Correct-by-Constructions Synthesis: Aircraft Electric Power System Challenge Problem

Mumu Xu, Necmiye Ozay, Richard M. Murray
Caltech CDS

Joint work: Pierluigi Nuzzo, Alberto Sangiovanni-Vincentelli (UCB), Ufuk Topcu (Penn), Robert Rogersten (KTH), Quentin Maillet (Mines ParisTech)

Thanks to: Rich Poisson of UTAS for feedback and discussions

iCyPhy WebEx Meeting
4 February, 2013

iCyPhy

Motivation

- Want to design sensing and control architectures with **safety**, **reliability** and **performance** <u>guarantees!</u>

- **Goal:** to efficiently engineer certifiable systems

- **Approach:** correct by construction controller synthesis:
  – design & verify vs. specify & synthesize

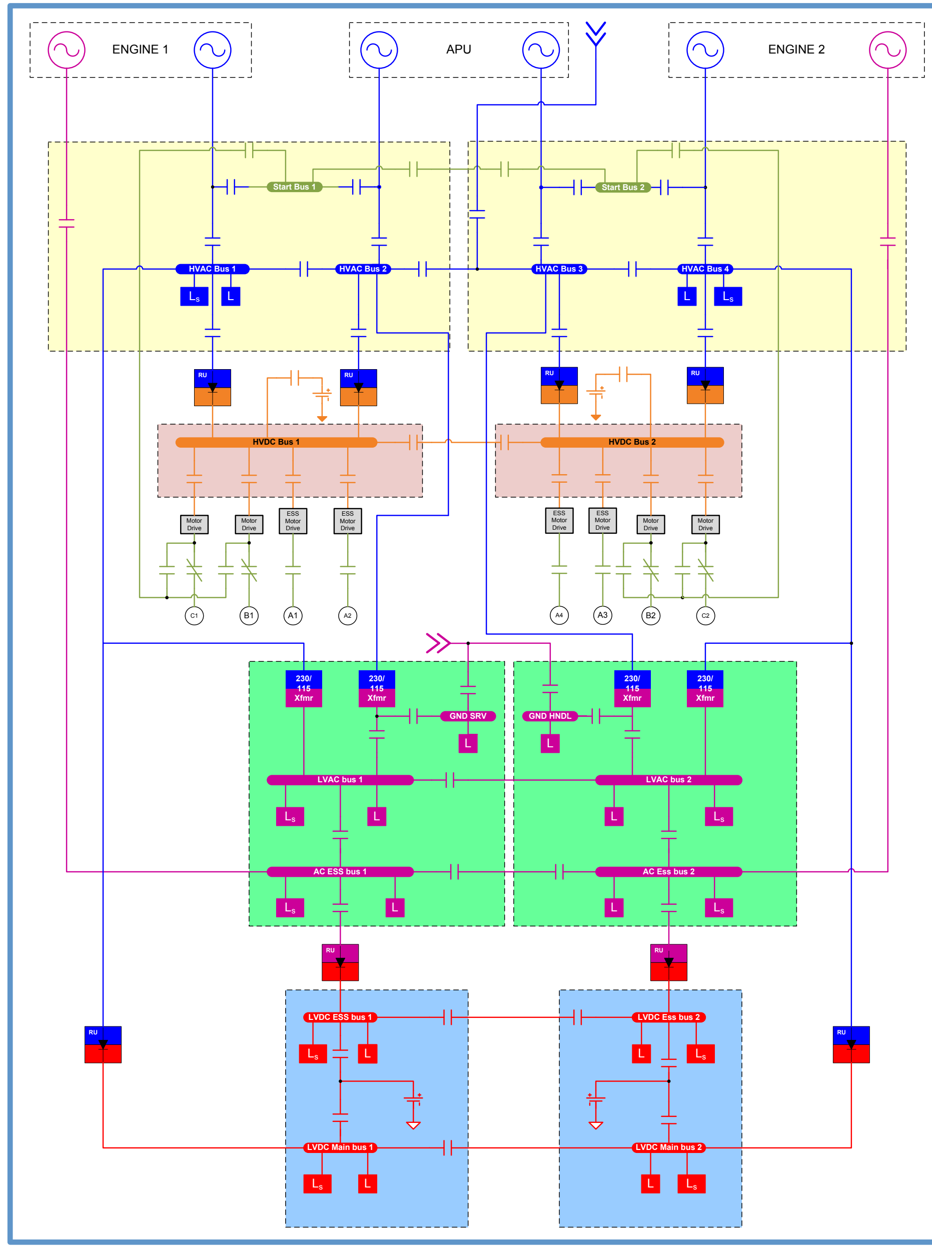


Figure courtesy of Rich Poisson, Hamilton-Sundstrand. Adapted from Honeywell Patent US 7,439,634 B2

