FuseIC3

An Algorithm for Checking Large Design Spaces

Rohit Dureja and Kristin Yvonne Rozier

IOWA STATE UNIVERSITY + Laboratory for Temporal Logic

fmcad.17

October 5, 2017

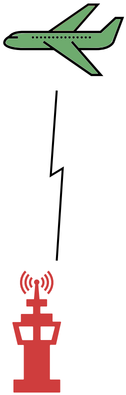
What is a Design Space?

Airspace Allocation

Design of allocation policies to ensure all airborne agents are safely separated.

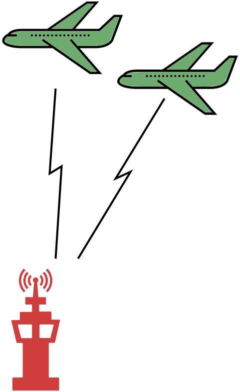
Airspace Allocation

Design of allocation policies to ensure all airborne agents are safely separated.



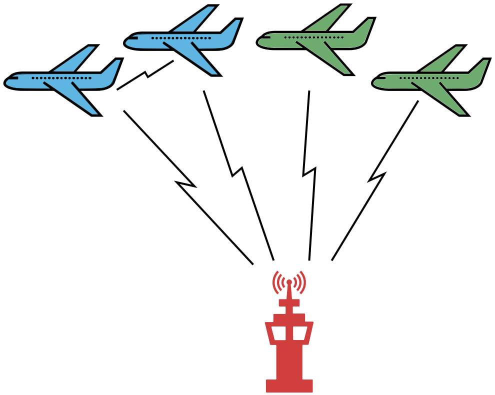
Airspace Allocation

Design of allocation policies to ensure all airborne agents are safely separated.



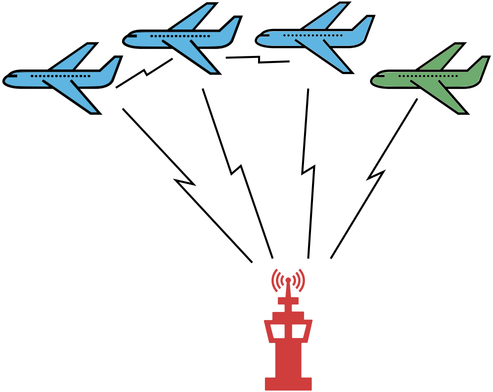
Airspace Allocation

Design of allocation policies to ensure all airborne agents are safely separated.



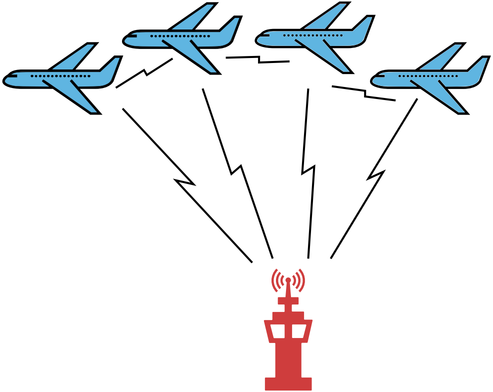
Airspace Allocation

Design of allocation policies to ensure all airborne agents are safely separated.



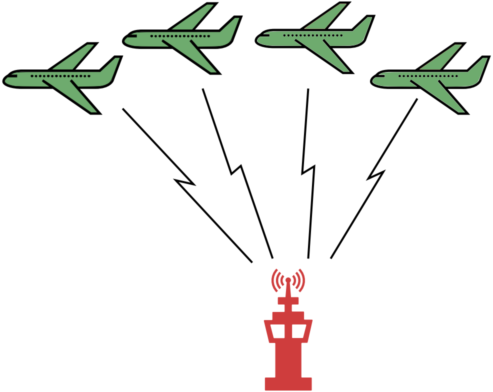
Airspace Allocation

Design of allocation policies to ensure all airborne agents are safely separated.



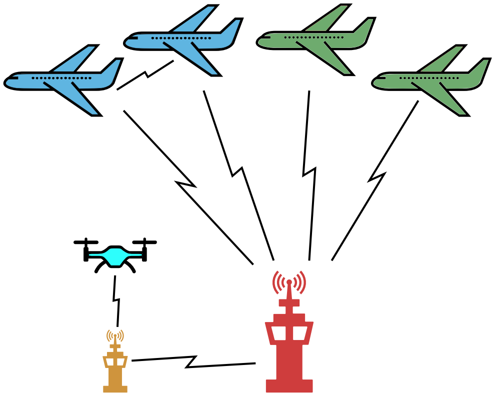
Airspace Allocation

Design of allocation policies to ensure all airborne agents are safely separated.



Airspace Allocation

Design of allocation policies to ensure all airborne agents are safely separated.



Airspace Allocation

Design of allocation policies to ensure all airborne agents are safely separated.

Lots of design choices!

What is a Design Space?

What is a Design Space?

Set of Design Choices for a System.

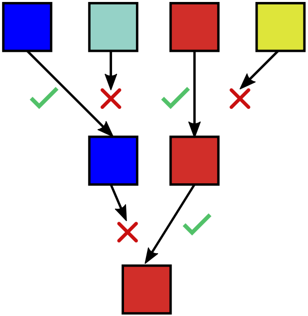
Design Problem

Design Problem

Complex systems are often modeled as **design spaces**.

Design Problem

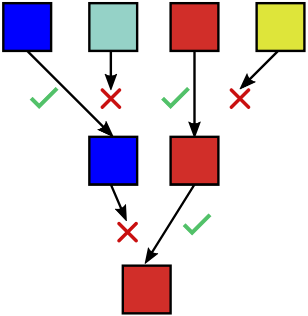
Complex systems are often modeled as **design spaces**.



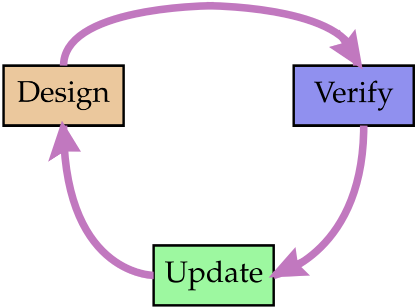
Alternative Comparison

Design Problem

Complex systems are often modeled as **design spaces**.



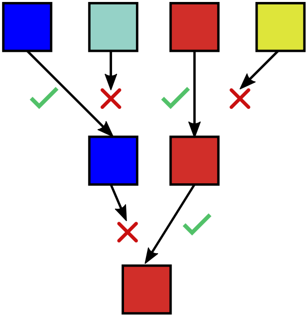
Alternative Comparison



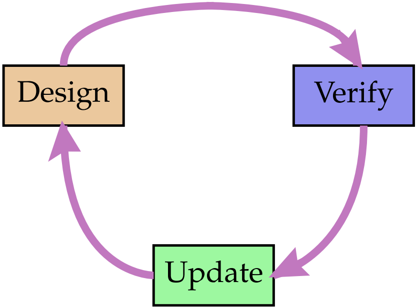
Incremental Development

Design Problem

Complex systems are often modeled as **design spaces**.



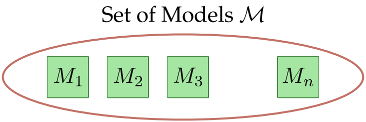
Alternative Comparison



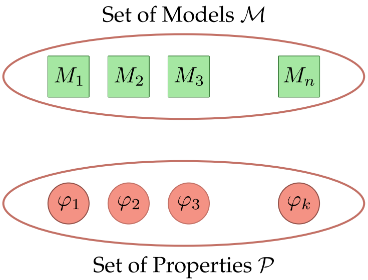
Incremental Development

Model checking!

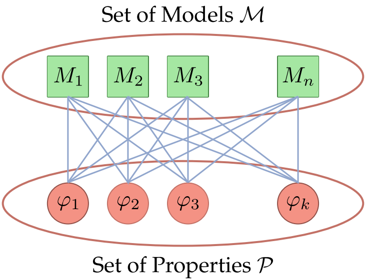
Classical Model Checking of a Design Space



Classical Model Checking of a Design Space

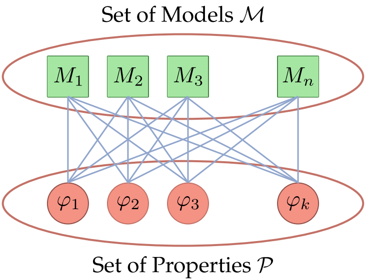


Classical Model Checking of a Design Space



For every $M \in \mathcal{M}$ and $\varphi \in \mathcal{P}$
check if $M \models \varphi$?

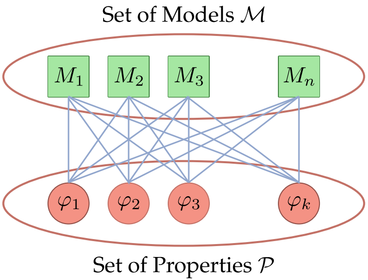
Classical Model Checking of a Design Space



For every $M \in \mathcal{M}$ and $\varphi \in \mathcal{P}$ check if $M \models \varphi$?

- Inefficient for large design spaces
  - may not scale to handle combinatorial size of the design space.

Classical Model Checking of a Design Space



For every $M \in \mathcal{M}$ and $\varphi \in \mathcal{P}$
check if $M \models \varphi$?

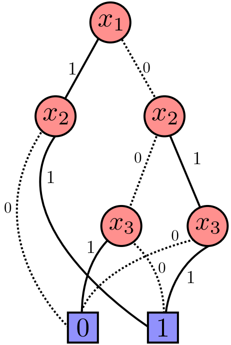
- Inefficient for large design spaces
  - may not scale to handle combinatorial size of the design space.

Can we do better?

