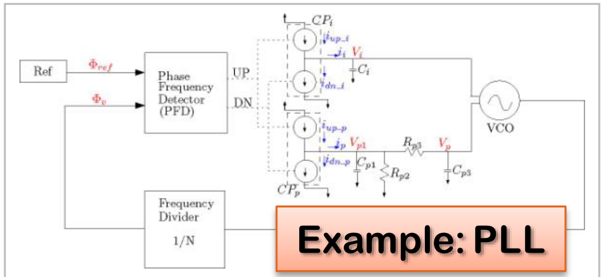
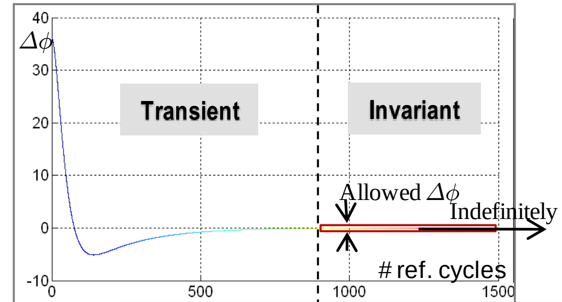
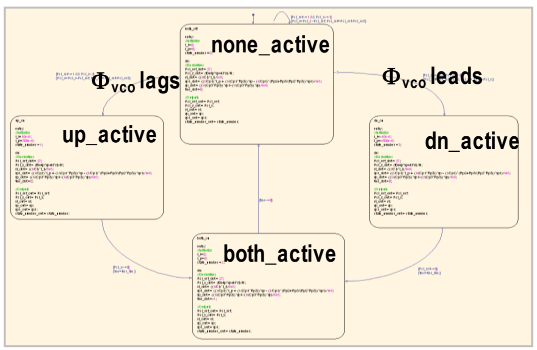
Investigation of Formal Verification for Self-Healing Analog/RF Systems



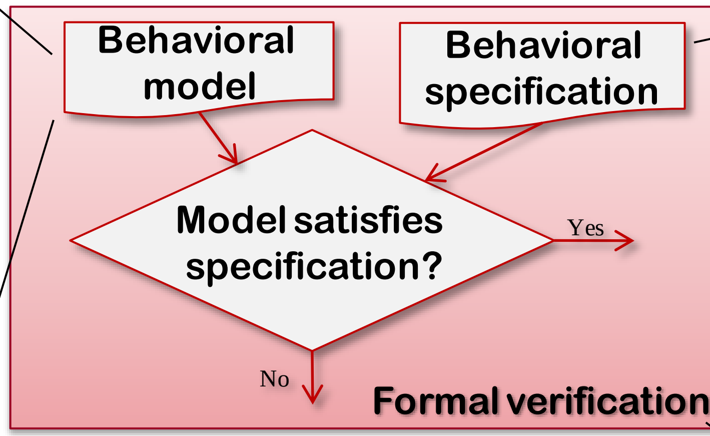
Example: PLL

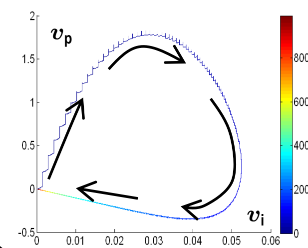


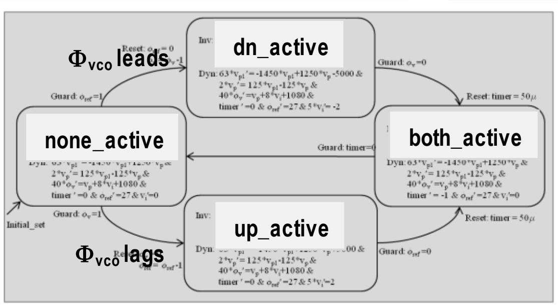
PLL locking specification

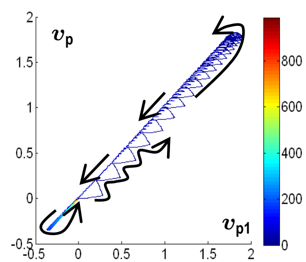


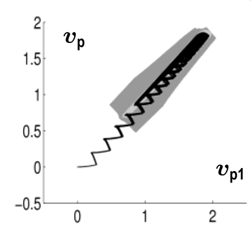
PLL: Behavioral model(s)

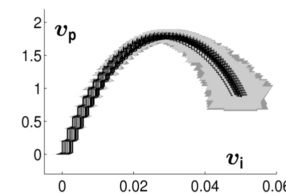












Preliminary results

Motivation for formal verification

ANALYSIS TASK	ANALYSIS METHOD
Analysis of a single operating point	**Simulation**
Analyze the correctness of design	Simulate one particular behavior
Analysis with process variations	**Monte Carlo simulation**
Analyze robustness against process variations	Simulate many behaviors
Analysis over complete post-silicon tuning range	**Formal verification?**
Determine whether there are acceptable solutions in the tuning range	State space too large for simulation! Verify all possible behaviors of a reasonably accurate behavioral model