Overall Design Flow

- Given text based specifications:
  - Formalize requirements and associate them with system entities (e.g. components)
  - Find a ``feasible” topology (design-space exploration, topology synthesis)
  - Given the topology and specifications, **synthesize control protocol** with correctness guarantees
  - Export the controller to high fidelity models for simulation and further tests
  - Implement on hardware

Problem Description

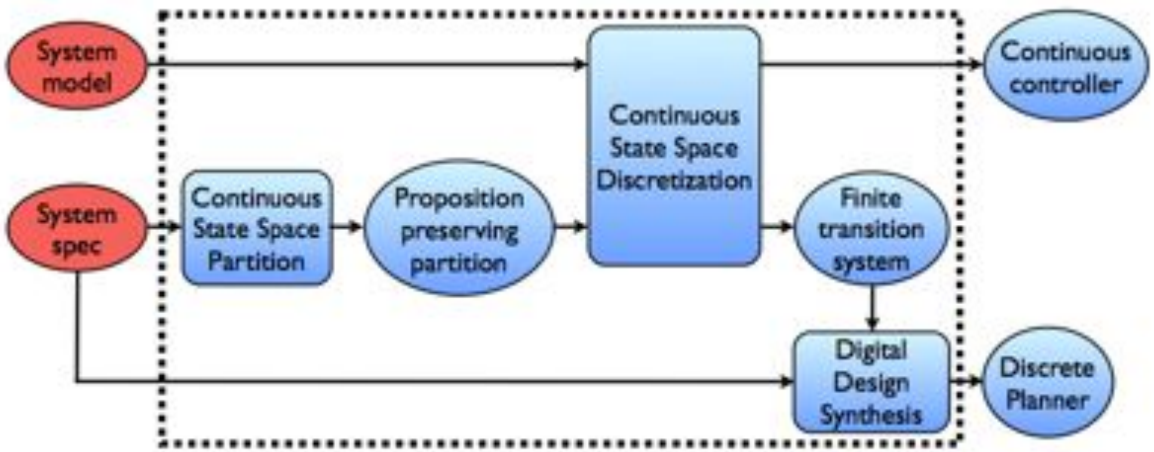
- Problem: Given a system model and LTL specification φ, design a controller to ensure that any system execution will satisfy φ.

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

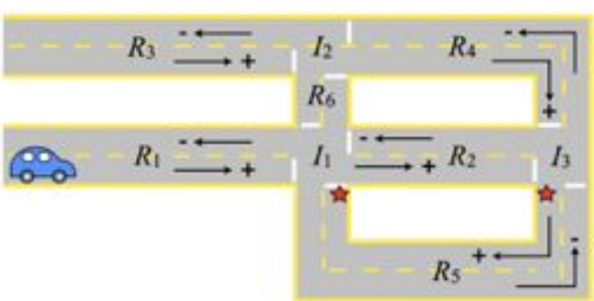
$$s \in \mathbb{R}^n, U \subseteq \mathbb{R}^m, D \subseteq \mathbb{R}^p$$

