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

Model checking!
Classical Model Checking of a Design Space

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

Can we do better?

Rohit Dureja & Kristin Y. Rozier

FuseIC3: An Algorithm for Checking Large Design Spaces
Classical Model Checking of a Design Space

Set of Models $\mathcal{M}$

$M_1$, $M_2$, $M_3$, $M_n$

Set of Properties $\mathcal{P}$

$\varphi_1$, $\varphi_2$, $\varphi_3$, $\varphi_k$
Classical Model Checking of a Design Space

Set of Models $\mathcal{M}$

$M_1$  $M_2$  $M_3$  $M_n$

Set of Properties $\mathcal{P}$

$\varphi_1$  $\varphi_2$  $\varphi_3$  $\varphi_k$

For every $M \in \mathcal{M}$ and $\varphi \in \mathcal{P}$ check if $M \models \varphi$?
Classical Model Checking of a Design Space

Set of Models $\mathcal{M}$

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.
Motivation

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

$x_1 < x_2 < x_3$

FuseIC3 is SAT-based

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

1. 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

Software product line verification

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

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 $\varphi$
High-level View of IC3/PDR

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

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. $R_i \land \delta_M \models R'_{i+1}$
4. for $i < x$, $R_i \models \varphi$
High-level View of IC3/PDR

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

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. $R_i \land \delta_M \models R'_{i+1}$
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 $\varphi$

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. $R_i \land \delta_M \models R'_{i+1}$
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 $\varphi$

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. $R_i \land \delta_M \models R'_{i+1}$
4. for $i < x$, $R_i \models \varphi$
Overview of FuseIC3

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

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. $R_i \land \delta_M \models R'_{i+1}$
4. for $i < x$, $R_i \models \varphi$

Core Idea of FuseIC3
Overview of FuseIC3

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

Frame Sequence $R$

Invariant $\mathcal{I}$

Error Trace $\mathcal{E}$
Overview of FuseIC3

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

Model $N = (\Sigma, Q_N, Q_{0N}, \delta_N)$ and Safety property $\varphi$
Overview of FuseIC3

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

\[ Q_{0M} \xrightarrow{\delta_M} R_1 \xrightarrow{\delta_M} R_2 \xrightarrow{\delta_M} R_3 \xrightarrow{\delta_M} R_x \]

Frame Sequence $R$

Invariant $\mathcal{I}$

Error Trace $\mathcal{E}$

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

Goal: Compute frame sequence $S$ for model $N$
Overview of FuseIC3

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

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

Goal: Compute frame sequence $S$ for model $N$
Overview of FuseIC3

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

Frame Sequence $R$

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

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 \(\varphi\)
**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 \( \varphi \)

1. Check \( M_1 \) with \( \varphi \)
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 \( \varphi \)

1. Check \( M_1 \) with \( \varphi \rightarrow M_1 |= \varphi \)
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 \( \varphi \)

1. Check \( M_1 \) with \( \varphi \) \( \rightarrow \) \( M_1 \models \varphi \)
2. Check \( M_2 \) with \( \varphi \) \( \rightarrow \)
**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 \( \varphi \)

1. Check \( M_1 \) with \( \varphi \) \( \rightarrow \) \( M_1 \models \varphi \)
2. Check \( M_2 \) with \( \varphi \) \( \rightarrow \)

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

1. Check \( M_1 \) with \( \varphi \rightarrow M_1 \models \varphi \)
2. Check \( M_2 \) with \( \varphi \rightarrow M_2 \not\models \varphi \)
Internal State

Internal State
(last known)

Frame Sequence \( R \)

\( R_1 \) \( \rightarrow \) \( R_2 \) \( \rightarrow \) \( R_3 \) \( \rightarrow \) \( R_x \)

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

Invariant \( I \)
(last known)

Error Trace \( E \)
(last known)

Internal state maintained by FuseIC3
Basic Checks

Internal State

(last known)

Frame Sequence $R$

(last known)

Invariant $I$

(last known)

