Incremental Methods for Checking Real-time Consistency

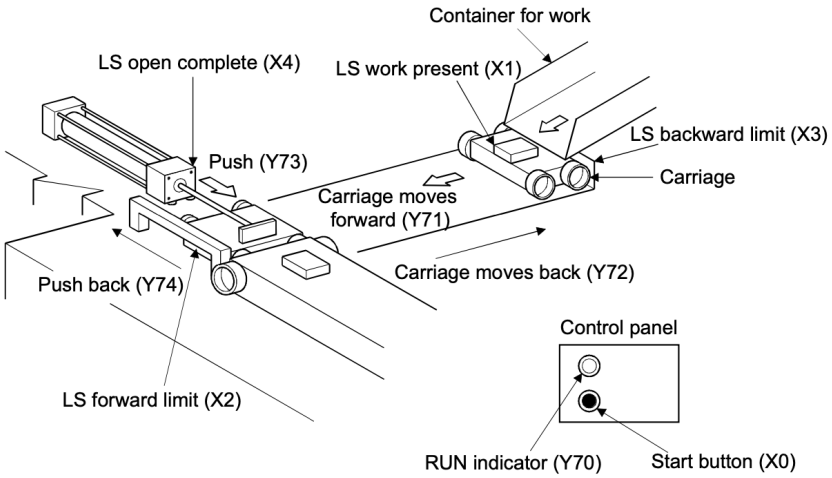
Thierry Jéron[1] Nicolas Markey[1] David Mentré[2]
Reya Noguchi[2] Ocan Sankur[1]

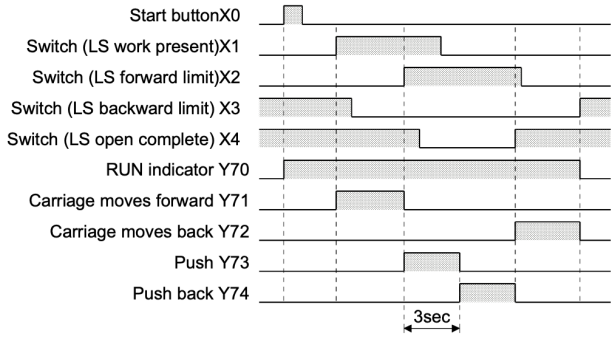
[1]Univ. Rennes, Inria & CNRS, Rennes (France)
[2]Mitsubishi Electric R&D Centre Europe (MERCE), Rennes (France)

Funded by French ANR project Ticktac
and MERCE/Inria collaboration

Context and Motivation

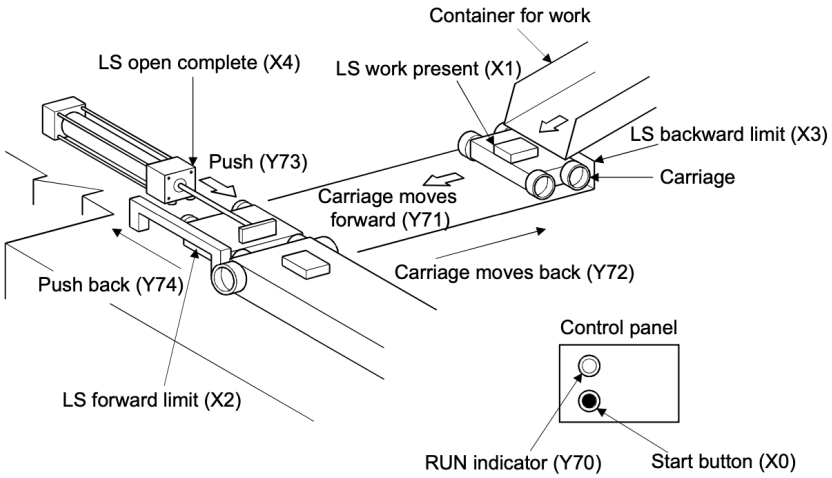
Requirement engineering for e.g. Programmable Logic Controllers in factory automation systems