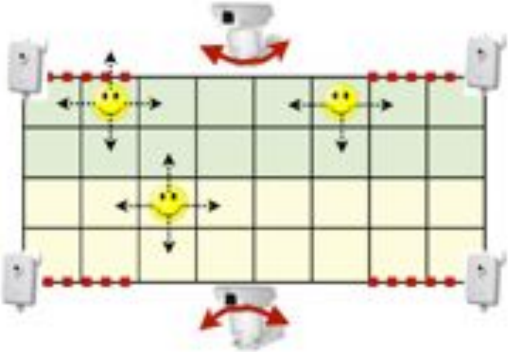
$$\varphi = \Big(\underbrace{\psi^e_{init}}_{\text{assumptions on initial condition}} \wedge \underbrace{\Box\psi^e_s \wedge \bigwedge_{i \in I_f} \Box\Diamond\psi^e_{f,i}}_{\text{assumptions on environment}} \Big) \implies \underbrace{\Big(\psi^s_{init} \wedge \Box\psi^s_s \wedge \bigwedge_{i \in I_g}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

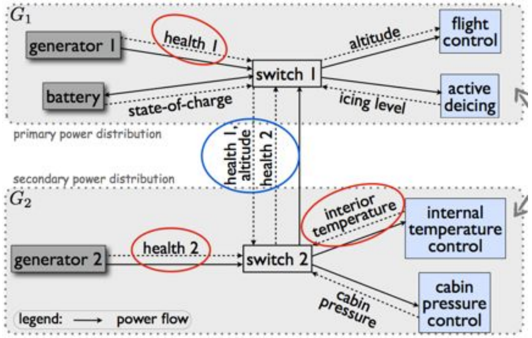
Temporal Logic Planning (TuLiP) Toolbox



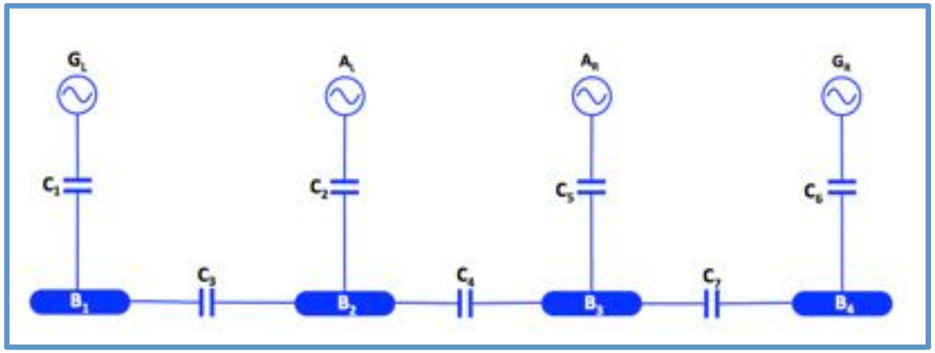
- Past Applications
  - Autonomous vehicles - traffic planner (intersections and roads, with other vehicles)
  - Distributed camera networks - cooperating cameras to track people in region
  - Electric power transfer - fault-tolerant control of generator + switches + loads







Control Synthesis Problem



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 turning on and off the contactors) by *sensing* and *reacting to* the faults and changes in system status in a way to ensure that the requirements are met.

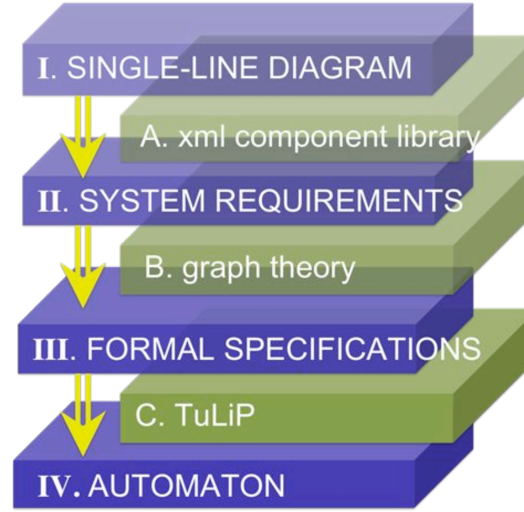
Goals (partly) achieved so far

- Demonstrated
  - **applicability** (formalize the EPS control problem),
  - **usability** (Domain Specific Language)

  of "correct-by-construction" controller synthesis tools within EPS context and,
  - **integration with simulation tools** (simulink),
  - **implementability** (hardware test-bed)

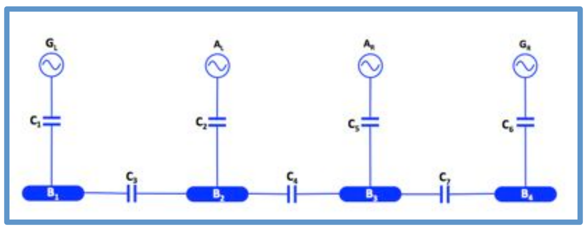
  of synthesized control protocols.