Related Work

1. Reusing BDD variable orderings

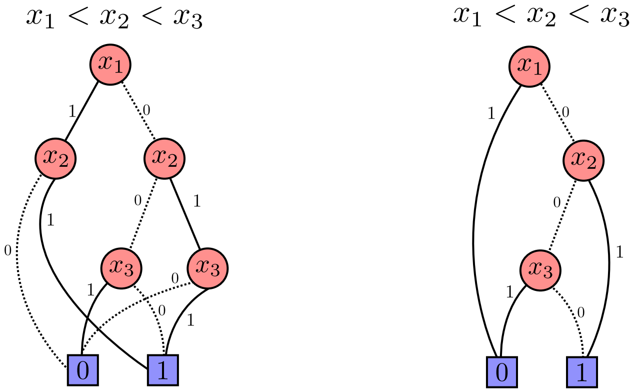
$$x_1 < x_2 < x_3$$



(Beer et al., 1996; Yang et al., 1998)

Related Work

1. Reusing BDD variable orderings

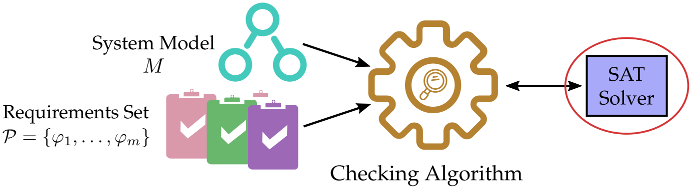


FuseIC3 is SAT-based

(Beer et al., 1996; Yang et al., 1998)

Related Work

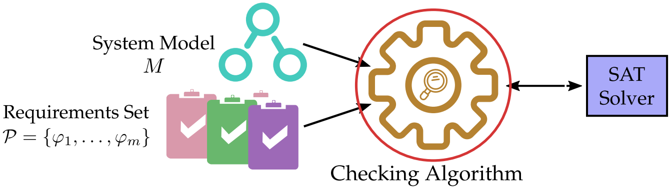
2. SAT solver optimizations and clause reuse



(Marques-Silva, 2007; Schrammel et al., 2016; Chockler et al., 2011; Khasidashvili et al., 2006; Khasidashvili & Nadel, 2012)

Related Work

2. SAT solver optimizations and clause reuse

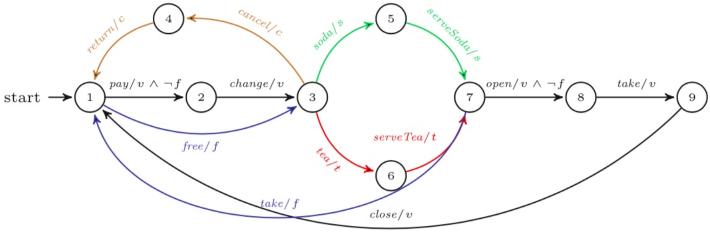


FuseIC3 reuses model checking artifacts

(Marques-Silva, 2007; Schrammel et al., 2016; Chockler et al., 2011; Khasidashvili et al., 2006; Khasidashvili & Nadel, 2012)

Related Work

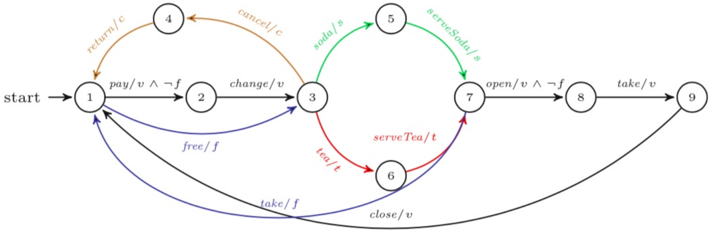
3. Software product line verification



(Ben-David et al., 2015; Classen et al., 2012, 2011, 2010; Dimovski et al., 2015)

Related Work

3. Software product line verification



FuseIC3 does not require custom modeling

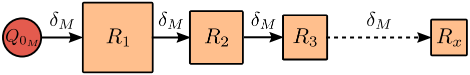
(Ben-David et al., 2015; Classen et al., 2012, 2011, 2010; Dimovski et al., 2015)

High-level View of IC3/PDR

Model $M = (\Sigma, Q_M, Q_{0_M}, \delta_M)$ and Safety property φ

High-level View of IC3/PDR

Model $M = (\Sigma, Q_M, Q_{0_M}, \delta_M)$ and Safety property φ



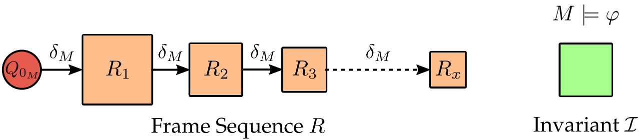
Frame Sequence R

Frame Invariants

1. for $i > 0$, R_i is CNF, over-approximated states reachable in up to i steps
2. $R_{i+1} \subseteq R_i$ (monotonic)
3. $R_i \wedge \delta_M \models R'_{i+1}$
4. for $i < x$, $R_i \models \varphi$

High-level View of IC3/PDR

Model $M = (\Sigma, Q_M, Q_{0_M}, \delta_M)$ and Safety property φ



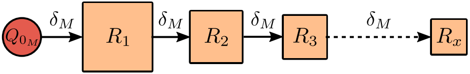
Frame Invariants

1. for $i > 0$, R_i is CNF, over-approximated states reachable in up to i steps
2. $R_{i+1} \subseteq R_i$ (monotonic)
3. $R_i \wedge \delta_M \models R'_{i+1}$
4. for $i < x$, $R_i \models \varphi$

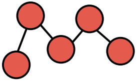
High-level View of IC3/PDR

Model $M = (\Sigma, Q_M, Q_{0_M}, \delta_M)$ and Safety property φ

$M \not\models \varphi$



Frame Sequence R



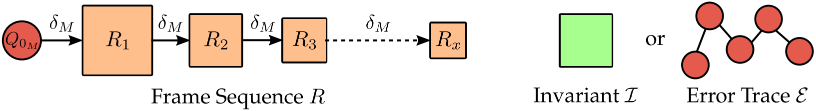
Error Trace $\mathcal{E}$

Frame Invariants

1. for $i > 0$, R_i is CNF, over-approximated states reachable in up to i steps
2. $R_{i+1} \subseteq R_i$ (monotonic)
3. $R_i \wedge \delta_M \models R'_{i+1}$
4. for $i < x$, $R_i \models \varphi$

High-level View of IC3/PDR

Model $M = (\Sigma, Q_M, Q_{0_M}, \delta_M)$ and Safety property φ

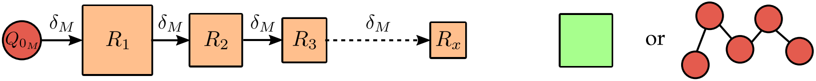


Frame Invariants

1. for $i > 0$, R_i is CNF, over-approximated states reachable in up to i steps
2. $R_{i+1} \subseteq R_i$ (monotonic)
3. $R_i \wedge \delta_M \models R'_{i+1}$
4. for $i < x$, $R_i \models \varphi$

Overview of FuseIC3

Model $M = (\Sigma, Q_M, Q_{0_M}, \delta_M)$ and Safety property φ



Frame Sequence R

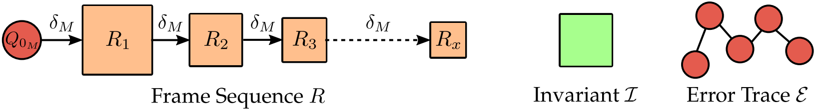
Invariant $\mathcal{I}$ Error Trace $\mathcal{E}$

Frame Invariants

1. for $i > 0$, R_i is CNF, over-approximated states reachable in up to i steps
2. $R_{i+1} \subseteq R_i$ (monotonic)
3. $R_i \wedge \delta_M \models R'_{i+1}$ ⟵ Core Idea of FuseIC3
4. for $i < x$, $R_i \models \varphi$

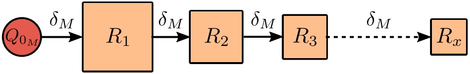
Overview of FuseIC3

Model $M = (\Sigma, Q_M, Q_{0_M}, \delta_M)$ and Safety property φ

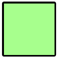


Overview of FuseIC3

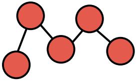
Model $M = (\Sigma, Q_M, Q_{0_M}, \delta_M)$ and Safety property φ



Frame Sequence R



Invariant $\mathcal{I}$

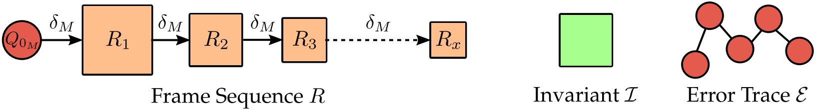


Error Trace $\mathcal{E}$

Model $N = (\Sigma, Q_N, Q_{0_N}, \delta_N)$ and Safety property φ

Overview of FuseIC3

Model $M = (\Sigma, Q_M, Q_{0_M}, \delta_M)$ and Safety property φ

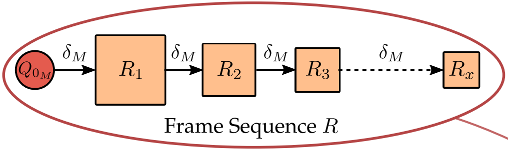


Model $N = (\Sigma, Q_N, Q_{0_N}, \delta_N)$ and Safety property φ

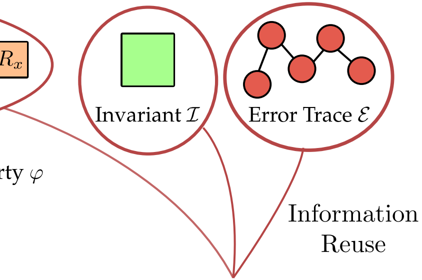
Goal: Compute frame sequence S for model N

Overview of FuseIC3

Model $M = (\Sigma, Q_M, Q_{0_M}, \delta_M)$ and Safety property φ



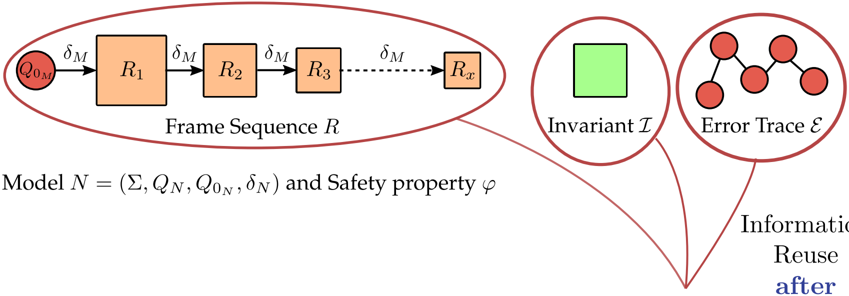
Model $N = (\Sigma, Q_N, Q_{0_N}, \delta_N)$ and Safety property φ



Goal: Compute frame sequence S for model N

Overview of FuseIC3

Model $M = (\Sigma, Q_M, Q_{0_M}, \delta_M)$ and Safety property φ



Model $N = (\Sigma, Q_N, Q_{0_N}, \delta_N)$ and Safety property φ

Information Reuse **after Repair**

Goal: Compute frame sequence S for model N

Assumptions and Intuition

Assumption 1

The different models in the design space are related, i.e., have overlapping reachable states.

