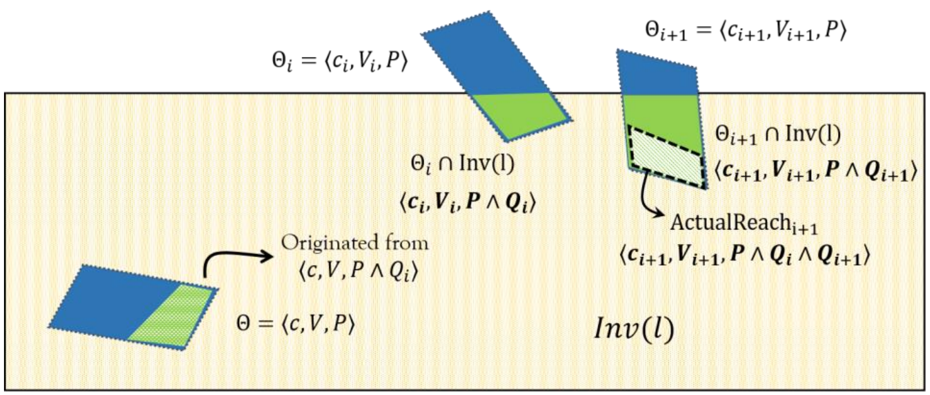
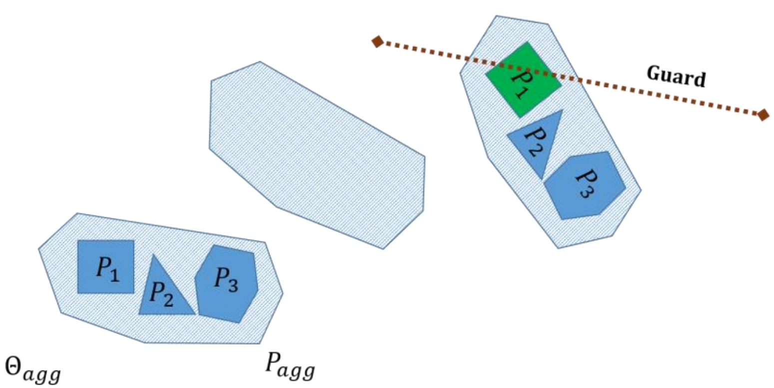
Rigorous Simulation-Based Analysis of Linear Hybrid Systems

Stanley Bak

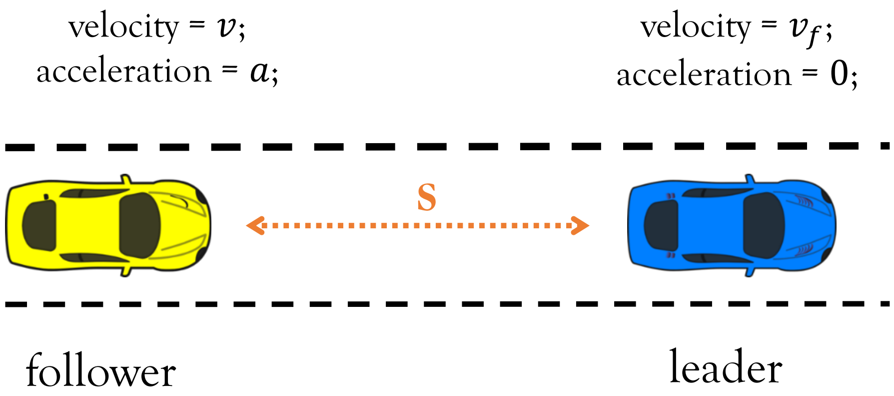
Parasara Sridhar Duggirala

UCONN
UNIVERSITY OF CONNECTICUT





Motivating Example: Leader-Follower System



Dynamics of the system

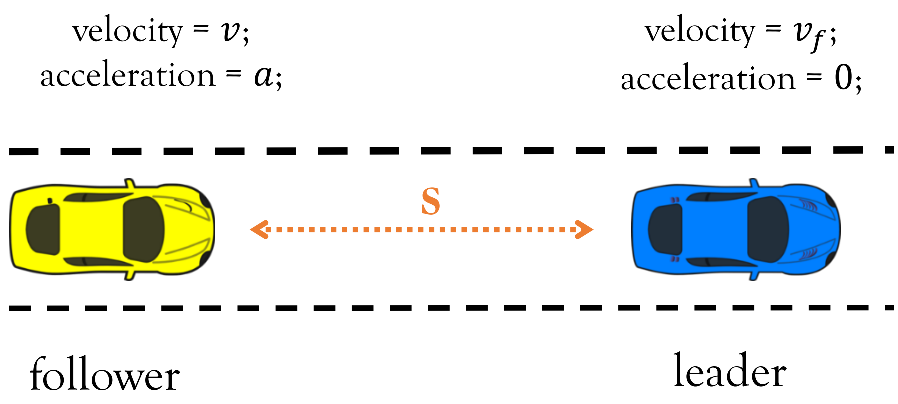
$\dot{s} = v_f - v;$

$\dot{v} = a - k_{aero} v;$

$\dot{a} = u;$

k_{aero} is the air–drag

Motivating Example: Leader-Follower System



Dynamics of the system

$\dot{s} = v_f - v;$

$\dot{v} = a - k_{aero} v;$

$\dot{a} = u;$

k_{aero} is the air-drag

Control Law

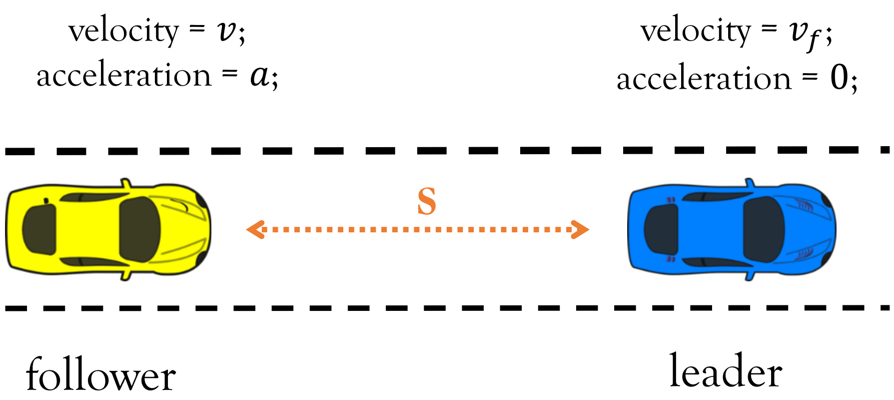
if(cond1) then

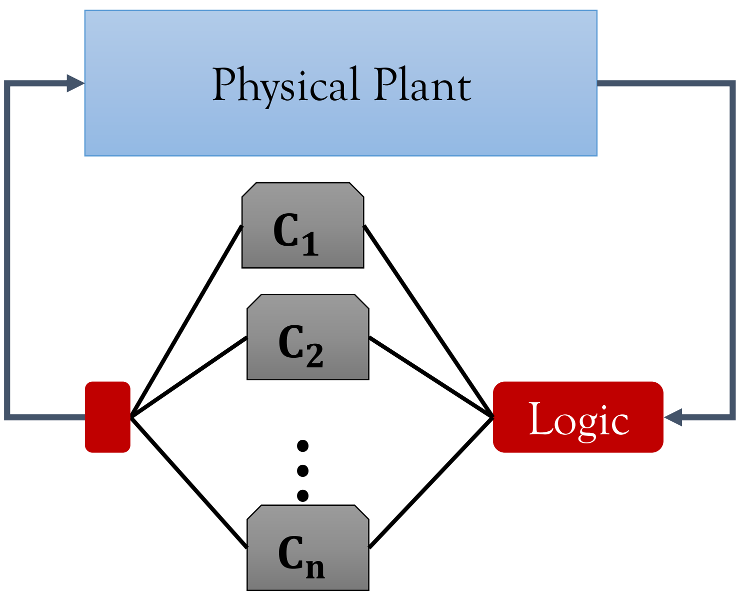
$u = -2a - 2(v - v_f);$

if(cond2) then

$u = -3a - 2(v - v_f);$

Motivating Example: Leader-Follower System





Dynamics of the system

$\dot{s} = v_f - v;$

$\dot{v} = a - k_{aero} v;$

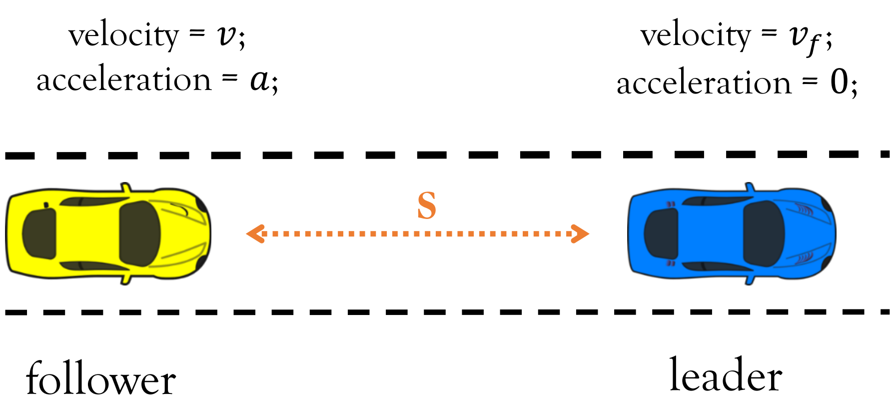
$\dot{a} = u;$

k_{aero} is the air–drag

Control Law

if(cond1) then
  $u = -2a - 2(v - v_f);$
if(cond2) then
  $u = -3a - 2(v - v_f);$

Motivating Example: Leader-Follower System



Dynamics of the system

$\dot{s} = v_f - v;$

$\dot{v} = a - k_{aero} v;$

$\dot{a} = u;$

k_{aero} is the air-drag

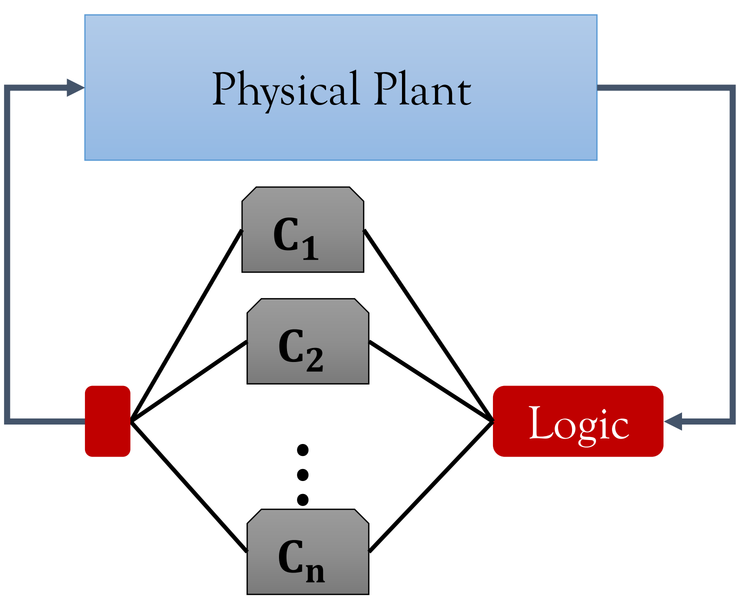
Control Law

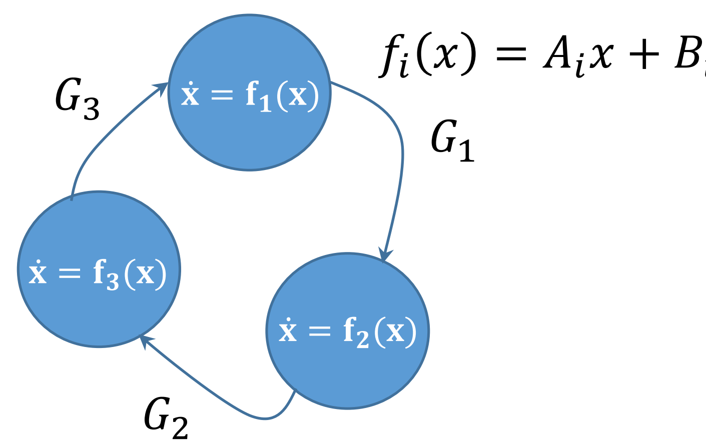
if(cond1) then

$u = -2a - 2(v - v_f);$

if(cond2) then

$u = -3a - 2(v - v_f);$

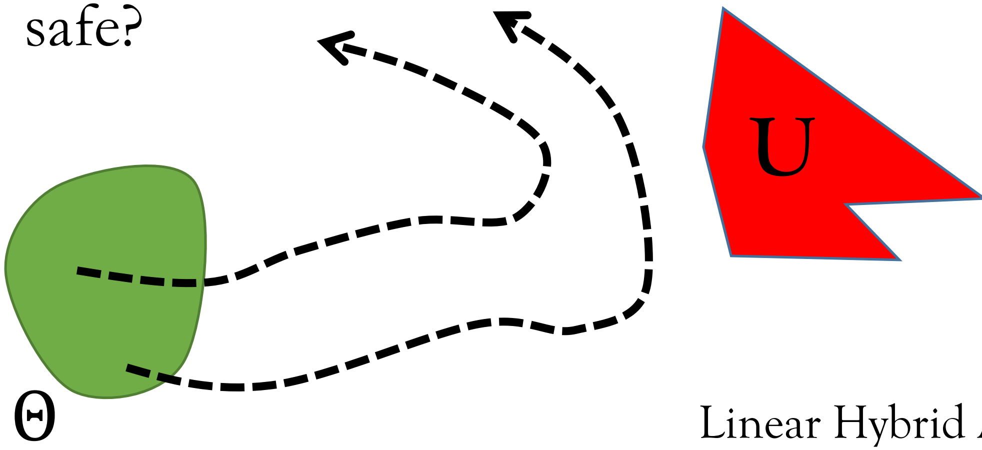


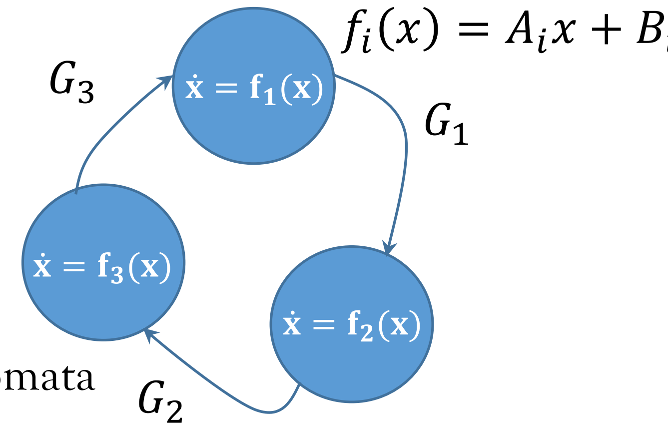


Linear Hybrid Automata

Safety Verification Problem

- Given a Linear Hybrid Automata H, with initial set Θ and unsafe set U, are all the behaviors starting from Θ for bounded time T_b are safe?

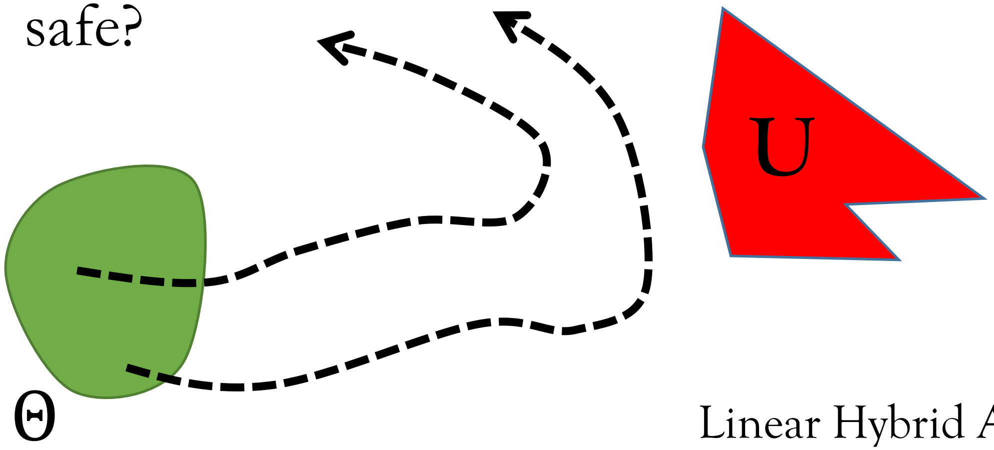


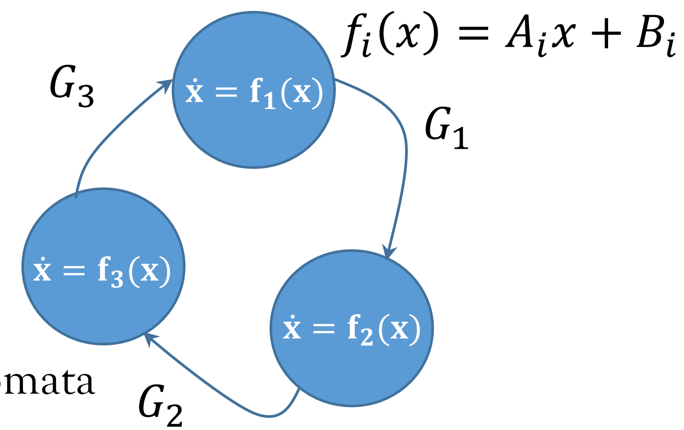


Linear Hybrid Automata

Safety Verification Problem

- Given a Linear Hybrid Automata H, with initial set Θ and unsafe set U, are all the behaviors starting from Θ for bounded time T_b are safe?