Domain Specific Language

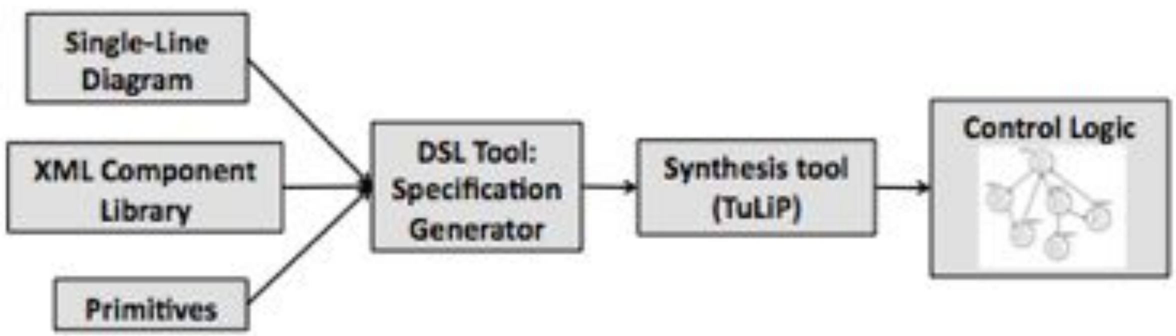


- Text-based specs are ambiguous.
- Formal languages, hard to use if you are unfamiliar
- SLDs and synthesis tools don't speak the same language.
- Idea: Use **primitives** to represent requirements

Sample Primitives:
- reliability(10^{-9}, some comps)
- noparallel(some gens)
- buspower(some buses, 50sec)



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



DSL facilitates consistency between views.

TIME / STATE	untimed	discrete-time
discrete		✓

Reactive Synthesis

Specs:

- Buses never unpowered for more than 50 ms
- Non-paralleling of AC sources
- Priority of generators

} guarantees (φ_s)

- Probability of failure: maintain reliability level } assumptions (φ_e)