How we can use formal verification

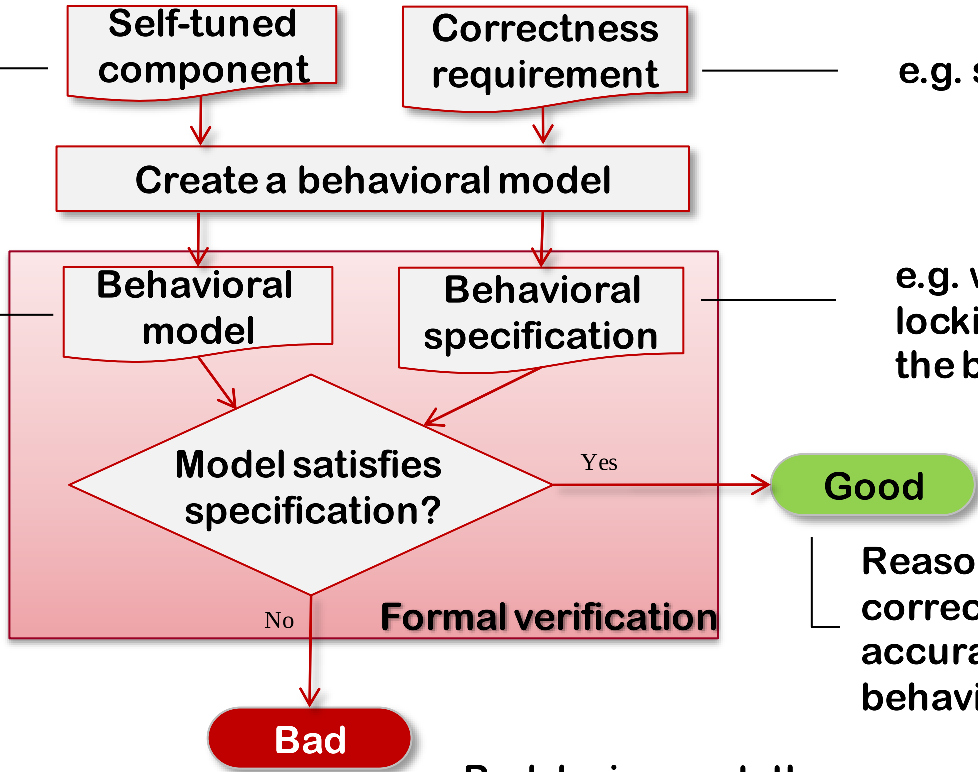
Verification-aided design of self-tuned components

e.g. self-healing PLL

e.g. successful locking

Reasonably accurate abstraction of the real circuit

e.g. what successful locking would mean in the behavioral model

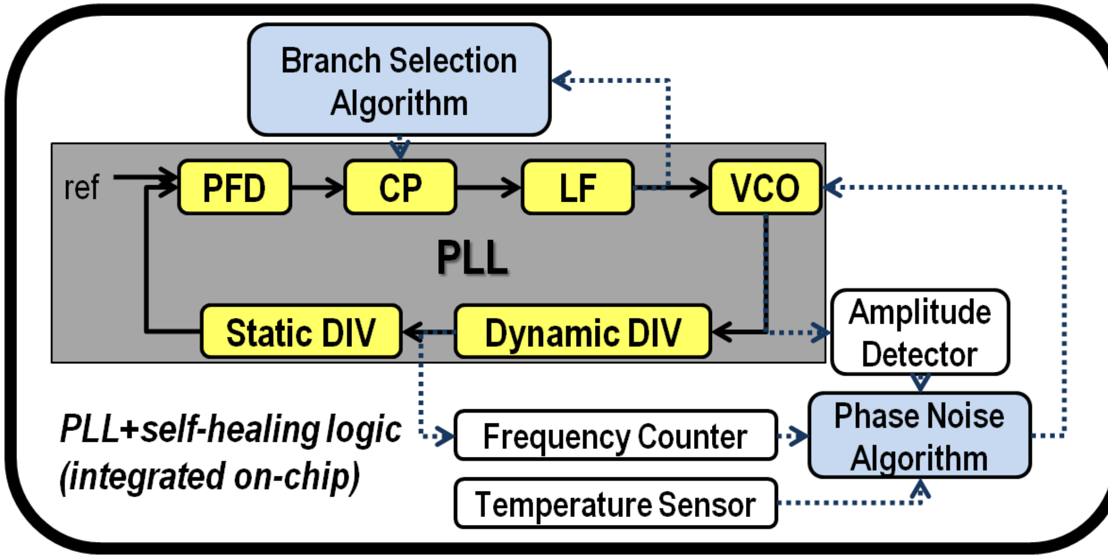


Reasonable assurance of correctness within the accuracy provided by the behavioral model