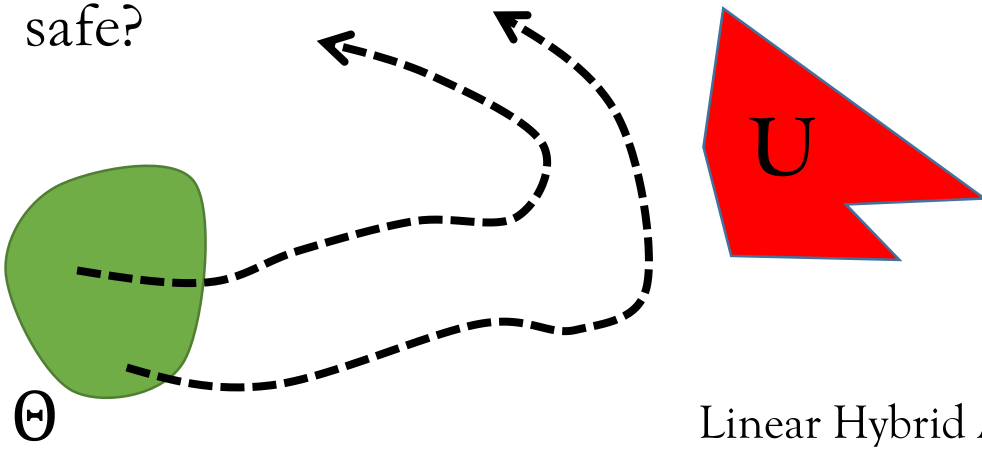


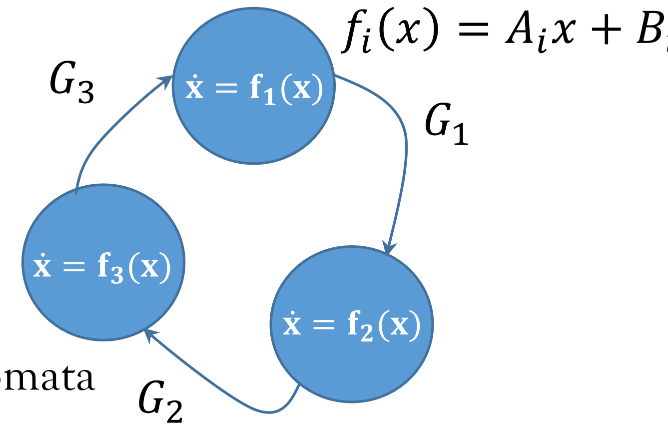
Linear Hybrid Automata

- One technique: Use a safety verification tool such as SpaceEx, Flow*, or CORA, etc.

Safety Verification Problem

- Given a Linear Hybrid Automata H, with initial set Θ and unsafe set U, are all the behaviors starting from Θ for bounded time T_b are safe?



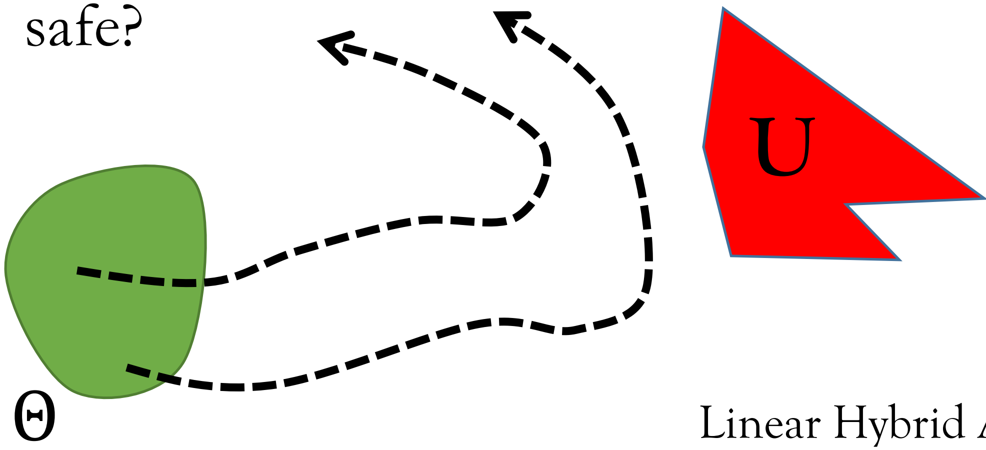


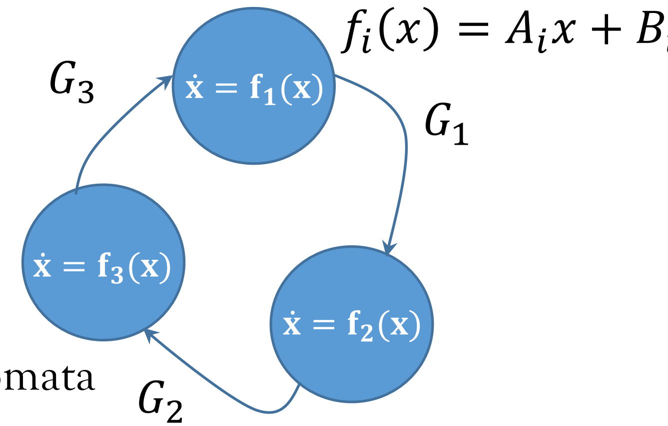
Linear Hybrid Automata

- One technique: Use a safety verification tool such as SpaceEx, Flow*, or CORA, etc.
- **However, most of design analysis is done using simulations.**

Safety Verification Problem

- Given a Linear Hybrid Automata H, with initial set Θ and unsafe set U, are all the behaviors starting from Θ for bounded time T_b are safe?





Linear Hybrid Automata

- One technique: Use a safety verification tool such as SpaceEx, Flow*, or CORA, etc.
- **However, most of design analysis is done using simulations.**

This paper

Simulations ↔ Verification

Simulation-Equivalent Reachability (Safety)

<u>Assumptions</u>

1. We are provided with a simulation engine (oracle) that provides a discrete time simulation for a differential equation $\dot{x} = Ax + B$.
2. All the sets encountered such as invariants, guards, initial set, and unsafe set are all conjunctions of **linear predicates**.

Simulation-Equivalent Reachability (Safety)

Assumptions

1. We are provided with a simulation engine (oracle) that provides a discrete time simulation for a differential equation $\dot{x} = Ax + B$.
2. All the sets encountered such as invariants, guards, initial set, and unsafe set are all conjunctions of **linear predicates**.

Contributions

1. Compute simulation-equivalent reachable set (safety verification).
2. New technique called forward constraint propagation for handling invariants.
3. New on-the-fly aggregation and deaggregation techniques.
4. Sound and complete with respect to the simulation engine provided.

Simulation-Equivalent Reachability (Safety)

Assumptions

1. We are provided with a simulation engine (oracle) that provides a discrete time simulation for a differential equation $\dot{x} = Ax + B$.
2. All the sets encountered such as invariants, guards, initial set, and unsafe set are all conjunctions of **linear predicates**.

Contributions

1. Compute simulation-equivalent reachable set (safety verification).
2. **New technique called forward constraint propagation for handling invariants.**
3. **New on-the-fly aggregation and deaggregation techniques.**
4. Sound and complete with respect to the simulation engine provided.

Overview

- ✓ Motivation and Contributions.
- Dynamic analysis technique for linear systems verification.
- Observations of the dynamic analysis technique.
- Invariant constraint propagation.
- Dynamic deaggregation.
- Experimental evaluation.
- Conclusions and Future work.

Dynamic Analysis Technique For Linear System

Dynamic Analysis Technique

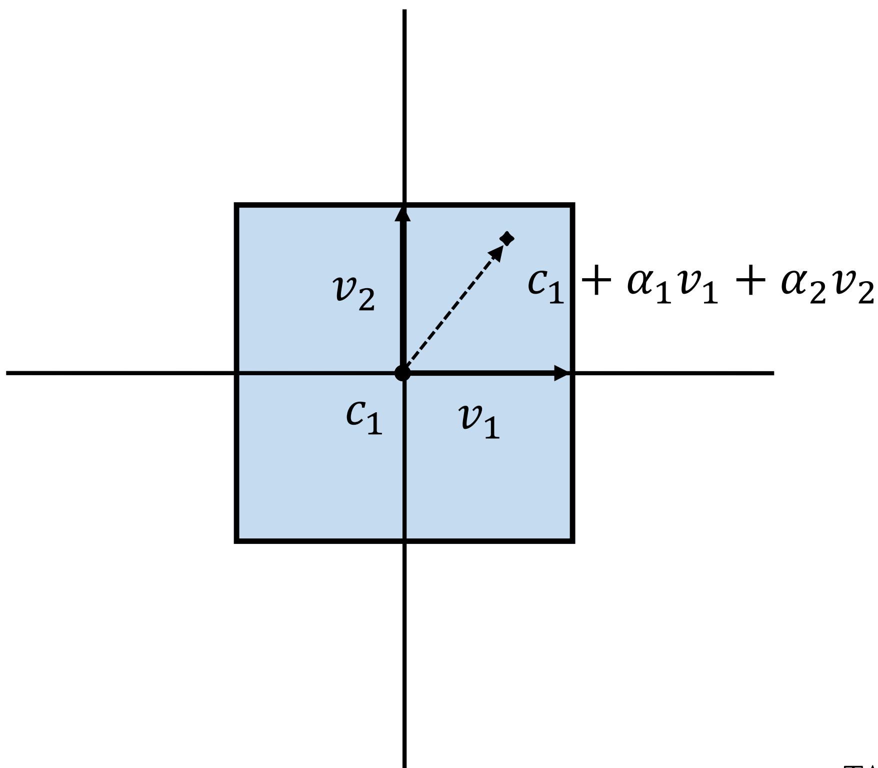
1. The representation: **Generalized stars**.
2. The property of linear systems: **Superposition principle**.
3. The reachable set computing technique: **Safety verification of an $\boldsymbol{n}$ dimensional system using $\boldsymbol{n+1}$ simulations**.

P.S.Duggirala, M.Viswanathan, *"Parsimonious, Simulation Based Verification of Linear Systems"*, International Conference on Computer Aided Verification (CAV) 2016.

Representation: Generalized Stars

- Generalized star is represented as $\langle c, V, P \rangle$
- c – center, V – set of vectors, P – predicate.

$$\langle c, V, P \rangle = \{\, x \mid \exists \bar{\alpha} = (\alpha_1, \ldots, \alpha_n), \mathrm{c} + \Sigma_i \alpha_i v_i = x, P(\bar{\alpha}) = \top \}$$

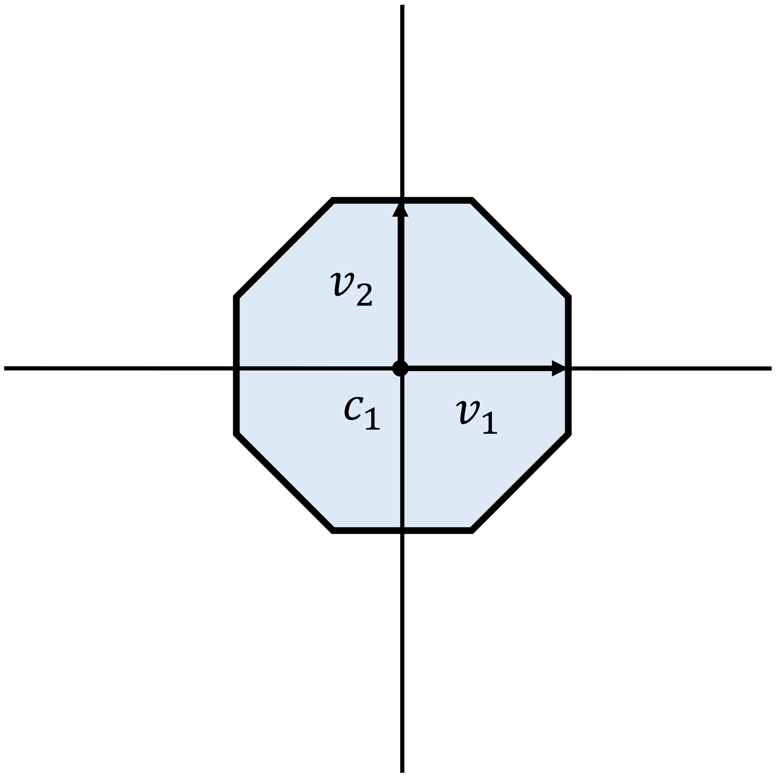


$$P(\langle \alpha_1, \alpha_2 \rangle) \triangleq |\alpha_1| \leq 1 \wedge |\alpha_2| \leq 1$$

Representation: Generalized Stars

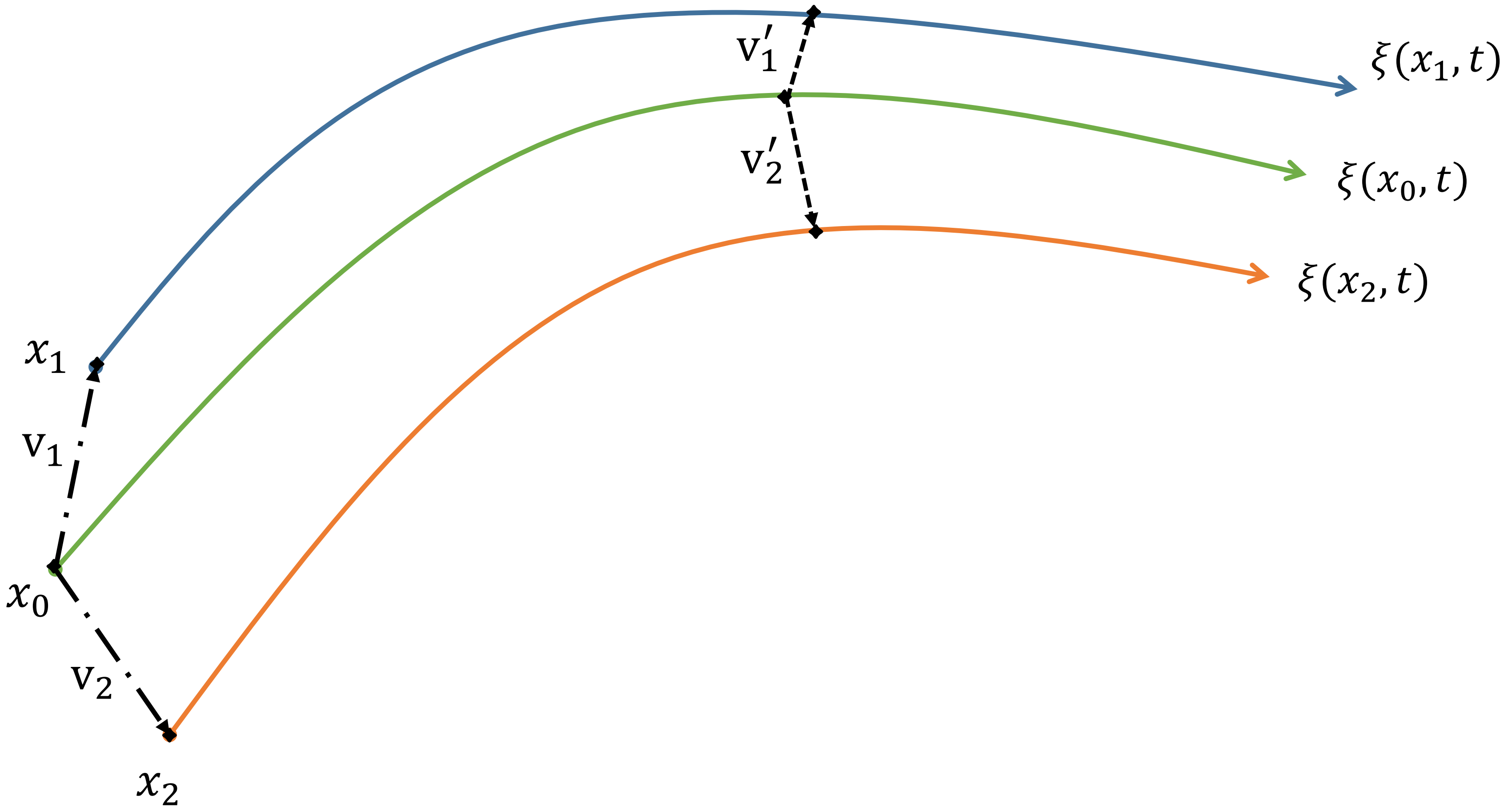
- Generalized star is represented as $\langle c, V, P \rangle$
- c – center, V – set of vectors, P – predicate.

$$\langle c, V, P \rangle = \{\, x \mid \exists \bar{\alpha} = (\alpha_1, \dots, \alpha_n), \mathrm{c} + \Sigma_i \alpha_i v_i = x, P(\bar{\alpha}) = \top \}$$