Error Trace $E$

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

Goal: Compute frame sequence $S$ for mode $N$
Basic Checks

Model $N = (\Sigma, Q_N, Q_0_N, \delta_N)$ and Safety property $\varphi$

**Goal:** Compute frame sequence $S$ for mode $N$

**CheckInvariant**

$I$ is inductive invariant w.r.t model $N$
Basic Checks

Internal State

(last known)

\[ R_1 \rightarrow R_2 \rightarrow R_3 \rightarrow R_x \]

Frame Sequence \( R \)

(last known)

Invariant \( I \)

(last known)

Error Trace \( E \)

Model \( N = (\Sigma, Q_N, Q_0_N, \delta_N) \) and Safety property \( \varphi \)

**Goal:** Compute frame sequence \( S \) for mode \( N \)

\[
\text{CHECKINVARIANT}
\]

\( I \) is inductive invariant w.r.t model \( N \)

\( N \models \varphi \)
Basic Checks

Model \( N = (\Sigma, Q_N, Q_0_N, \delta_N) \) and Safety property \( \varphi \)

Goal: Compute frame sequence \( S \) for mode \( N \)

---

**CheckInvariant**

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

\( \checkmark \quad N \models \varphi \)

---

**SimulateCex**

\( \mathcal{E} \) is valid trace w.r.t to model \( N \)
Basic Checks

Internal State

Frame Sequence $R$

(last known)

Rohit Dureja & Kristin Y. Rozier

FuseIC3: An Algorithm for Checking Large Design Spaces

Model $N = (\Sigma, Q_N, Q_0_N, \delta_N)$ and Safety property $\varphi$

Goal: Compute frame sequence $S$ for mode $N$

\begin{align*}
\text{CHECK\textsc{Invariant}} & \\
\mathcal{I} \text{ is inductive invariant w.r.t model } N & \checkmark N \models \varphi
\end{align*}

\begin{align*}
\text{SIMULATE\textsc{Cex}} & \\
\mathcal{E} \text{ is valid trace w.r.t to model } N & \checkmark N \not\models \varphi
\end{align*}
Basic Checks

**Internal State**

(frame sequence \( R \))

- \( R_1 \)
- \( R_2 \)
- \( R_3 \)

(last known)

(frame sequence \( R \))

- \( R_x \)

(last known)

.Invariant \( \mathcal{I} \)

.Error Trace \( \mathcal{E} \)

Model \( N = (\Sigma, Q_N, Q_0_N, \delta_N) \) and Safety property \( \varphi \)

**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 \( N \)

\( N \not\models \varphi \)

**Instant Verification**
Frame Reuse

Model $N = (\Sigma, Q_N, Q_0, \delta_N)$ and Safety property $\varphi$

**Goal:** Compute frame sequence $S$ for mode $N$
Frame Reuse

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

Goal: Compute frame sequence $S$ for mode $N$

Frame Sequence $S$
Frame Reuse

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

Goal: Compute frame sequence $S$ for mode $N$
Frame Reuse

Model \( N = (\Sigma, Q_N, Q_0_N, \delta_N) \) and Safety property \( \varphi \)

**Goal:** Compute frame sequence \( S \) for mode \( N \)

We want to compute \( S_1 \) using known information
Frame Reuse

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

**Goal:** Compute frame sequence $S$ for mode $N$

We want to compute $S_1$ using known information

$$S_0 \land \delta_N \rightarrow R'_1?$$
Frame Reuse

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

Goal: Compute frame sequence $S$ for mode $N$

We want to compute $S_1$ using known information

$S_0 \land \delta_N \rightarrow R_1'$?
Frame Reuse

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

Goal: Compute frame sequence $S$ for mode $N$

We want to compute $S_1$ using known information

$$S_0 \land \delta_N \rightarrow R'_1? \times$$
Frame Reuse

Motivation

Preliminaries

Algorithm

Results

Internal State

(last known)

Frame Sequence $R$

(last known)

Invariant $I$

(last known)

Error Trace $E$

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

Goal: Compute frame sequence $S$ for mode $N$

We want to compute $S_1$

using known information

Frame Sequence $S$

$S_0 \land \delta_N \rightarrow R'_1$?