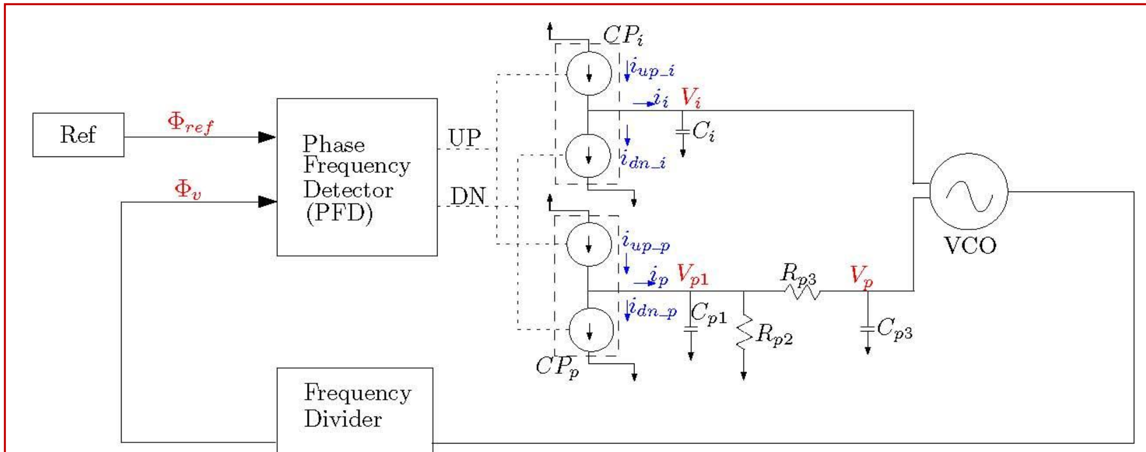
Bad design w.r.t. the correctness requirement (e.g. some behaviors don't satisfy locking spec.) Need to catch this!

Target application: self-healing PLL

- **Verify locking behavior over**
  - arbitrary initial states
  - range of parameter values
  - with self-healing logic



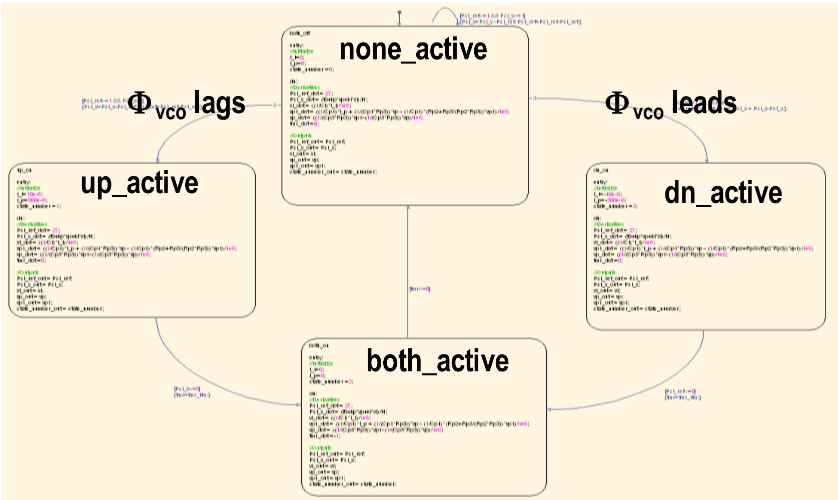
- **Behavioral model**
  - Continuous state variables: Φ_{ref}, Φ_v, V_i, V_{p1}, V_p
  - Discrete switching due to charge pump operation