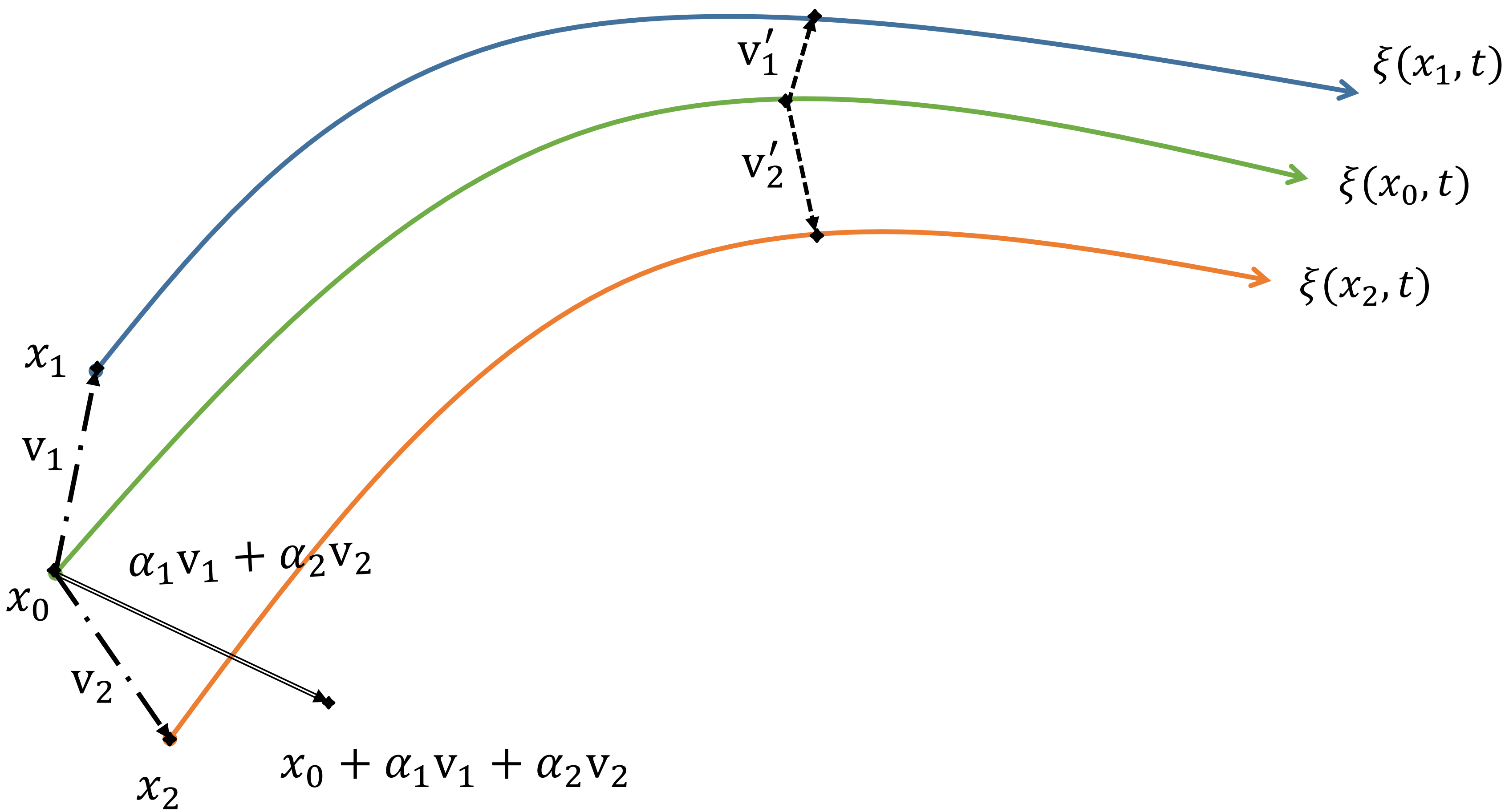


$$P(\langle \alpha_1, \alpha_2 \rangle) \triangleq |\alpha_1| \leq 1 \wedge |\alpha_2| \leq 1 \wedge |\alpha_1 + \alpha_2| \leq 1.5$$

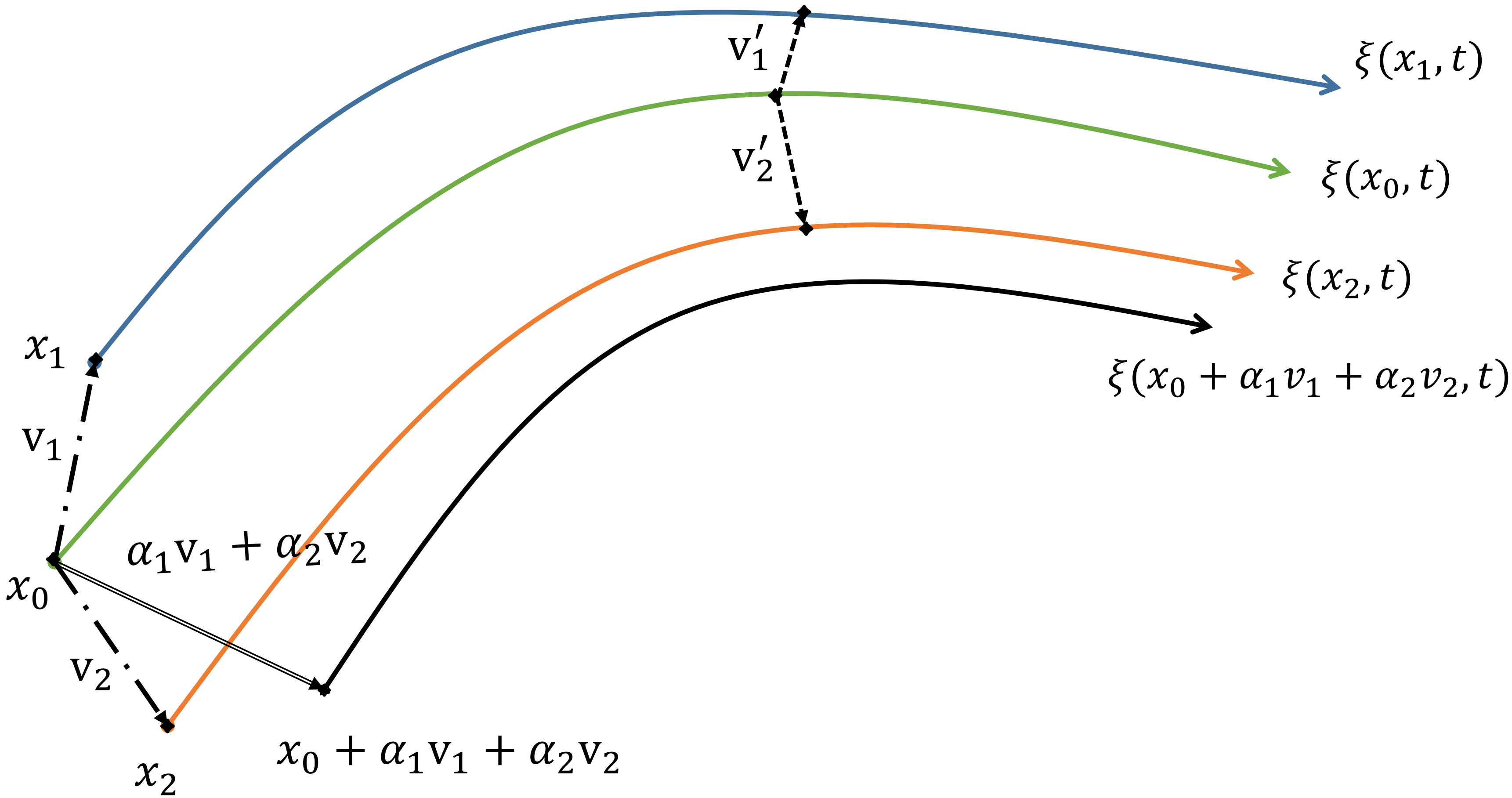
Property: Superposition



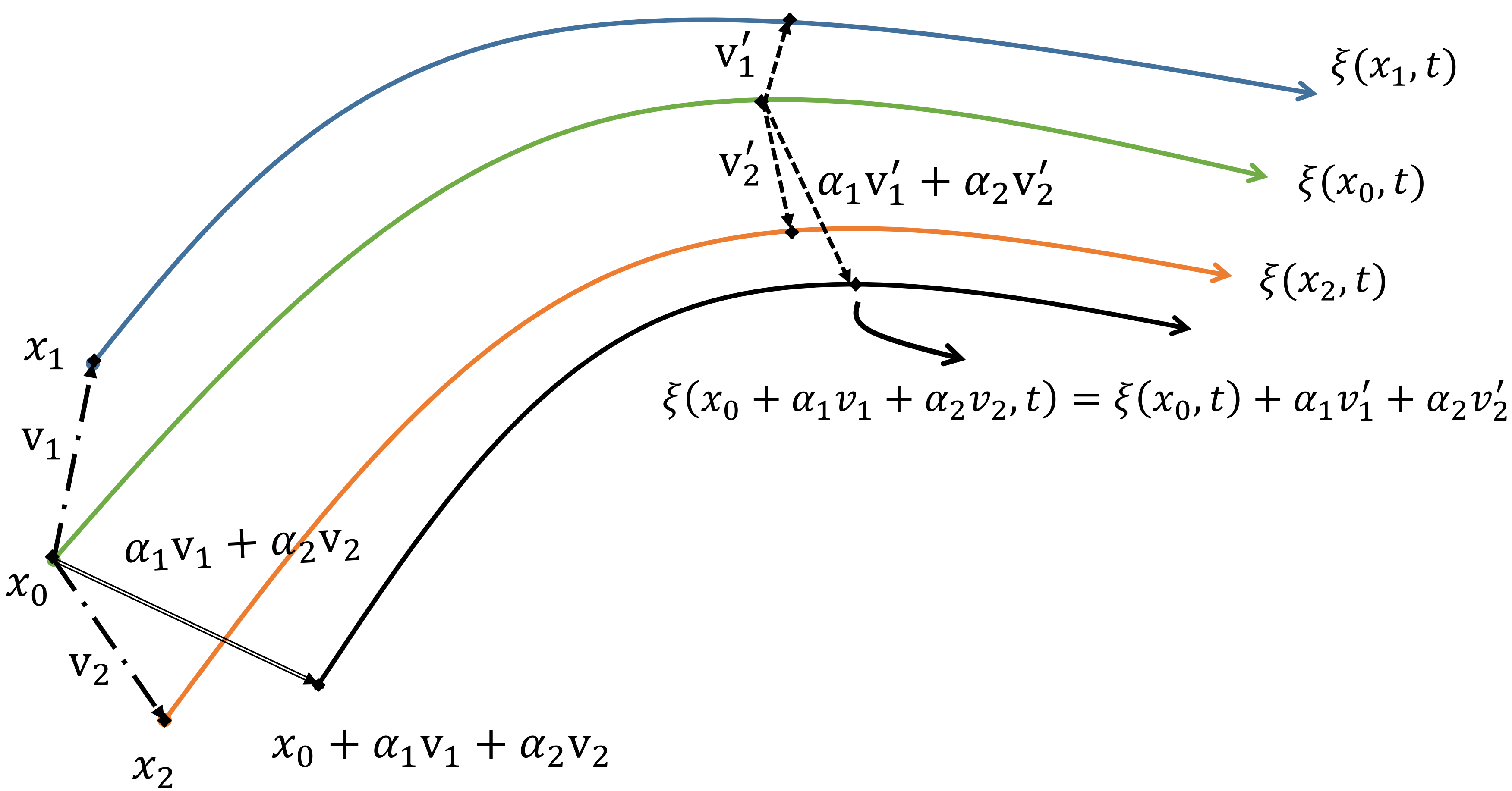
Property: Superposition



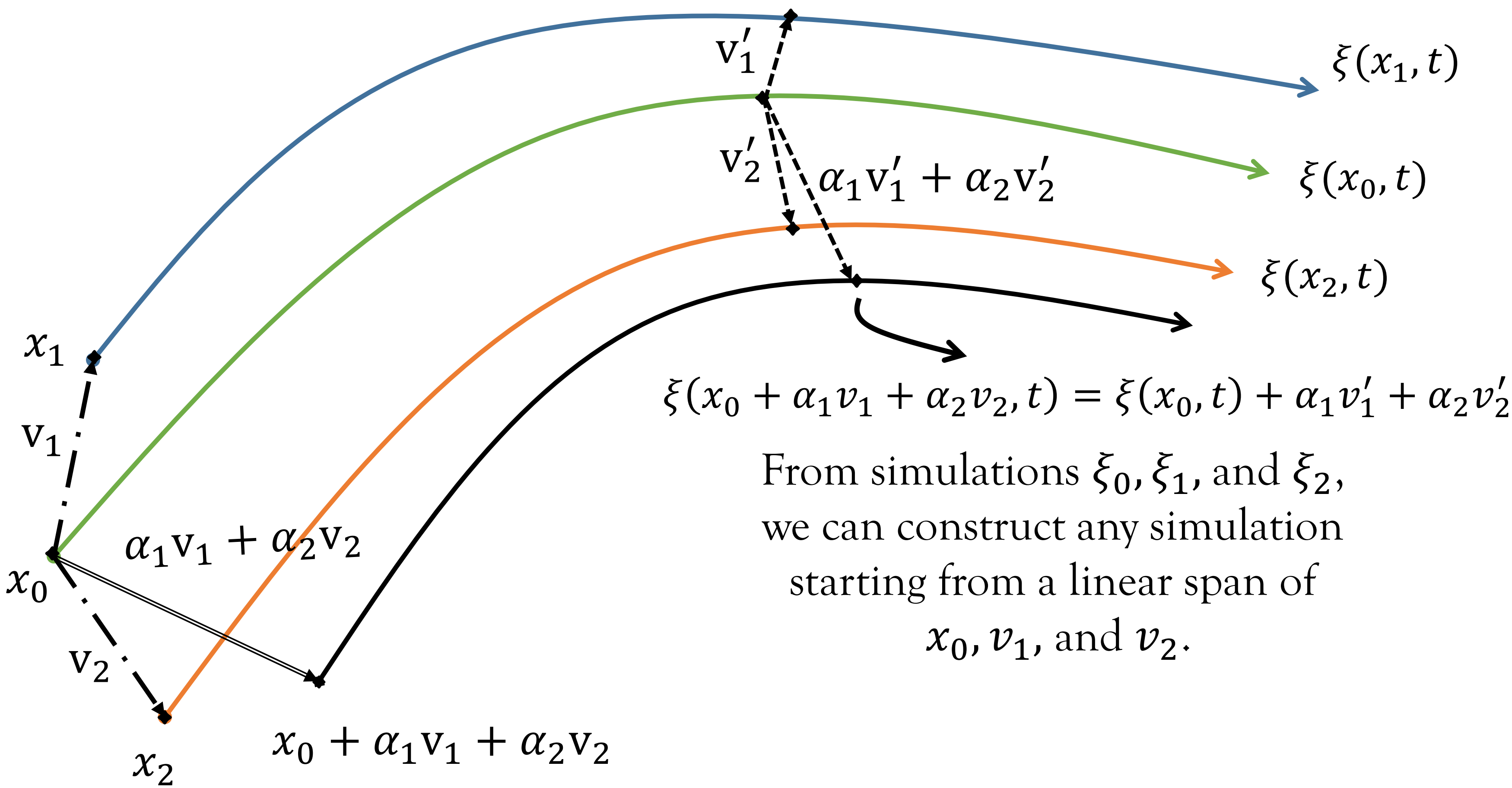
Property: Superposition



Property: Superposition



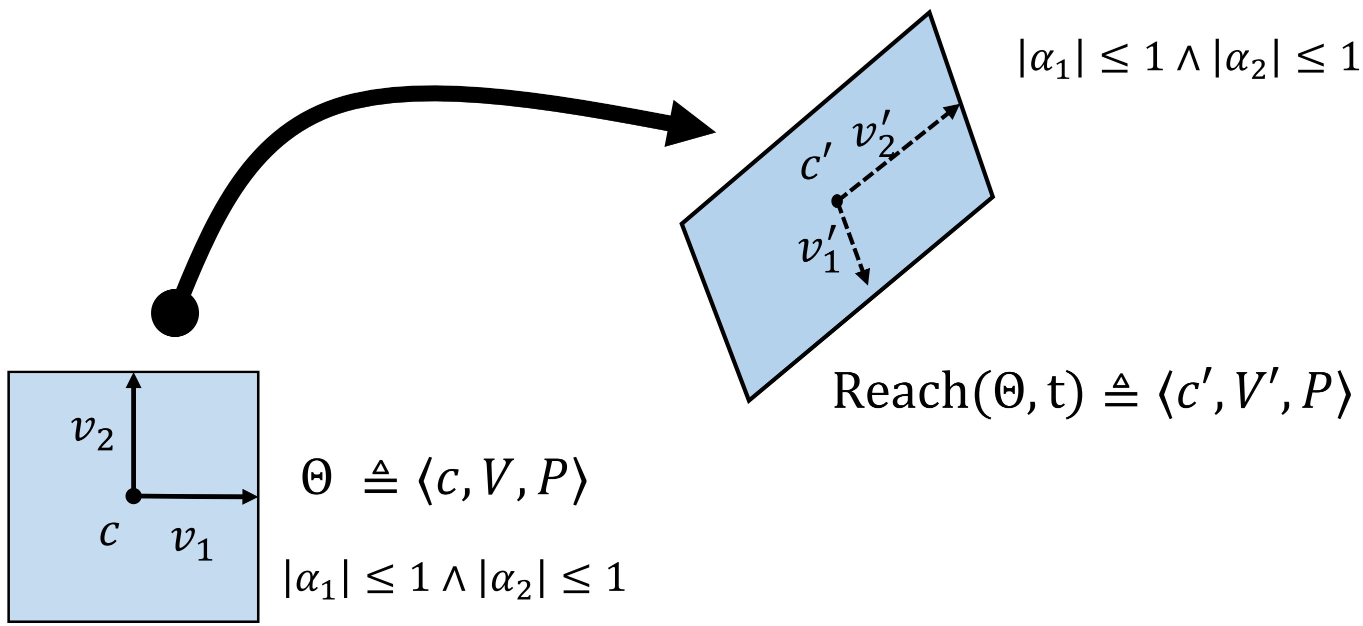
Property: Superposition



From simulations ξ_0, ξ_1, and ξ_2, we can construct any simulation starting from a linear span of x_0, v_1, and v_2.