Assumptions and Intuition

Assumption 1

The different models in the design space are related, i.e., have overlapping reachable states.

Assumption 2

The models in the design space are checked sequentially.

Assumptions and Intuition

Assumption 1

The different models in the design space are related, i.e., have overlapping reachable states.

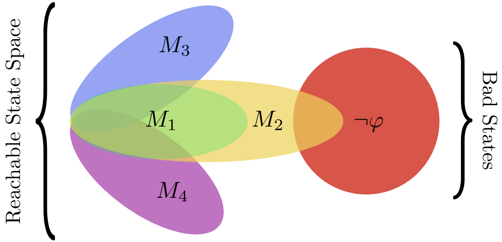
Assumption 2

The models in the design space are checked sequentially.

Intuition

Set of related models $\{M_1, M_2, M_3, M_4\}$

Safety property φ



Assumptions and Intuition

Assumption 1

The different models in the design space are related, i.e., have overlapping reachable states.

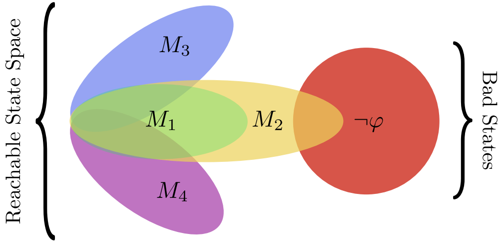
Assumption 2

The models in the design space are checked sequentially.

Intuition

Set of related models $\{M_1, M_2, M_3, M_4\}$

Safety property φ



1. Check M_1 with φ

Assumptions and Intuition

Assumption 1

The different models in the design space are related, i.e., have overlapping reachable states.

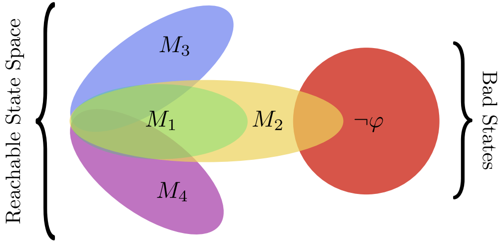
Assumption 2

The models in the design space are checked sequentially.

Intuition

Set of related models $\{M_1, M_2, M_3, M_4\}$

Safety property φ



1. Check M_1 with $\varphi \longrightarrow M_1 \models \varphi$

Assumptions and Intuition

Assumption 1

The different models in the design space are related, i.e., have overlapping reachable states.

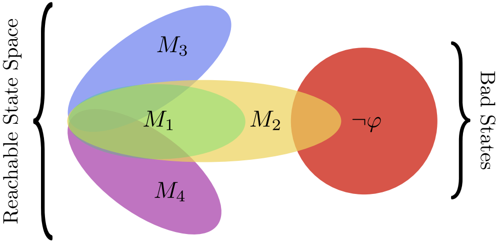
Assumption 2

The models in the design space are checked sequentially.

Intuition

Set of related models $\{M_1, M_2, M_3, M_4\}$

Safety property φ



1. Check M_1 with $\varphi \longrightarrow$ $M_1 \models \varphi$
2. Check M_2 with $\varphi \longrightarrow$

Assumptions and Intuition

Assumption 1

The different models in the design space are related, i.e., have overlapping reachable states.

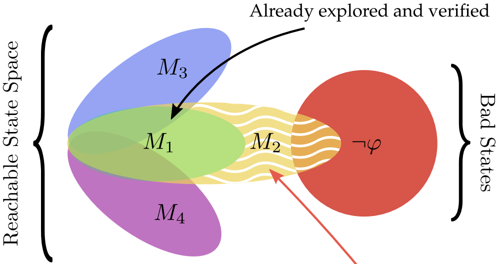
Assumption 2

The models in the design space are checked sequentially.

Intuition

Set of related models $\{M_1, M_2, M_3, M_4\}$

Safety property φ



1. Check M_1 with $\varphi \longrightarrow$ $M_1 \models \varphi$
2. Check M_2 with $\varphi \longrightarrow$

When checking M_2, FuseIC3 reuses the already explored and verified state space of M_1 and only checks

Assumptions and Intuition

Assumption 1

The different models in the design space are related, i.e., have overlapping reachable states.

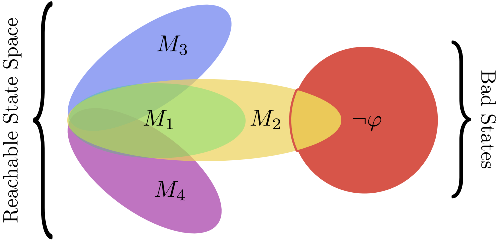
Assumption 2

The models in the design space are checked sequentially.

Intuition

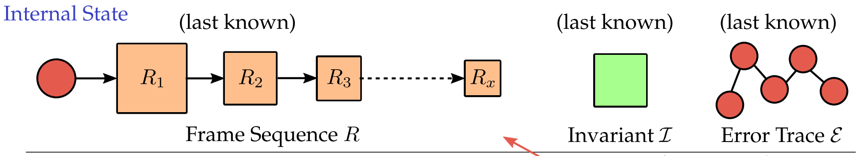
Set of related models $\{M_1, M_2, M_3, M_4\}$

Safety property φ



1. Check M_1 with $\varphi \longrightarrow$ $M_1 \models \varphi$
2. Check M_2 with $\varphi \longrightarrow$ $M_2 \not\models \varphi$

Internal State

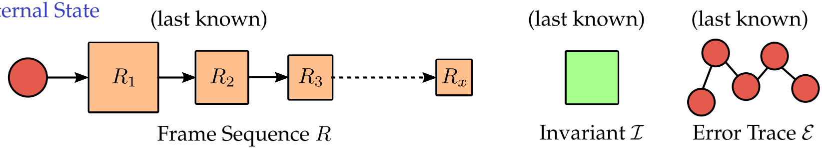


Model $N = (\Sigma, Q_N, Q_{0_N}, \delta_N)$ and Safety property φ

Internal state maintained by FuseIC3

Basic Checks

Internal State

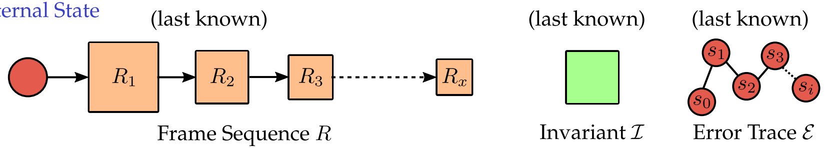


Model $N = (\Sigma, Q_N, Q_{0_N}, \delta_N)$ and Safety property φ

Goal: Compute frame sequence S for mode N

Basic Checks

Internal State



Model $N = (\Sigma, Q_N, Q_{0_N}, \delta_N)$ and Safety property φ

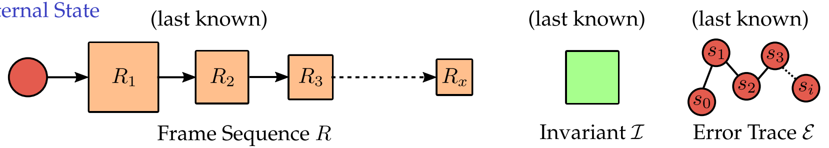
Goal: Compute frame sequence S for mode N

CHECKINVARIANT

$\mathcal{I}$ is inductive invariant w.r.t model N

Basic Checks

Internal State



Model $N = (\Sigma, Q_N, Q_{0_N}, \delta_N)$ and Safety property φ

Goal: Compute frame sequence S for mode N

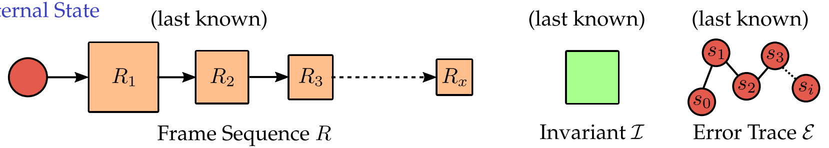
CHECKINVARIANT

$\mathcal{I}$ is inductive invariant w.r.t model N

✓ $N \models \varphi$

Basic Checks

Internal State



Model $N = (\Sigma, Q_N, Q_{0_N}, \delta_N)$ and Safety property φ

Goal: Compute frame sequence S for mode N

CheckInvariant

$\mathcal{I}$ is inductive invariant w.r.t model N

✓ $N \models \varphi$

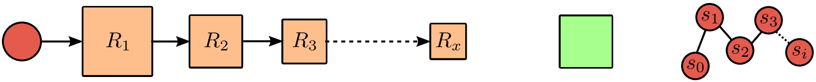
SimulateCex

$\mathcal{E}$ is valid trace w.r.t to model N

Basic Checks

Internal State

(last known) (last known) (last known)



Frame Sequence R Invariant $\mathcal{I}$ Error Trace $\mathcal{E}$

Model $N = (\Sigma, Q_N, Q_{0_N}, \delta_N)$ and Safety property φ

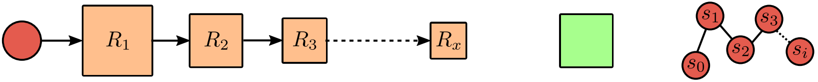
Goal: Compute frame sequence S for mode N