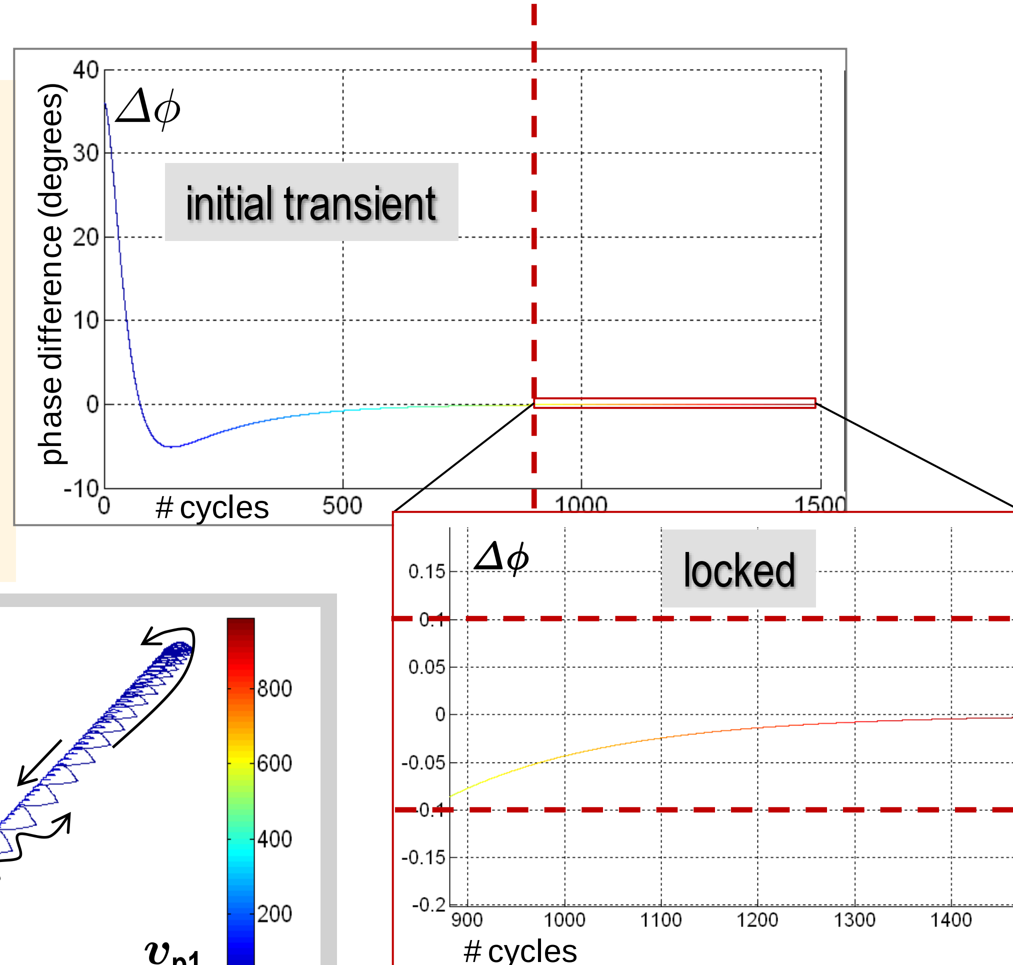


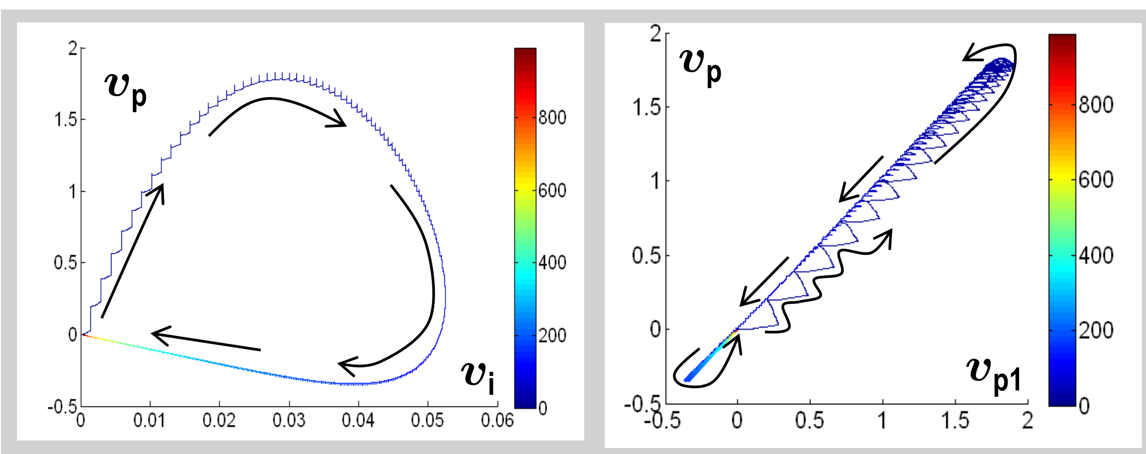
PLL w/o self-healing logic

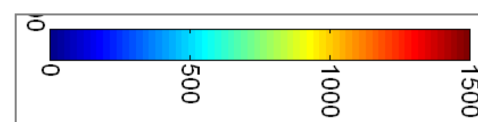
Simulation of the behavioral model

Simulink/Stateflow model





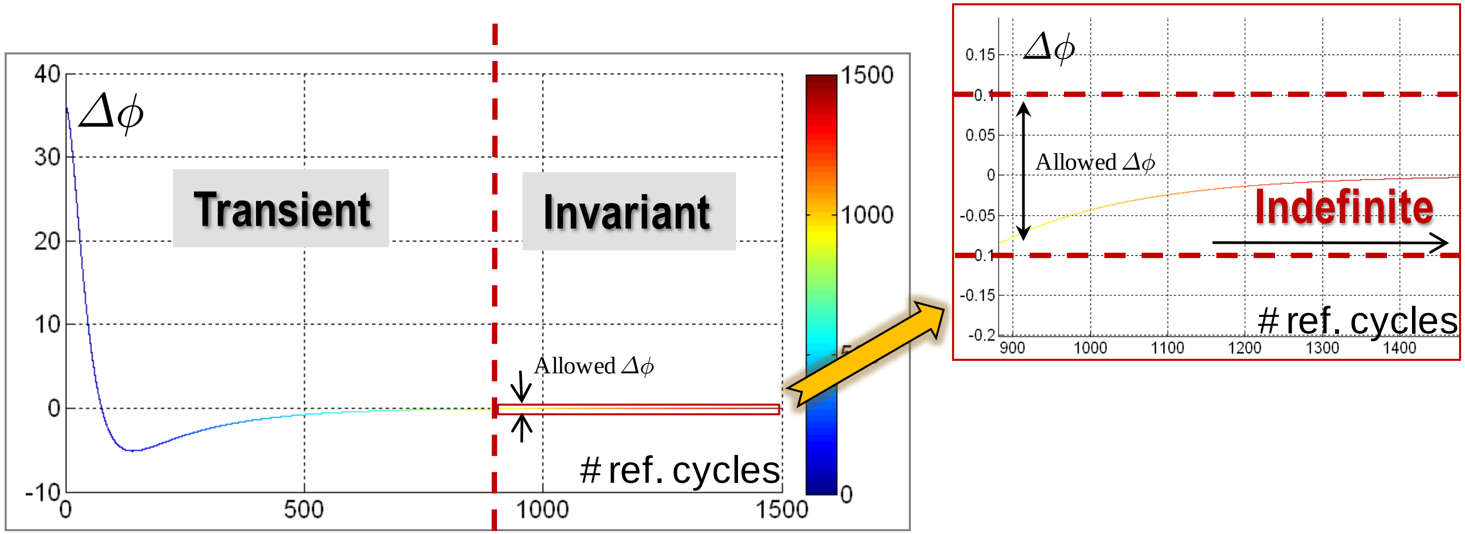




Provides quick insight about the behavior, but can only simulate one behavior at a time. Over a wide range, need to simulate many behaviors one-by-one. This is costly.

Verification approach

Decompose the locking specification into two parts



Transient verification

- Bounded-time verification of whether all behaviors enter the invariant target

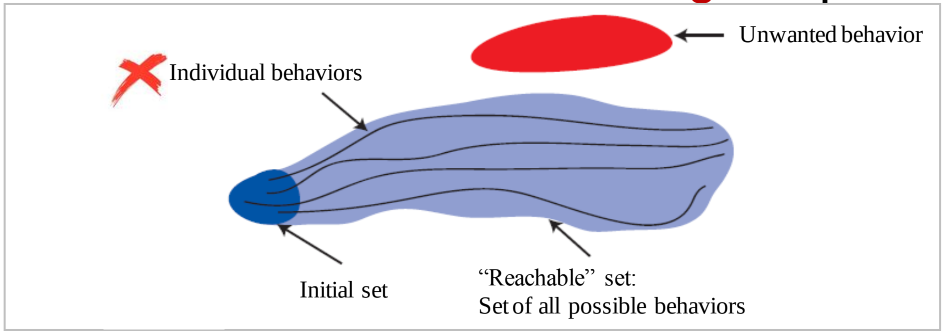
Invariant verification

- Identify regions of state space that guarantee staying in the limit indefinitely
- This becomes a target set for transient verification

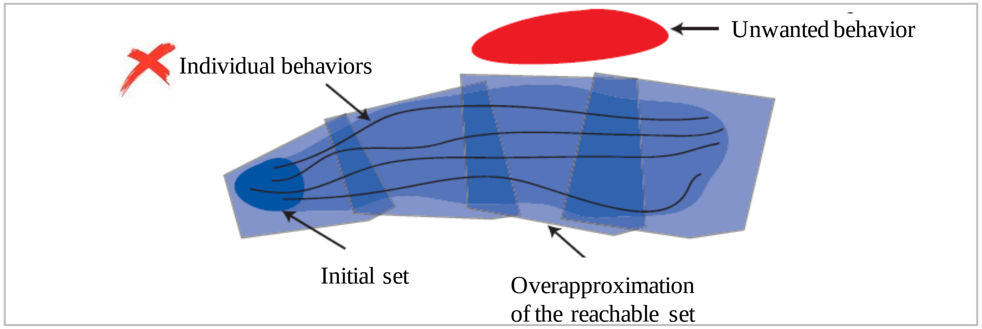
Verification using reachability analysis

General approach

- **Compute the set of all behaviors (not one-by-one)**
