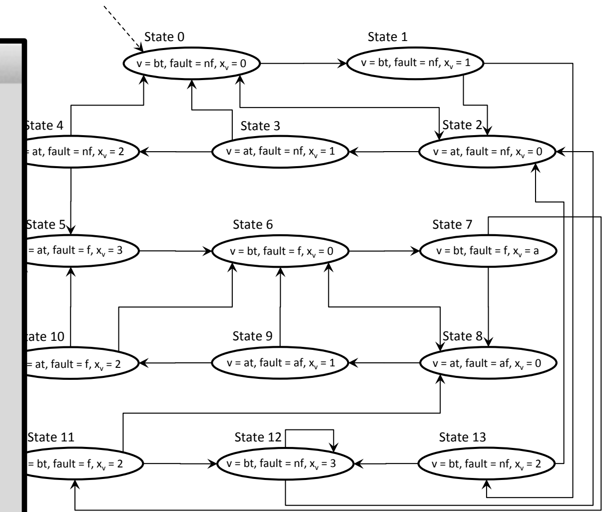
2. Input: Primitives

```
################################################
# Declare System Variables
################################################
```




```
dc_power(AES)
dcbusalways(busdc, AES)
```

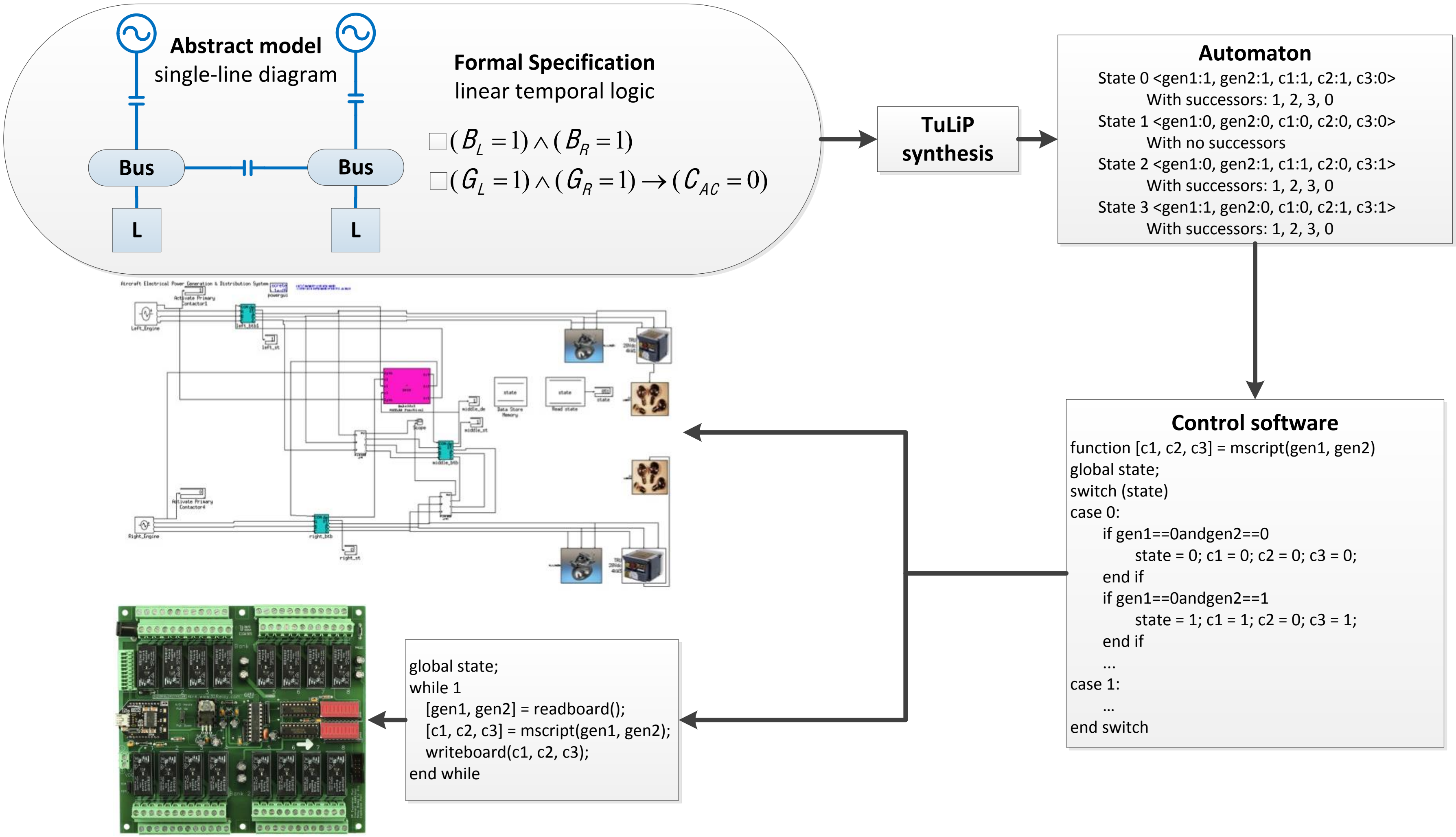
3. Output: Controller



Interfaces with SAT solver and TuLiP

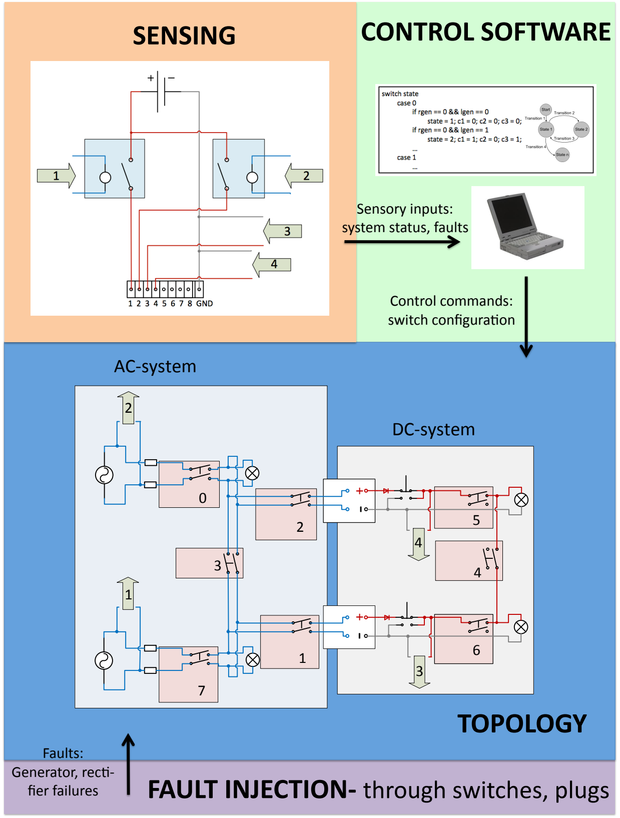
Interfaces to Simulink

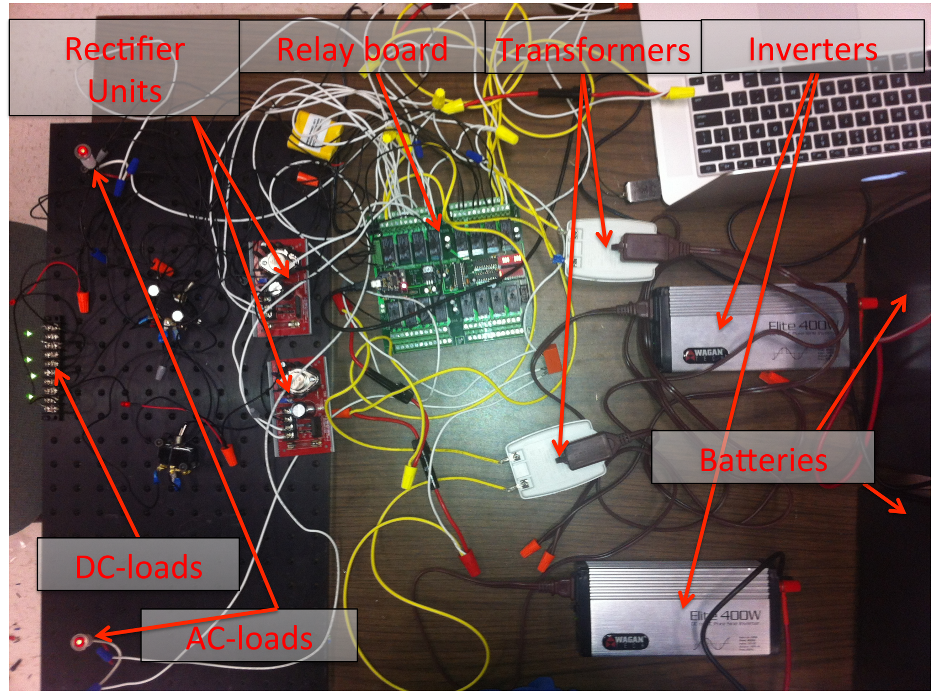
HARDWARE TESTBED



[Rogersten, Xu, Ozay, Topcu, Murray JAIS2014]