CheckInvariant	SimulateCex
$\mathcal{I}$ is inductive invariant w.r.t model N	$\mathcal{E}$ is valid trace w.r.t to model N
✓ $N \models \varphi$	✓ $N \not\models \varphi$

Basic Checks

Internal State

(last known) (last known) (last known)



Frame Sequence R Invariant $\mathcal{I}$ Error Trace $\mathcal{E}$

Model $N = (\Sigma, Q_N, Q_{0_N}, \delta_N)$ and Safety property φ

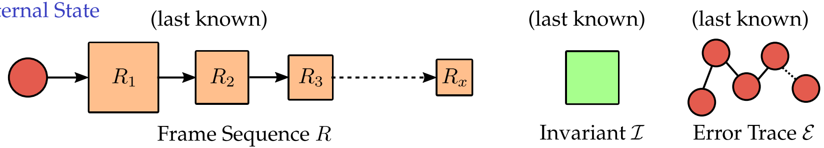
Goal: Compute frame sequence S for mode N

CheckInvariant	SimulateCex
$\mathcal{I}$ is inductive invariant w.r.t model N	$\mathcal{E}$ is valid trace w.r.t to model N
✓ $N \models \varphi$	✓ $N \not\models \varphi$

Instant Verification

Frame Reuse

Internal State

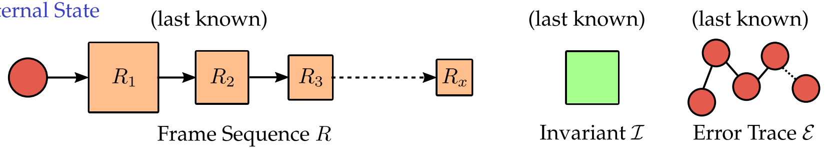


Model $N = (\Sigma, Q_N, Q_{0_N}, \delta_N)$ and Safety property φ

Goal: Compute frame sequence S for mode N

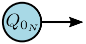
Frame Reuse

Internal State



Model $N = (\Sigma, Q_N, Q_{0_N}, \delta_N)$ and Safety property φ

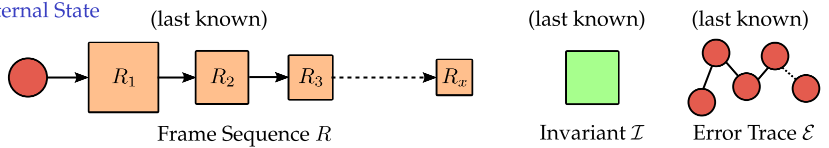
Goal: Compute frame sequence S for mode N



Frame Sequence S

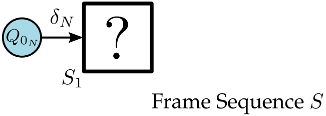
Frame Reuse

Internal State



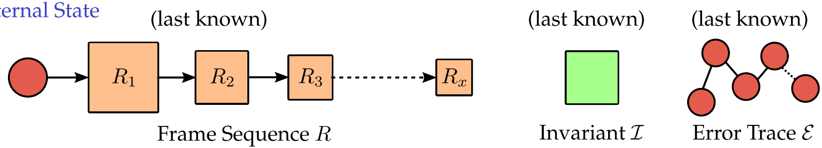
Model $N = (\Sigma, Q_N, Q_{0_N}, \delta_N)$ and Safety property φ

Goal: Compute frame sequence S for mode N



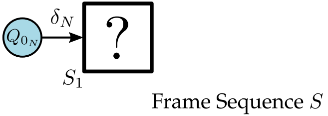
Frame Reuse

Internal State



Model $N = (\Sigma, Q_N, Q_{0_N}, \delta_N)$ and Safety property φ

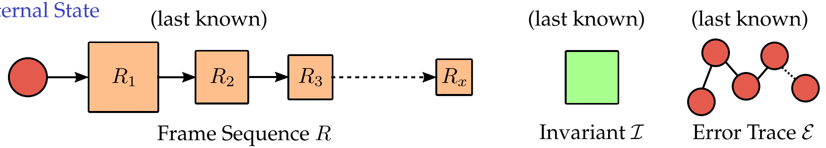
Goal: Compute frame sequence S for mode N



We want to compute S_1 using known information

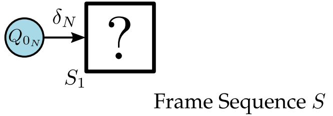
Frame Reuse

Internal State



Model $N = (\Sigma, Q_N, Q_{0_N}, \delta_N)$ and Safety property φ

Goal: Compute frame sequence S for mode N

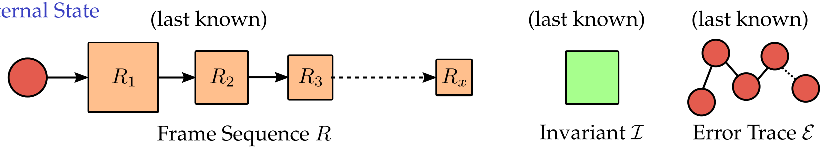


We want to compute S_1 using known information

$$S_0 \wedge \delta_N \rightarrow R_1'?$$

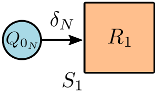
Frame Reuse

Internal State



Model $N = (\Sigma, Q_N, Q_{0_N}, \delta_N)$ and Safety property φ

Goal: Compute frame sequence S for mode N



Frame Sequence S

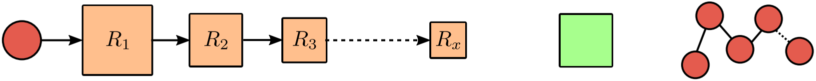
We want to compute S_1 using known information

$$S_0 \wedge \delta_N \rightarrow R_1'?$$ ✓

Frame Reuse

Internal State

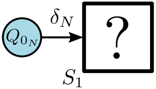
(last known) (last known) (last known)



Frame Sequence R Invariant $\mathcal{I}$ Error Trace $\mathcal{E}$

Model $N = (\Sigma, Q_N, Q_{0_N}, \delta_N)$ and Safety property φ

Goal: Compute frame sequence S for mode N



Frame Sequence S

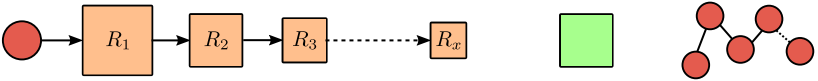
We want to compute S_1 using known information

$$S_0 \wedge \delta_N \rightarrow R_1'?$$

Frame Reuse

Internal State

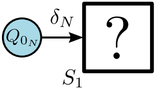
(last known) (last known) (last known)



Frame Sequence R Invariant $\mathcal{I}$ Error Trace $\mathcal{E}$

Model $N = (\Sigma, Q_N, Q_{0_N}, \delta_N)$ and Safety property φ

Goal: Compute frame sequence S for mode N



Frame Sequence S

We want to compute S_1 using known information

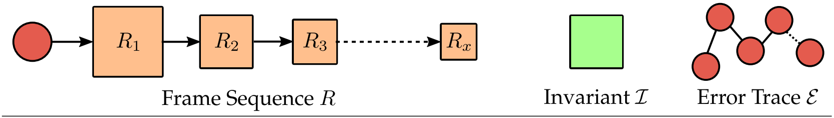
$$S_0 \wedge \delta_N \rightarrow R_1'?$$

Repair R_1

Frame Repair

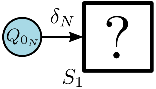
Internal State

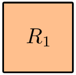
(last known) (last known) (last known)



Model $N = (\Sigma, Q_N, Q_{0_N}, \delta_N)$ and Safety property φ

Goal: Compute frame sequence S for mode N



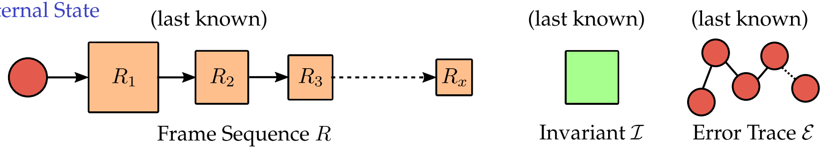


Frame Sequence S

Repair R_1 to $\hat{R}_1$ s.t. $S_0 \wedge \delta_N \rightarrow \hat{R}_1{}'$ is valid

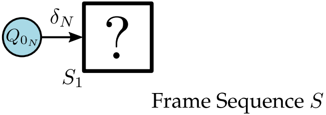
Frame Repair

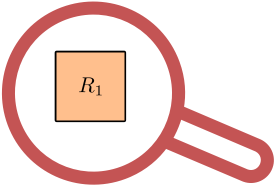
Internal State



Model $N = (\Sigma, Q_N, Q_{0_N}, \delta_N)$ and Safety property φ

Goal: Compute frame sequence S for mode N

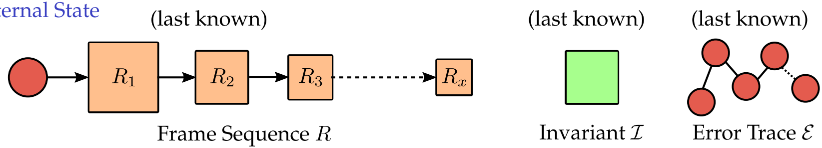




Repair R_1 to $\hat{R}_1$ s.t. $S_0 \wedge \delta_N \rightarrow \hat{R}_1{}'$ is valid

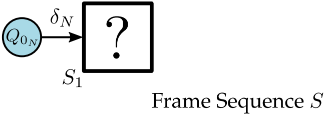
Frame Repair

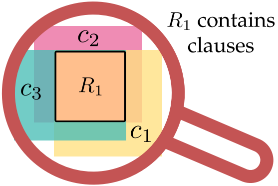
Internal State



Model $N = (\Sigma, Q_N, Q_{0_N}, \delta_N)$ and Safety property φ

Goal: Compute frame sequence S for mode N



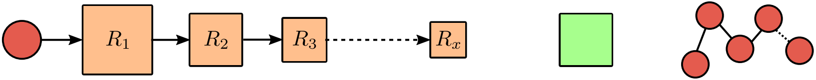


Repair R_1 to $\hat{R}_1$ s.t. $S_0 \wedge \delta_N \rightarrow \hat{R}_1{}'$ is valid