  - for a range of initial conditions and a range of possible dynamics



- **If reachable set is hard to compute (typically the case)**
  - over-approximate the set using polyhedra



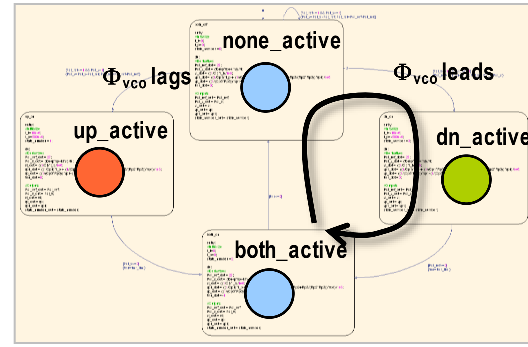
Challenges in reachability analysis

- **Hybrid dynamics**
  - Verification complexity exponential in the number of continuous state variables for polyhedral computations
  - With zonotope (polyhedra with special structure) computations*, there's major speed-up in continuous reachability (cubic complexity); but complexity still exponential for hybrid dynamics
- **Very long transient**
  - Thousands of discrete transitions; over-approximation becomes less accurate with each discrete transition
- **Liveness specification (locking)**
  - Need to verify indefinite (infinite-time) behavior
  - Over-approximation grows with time

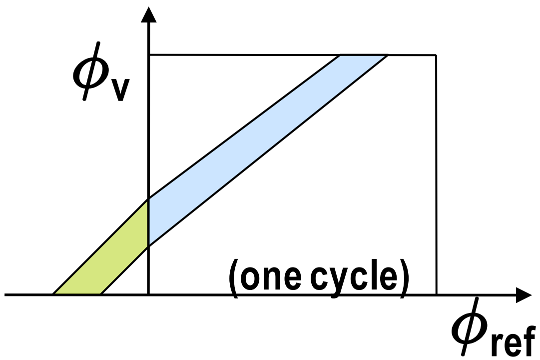
* *Antoine Girard,* Reachability of Uncertain Linear Systems Using Zonotopes. HSCC 2005