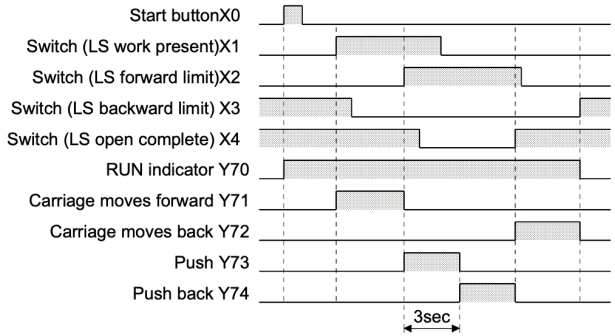




Context and Motivation

Requirement engineering for e.g. Programmable Logic Controllers in factory automation systems

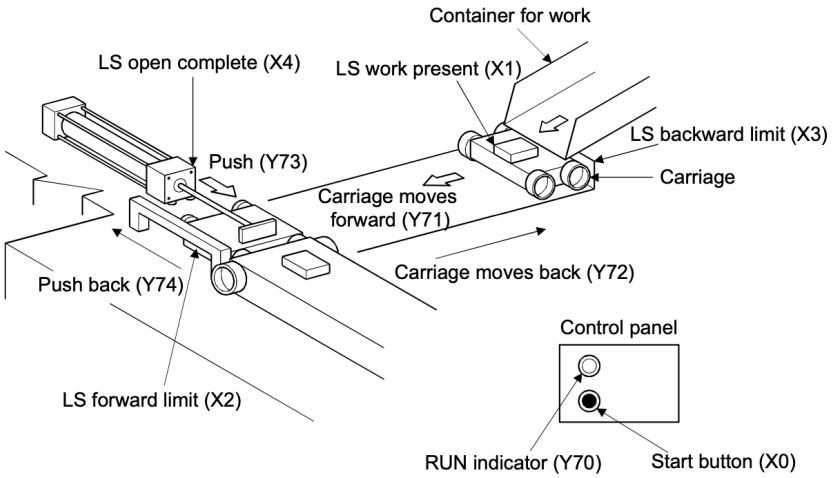


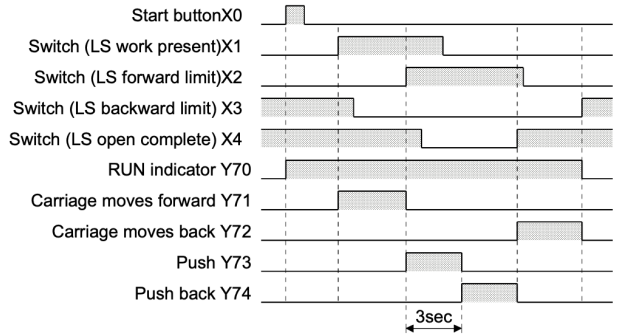


- Requirements express **functional timing constraints** of Boolean variables observed at discrete time steps, e.g.
  "*The arm shall push for 3s iff the carriage has reached the forward limit*".
- Used as the basis for implementation, verification and testing

Context and Motivation

Requirement engineering for e.g. Programmable Logic Controllers in factory automation systems





- Requirements express **functional timing constraints** of Boolean variables observed at discrete time steps, e.g.
  "*The arm shall push for 3s iff the carriage has reached the forward limit*".
- Used as the basis for implementation, verification and testing