Frame Repair

Internal State

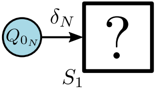
(last known) (last known) (last known)



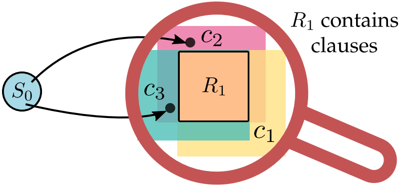
Frame Sequence R Invariant $\mathcal{I}$ Error Trace $\mathcal{E}$

Model $N = (\Sigma, Q_N, Q_{0_N}, \delta_N)$ and Safety property φ

Goal: Compute frame sequence S for mode N



Frame Sequence S



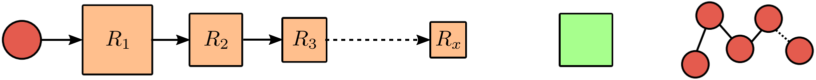
Reason for $S_0 \wedge \delta_N \not\to R_1$

Repair R_1 to $\hat{R}_1$ s.t. $S_0 \wedge \delta_N \to \hat{R}_1{}'$ is valid

Frame Repair

Internal State

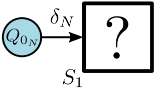
(last known) (last known) (last known)



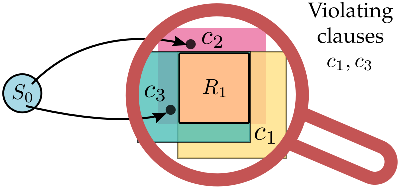
Frame Sequence R Invariant $\mathcal{I}$ Error Trace $\mathcal{E}$

Model $N = (\Sigma, Q_N, Q_{0_N}, \delta_N)$ and Safety property φ

Goal: Compute frame sequence S for mode N



Frame Sequence S



Violating clauses c_1, c_3

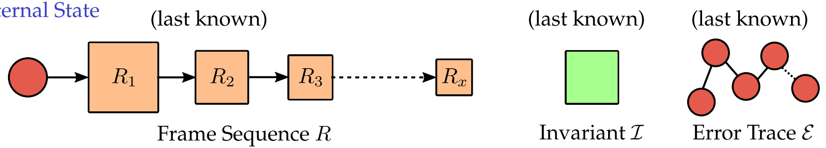
Reason for $S_0 \wedge \delta_N \not\to R_1$

Repair R_1 to $\hat{R}_1$ s.t. $S_0 \wedge \delta_N \to \hat{R}_1{}'$ is valid

FINDCLAUSES

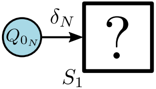
Frame Repair

Internal State



Model $N = (\Sigma, Q_N, Q_{0_N}, \delta_N)$ and Safety property φ

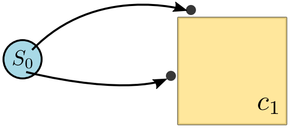
Goal: Compute frame sequence S for mode N



Frame Sequence S

Violating clauses

c_1, c_3

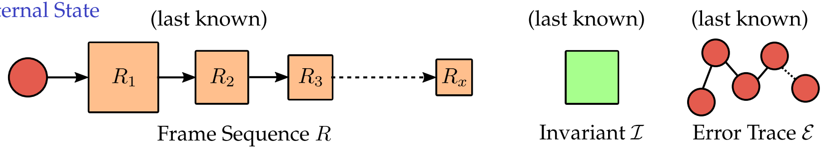


$S_0 \wedge \delta_N \not\rightarrow c_1'$

Repair R_1 to $\hat{R_1}$ s.t. $S_0 \wedge \delta_N \rightarrow \hat{R_1}'$ is valid

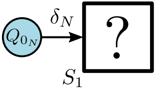
Frame Repair

Internal State

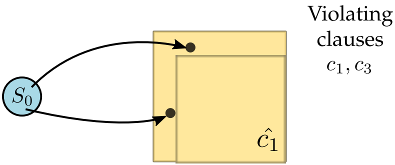


Model $N = (\Sigma, Q_N, Q_{0_N}, \delta_N)$ and Safety property φ

Goal: Compute frame sequence S for mode N



Frame Sequence S



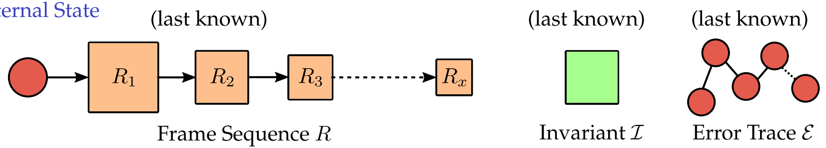
$S_0 \wedge \delta_N \rightarrow \hat{c_1}'$

Repair R_1 to $\hat{R_1}$ s.t. $S_0 \wedge \delta_N \rightarrow \hat{R_1}'$ is valid

EXPANDCLAUSE

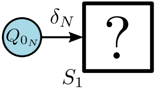
Frame Repair

Internal State

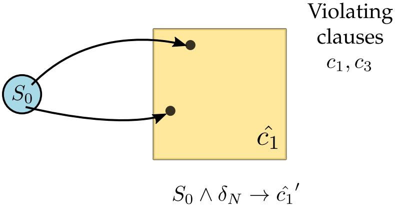


Model $N = (\Sigma, Q_N, Q_{0_N}, \delta_N)$ and Safety property φ

Goal: Compute frame sequence S for mode N



Frame Sequence S

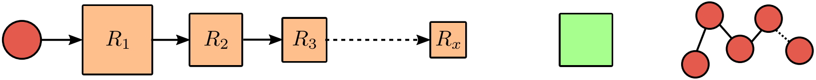


Repair R_1 to $\hat{R_1}$ s.t. $S_0 \wedge \delta_N \rightarrow \hat{R_1}'$ is valid

Frame Repair

Internal State

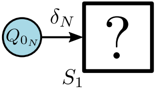
(last known) (last known) (last known)



Frame Sequence R Invariant $\mathcal{I}$ Error Trace $\mathcal{E}$

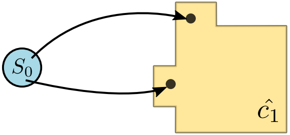
Model $N = (\Sigma, Q_N, Q_{0_N}, \delta_N)$ and Safety property φ

Goal: Compute frame sequence S for mode N



Frame Sequence S

Violating clauses c_1, c_3



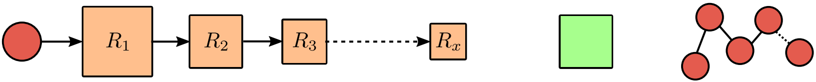
$S_0 \wedge \delta_N \rightarrow \hat{c_1}'$

Repair R_1 to $\hat{R_1}$ s.t. $S_0 \wedge \delta_N \rightarrow \hat{R_1}'$ is valid

SHRINKCLAUSE

Frame Repair

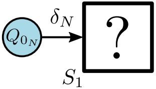
Internal State



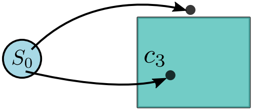
Frame Sequence R Invariant $\mathcal{I}$ Error Trace $\mathcal{E}$

Model $N = (\Sigma, Q_N, Q_{0_N}, \delta_N)$ and Safety property φ

Goal: Compute frame sequence S for mode N



Frame Sequence S



Violating clauses c_1, c_3

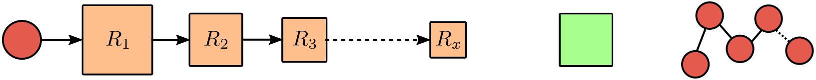
$S_0 \wedge \delta_N \not\to c_3'$

Repair R_1 to $\hat{R}_1$ s.t. $S_0 \wedge \delta_N \to \hat{R}_1{}'$ is valid

Frame Repair

Internal State

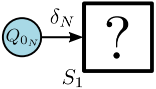
(last known) (last known) (last known)



Frame Sequence R Invariant $\mathcal{I}$ Error Trace $\mathcal{E}$

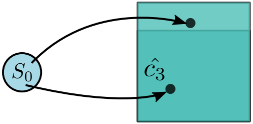
Model $N = (\Sigma, Q_N, Q_{0_N}, \delta_N)$ and Safety property φ

Goal: Compute frame sequence S for mode N



Frame Sequence S

Violating clauses c_1, c_3



$S_0 \wedge \delta_N \rightarrow \hat{c_3}'$

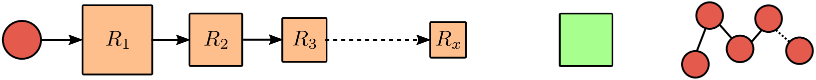
Repair R_1 to $\hat{R_1}$ s.t. $S_0 \wedge \delta_N \rightarrow \hat{R_1}'$ is valid

ExpandClause

Frame Repair

Internal State

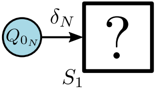
(last known) (last known) (last known)



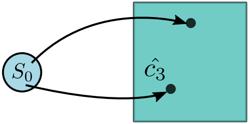
Frame Sequence R Invariant $\mathcal{I}$ Error Trace $\mathcal{E}$

Model $N = (\Sigma, Q_N, Q_{0_N}, \delta_N)$ and Safety property φ

Goal: Compute frame sequence S for mode N



Frame Sequence S



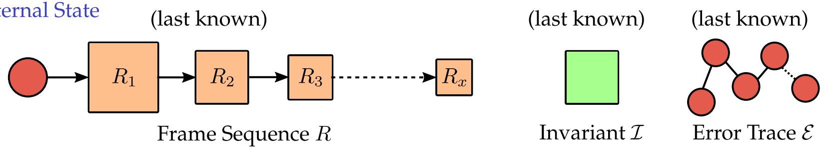
Violating clauses c_1, c_3

$S_0 \wedge \delta_N \rightarrow \hat{c_3}'$

Repair R_1 to $\hat{R_1}$ s.t. $S_0 \wedge \delta_N \rightarrow \hat{R_1}'$ is valid