Technique: Basic Idea

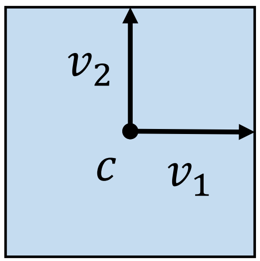
- Given initial set $\Theta = \langle c, V, P \rangle$, the **Reach** is computed not as new predicate, but is done by changing the *center* and the *basis* vectors.



P.S.Duggirala, M.Viswanathan, "*Parsimonious, Simulation Based Verification of Linear Systems*", International Conference on Computer Aided Verification (CAV) 2016.

Technique
Representation + Superposition

Given $\Theta \triangleq \langle c, V, P \rangle$ to compute reachable set



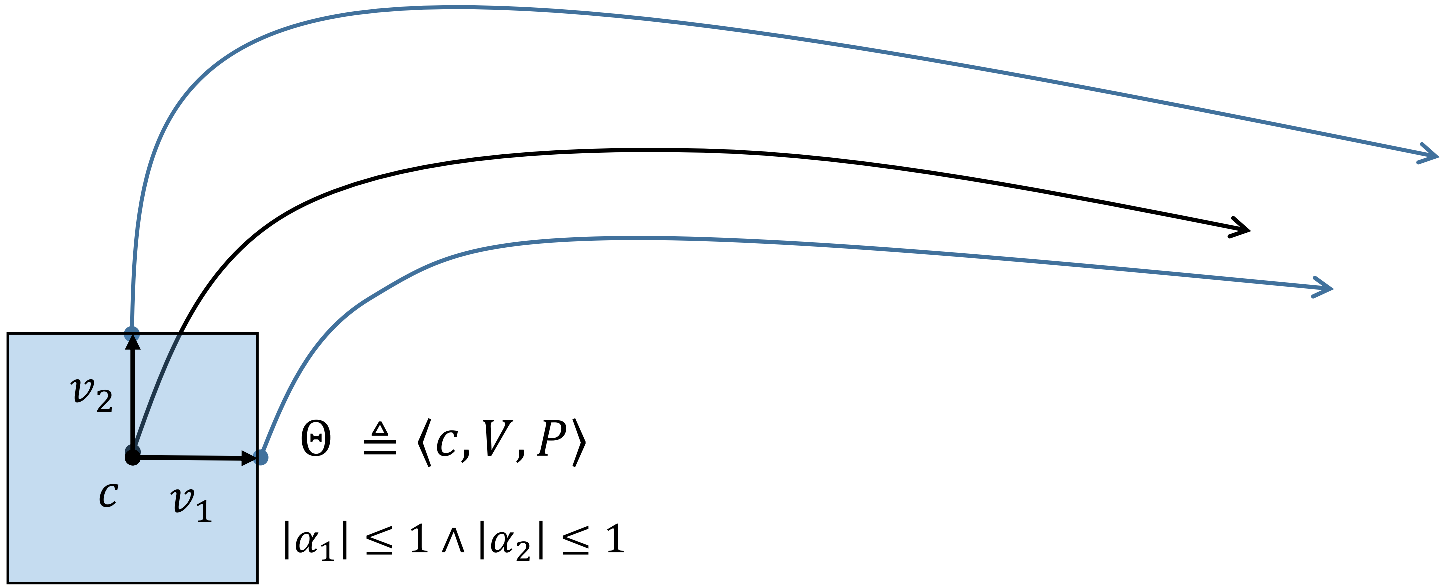
$\Theta \triangleq \langle c, V, P \rangle$

$|\alpha_1| \leq 1 \wedge |\alpha_2| \leq 1$

Technique
Representation + Superposition

Given $\Theta \triangleq \langle c, V, P \rangle$ to compute reachable set

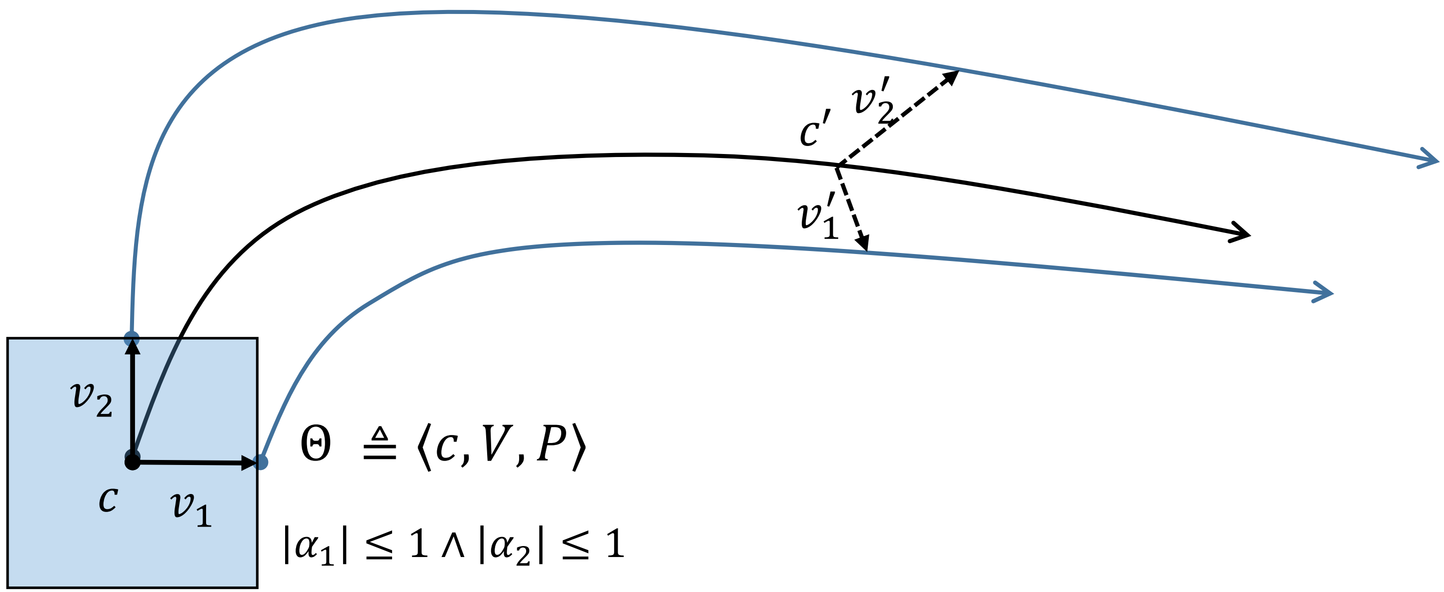
1. Simulate from c
2. Simulate from $c + v_i$ for each i



Technique
Representation + Superposition

Given $\Theta \triangleq \langle c, V, P \rangle$ to compute reachable set

1. Simulate from c
2. Simulate from $c + v_i$ for each i



Reachable set at time t is given by $\langle c', V', P \rangle$ where

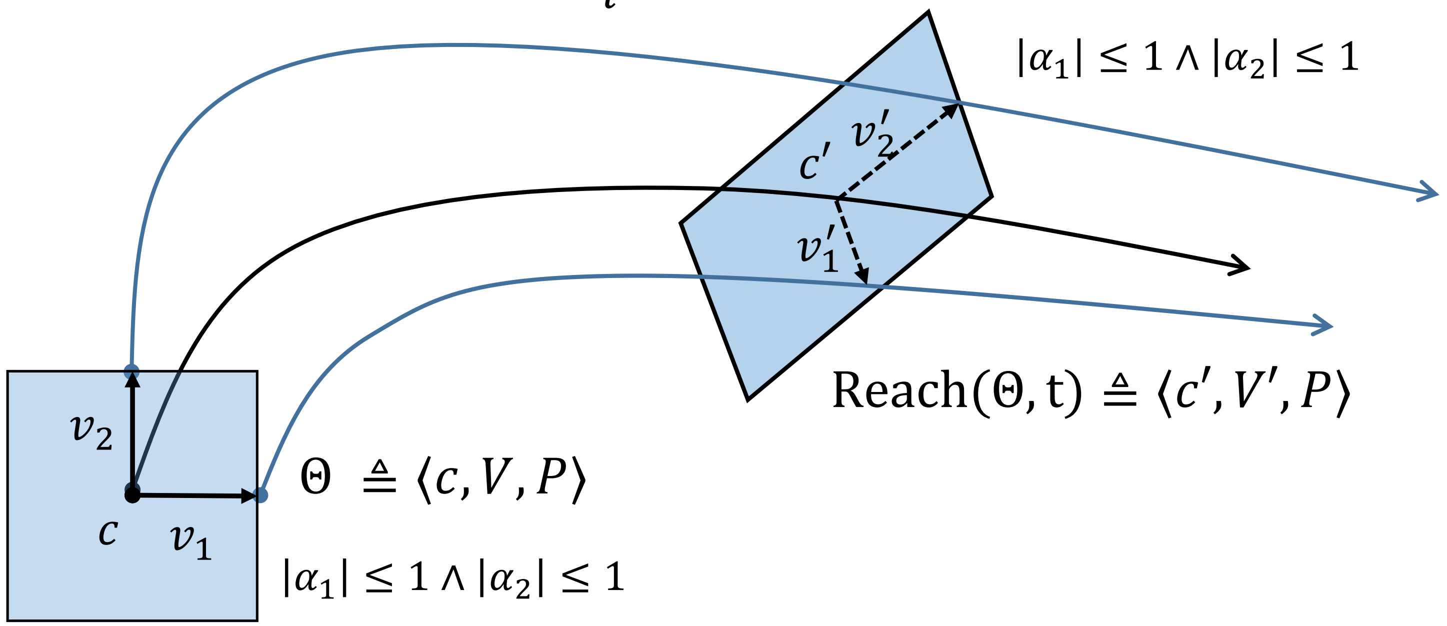
1. c' is the simulation corresponding to c

2. ${v_i}'$ is the difference of simulations from $c + v_i$ and from c

Technique
Representation + Superposition

Given $\Theta \triangleq \langle c, V, P \rangle$ to compute reachable set

1. Simulate from c
2. Simulate from $c + v_i$ for each i



Reachable set at time t is given by $\langle c', V', P \rangle$ where

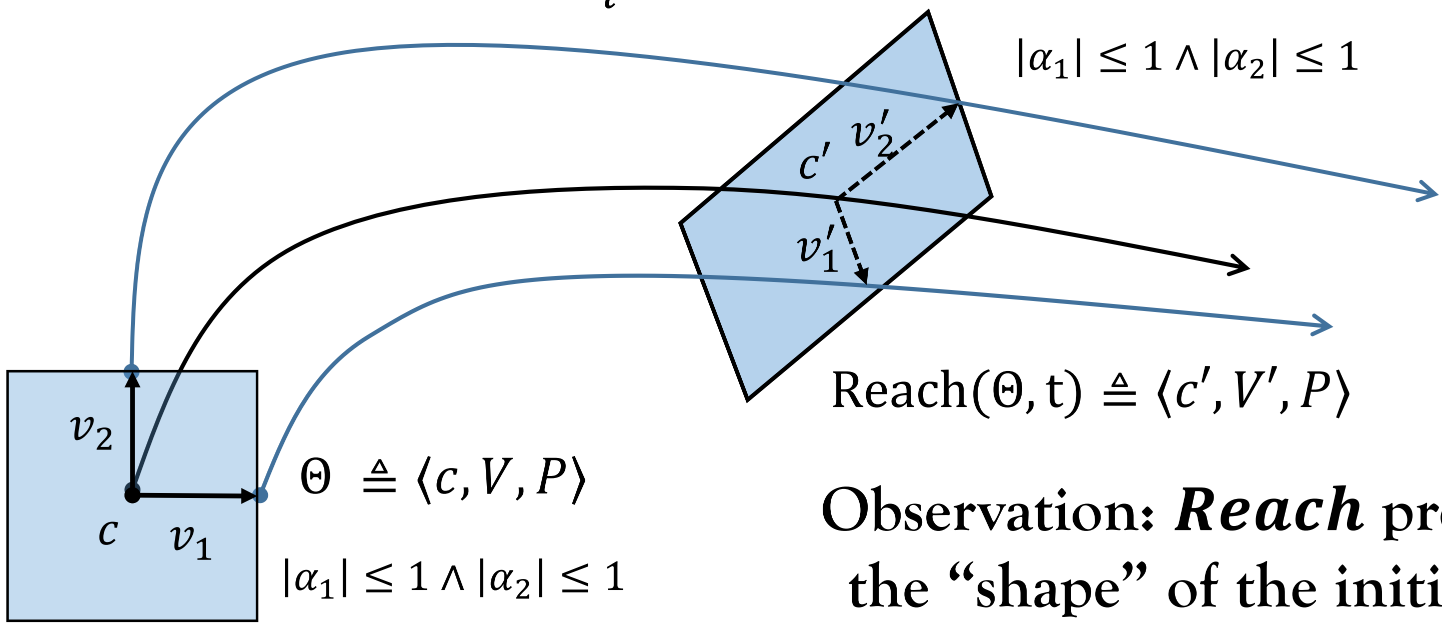
1. c' is the simulation corresponding to c

2. ${v_i}'$ is the difference of simulations from $c + v_i$ and from c

Technique
Representation + Superposition

Given $\Theta \triangleq \langle c, V, P \rangle$ to compute reachable set

1. Simulate from c
2. Simulate from $c + v_i$ for each i



Observation: *Reach* preserves the "shape" of the initial set.

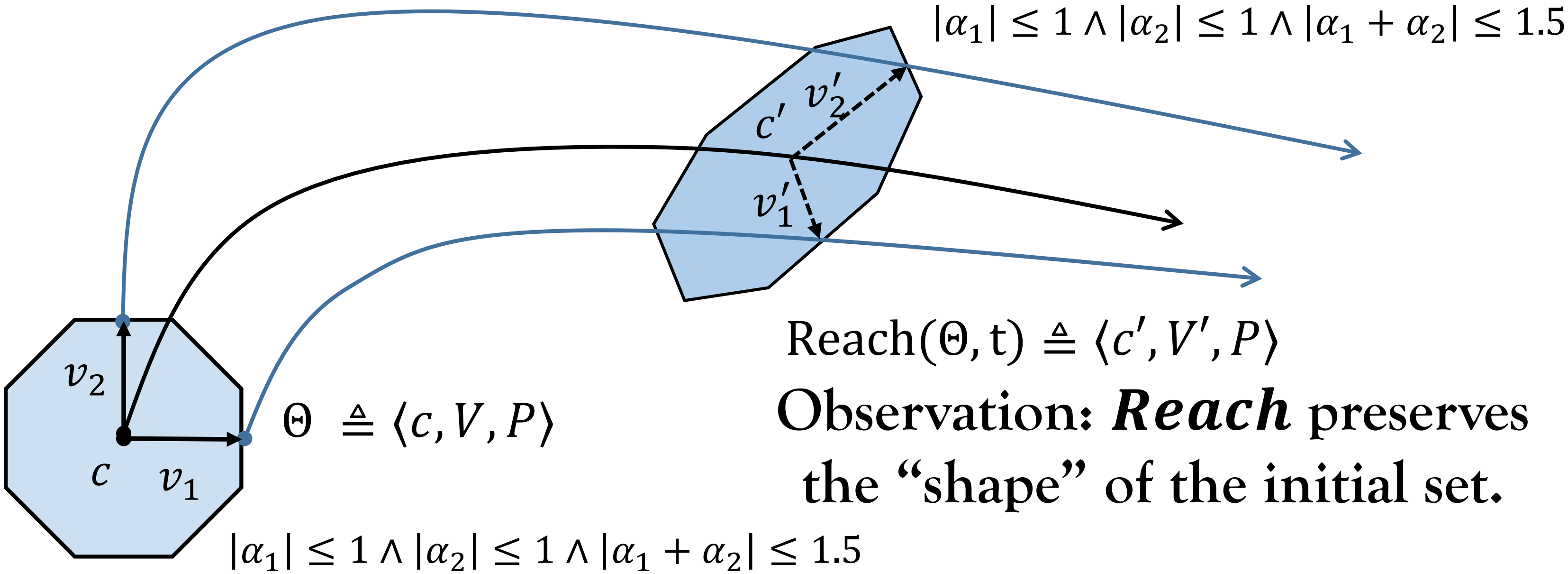
Reachable set at time t is given by $\langle c', V', P \rangle$ where

1. c' is the simulation corresponding to c
2. ${v_i}'$ is the difference of simulations from $c + v_i$ and from c

Technique
Representation + Superposition

Given $\Theta \triangleq \langle c, V, P \rangle$ to compute reachable set

1. Simulate from c
2. Simulate from $c + v_i$ for each i



$\mathrm{Reach}(\Theta, \mathrm{t}) \triangleq \langle c', V', P \rangle$

Observation: *Reach* preserves the "shape" of the initial set.