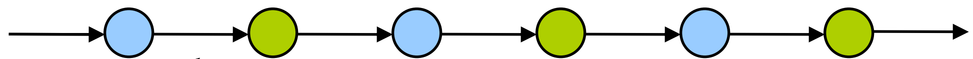
Transient verification using CORA*

Fighting excessive growth of the reachability tree



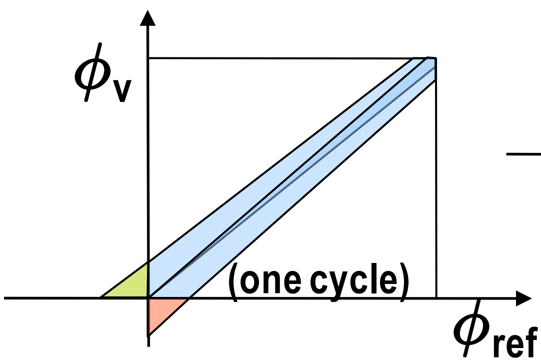
When PLL is far from locking

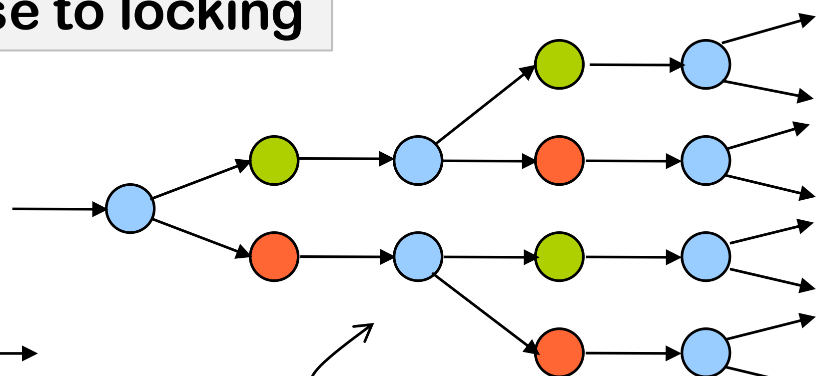




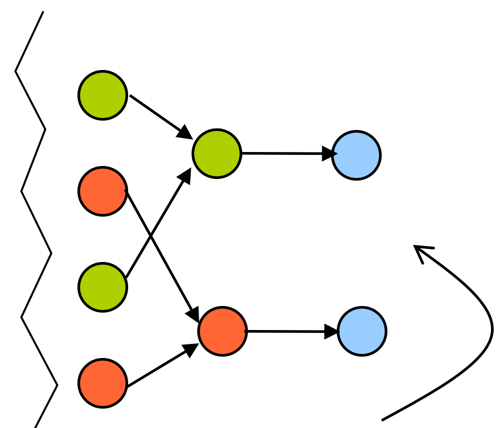
Discrete transition graph: a single branch

When PLL is close to locking





Discrete transition graph: rapidly growing tree

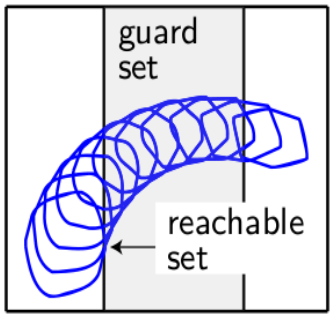


New solution with merging of paths

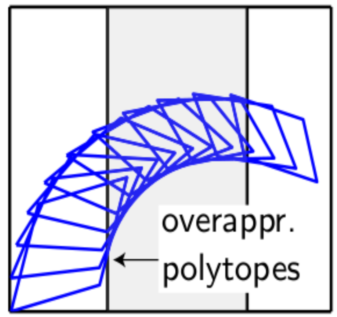
* CORA : COntinuous Reachability Analyzer. *Althoff, M.* Reachability Analysis and its Application to the Safety Assessment of Autonomous Cars, TU München, 2010

Transient verification using CORA
Making guard set overapproximations tighter

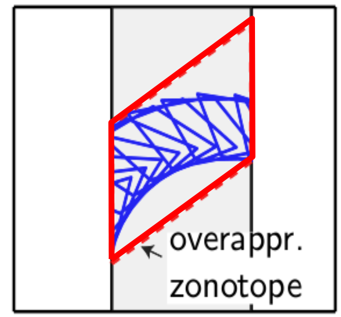
Overapproximation using a single zonotope