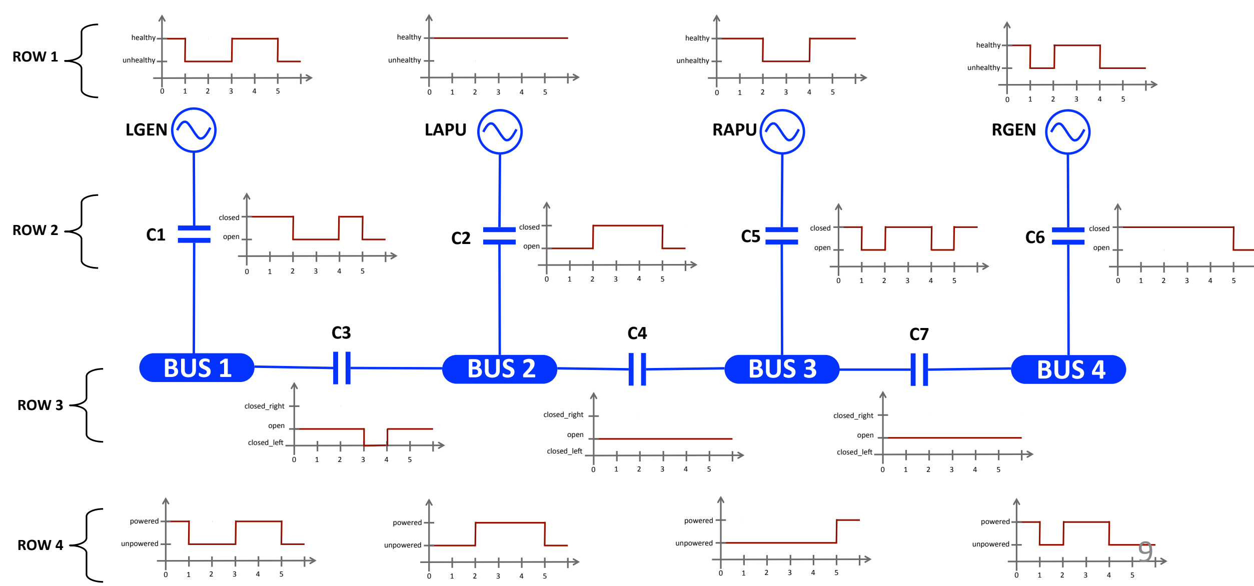
Formal Spec in LTL

$$\varphi_e \longrightarrow \varphi_s$$

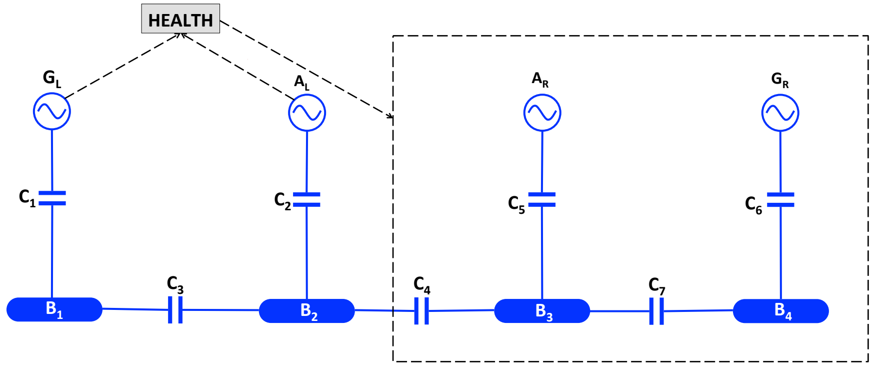
- Environment (generation): G_L, A_L, A_R, G_R (healthy, unhealthy)
- Controlled (contactors): C_1- C_7 (closed, open)
- Dependent (buses): B_1- B_4 (powered, unpowered)

Find a controller that would react to all allowable environment behavior (encoded in assumption) to guarantee specification is met or declare non-existence!

Output: Control logic (represented as an automaton).



Distributed Synthesis



Problem Statement: Given a global spec $\varphi_e \rightarrow \varphi_s$, and an inter-connection structure, find local controllers to satisfy the spec.

Main Results: Decompose the global spec into local ones $\varphi_{e_i} \rightarrow \varphi_{s_i}$ for each control unit such that

$$\bigwedge_i \varphi_{e_i} \rightarrow \varphi_e \rightarrow \varphi_s \rightarrow \bigwedge_i \varphi_{s_i}$$

–If the local specs satisfy certain conditions and there exist local controllers to satisfy the local specs, local implementations satisfy global spec!

–If the local specs are unrealizable

- Refine the local specs:

$$(\phi_2' \wedge \varphi_{e_1}) \rightarrow (\varphi_{s_1} \wedge \phi_1)$$

$$(\phi_1' \wedge \varphi_{e_2}) \rightarrow (\varphi_{s_2} \wedge \phi_2)$$

1. Master/Slave

$$\phi_r \wedge \varphi_{e_l} \rightarrow \varphi_{s_l}$$

$$\varphi_{e_r} \rightarrow \varphi_{s_r} \wedge \phi_r$$

$$\varphi_{e_r} = \Box(A_R = 1 \vee G_R = 1)$$

$$\phi_r = \Box\{((H_1 = 0) \wedge (B_3 = 1)) \rightarrow (\tilde{C}_4 = -1)\}$$

2. Bi-Directional Power Flow

$$\phi_r \wedge \varphi_{e_l} \rightarrow \varphi_{s_l} \wedge \phi_l$$

$$\phi_l \wedge \varphi_{e_r} \rightarrow \varphi_{s_r} \wedge \phi_r$$

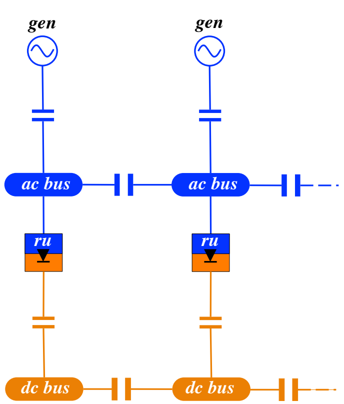
$$\phi_r = \Box\{G_R = 1 \vee A_R = 1 \vee B_2 = 1\}$$

$$\phi_l = \Box\{G_L = 0 \wedge A_L = 0 \rightarrow (C_4 = -1)\}$$

TIME / STATE	untimed	discrete-time
discrete	✓	

Untimed Synthesis

- Steady state solutions
  - Underlying assumption: transients/delays are negligible or can be handled at a different level of abstraction
- Reduces to SAT → More scalable (good heuristics, highly optimized software)
- For reactivity: solve for "each" allowable environment configuration (symmetries → # of confs ↓)



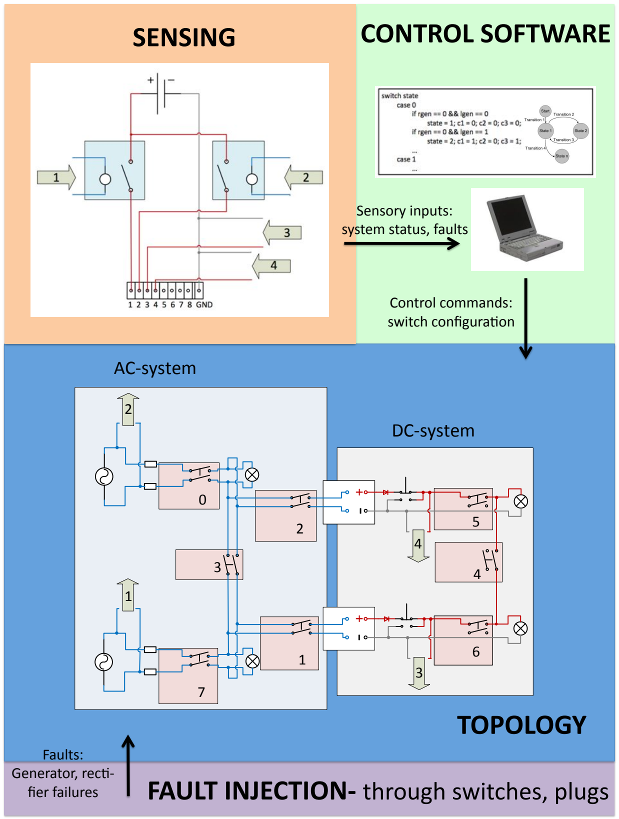
Base Units	Yices Env.	Time(Y/T)	Mem. (Y/T)
4	25	.25/10.7	25MB/215MB
5	36	.82/1015	36MB/16GB
10	121	205.7/–	53MB/–
12	169	1410/–	158MB/–
15	256	62208/–	1.2GB/–