Reachable set at time t is given by $\langle c', V', P \rangle$ where

1. c' is the simulation corresponding to c
2. $v_i{}'$ is the difference of simulations from $c + v_i$ and from c

Using Discrete Time Simulation Engine

Initial set $\Theta \triangleq \langle c, V, P \rangle$; Simulation engine ρ; step size h;

For computing the reachable set at time $j \cdot h$ instant

1. Generate simulation $\rho(c, j \cdot h)$;
2. For each $v_i \in V$, generate simulation $\rho(c + v_i, j \cdot h)$;
3. Reachable set denoted as Θ_j is defined as $\langle c', V', P \rangle$ where
   1. $c' = \rho(c, j \cdot h)$;
   2. $v_i' = \rho(c + v_i,\ j \cdot h) - \rho(c, j \cdot h)$;

Using Discrete Time Simulation Engine

Initial set $\Theta \triangleq \langle c, V, P \rangle$; Simulation engine ρ; step size h;

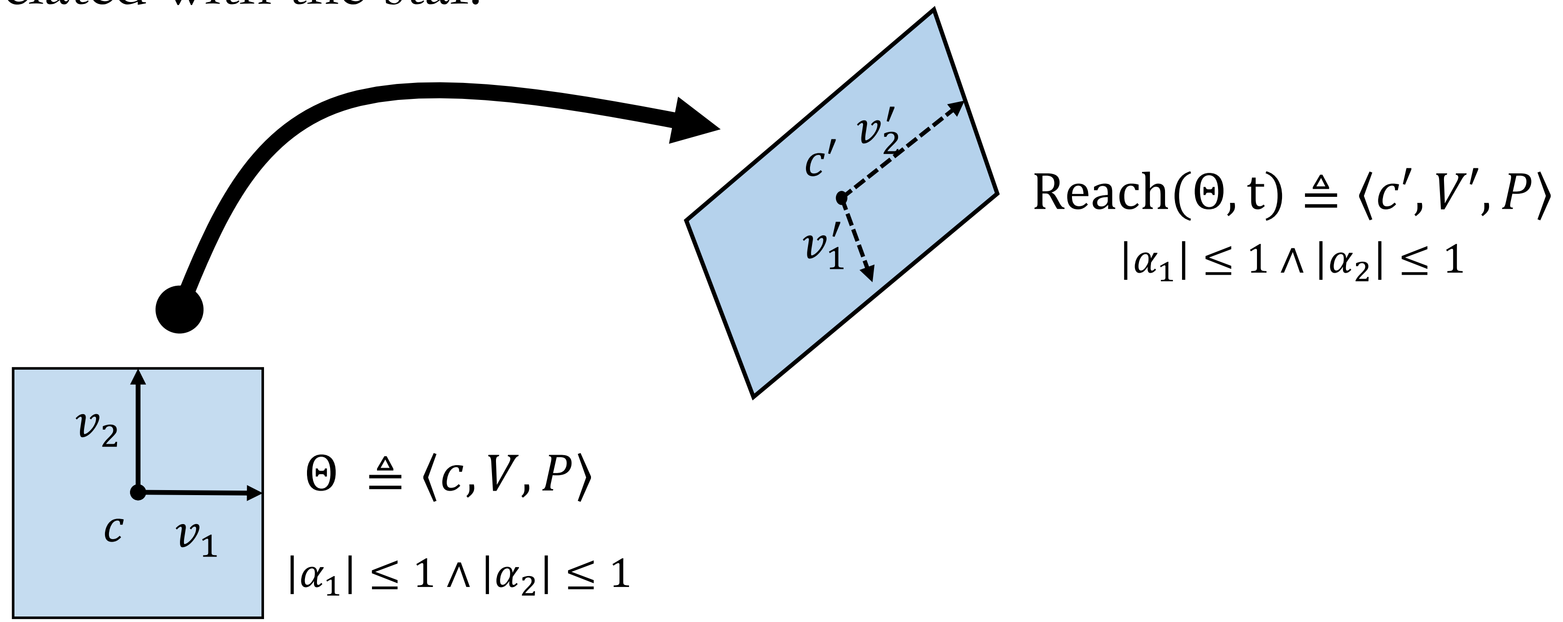
For computing the reachable set at time $j \cdot h$ instant

1. Generate simulation $\rho(c, j \cdot h)$;
2. For each $v_i \in V$, generate simulation $\rho(c + v_i, j \cdot h)$;
3. Reachable set denoted as Θ_j is defined as $\langle c', V', P \rangle$ where
   1. $c' = \rho(c, j \cdot h)$;
   2. $v_i' = \rho(c + v_i,\ j \cdot h) - \rho(c, j \cdot h)$;

Given initial set $\boldsymbol{\Theta}$, procedure $\mathbf{Reach}(\boldsymbol{\Theta}, \mathbf{h}, \mathbf{k} \cdot \boldsymbol{h})$ returns $\boldsymbol{\Theta_1}, \boldsymbol{\Theta_2}, \ldots, \boldsymbol{\Theta_k}$ where $\boldsymbol{\Theta_j} = \langle \boldsymbol{c_j}, \boldsymbol{V_j}, \boldsymbol{P} \rangle$ is the reachable set from Θ at time instance $j \cdot h$.

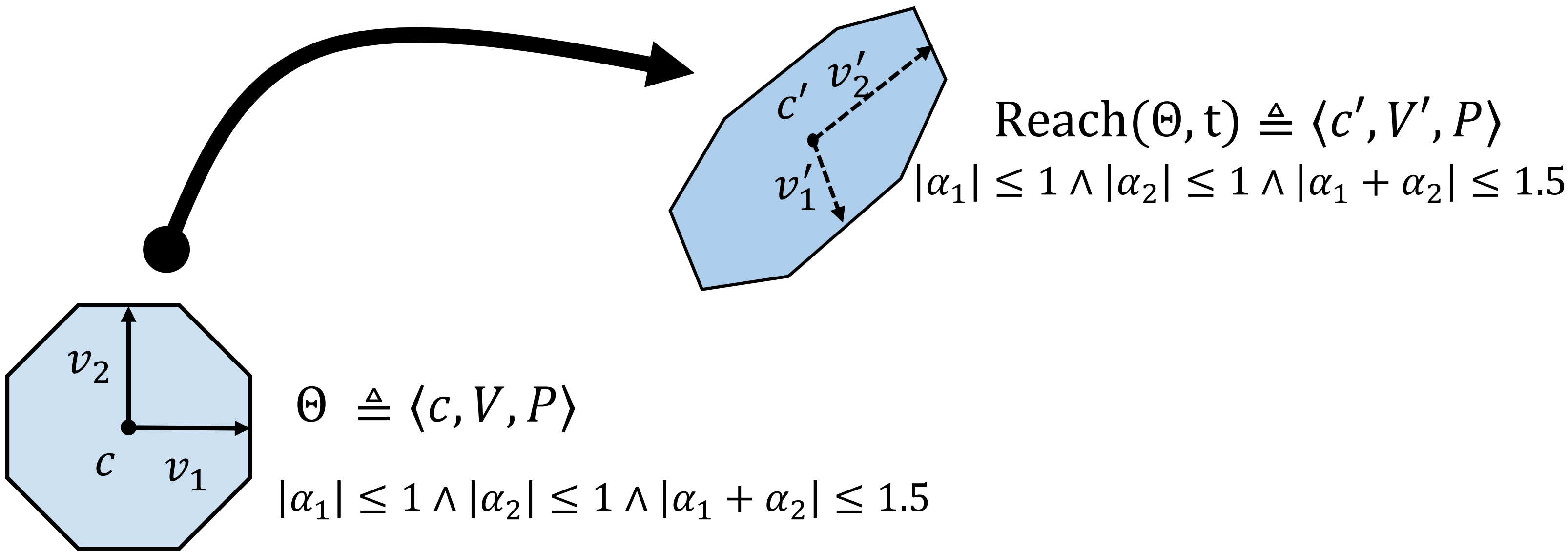
Observations

1. The discrete time reachable set doesn't change the predicate associated with the star.



Observations

1. The discrete time reachable set doesn't change the predicate associated with the star.

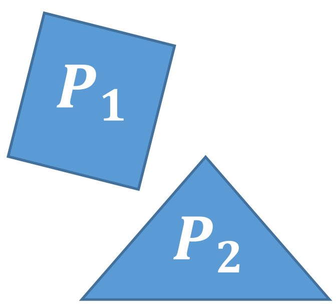


To compute reachable set of a new initial set, just changing the predicate suffices!

Observations

2. It is easy to aggregate and de-aggregate sets on-the-fly.

$\mathbf{\Theta_1} = \langle \boldsymbol{c}, \boldsymbol{V}, \boldsymbol{P_1} \rangle$



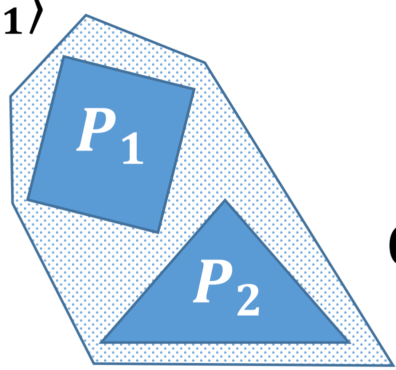
$\mathbf{\Theta_2} = \langle \boldsymbol{c}, \boldsymbol{V}, \boldsymbol{P_2} \rangle$

Notice: all have same center and basis in their representation

Observations

2. It is easy to aggregate and de-aggregate sets on-the-fly.

$\boldsymbol{\Theta_1 = \langle c, V, P_1 \rangle}$



$\boldsymbol{\Theta_{agg} = \langle c, V, P_{agg} \rangle}$

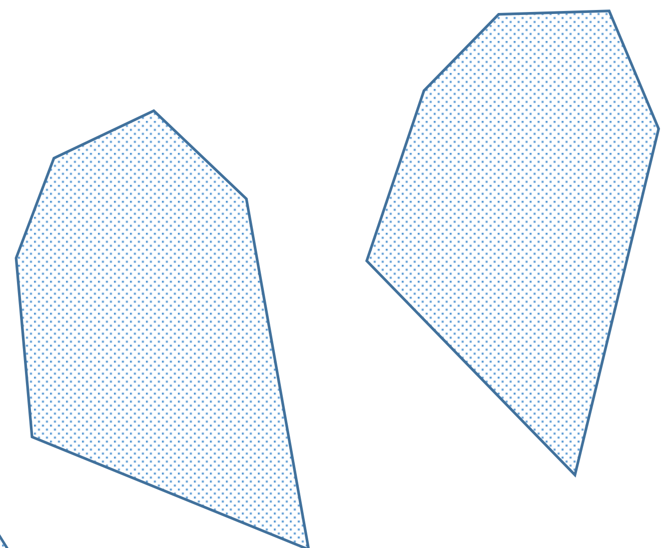
$\boldsymbol{(P_1 \vee P_2) \Rightarrow P_{agg}}$

$\boldsymbol{\Theta_2 = \langle c, V, P_2 \rangle}$

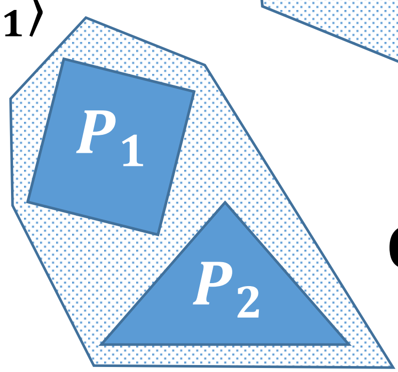
Notice: all have same center and basis in their representation

Observations

2. It is easy to aggregate and de-aggregate sets on-the-fly.



$$\Theta_1 = \langle c, V, P_1 \rangle$$



$$\Theta_{agg} = \langle c, V, P_{agg} \rangle$$

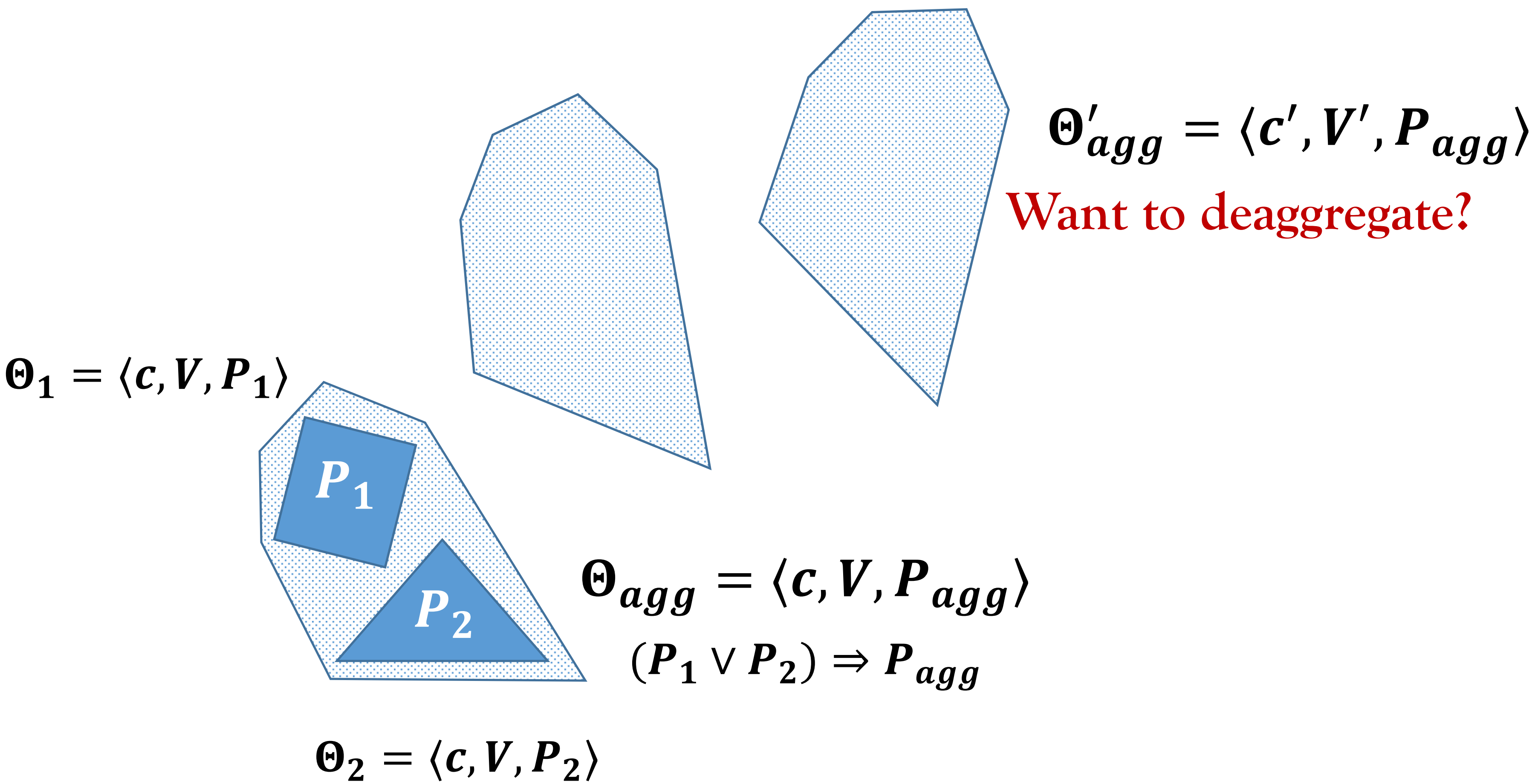
$$(P_1 \vee P_2) \Rightarrow P_{agg}$$

$$\Theta_2 = \langle c, V, P_2 \rangle$$

Notice: all have same center and basis in their representation

Observations

2. It is easy to aggregate and de-aggregate sets on-the-fly.

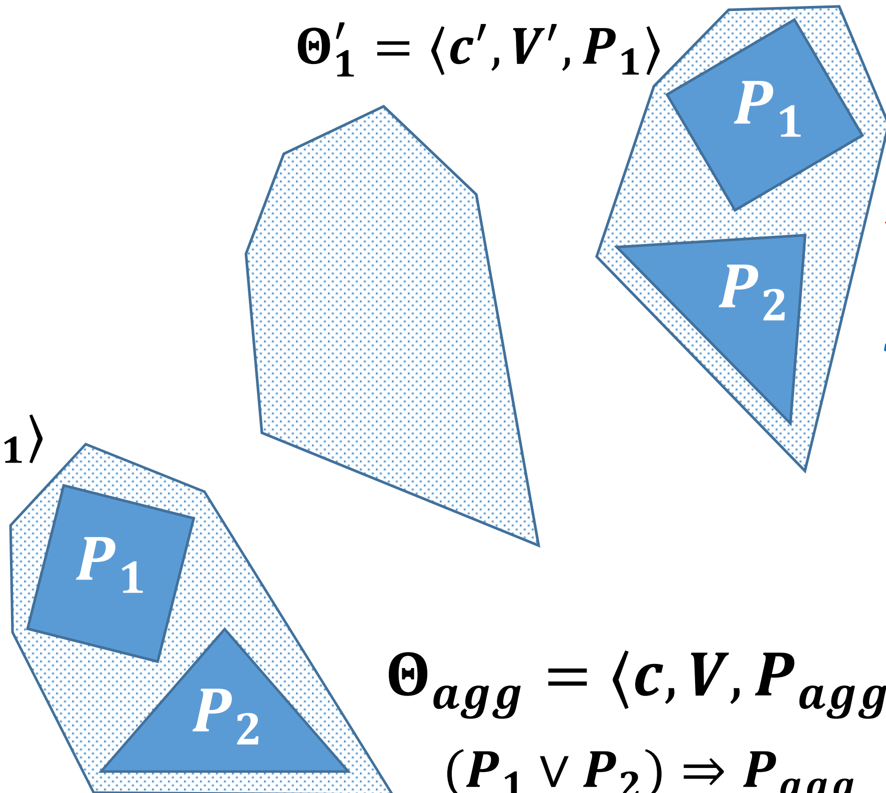


Notice: all have same center and basis in their representation

Observations

2. It is easy to aggregate and de-aggregate sets on-the-fly.

$\Theta_1' = \langle c', V', P_1 \rangle$



$\Theta_{agg}' = \langle c', V', P_{agg} \rangle$

Want to deaggregate?