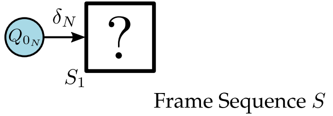
Frame Repair

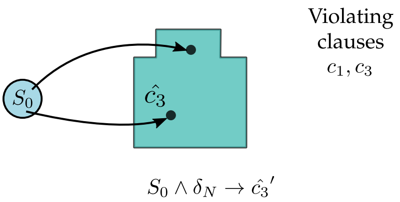
Internal State



Model $N = (\Sigma, Q_N, Q_{0_N}, \delta_N)$ and Safety property φ

Goal: Compute frame sequence S for mode N

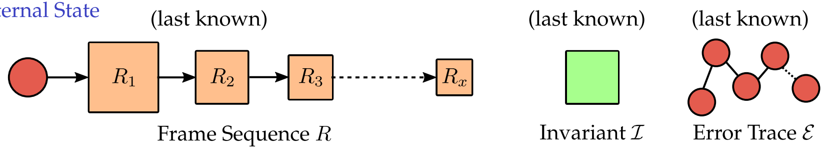




Repair R_1 to $\hat{R_1}$ s.t. $S_0 \wedge \delta_N \rightarrow \hat{R_1}'$ is valid

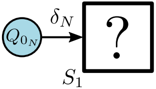
Frame Repair

Internal State

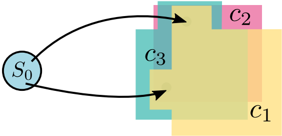


Model $N = (\Sigma, Q_N, Q_{0_N}, \delta_N)$ and Safety property φ

Goal: Compute frame sequence S for mode N



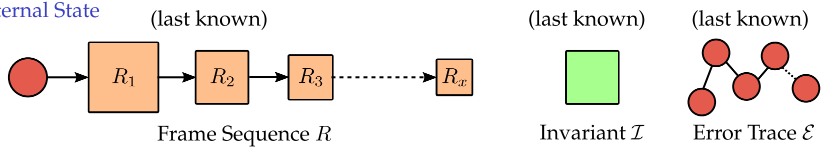
Frame Sequence S



Repair R_1 to $\hat{R}_1$ s.t. $S_0 \wedge \delta_N \rightarrow \hat{R}_1{}'$ is valid

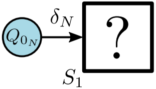
Frame Repair

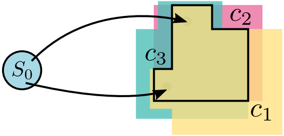
Internal State



Model $N = (\Sigma, Q_N, Q_{0_N}, \delta_N)$ and Safety property φ

Goal: Compute frame sequence S for mode N

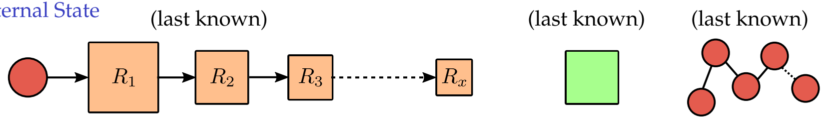




Repair R_1 to $\hat{R}_1$ s.t. $S_0 \wedge \delta_N \rightarrow \hat{R}_1{}'$ is valid

Frame Repair

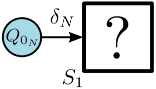
Internal State



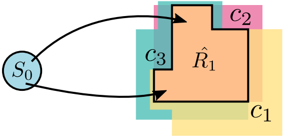
Frame Sequence R Invariant $\mathcal{I}$ Error Trace $\mathcal{E}$

Model $N = (\Sigma, Q_N, Q_{0_N}, \delta_N)$ and Safety property φ

Goal: Compute frame sequence S for mode N



Frame Sequence S

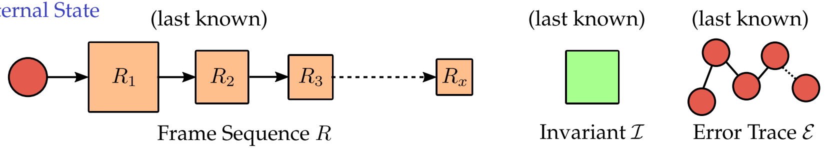


$S_0 \wedge \delta_N \rightarrow \hat{R}_1{}'$

Repair R_1 to $\hat{R}_1$ s.t. $S_0 \wedge \delta_N \rightarrow \hat{R}_1{}'$ is valid

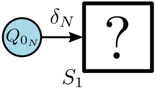
Frame Repair

Internal State

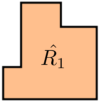


Model $N = (\Sigma, Q_N, Q_{0_N}, \delta_N)$ and Safety property φ

Goal: Compute frame sequence S for mode N



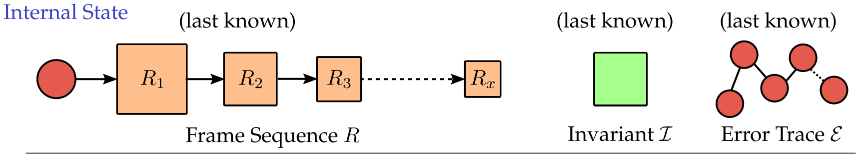
Frame Sequence S



$$S_0 \wedge \delta_N \rightarrow \hat{R}_1{}'$$

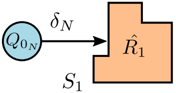
Repair R_1 to $\hat{R}_1$ s.t. $S_0 \wedge \delta_N \rightarrow \hat{R}_1{}'$ is valid

Frame Repair



Model $N = (\Sigma, Q_N, Q_{0_N}, \delta_N)$ and Safety property φ

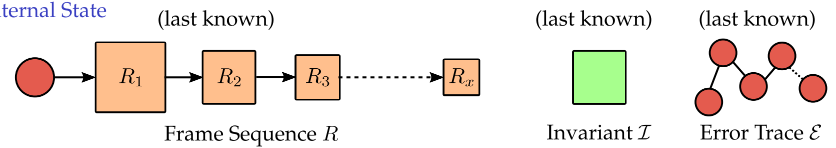
Goal: Compute frame sequence S for mode N



Frame Sequence S

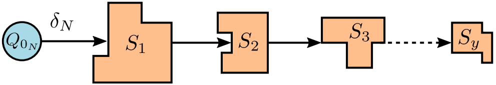
Strengthen Frames

Internal State



Model $N = (\Sigma, Q_N, Q_{0_N}, \delta_N)$ and Safety property φ

Goal: Compute frame sequence S for mode N

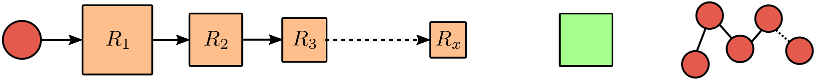


Frame Sequence S

Strengthen Frames

Internal State

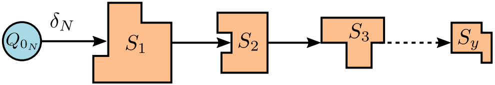
(last known) (last known) (last known)