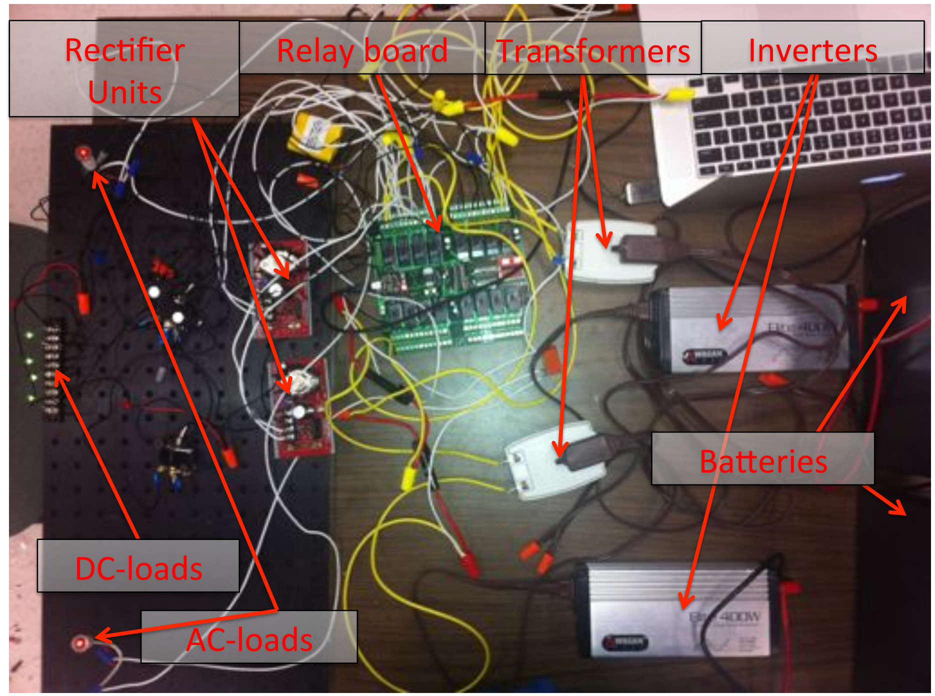
Y: SAT solver yices
T: standard TuLiP synthesis
(time in sec)

Untimed control synthesis for full-scale challenge problem takes 0.9sec (per conf.) and 39MB of memory.

Hardware test-bed

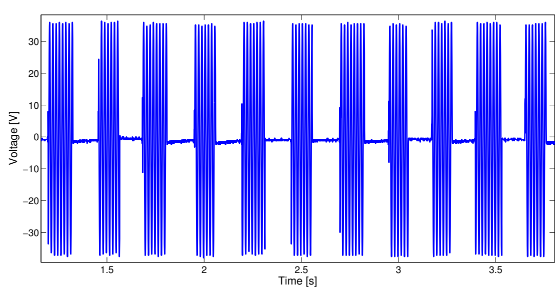
Detailed Model



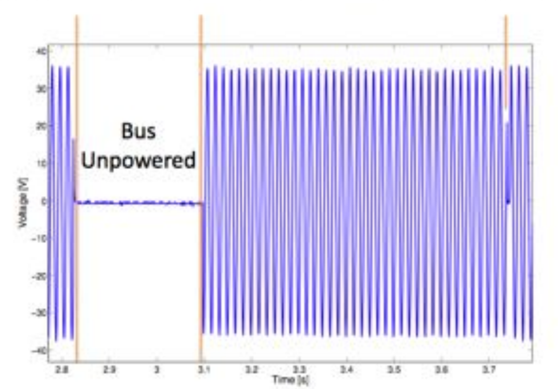


Timing characterization of the system

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

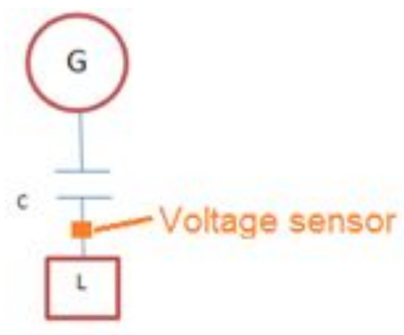


Fault - Controller reacts - Generator back on



Sensing (Fault detection/State Estimation)

- The controller needs to know the state.
- Measurement and state estimation
  - bridge the gap