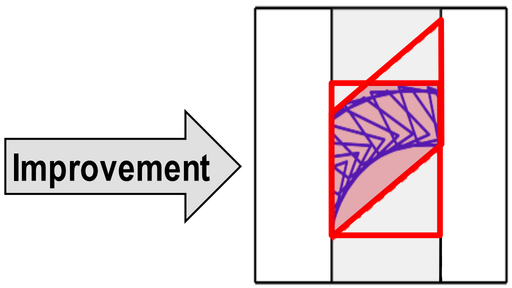
Reachable set using zonotopes



Overapproximate using polytopes

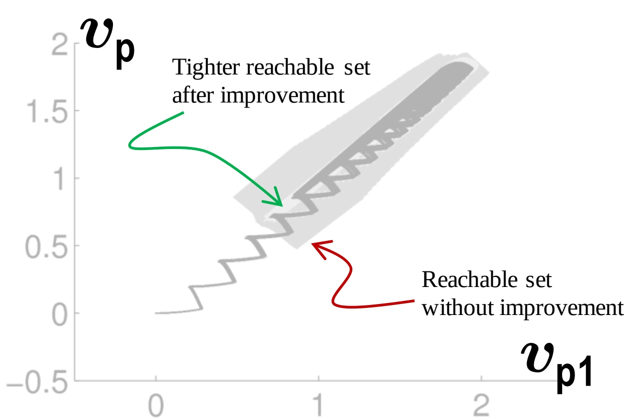


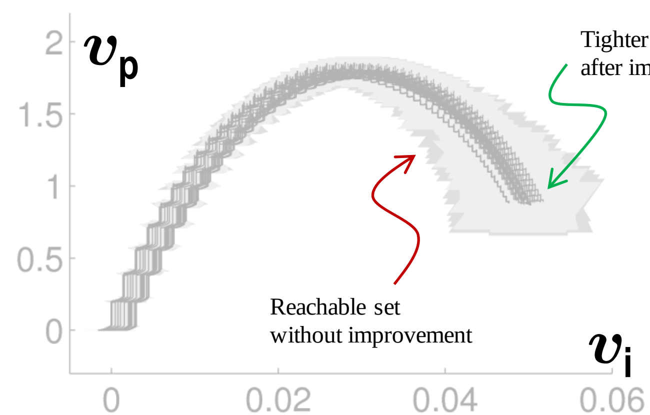
Overapproximate the intersection with a zonotope