$\times$

Repair $R_1$
Frame Repair

Motivation

Preliminaries

Algorithm

Results

Internal State

(last known)

Frame Sequence $R$

(last known)

Invariant $I$

(last known)

Error Trace $E$

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

Goal: Compute frame sequence $S$ for mode $N$

Frame Sequence $S$

Repair $R_1$ to $\hat{R}_1$ s.t. $S_0 \land \delta_N \rightarrow \hat{R}_1$ is valid
Frame Repair

Internal State

(last known)

Frame Sequence $R$

(last known)

Invariant $I$

(last known)

Error Trace $E$

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

Goal: Compute frame sequence $S$ for mode $N$

Frame Sequence $S$

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

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

**Goal:** Compute frame sequence $S$ for mode $N$

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

Internal State

(last known)

| $R_1$ | $R_2$ | $R_3$ | $R_x$ |

Frame Sequence $R$

(last known)

Invariant $I$

(last known)

Error Trace $E$

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

Goal: Compute frame sequence $S$ for mode $N$

Frame Sequence $S$

$Q_{0_N} \xrightarrow{\delta_N} ? \xrightarrow{} S_1$

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

Reason for $S_0 \land \delta_N \not\rightarrow R_1$
Frame Repair

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

**Goal:** Compute frame sequence $S$ for mode $N$

**Internal State**
- $(last\ known)$

**Frame Sequence $R$**
- $R_1 \rightarrow R_2 \rightarrow R_3 \rightarrow R_x$

**Invariant $I$**
- $(last\ known)$

**Error Trace $E$**
- $(last\ known)$

**Frame Sequence $S$**
- $S_0 \rightarrow \delta_N \rightarrow \cdots$

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

**Reason for $S_0 \land \delta_N \not\rightarrow R_1$**

$\text{FINDCLAUSES}$
Frame Repair

Motivation
Preliminaries
Algorithm
Results

Frame Repair

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

Goal: Compute frame sequence \( S \) for mode \( N \)

Violating clauses \( c_1, c_3 \)

\[ S_0 \land \delta_N \not\rightarrow c_1' \]

Repair \( R_1 \) to \( \hat{R}_1 \) s.t. \( S_0 \land \delta_N \rightarrow \hat{R}_1' \) is valid
Frame Repair

Internal State

(last known)

Frame Sequence $R$

(last known)

Invariant $I$

(last known)

Error Trace $E$

Model $N = (\Sigma, Q_N, Q_0_N, \delta_N)$ and Safety property $\varphi$

Goal: Compute frame sequence $S$ for mode $N$

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

EXPANDCLAUSE
Frame Repair

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

**Goal:** Compute frame sequence $S$ for mode $N$

Repair $R_1$ to $\hat{R}_1$ s.t. $S_0 \land \delta_N \rightarrow \hat{R}_1$ is valid
Frame Repair

Internal State

\begin{align*}
\text{Frame Sequence } R & : R_1 \rightarrow R_2 \rightarrow R_3 \rightarrow R_x \\
& \text{(last known)} \\
\text{Invariant } I & : \text{(last known)} \\
\text{Error Trace } \mathcal{E} & : \text{(last known)}
\end{align*}

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

**Goal:** Compute frame sequence \( S \) for mode \( N \)

Frame Sequence \( S \)

Repair \( R_1 \) to \( \hat{R}_1 \) s.t. \( S_0 \land \delta_N \rightarrow \hat{R}_1 \) is valid