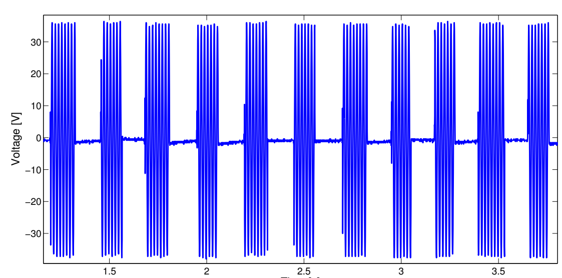
Detailed Model



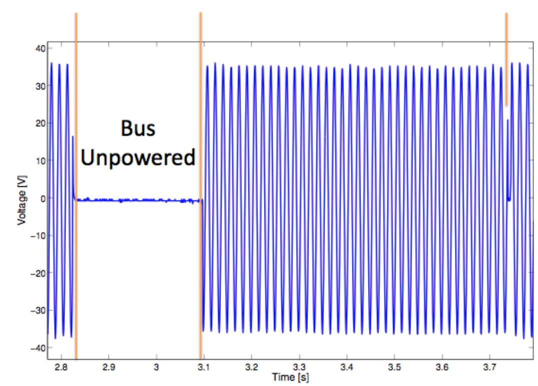


Timing characterization of the system

1 Relay	Unpowered time/ Close time [ms]	Powered time/ Open time [ms]
Mean	27	18.1
Max	28	19.4
Min	25.8	16.3
2 Relays	Unpowered time/ Close time [ms]	Powered time/ Open time [ms]
Mean	116.4	130.5
Max	130.9	148.6
Min	102.6	116.7



Fault - Controller reacts - Generator back on



LIMITATIONS/SOLUTIONS

- Full synthesis – double exponential
- GR(1) synthesis – polynomial
- State space – scales exponentially with clocks

- Solve synthesis problem
  - Untimed
    - SAT solver
  - Distributed
    - Decompose into smaller systems
    - Counter-strategy guided refinement

No. of Clocks	Clock "Ticks"	Aut. Size	Time [sec]
1	1	32	1.5
1	3	64	1.7
1	5	96	1.7
1	10	176	2.8
1	20	336	3.1
2	1	79	2
2	3	96	2
2	5	224	2.1
2	10	384	2.5
2	20	704	2.5
3	1	478	3.5
3	3	2858	7
3	5	7180	160
3	10	45492	1084
3	20	88604	4796
4	1	1798	7.2
4	3	22008	308
4	5	93386	4778