Detecting **conflicts** in sets of requirements is important and challenging

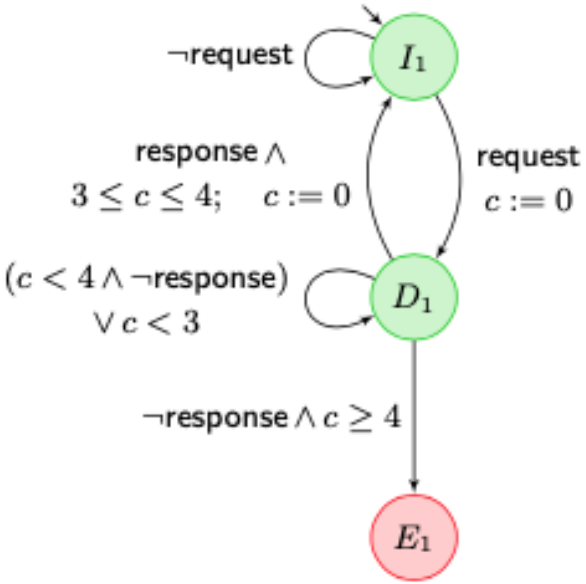
Requirements as Timed Automata

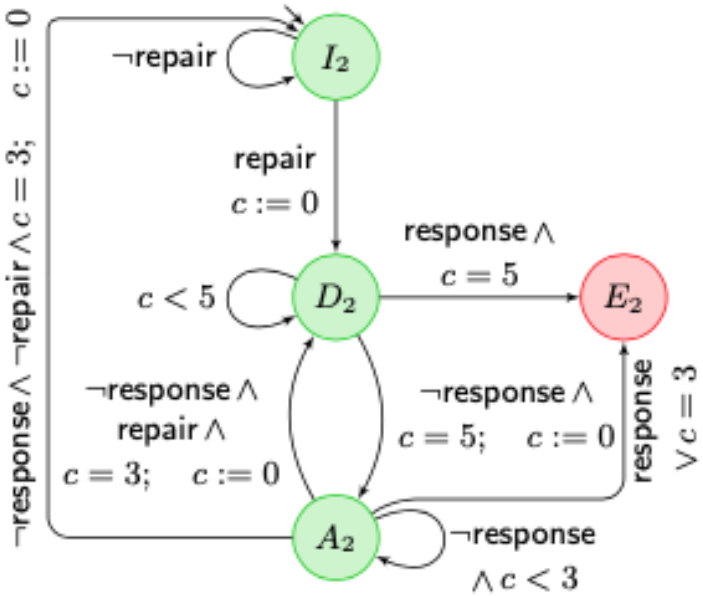
Patterns for Timed Temporal Properties: *Simplified Universal Patterns*

(Bienmüller et al. 2016)

$\rightarrow$ expressible as timed automata

R_1 : request $\xrightarrow{[3;4]}$ response R_2 : repair $\xrightarrow{[5;5]}$ $\neg$response$[3;3]$.

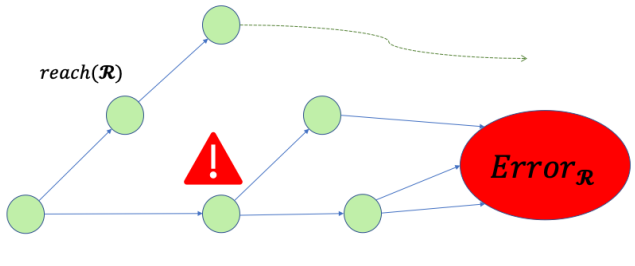




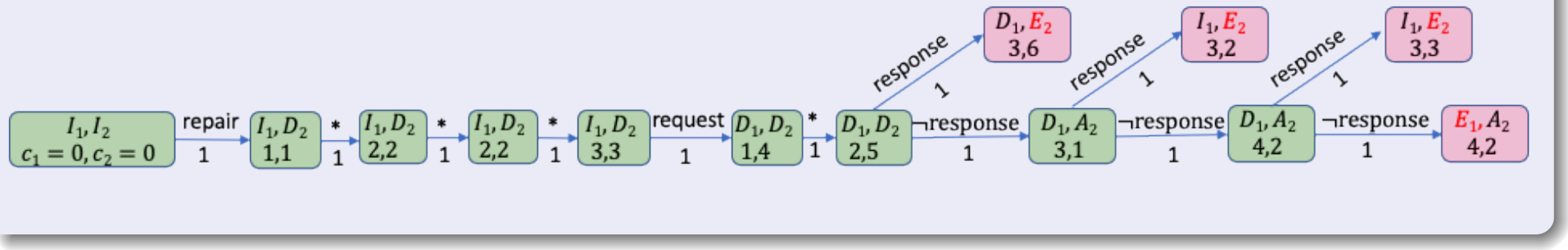
Real-time consistency

A set $\mathcal{R}$ of requirements is *rt-consistent* if all finite executions that do not violate $\mathcal{R}$ have **infinite** extensions that satisfy $\mathcal{R}$

(Post, Hoenicke, Podelski FASE 2011)