**SHRINKCLAUSE**
Frame Repair

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

**Goal:** Compute frame sequence $S$ for mode $N$

$S_0 \delta_N S_1$

Frame Sequence $S$

$S_0 \land \delta_N \not\Rightarrow c_3'$

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

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

Goal: Compute frame sequence $S$ for mode $N$

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

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

**Goal:** Compute frame sequence $S$ for mode $N$

**Frame Sequence**

$$S_0$$

$$\delta_N$$

$$S_1$$

$$Q_{0_N}$$

**Error Trace**

$$\hat{c}_3$$

$$S_0 \land \delta_N \rightarrow \hat{c}_3'$$

**Repair**

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

FuseIC3: An Algorithm for Checking Large Design Spaces
Frame Repair

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

Goal: Compute frame sequence $S$ for mode $N$

Repair $R_1$ to $\hat{R}_1$ s.t. $S_0 \land \delta_N \rightarrow \hat{R}_1$ is valid
Frame Repair

Internal State

(\text{last known})

\begin{align*}
R_1 &\rightarrow R_2 & R_3 &\rightarrow R_x \\
\text{Frame Sequence } R
\end{align*}

(\text{last known})

Invariant \mathcal{I}

(\text{last known})

Error Trace \mathcal{E}

Model \( N = (\Sigma, Q_N, Q_0, \delta_N) \) and Safety property \( \varphi \)

\textbf{Goal:} Compute frame sequence \( S \) for mode \( N \)

\begin{align*}
Q_0 &\rightarrow ? \\
\delta_N &\rightarrow S_1 \\
\text{Frame Sequence } S
\end{align*}

Repair \( R_1 \) to \( \hat{R}_1 \) s.t. \( S_0 \land \delta_N \rightarrow \hat{R}_1' \) is valid
Frame Repair

Internal State

*(last known)*

Frame Sequence $R$

$(R_1 \rightarrow R_2 \rightarrow R_3 \rightarrow R_x)$

Invariant $I$

$(last known)$

Error Trace $E$

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

**Goal:** Compute frame sequence $S$ for mode $N$

Frame Sequence $S$

$Q_{0_N} \xrightarrow{\delta_N} ? \xrightarrow{S_1}$

$S_0 \xrightarrow{c_1} c_2 \xrightarrow{c_3}$

$R_1$ to $\hat{R}_1$ s.t. $S_0 \land \delta_N \rightarrow \hat{R}_1 \; \text{is valid}$
Frame Repair

Internal State

\[ \text{Frame Sequence } R \]

\[ \begin{align*}
R_1 & \rightarrow R_2 \rightarrow R_3 \rightarrow \cdots \rightarrow R_x
\end{align*} \]

(last known)

Invariant \( I \)

(last known)

Error Trace \( E \)

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

Goal: Compute frame sequence \( S \) for mode \( N \)

Frame Sequence \( S \)

\[ \begin{align*}
Q_{0N} & \xrightarrow{\delta_N} \mathbf{?} \\
S_1 & \rightarrow \cdots
\end{align*} \]

\[ S_0 \rightarrow \cdots \]

\[ \hat{R}_1 \rightarrow \cdots \]

\[ c_1, c_2, c_3 \]

\[ S_0 \land \delta_N \rightarrow \hat{R}_1' \]

Repair \( R_1 \) to \( \hat{R}_1 \) s.t. \( S_0 \land \delta_N \rightarrow \hat{R}_1' \) is valid
Frame Repair

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

**Goal:** Compute frame sequence $S$ for mode $N$

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

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

Goal: Compute frame sequence $S$ for mode $N$
Strengthen Frames

Model $N = (\Sigma, Q_N, Q_0_N, \delta_N)$ and Safety property $\varphi$

**Goal:** Compute frame sequence $S$ for mode $N$
Strengthen Frames

Motivation

Preliminaries

Algorithm

Results

Internal State

(last known)

Frame Sequence $R$

(last known)

Invariant $\mathcal{I}$

(last known)

Error Trace $\mathcal{E}$

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

Goal: Compute frame sequence $S$ for mode $N$

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

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

**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

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

Goal: Compute frame sequence $S$ for mode $N$

- Invariant: $S_{i+1} \equiv S_i$
- Update last known
  1. Frame Sequence
  2. Invariant
Strengthen Frames

Internal State

Frame Sequence $R$

(last known)