Just change the predicates!

$\Theta_2' = \langle c', V', P_2 \rangle$

$\Theta_1 = \langle c, V, P_1 \rangle$

$\Theta_{agg} = \langle c, V, P_{agg} \rangle$

$(P_1 \vee P_2) \Rightarrow P_{agg}$

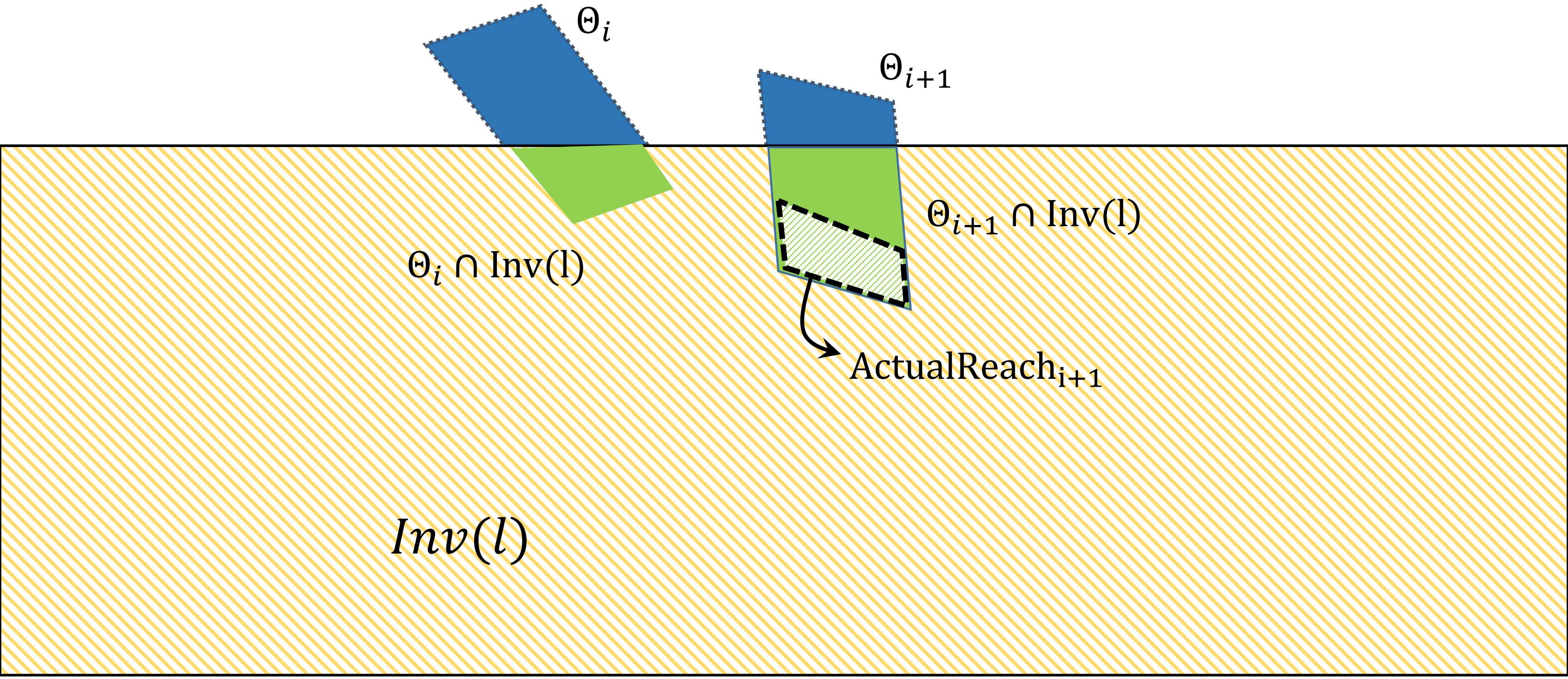
$\Theta_2 = \langle c, V, P_2 \rangle$

Notice: all have same center and basis in their representation

Handling Invariants and Discrete Transitions

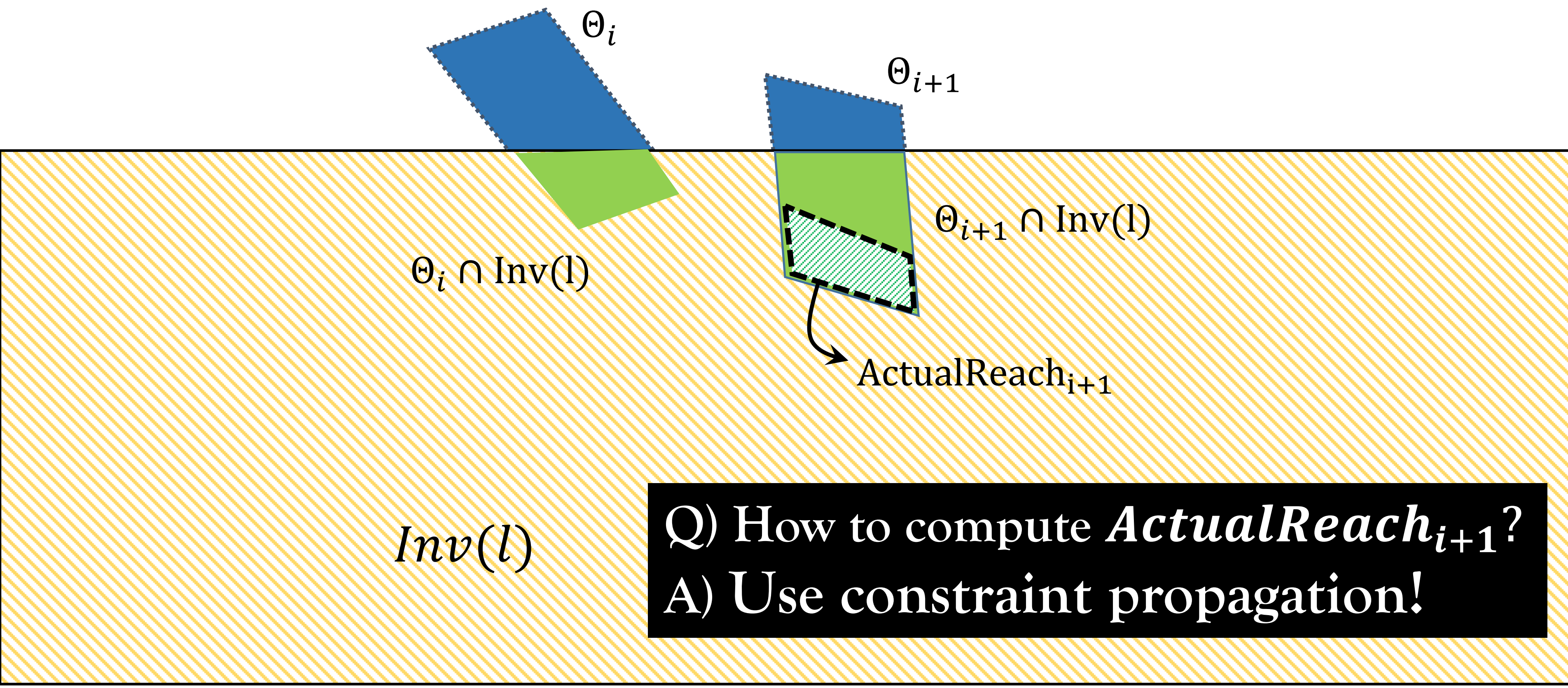
The Problems With Invariants

- Given $\Theta_1, \Theta_2, \ldots, \Theta_k$ as discrete time reachable sets for a given mode, performing just $\Theta_j \cap Inv$ only gives an overapproximation.



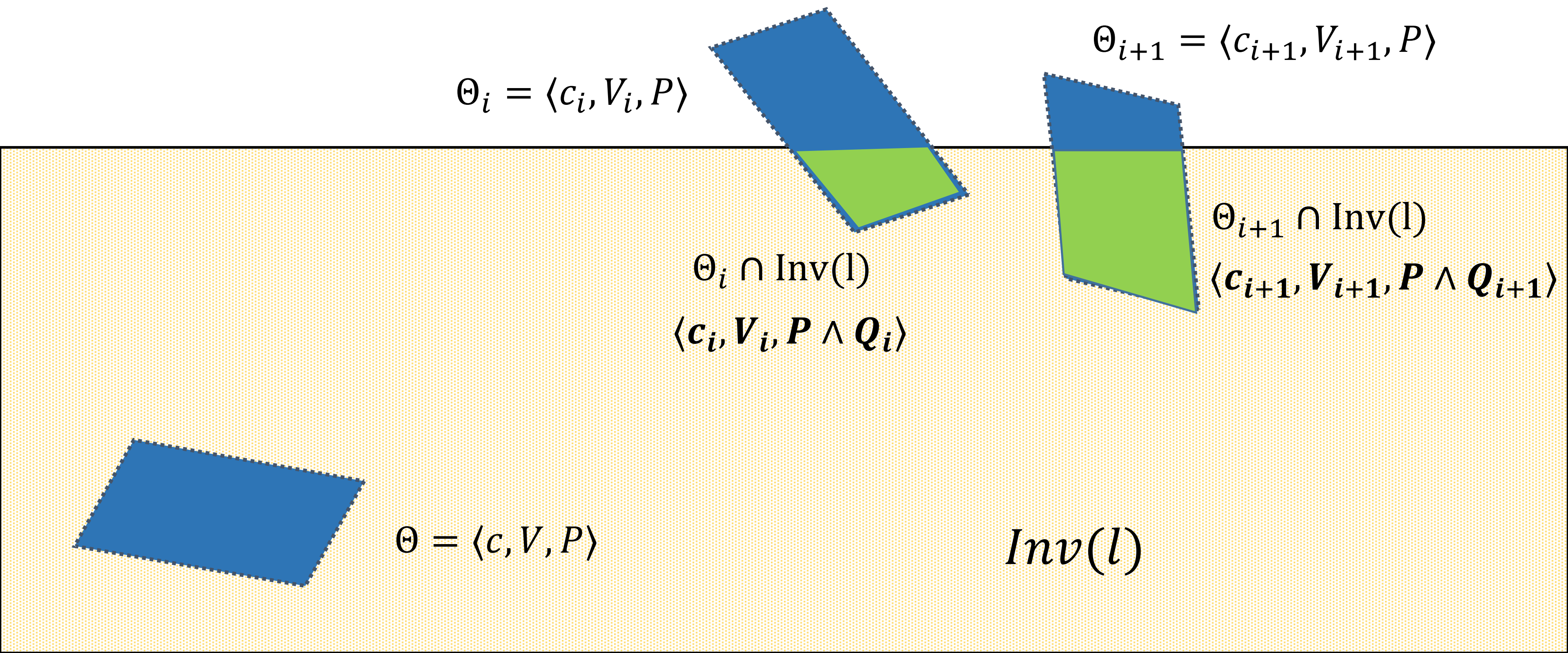
The Problems With Invariants

- Given $\Theta_1, \Theta_2, \ldots, \Theta_k$ as discrete time reachable sets for a given mode, performing just $\Theta_j \cap Inv$ only gives an overapproximation.



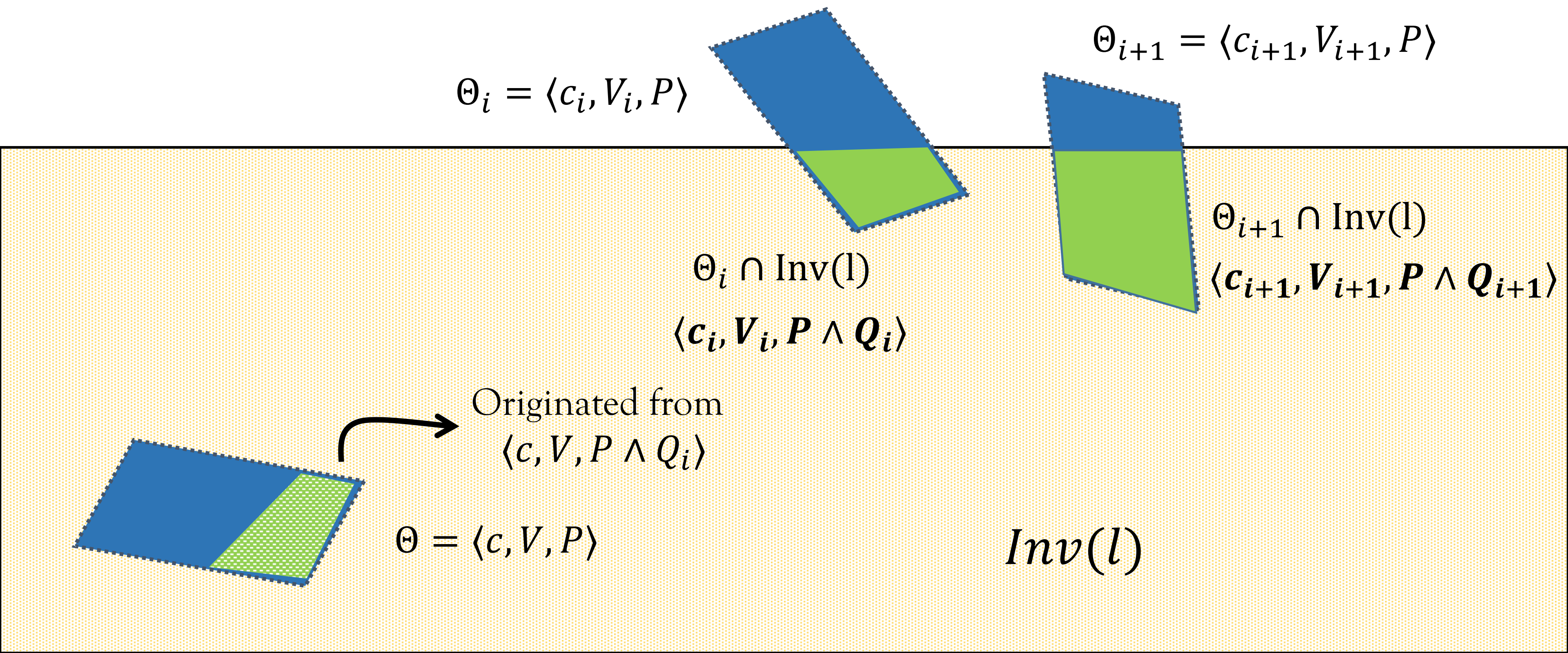
Forward Constraint Propagation

1. Convert Inv into the center and basis of i^{th} star as $\langle c_i, V_i, Q_i \rangle$.
2. $\Theta \cap Inv = \langle c_i, V_i, P \wedge Q_i \rangle$