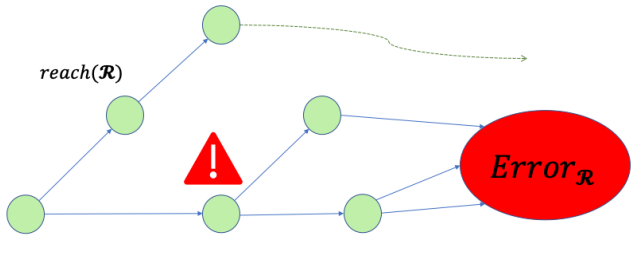
$\{R_1, R_2\}$ is inconsistent



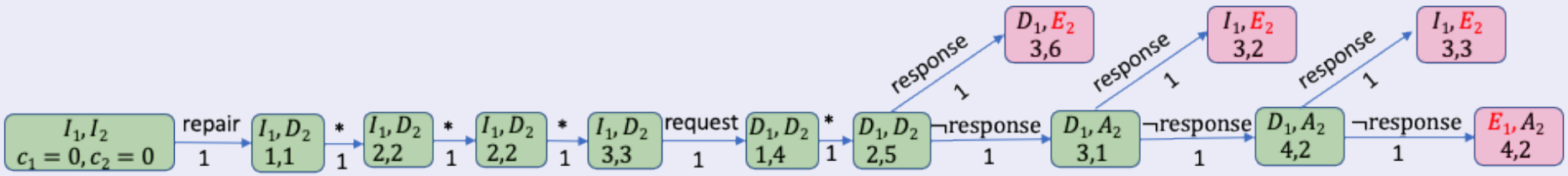
Real-time consistency

A set $\mathcal{R}$ of requirements is *rt-consistent* if all finite executions that do not violate $\mathcal{R}$ have **infinite** extensions that satisfy $\mathcal{R}$

(Post, Hoenicke, Podelski FASE 2011)



$\{R_1, R_2\}$ is inconsistent

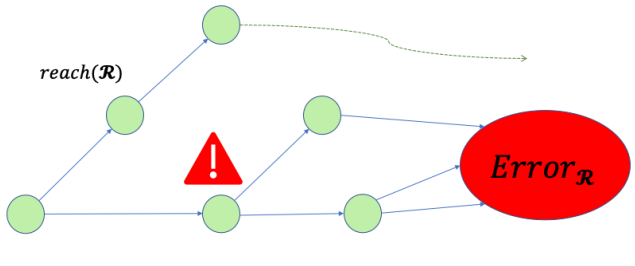


- RT-Inconsistency: R_1 cannot be satisfied without violating R_2, and vice versa
- Inevitability of the violation of $\mathcal{R}$ should be anticipated by $\mathcal{R}$

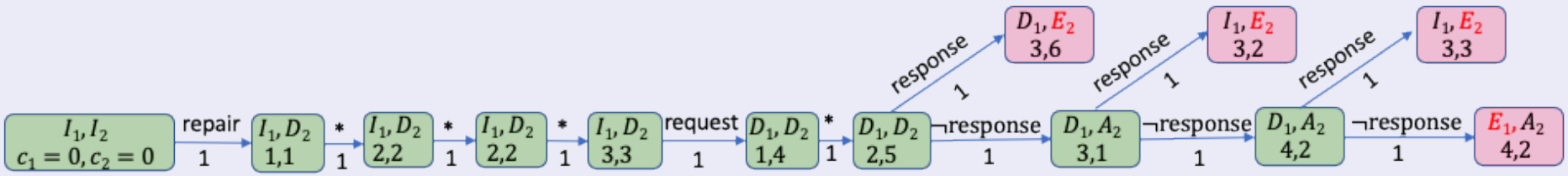
Real-time consistency

A set $\mathcal{R}$ of requirements is *rt-consistent* if all finite executions that do not violate $\mathcal{R}$ have **infinite** extensions that satisfy $\mathcal{R}$

(Post, Hoenicke, Podelski FASE 2011)