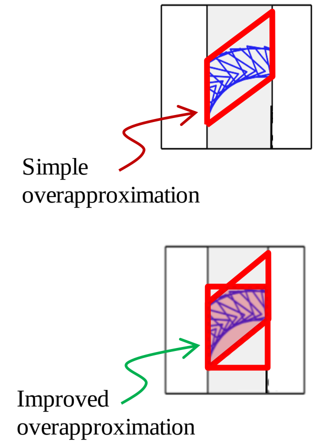


Overapproximate using multiple intersecting zonotopes

Tighter overapproximation using multiple intersecting zonotopes

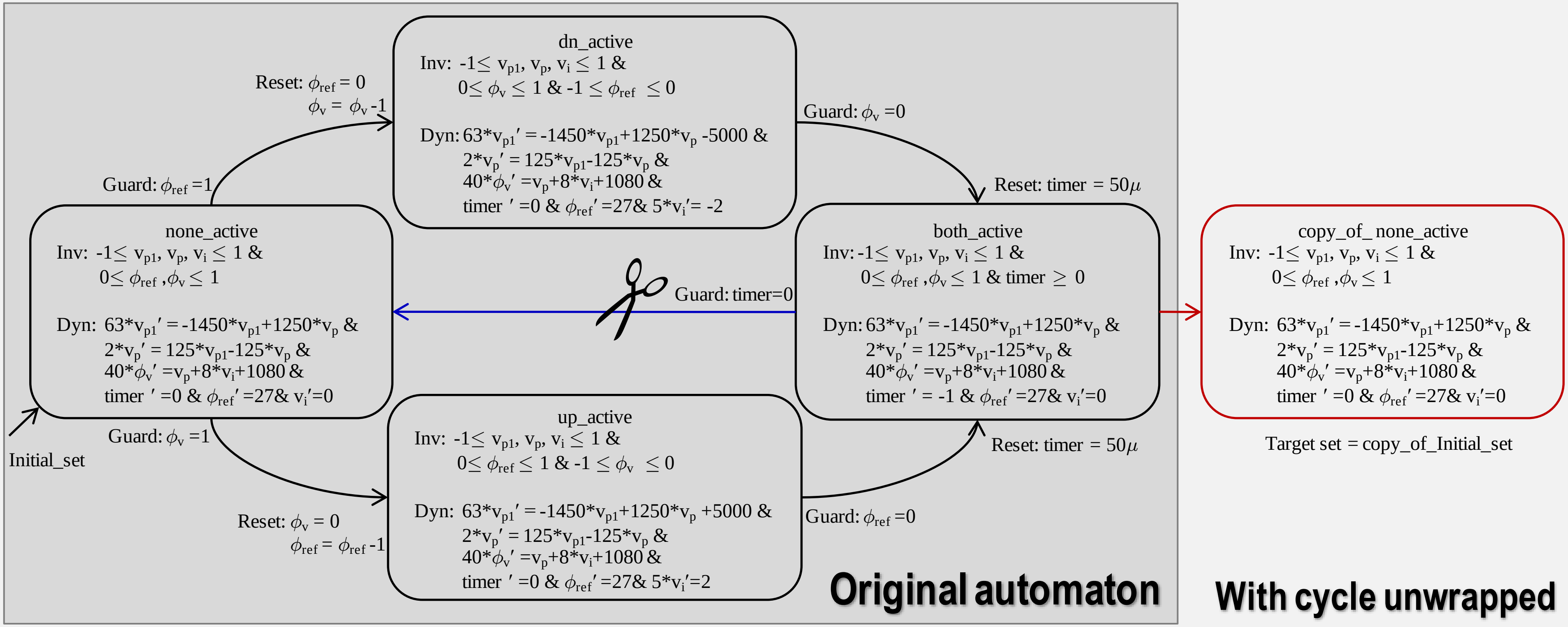




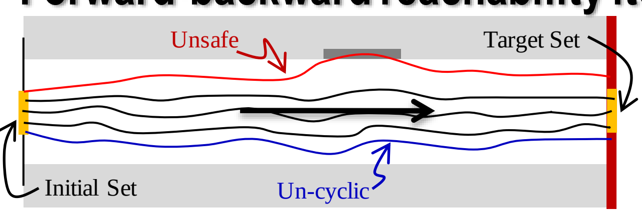


Reachability analysis results for first 50 cycles of ref.

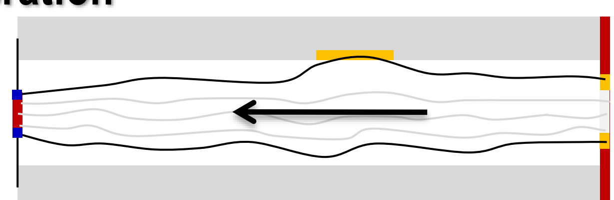
Invariant verification: Forward-backward iteration



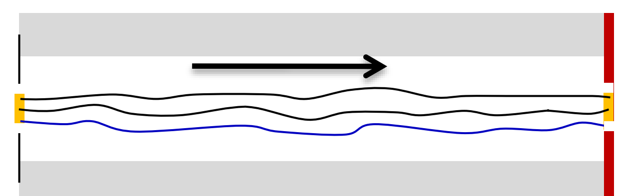
Forward-backward reachability iteration



1. Forward reachability
Check for unsafe/uncyclic behavior



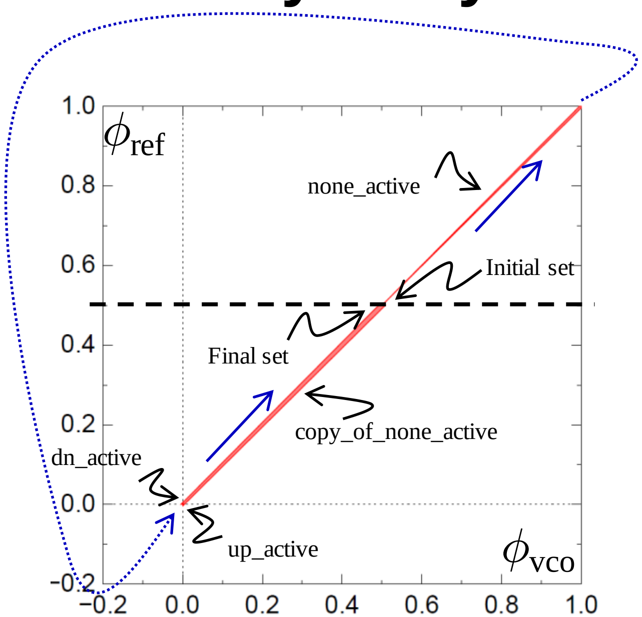
2. Backward reachability
Find unsafe/uncyclic part of initial set

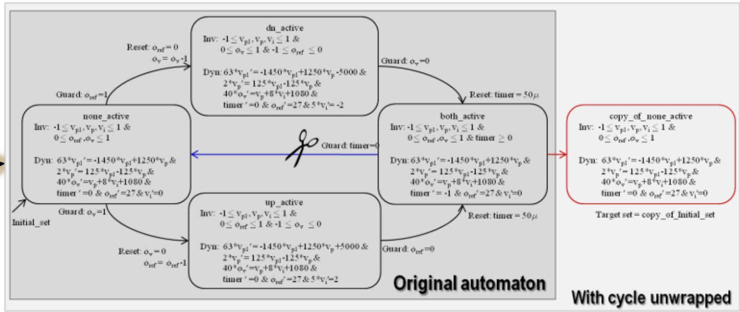


3. Forward reachability
Exclude unsafe/uncyclic initial set, update target set, continue…

Invariant verification using PHAVer*

- **PHAVer (Polyhedral Hybrid Automaton Verifier)**
  - Uses exact rational arithmetic up to arbitrary precision.
  - Supports forward and backward reachability computation.
  - However, needs to overapproximate linear dynamics by (even simpler) piecewise constant bounds on derivatives.
- **Reachability analysis with cycle unwrapped**





<u>Challenge with PHAVer implementation</u>:
When already locked, charge pumps active for a very short fraction of time.
Overapproximation wider than contraction due to charge pump action.

* *Goran Frehse,* PHAVer: algorithmic verification of hybrid systems past HyTech. STTT 10(3): 263-279 (2008)

Next Steps

- **Completion of invariant and transient verification**
- **More detailed model including**
  - Charge pump saturation
  - VCO nonlinearity
- **Compositional verification: digital-analog decoupling**