Frame Sequence R Invariant $\mathcal{I}$ Error Trace $\mathcal{E}$

Model $N = (\Sigma, Q_N, Q_{0_N}, \delta_N)$ and Safety property φ

Goal: Compute frame sequence S for mode N

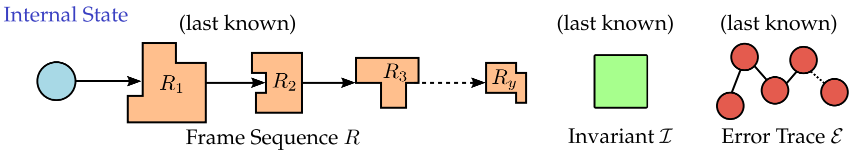


Frame Sequence S

$N \models \varphi$

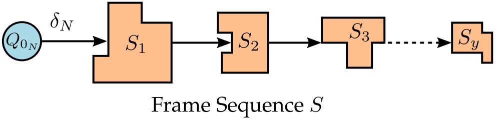
Invariant: $S_{i+1} \equiv S_i$

Strengthen Frames



Model $N = (\Sigma, Q_N, Q_{0_N}, \delta_N)$ and Safety property φ

Goal: Compute frame sequence S for mode N



$N \models \varphi$

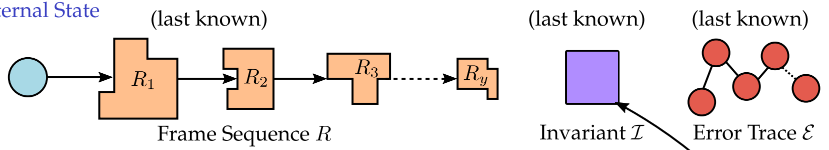
Invariant: $S_{i+1} \equiv S_i$

Update last known

1. Frame Sequence

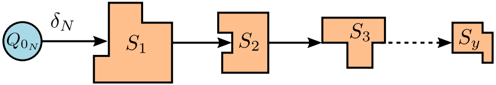
Strengthen Frames

Internal State



Model $N = (\Sigma, Q_N, Q_{0_N}, \delta_N)$ and Safety property φ

Goal: Compute frame sequence S for mode N



Frame Sequence S

$N \models \varphi$

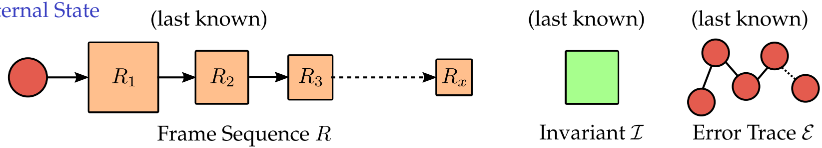
Invariant: $S_{i+1} \equiv S_i$

Update last known

1. Frame Sequence
2. Invariant

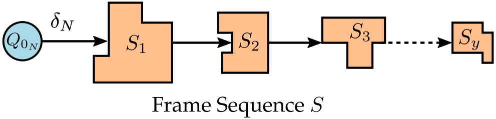
Strengthen Frames

Internal State



Model $N = (\Sigma, Q_N, Q_{0_N}, \delta_N)$ and Safety property φ

Goal: Compute frame sequence S for mode N

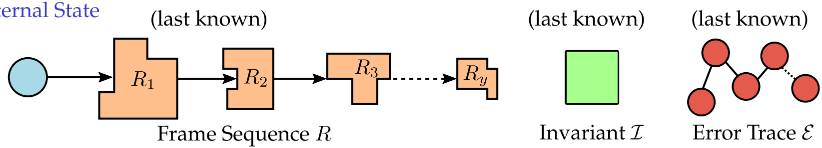


$N \not\models \varphi$

Error Trace: $\mathcal{E}_N$

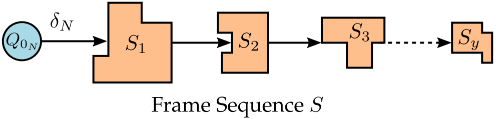
Strengthen Frames

Internal State



Model $N = (\Sigma, Q_N, Q_{0_N}, \delta_N)$ and Safety property φ

Goal: Compute frame sequence S for mode N



$N \not\models \varphi$

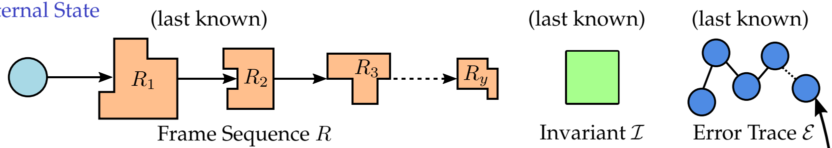
Error Trace: $\mathcal{E}_N$

Update last known

1. Frame Sequence

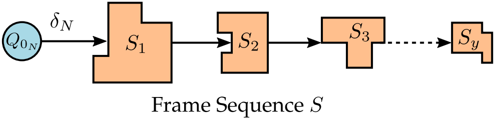
Strengthen Frames

Internal State



Model $N = (\Sigma, Q_N, Q_{0_N}, \delta_N)$ and Safety property φ

Goal: Compute frame sequence S for mode N



$N \not\models \varphi$

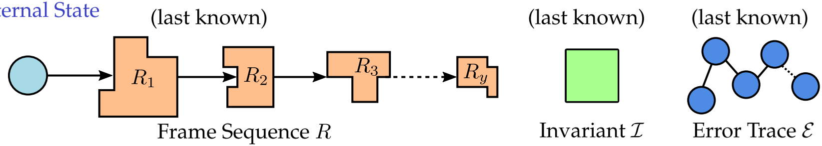
Error Trace: $\mathcal{E}_N$

Update last known

1. Frame Sequence
2. Error Trace

Strengthen Frames

Internal State



Ready for next model

Experiment Setup

- FuseIC3 is coded in C++ and uses MathSAT5 as SAT solver.

[1] https://es-static.fbk.eu/people/griggio/ic3ia/

Experiment Setup

- FuseIC3 is coded in C++ and uses MathSAT5 as SAT solver.
- Core IC3 implementation based on `ic3ia`[1]

Source code available at
`http://temporallogic.org/research/FMCAD17`

[1] `https://es-static.fbk.eu/people/griggio/ic3ia/`

Experiment Setup

- FuseIC3 is coded in C++ and uses MathSAT5 as SAT solver.
- Core IC3 implementation based on `ic3ia`[1]
- Other algorithms considered
  1. Typical IC3 (typ) (Een et al., 2011)
  2. Incremental IC3 (inc) (Chockler et al., 2011)

Source code available at
`http://temporallogic.org/research/FMCAD17`

[1] `https://es-static.fbk.eu/people/griggio/ic3ia/`

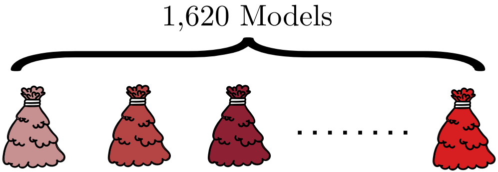
Experiment Setup

- FuseIC3 is coded in C++ and uses MathSAT5 as SAT solver.
- Core IC3 implementation based on `ic3ia`[1]
- Other algorithms considered
  1. Typical IC3 (typ) (Een et al., 2011)
  2. Incremental IC3 (inc) (Chockler et al., 2011)
- Benchmarks evaluated
  1. NASA NextGen Air Traffic Control (ATC) System (Gario et al., 2016)
  2. Selected benchmarks from HWMCC 2015
     - Each model was randomly mutated to generate a model-set.
  3. Boeing AIR 6110 Wheel Braking System (WBS) (Bozzano et al., 2015)

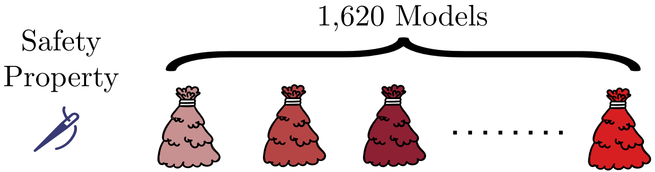
Source code available at
`http://temporallogic.org/research/FMCAD17`