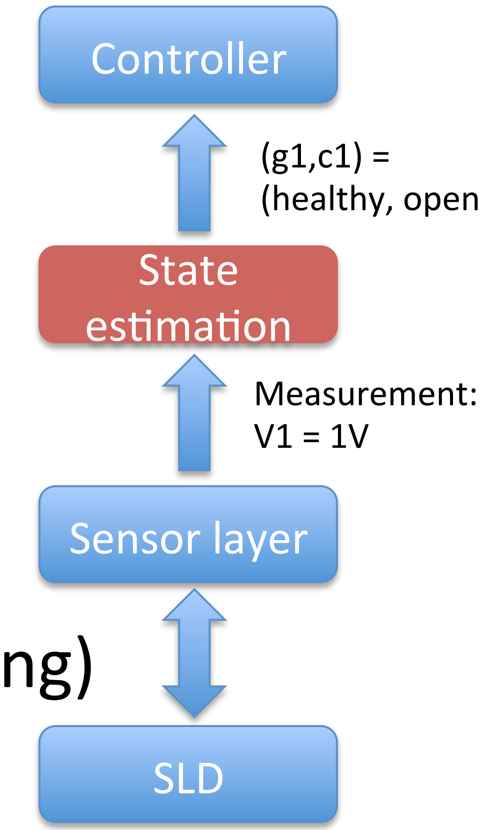


V=1 => (G healhty, C closed)

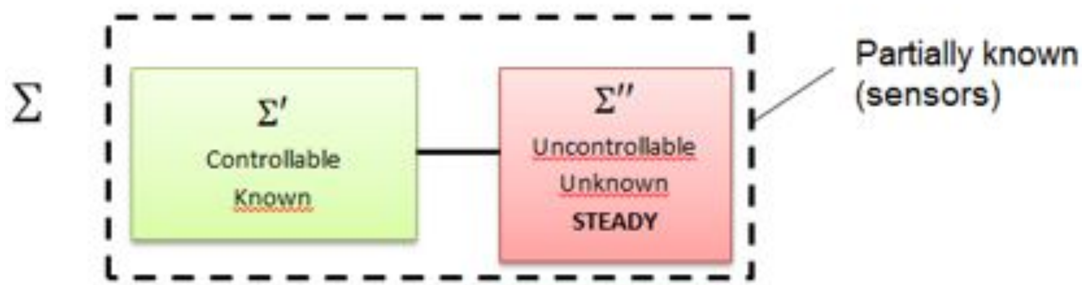
V=0 => (G healthy,C open)

(G faulty, C open)

(G faulty, C closed)



- State estimation with control (active sensing)



Goals

- Demonstrated
  - **applicability** (formalize the EPS control problem),
  - **usability** (Domain Specific Language)

of "correct-by-construction" controller synthesis within EPS context and,

  - **integration with simulation tools** (simulink),
  - **implementability** (hardware test-bed)

of synthesized control protocols.

Future Directions

- To do:
  - **scalability** (as the size and fidelity of the models increase),
  - **integration** of continuous dynamics and timing specifications
  - **different control architectures** (preliminary results w/ distributed but need more systematic methods to distribute the functionality, voting schemes)
  - expand the **types of faults** (including controller failures)
  - work on (active) **sensing** in different abstract views

Goals

- Demonstrated
  - **applicability** (formalize the EPS control problem),
  - **usability** (Domain Specific Language)

  of "correct-by-construction" controller synthesis within EPS context and,
  - **integration with simulation tools** (simulink),
  - **implementability** (hardware test-bed)

  of synthesized control protocols.

Future Directions

- To do:
  - language is **not "complete"** (should cover more specs and be more flexible and extendible),
  - better integration with Single Line Drawing tools and component libraries

Goals

- Demonstrated
  - **applicability** (formalize the EPS control problem),
  - **usability** (Domain Specific Language)

  of "correct-by-construction" controller synthesis within EPS context and,
  - **integration with simulation tools** (simulink),
  - **implementability** (hardware test-bed)

  of synthesized control protocols.

Future Directions

- To do:
  - better integration with other tools (**ptolemy, rhapsody?**)
  - compatibility with different models of computation

Goals

- Demonstrated
  - **applicability** (formalize the EPS control problem),
  - **usability** (Domain Specific Language)

of "correct-by-construction" controller synthesis within EPS context and,

  - **integration with simulation tools** (simulink),
  - **implementability** (hardware test-bed)

of synthesized control protocols.

Future Directions

- To do:

TIME / STATE	untimed	discrete-time	continuous-time
discrete			
discrete & continuous			

we can only export controllers for now

Formal analysis of:

- moving controllers across different abstractions (*model views*)
- in particular: how do violations of the assumptions within a *view* propagate while moving across?