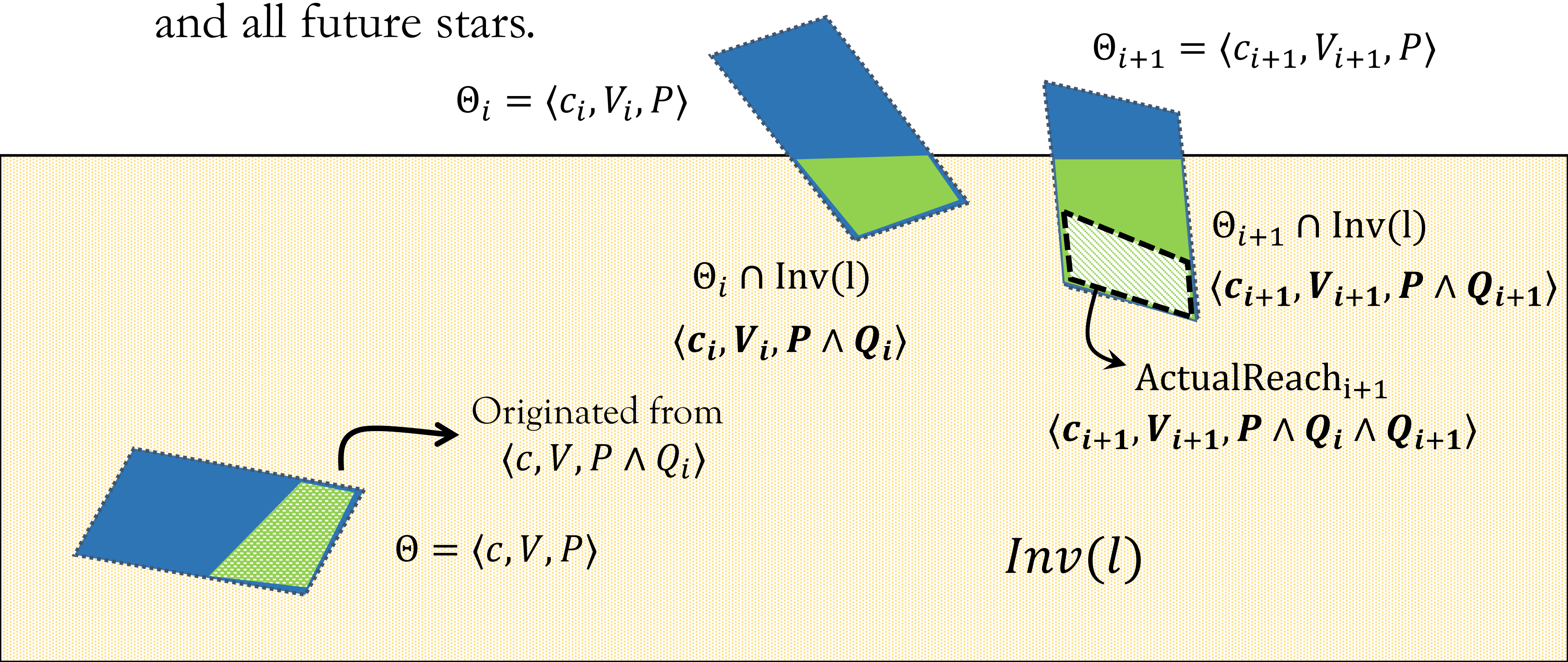
Forward Constraint Propagation

1. Convert Inv into the center and basis of i^{th} star as $\langle c_i, V_i, Q_i \rangle$.
2. $\Theta \cap Inv = \langle c_i, V_i, P \wedge Q_i \rangle$
3. These should originate from $\langle c, V, P \wedge Q_i \rangle$ in Θ



Forward Constraint Propagation

1. Convert Inv into the center and basis of i^{th} star as $\langle c_i, V_i, Q_i \rangle$.
2. $\Theta \cap Inv = \langle c_i, V_i, P \wedge Q_i \rangle$
3. These should originate from $\langle c, V, P \wedge Q_i \rangle$ in Θ
4. Propagate constraint Q_i forward --- add it to predicates of itself and all future stars.



Invariant Constraint Propagation

1. Compute reachable sets $\Theta_1, \Theta_2, \ldots, \Theta_k$.

2. Convert Inv into star representation of Θ_i as $\langle c_1, V_1, Q_1 \rangle, \langle c_2, V_2, Q_2 \rangle, \ldots, \langle c_k, V_k, Q_k \rangle$

3. For each Θ_i, add $Q_1 \wedge Q_2 \wedge \cdots \wedge Q_i$ into its predicate.

Invariant Constraint Propagation

1. Compute reachable sets $\Theta_1, \Theta_2, \ldots, \Theta_k$.
2. Convert Inv into star representation of Θ_i as $\langle c_1, V_1, Q_1 \rangle, \langle c_2, V_2, Q_2 \rangle, \ldots, \langle c_k, V_k, Q_k \rangle$
3. For each Θ_i, add $Q_1 \wedge Q_1 \wedge \cdots \wedge Q_i$ into its predicate.

Isn't this expensive?

Invariant Constraint Propagation

1. Compute reachable sets $\Theta_1, \Theta_2, \ldots, \Theta_k$.
2. Convert Inv into star representation of Θ_i as $\langle c_1, V_1, Q_1 \rangle, \langle c_2, V_2, Q_2 \rangle, \ldots, \langle c_k, V_k, Q_k \rangle$
3. For each Θ_i, add $Q_1 \wedge Q_1 \wedge \cdots \wedge Q_i$ into its predicate.

No. of predicates increase linearly with time?

Isn't this expensive?

Optimizations

1. If $\Theta_i \subseteq Inv$, then $P \wedge Q_i \equiv P$. Hence, no constraint is added.

2. If $\Theta_i \subseteq Inv^c$, then $P \wedge Q_i \equiv \bot$. Hence, no need to add Q_i.

Optimizations

1. If $\Theta_i \subseteq Inv$, then $P \wedge Q_i \equiv P$. Hence, no constraint is added.

2. If $\Theta_i \subseteq Inv^c$, then $P \wedge Q_i \equiv \bot$. Hence, no need to add Q_i.

3. Add a constraint Q_i to $P \wedge Q_1 \wedge \cdots \wedge Q_{i-1}$ if and only if
$$\neg(P \wedge Q_1 \wedge \cdots \wedge Q_{i-1} \Rightarrow Q_i)$$

Optimizations

1. If $\Theta_i \subseteq Inv$, then $P \wedge Q_i \equiv P$. Hence, no constraint is added.

2. If $\Theta_i \subseteq Inv^c$, then $P \wedge Q_i \equiv \bot$. Hence, no need to add Q_i.

3. Add a constraint Q_i to $P \wedge Q_1 \wedge \cdots \wedge Q_{i-1}$ if and only if
$$\neg(P \wedge Q_1 \wedge \cdots \wedge Q_{i-1} \Rightarrow Q_i)$$

4. **[Empirical heuristic]:** Compare successive constraints Q_i and Q_{i+1} and if Q_{i+1} is stronger than Q_i, replace Q_i with Q_{i+1}.

Discrete Transitions

- Discrete transitions are enabled when the reachable set overlaps with the guard condition.
- If reachable set from Θ overlaps with guard G_i at $\Theta_{i,1}, \Theta_{i,2}, \dots, \Theta_{i,l}$. That is, Θ has l successor sets.
- After m discrete transitions, the number of sets to keep track will be l^m. (exponential blow-up).

Discrete Transitions

- Discrete transitions are enabled when the reachable set overlaps with the guard condition.
- If reachable set from Θ overlaps with guard G_i at $\Theta_{i,1}, \Theta_{i,2}, \dots, \Theta_{i,l}$. That is, Θ has l successor sets.
- After m discrete transitions, the number of sets to keep track will be l^m. (exponential blow-up).

Solution: Aggregation

Aggregation – A Necessary Evil

- Necessary to reduce the number of sets to keep track of.

Aggregation – A Necessary Evil

- Necessary to reduce the number of sets to keep track of.
- Aggregation introduces overapproximation that we can never get rid of!
- Might cause spurious discrete transitions; cannot give concrete counterexamples.

Aggregation – A Necessary Evil

- Necessary to reduce the number of sets to keep track of.
- Aggregation introduces overapproximation that we can never get rid of!
- Might cause spurious discrete transitions; cannot give concrete counterexamples.

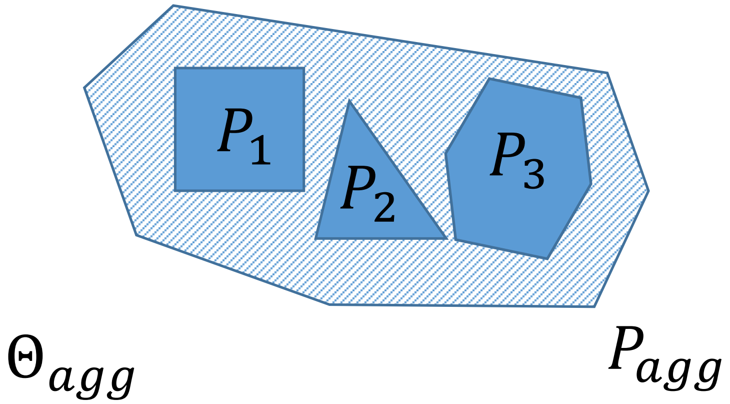
Aggregation – A Necessary Evil

- Necessary to reduce the number of sets to keep track of.
- Aggregation introduces overapproximation that we can never get rid of!
- Might cause spurious discrete transitions; cannot give concrete counterexamples.

Damned if you do!
Damned if you don't!