Frame Sequence $S$

(last known)

Invariant $\mathcal{I}$

(last known)

Error Trace $\mathcal{E}$

Model $N = (\Sigma, Q_N, Q_0_N, \delta_N)$ and Safety property $\varphi$

Goal: Compute frame sequence $S$ for mode $N$

$N \not\models \varphi$

Error Trace: $\mathcal{E}_N$
Model $N = (\Sigma, Q_N, Q_{0N}, \delta_N)$ and Safety property $\varphi$

**Goal:** Compute frame sequence $S$ for mode $N$

$N \not\models \varphi$

**Error Trace:** $\mathcal{E}_N$

**Update last known**
1. Frame Sequence
Motivation

Preliminaries

Algorithm

Results

Strengthen Frames

Internal State

Frame Sequence $R$

(last known)

(last known)

(last known)

Invariant $\mathcal{I}$

Error Trace $\mathcal{E}$

Model $N = (\Sigma, Q_N, Q_0_N, \delta_N)$ and Safety property $\varphi$

Goal: Compute frame sequence $S$ for mode $N$

Frame Sequence $S$

Error Trace: $\mathcal{E}_N$

Update last known

1. Frame Sequence

2. Error Trace
Strengthen Frames

Internal State

(last known)

Frame Sequence $R$

(last known)

Invariant $I$

(last known)

Error Trace $E$

Ready for next model
FuseIC3 is coded in C++ and uses MathSAT5 as SAT solver.
**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\textsuperscript{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

\textsuperscript{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) \((\text{Een et al., 2011})\)
  - 2 Incremental IC3 (inc) \((\text{Chockler et al., 2011})\)
- Benchmarks evaluated
  - 1 NASA NextGen Air Traffic Control (ATC) System \((\text{Gario et al., 2016})\)
  - 2 Selected benchmarks from HWMCC 2015
    - Each model was randomly mutated to generate a model-set.
  - 3 Boeing AIR 6110 Wheel Braking System (WBS) \((\text{Bozzano et al., 2015})\)

Source code available at

http://temporallogic.org/research/FMCAD17

\(^1\)https://es-static.fbk.eu/people/griggio/ic3ia/

---

Rohit Dureja & Kristin Y. Rozier
FuseIC3: An Algorithm for Checking Large Design Spaces
NASA ATC Benchmark

1,620 Models
NASA ATC Benchmark

Safety Property

1,620 Models
FuseIC3: An Algorithm for Checking Large Design Spaces
FuseIC3: An Algorithm for Checking Large Design Spaces
NASA ATC Benchmark

Motivation
Preliminaries
Algorithm
Results

Rohit Dureja & Kristin Y. Rozier
FuseIC3: An Algorithm for Checking Large Design Spaces
NASA ATC Benchmark

Motivation

Preliminaries

Algorithm

Results

FuseIC3: An Algorithm for Checking Large Design Spaces
Selected HWMCC Benchmarks
Selected HWMCC Benchmarks

1% mutations

Original Benchmark

200 Models
Selected HWMCC Benchmarks

- Typical IC3
- Incremental IC3
- FuseIC3

Model ID: 0 200 400 600 800 1000 1200
Checking time (minutes): 0 10 20 30 40 50 60 70 80 91

Rohit Dureja & Kristin Y. Rozier
FuseIC3: An Algorithm for Checking Large Design Spaces
Boeing WBS Benchmark

7 models
Boeing WBS Benchmark

~300 Safety Properties

7 models
Boeing WBS Benchmark

\[ \sim 300 \text{ Safety Properties} \]

7 models
Boeing WBS Benchmark

Motivation
Preliminaries
Algorithm
Results

Rohit Dureja & Kristin Y. Rozier
FuseIC3: An Algorithm for Checking Large Design Spaces
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.
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.

Thank You!

http://temporallogic.org/research/FMCAD17

Rohit Dureja & Kristin Y. Rozier

FuseIC3: An Algorithm for Checking Large 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.
  ▶ 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

References