[1] `https://es-static.fbk.eu/people/griggio/ic3ia/`

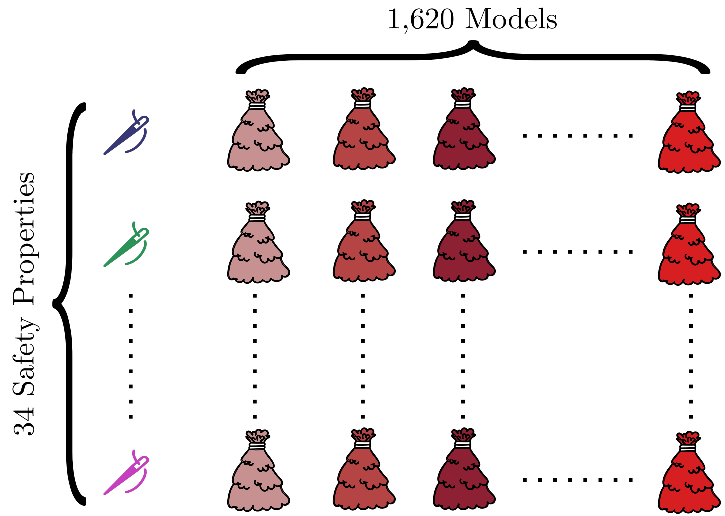
NASA ATC Benchmark



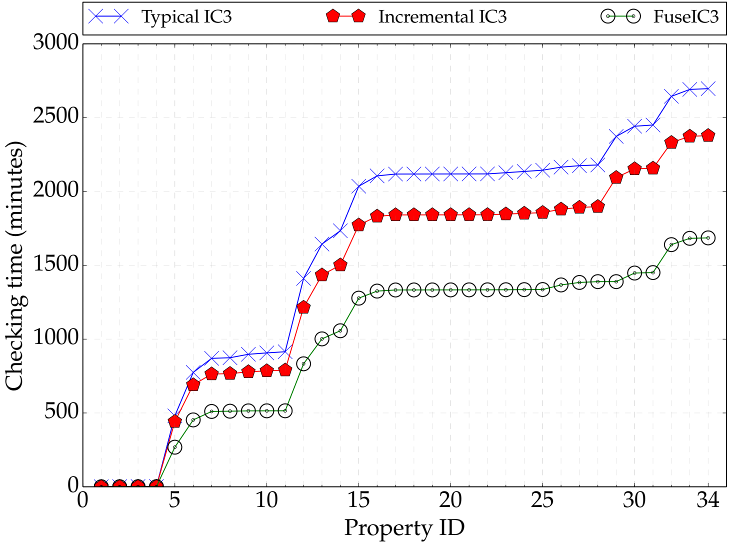
NASA ATC Benchmark



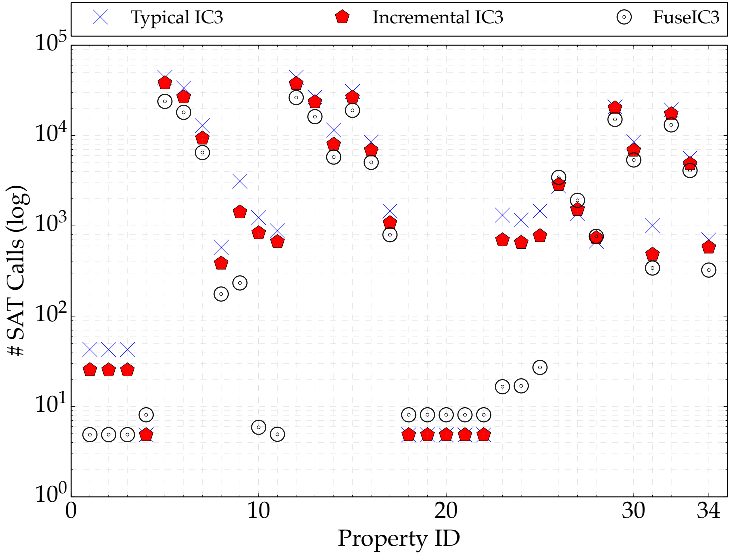
NASA ATC Benchmark



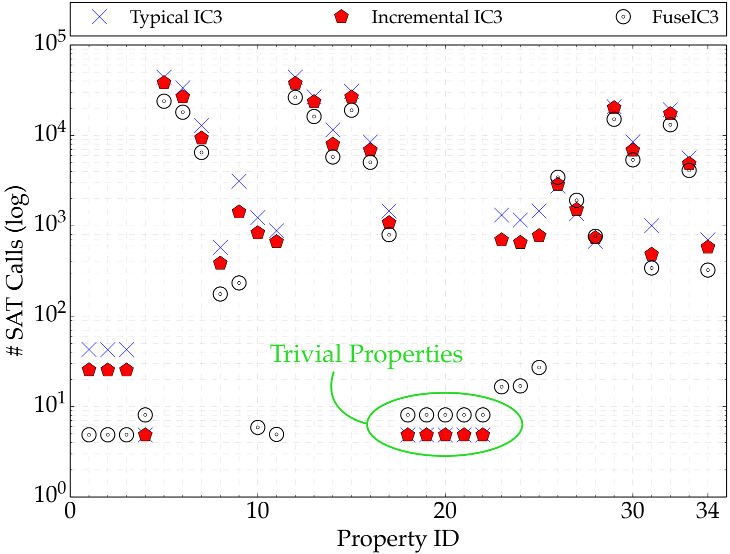
NASA ATC Benchmark



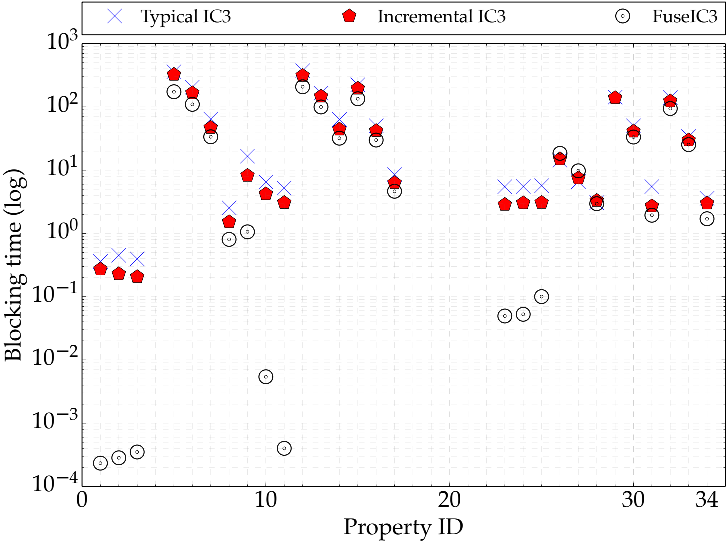
NASA ATC Benchmark



NASA ATC Benchmark



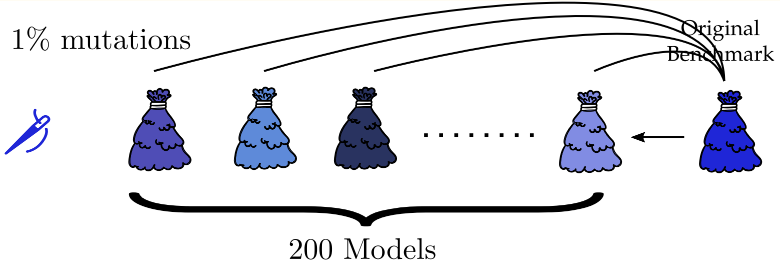
NASA ATC Benchmark



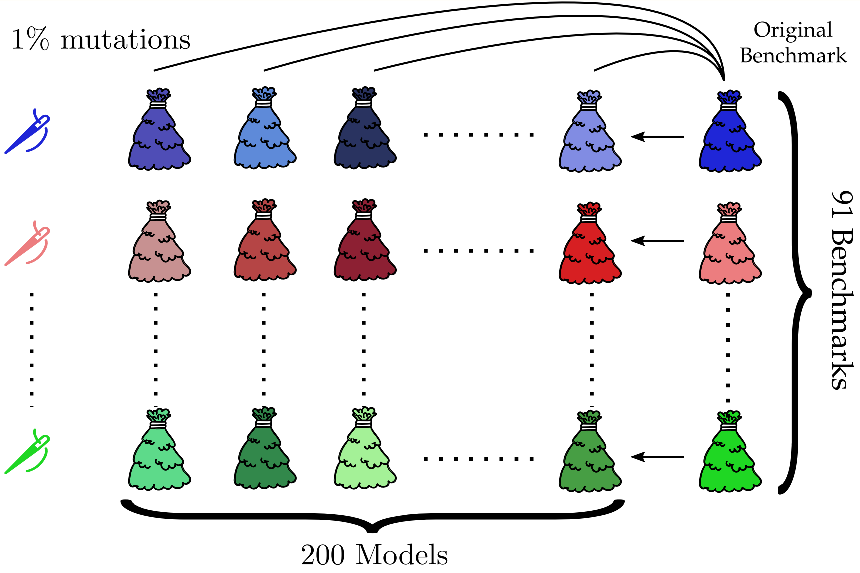
Selected HWMCC Benchmarks

Original
Benchmark

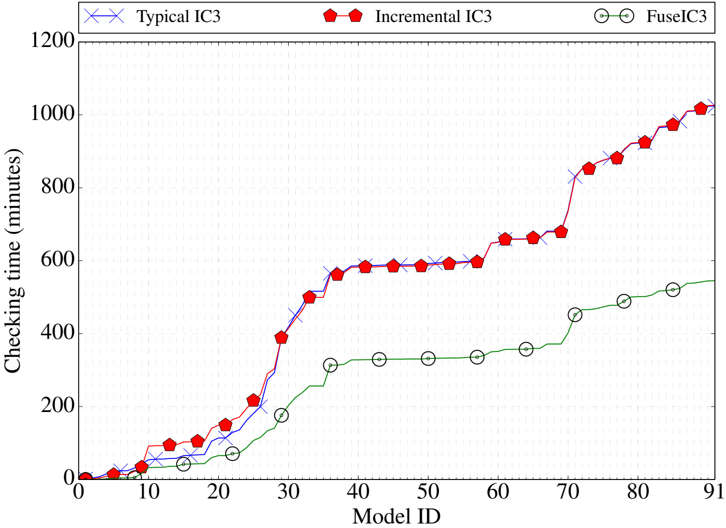
Selected HWMCC Benchmarks



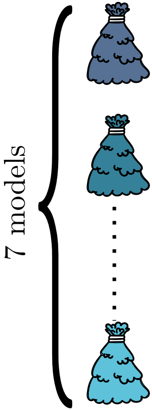
Selected HWMCC Benchmarks



Selected HWMCC Benchmarks



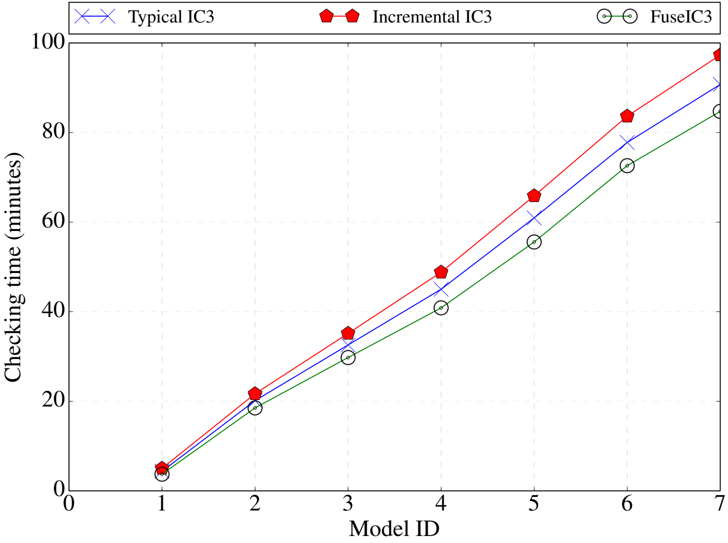
Boeing WBS Benchmark



Boeing WBS Benchmark

Boeing WBS Benchmark

Boeing WBS Benchmark



Discussion

- FuseIC3 is an efficient algorithm for checking design spaces

Discussion

- FuseIC3 is an efficient algorithm for checking design spaces
  - Incremental - can be used for
    - regression verification,
    - coverage computation, and
    - product line verification.

Discussion

- FuseIC3 is an efficient algorithm for checking design spaces
  - Incremental - can be used for
    - regression verification,
    - coverage computation, and
    - product line verification.
  - General & scalable - does not require special modeling formalisms.

Discussion

- FuseIC3 is an efficient algorithm for checking design spaces
  - Incremental - can be used for
    - regression verification,
    - coverage computation, and
    - product line verification.
  - General & scalable - does not require special modeling formalisms.
  - Reuses information - IC3 frames, invariants, and error traces.

Discussion

- FuseIC3 is an efficient algorithm for checking design spaces
  - Incremental - can be used for
    - regression verification,
    - coverage computation, and
    - product line verification.
  - General & scalable - does not require special modeling formalisms.
  - Reuses information - IC3 frames, invariants, and error traces.
- Future Work
  - How can we use intermediate SAT results to speed-up FuseIC3?
  - What model/property ordering heuristics may improve performance?
  - Is it possible to use FuseIC3 for liveness checking?

Discussion

- FuseIC3 is an efficient algorithm for checking design spaces
  - Incremental - can be used for
    - regression verification,
    - coverage computation, and
    - product line verification.
  - General & scalable - does not require special modeling formalisms.
  - Reuses information - IC3 frames, invariants, and error traces.
- Future Work
  - How can we use intermediate SAT results to speed-up FuseIC3?
  - What model/property ordering heuristics may improve performance?
  - Is it possible to use FuseIC3 for liveness checking?

Thank You!

`http://temporallogic.org/research/FMCAD17`

Beer, I., Ben-David, S., Eisner, C., & Landver, A. (1996). RuleBase: An industry-oriented formal verification tool. In *Dac*.

Ben-David, S., Sterin, B., Atlee, J. M., & Beidu, S. (2015). Symbolic model checking of product-line requirements using SAT-based methods. In *Icse* (Vol. 1, pp. 189–199).

Bozzano, M., Cimatti, A., Fernandes Pires, A., Jones, D., Kimberly, G., Petri, T., … Tonetta, S. (2015). Formal design and safety analysis of AIR6110 wheel brake system. In *CAV*.

Chockler, H., Ivrii, A., Matsliah, A., Moran, S., & Nevo, Z. (2011). Incremental Formal Verification of Hardware. In *Fmcad* (pp. 135–143).

Classen, A., Cordy, M., Heymans, P., Legay, A., & Schobbens, P.-Y. (2012). Model checking software product lines with snip. *(STTT)*, 1–24.

Classen, A., Heymans, P., Schobbens, P.-Y., & Legay, A. (2011). Symbolic model checking of software product lines. In *Icse* (pp. 321–330).

Classen, A., Heymans, P., Schobbens, P.-Y., Legay, A., & Raskin, J.-F. (2010). Model checking lots of systems: efficient verification of temporal properties in software product lines. In *Icse* (pp. 335–344).