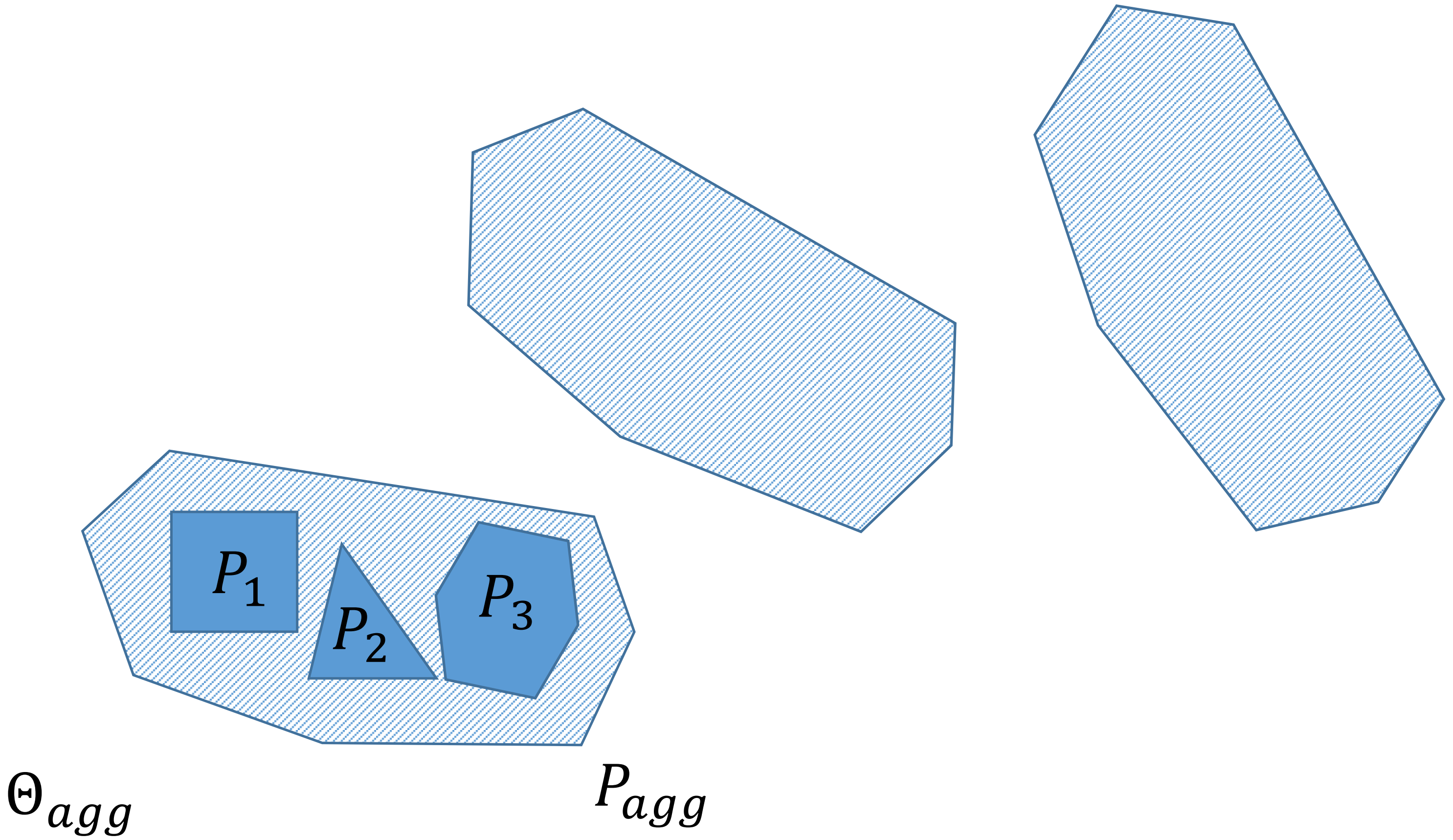
Dynamic Aggregation Illustration

1. Aggregate all the sets by default and compute reachable set.



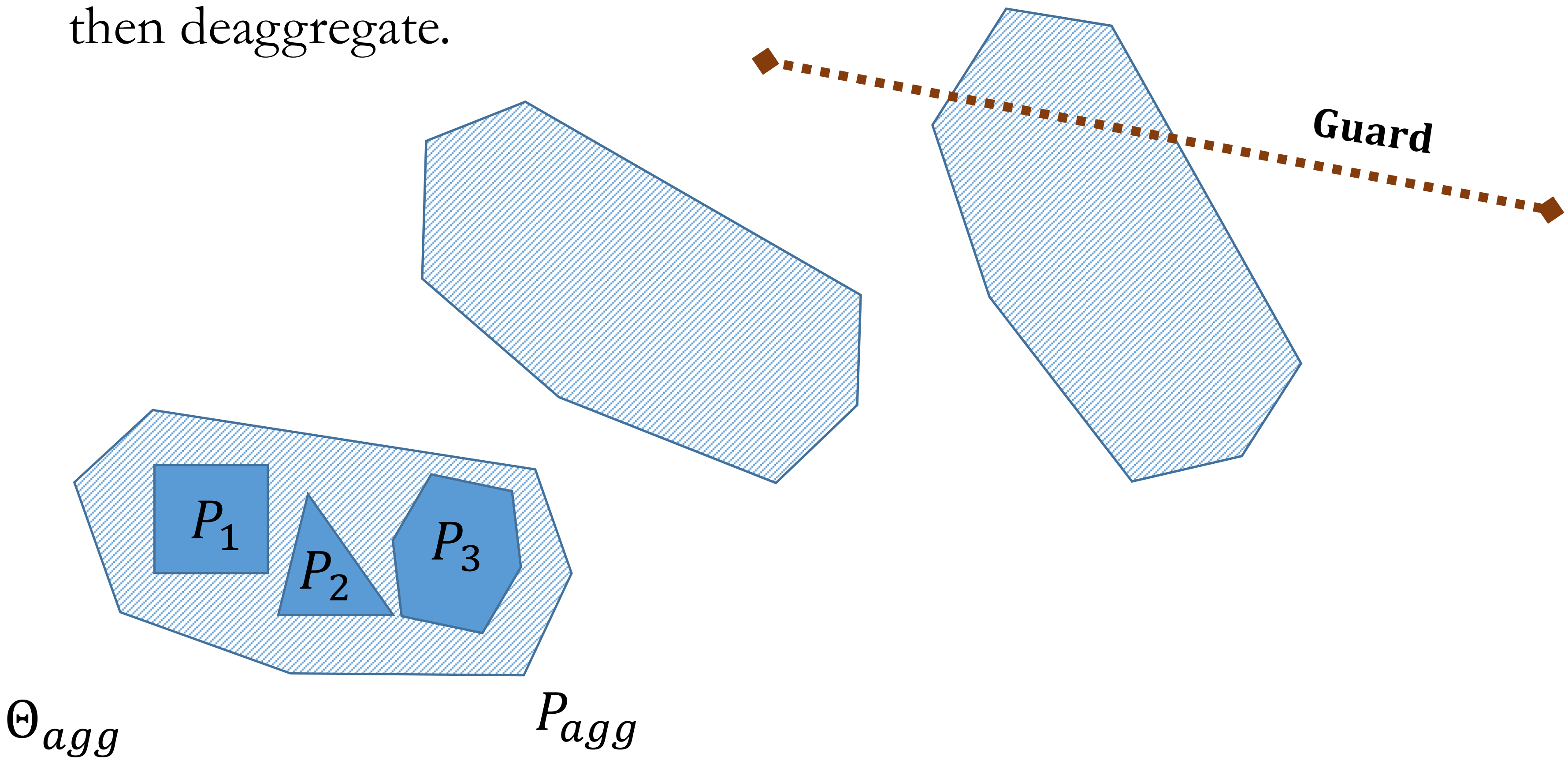
Dynamic Aggregation Illustration

1. Aggregate all the sets by default and compute reachable set.



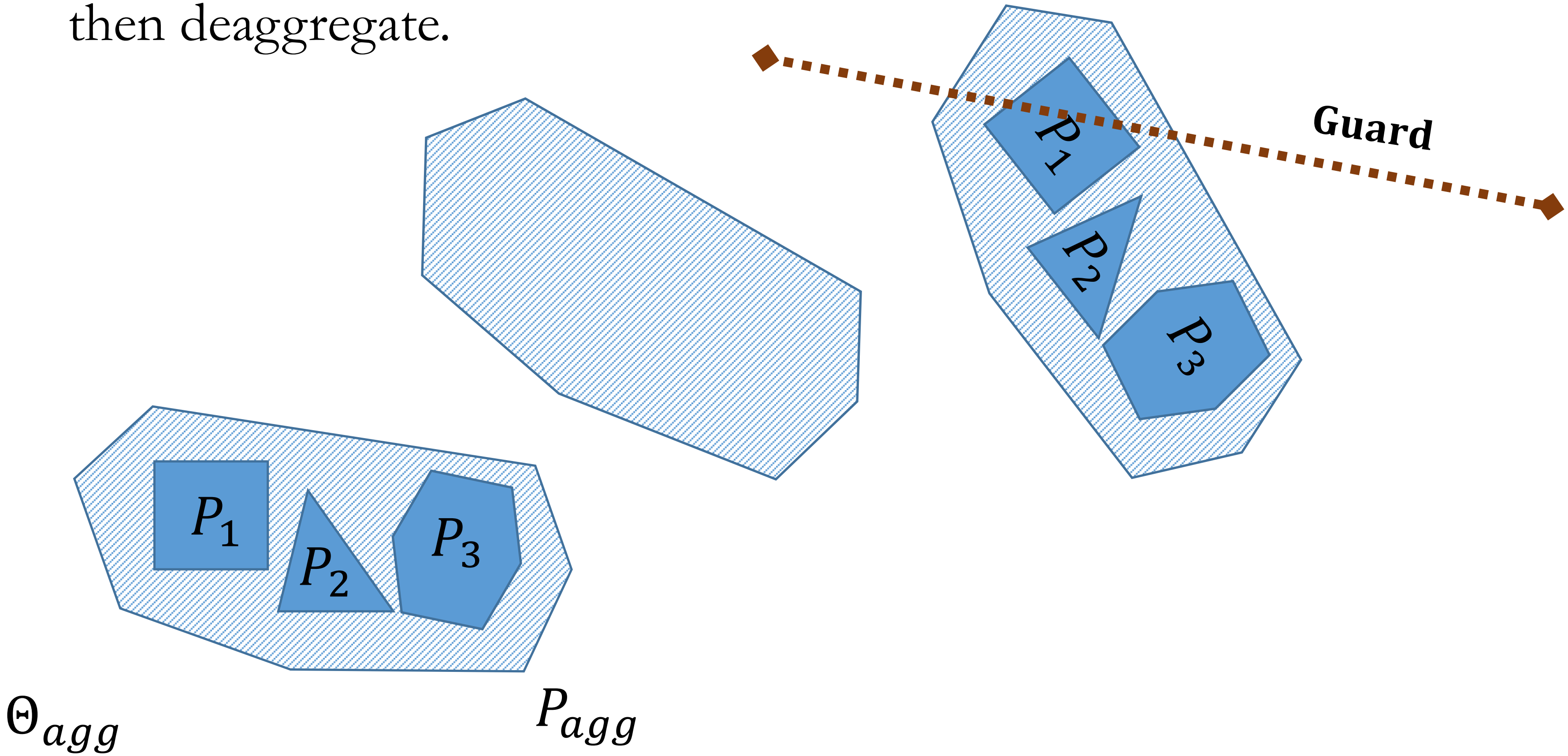
Dynamic Aggregation Illustration

1. Aggregate all the sets by default and compute reachable set.
2. When the aggregated set intersects with a guard or unsafe set, then deaggregate.



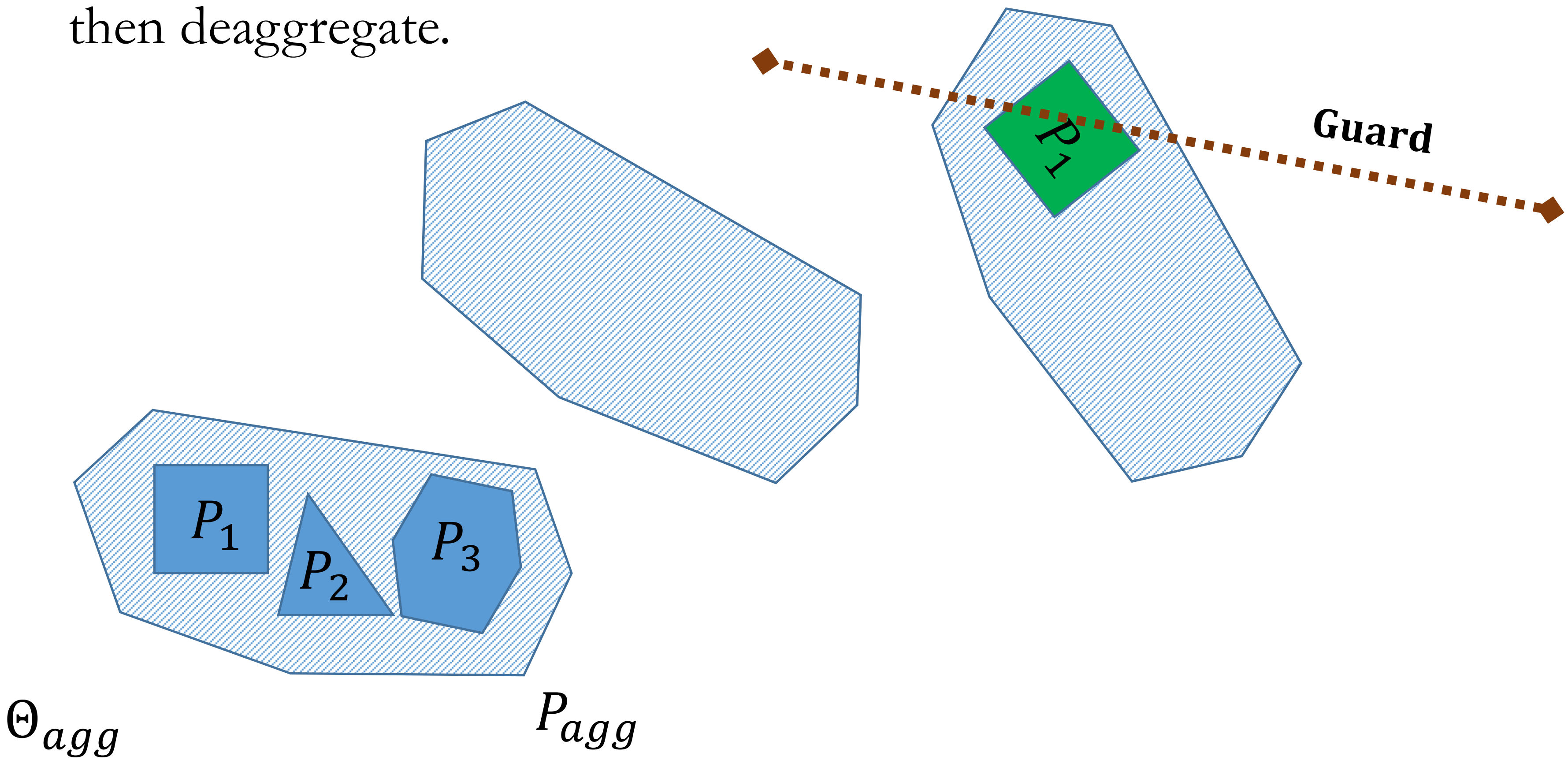
Dynamic Aggregation Illustration

1. Aggregate all the sets by default and compute reachable set.
2. When the aggregated set intersects with a guard or unsafe set, then deaggregate.



Dynamic Aggregation Illustration

1. Aggregate all the sets by default and compute reachable set.
2. When the aggregated set intersects with a guard or unsafe set, then deaggregate.

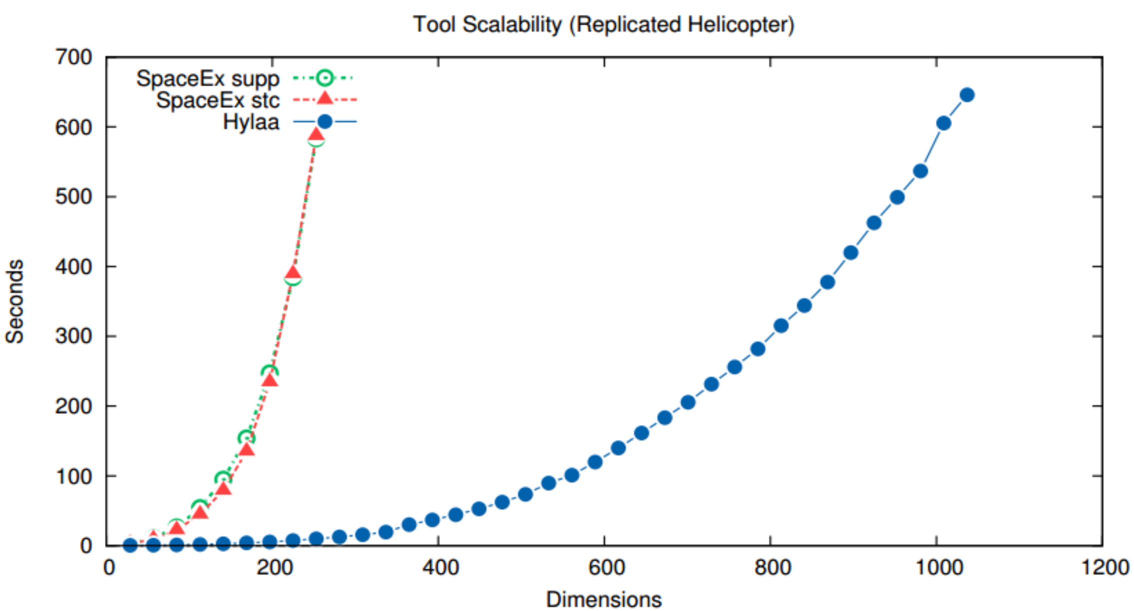


Overview

- ✓Motivation and Contributions.
- ✓Dynamic analysis technique for linear systems verification.
- ✓Observations of the dynamic analysis technique.
- ✓Invariant constraint propagation.
- ✓Dynamic deaggregation.
- ▪ Experimental evaluation.
- ▪ Conclusions and Future work.

Experimental Evaluation
HyLAA

Scalability with respect to number of dimensions.***

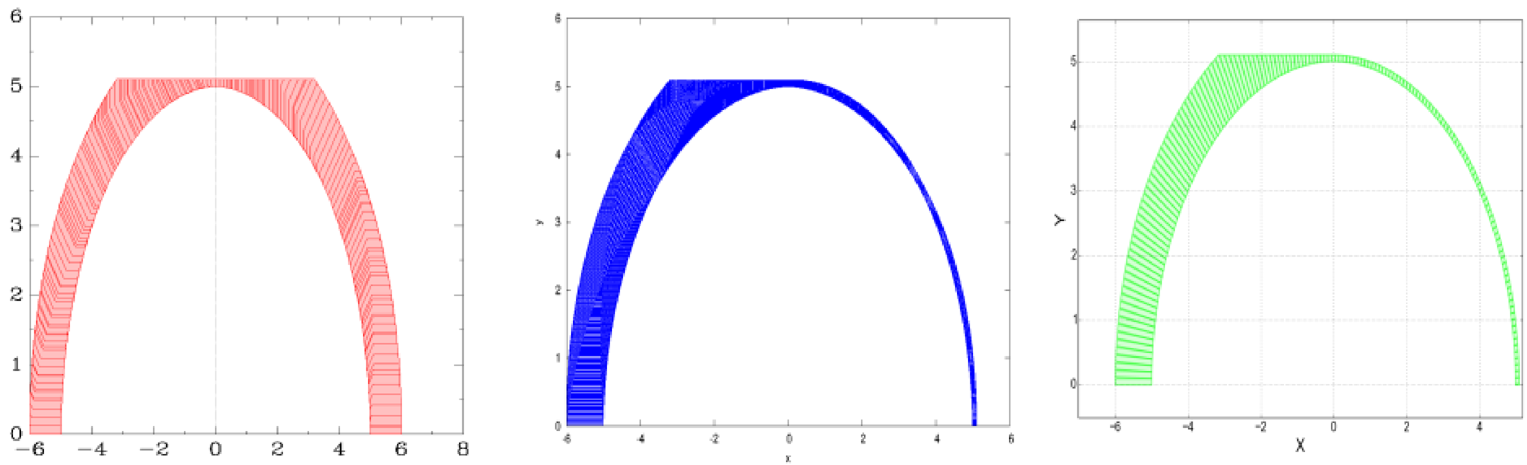


# Dims	supp	stc	HyLAA
29	2.98	2.60	0.42
57	10.93	9.48	0.67
141	94.83	79.23	2.65
253	583.27	587.42	9.79
449	-	-	52.67
1009	-	-	605.38

*** accurate comparison of tools is very hard owing to semantics and parameters during verification. HyPro might be a good solution.

http://stanleybak.com/hylaa/

HyLAA
Constraint Propagation

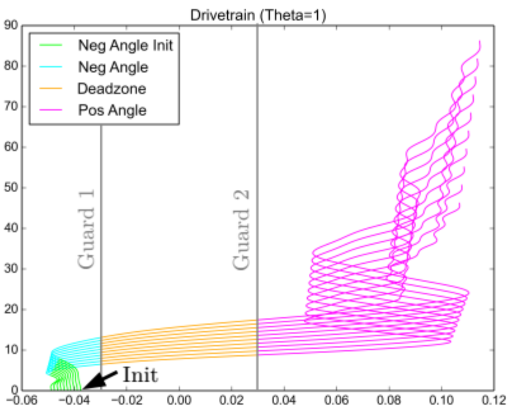


(a) SpaceEx `stc` (b) Flow* (c) HyLAA

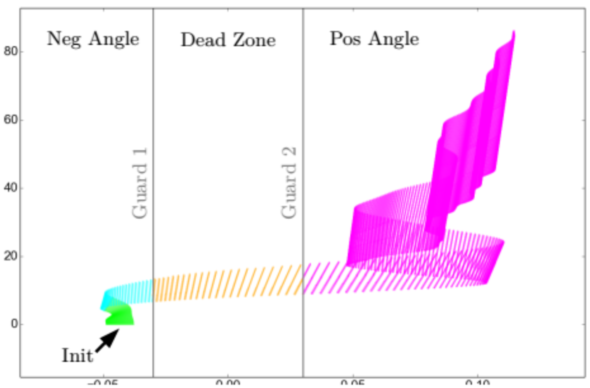
Step	No Trim	Trim
0.05	16	5
0.005	119	9
0.001	576	25
0.0005	1148	45

http://stanleybak.com/hylaa/

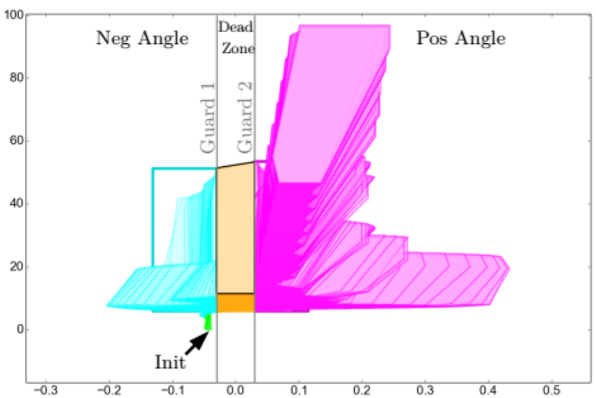
HyLAA
Aggregation and Deaggregation



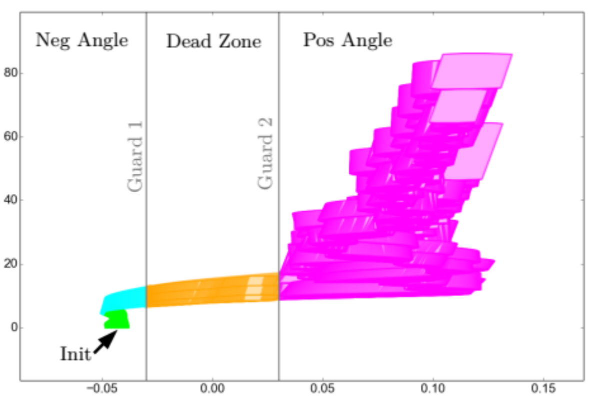
(a) Simulations



(b) Unaggregated



(c) Aggregated (incomplete)



(d) Deaggregated

- Without aggregation is very expensive
- Completely aggregated introduces new transitions and doesn't terminate.
- Dynamic deaggregation has 1.2x – 5x speedup based on the system.

http://stanleybak.com/hylaa/

HyLAA
Aggregation and Deaggregation

# Dims	10	12	14	16	18	20	24	30	42
Deaggregated	25.70	44.94	24.71	131.82	47.72	267.71	450.42	331.57	516.21
Unaggregated	112.94	79.24	98.63	145.87	214.80	409.55	561.47	384.55	672.60

- Automotive drivetrain system with additional masses ($8 + 2\theta$).
- In lower dimensions, the synchronous behavior of masses gives a better performance for aggregation.
- In higher dimensions, the benefits of aggregation are low because deaggregation is performed more often.

http://stanleybak.com/hylaa/

Conclusion

- Notion of simulation equivalent reachable set and safety verification.
- New invariant constraint propagation methods for handling invariants.
- Dynamic aggregation and deaggregation for handling discrete transitions.
- Implemented these in a tool called HyLAA and demonstrated the benefits of these techniques.

Future work

- Giving guarantees over *dense-time* semantics.
- Templates for aggregation and deaggregation.

Recently verified 10,000 dimensional system using enhancements on HyLAA.

http://stanleybak.com/hylaa/

Conclusion

- Notion of simulation equivalent reachable set and safety verification.
- New invariant constraint propagation methods for handling invariants.
- Dynamic aggregation and deaggregation for handling discrete transitions.
- Implemented these in a tool called HyLAA and demonstrated the benefits of these techniques.

Future work

- Giving guarantees over *dense-time* semantics.
- Templates for aggregation and deaggregation.

Recently verified 10,000 dimensional system using enhancements on HyLAA.

http://stanleybak.com/hylaa/