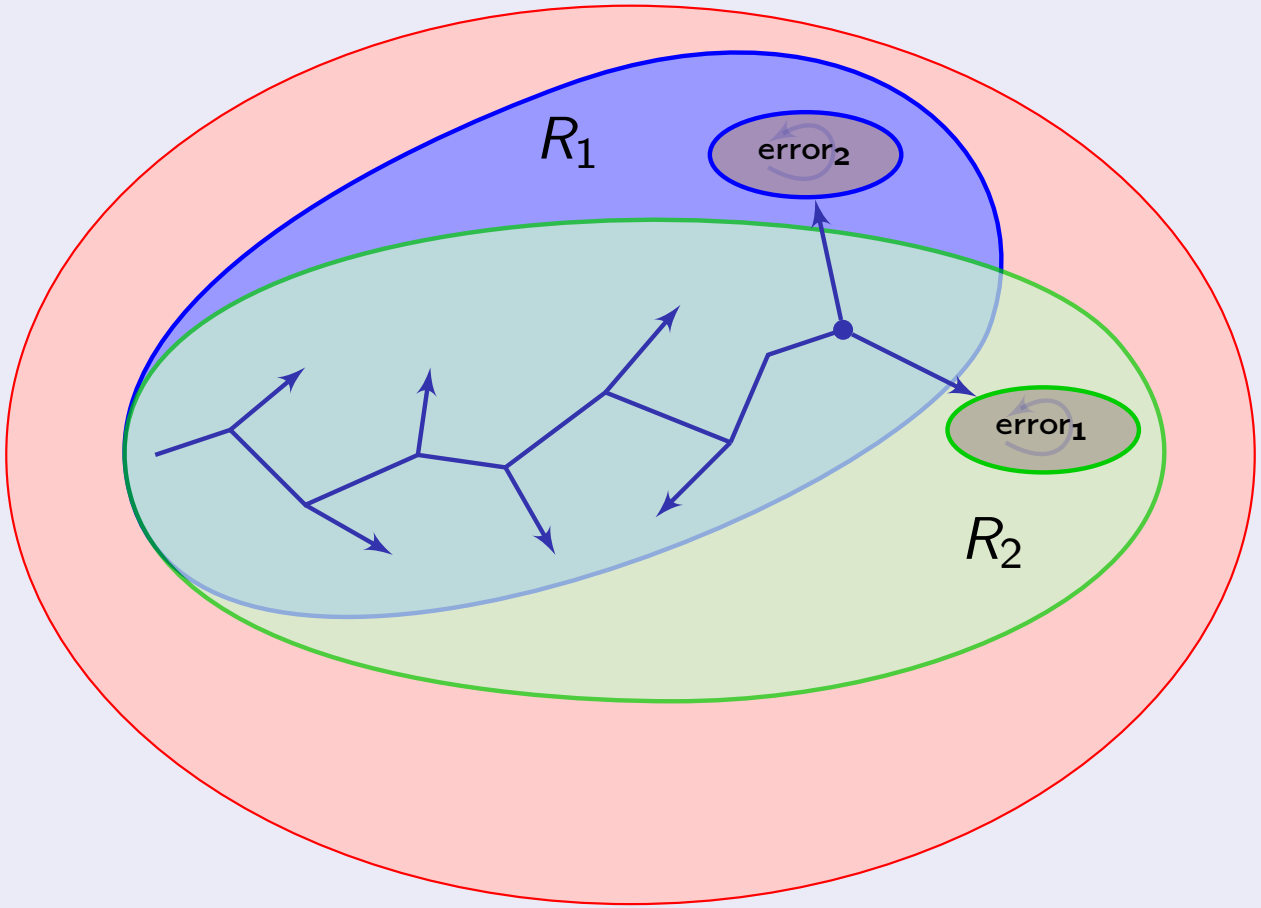
$\{R_1, R_2\}$ is inconsistent



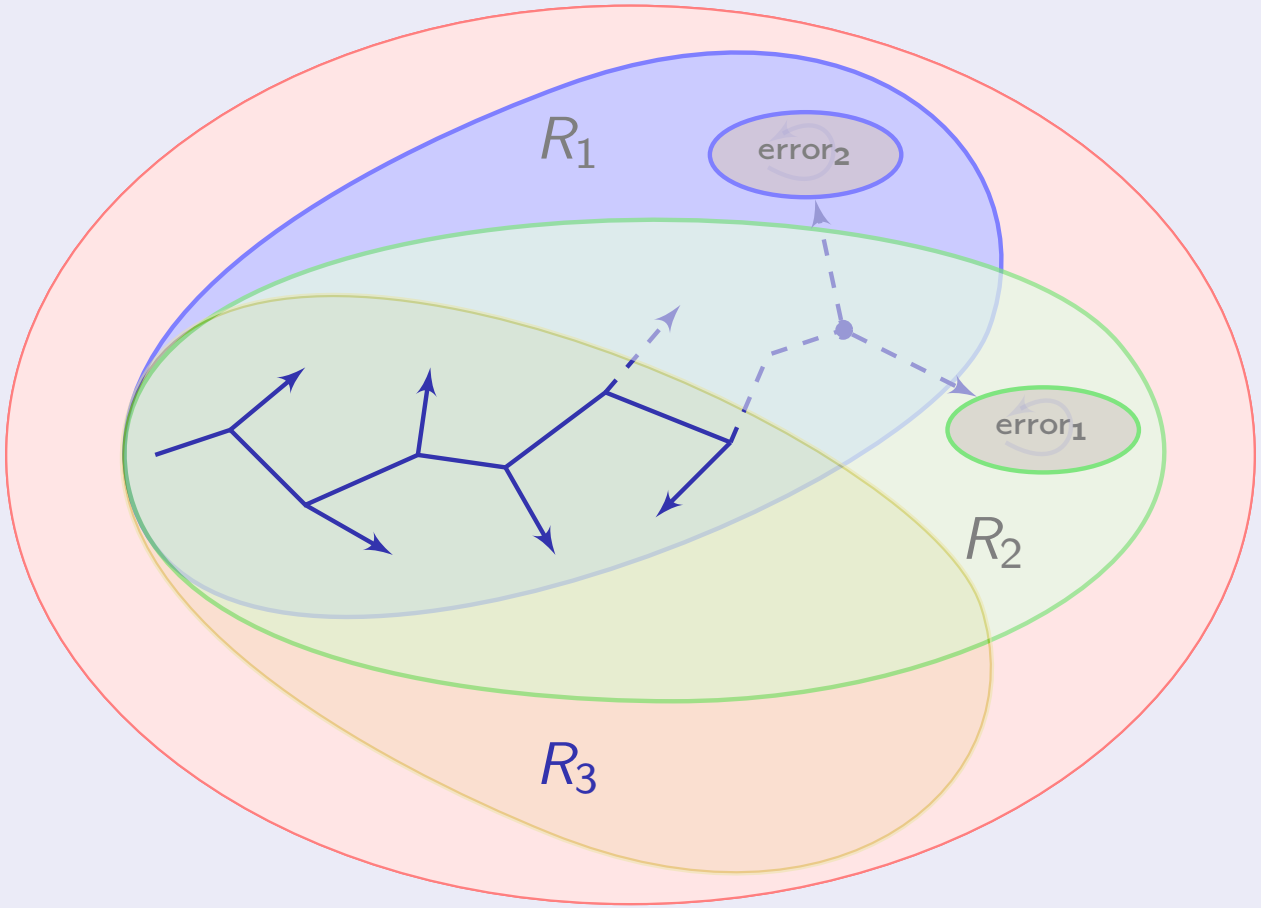
- RT-Inconsistency: R_1 cannot be satisfied without violating R_2, and vice versa
- Inevitability of the violation of $\mathcal{R}$ should be anticipated by $\mathcal{R}$

One way to fix the inconsistency: Add R_3 : *repair* $\rightarrow \neg$request$[5; 5]$

RT-Inconsistency



RT-Inconsistency



Contributions

Problem: Consistency checking is expensive due to state space explosion

Ideas

- Use cheaper notions / sufficient criteria
  (e.g. partial and bounded notions from previous work)
- Restrict to **subsets** of requirements (! false positive/negatives !)
- **Incremental** approach: Enlarge the subsets when needed

Contributions

Problem: Consistency checking is expensive due to state space explosion

Ideas

- Use cheaper notions / sufficient criteria
  (e.g. partial and bounded notions from previous work)
- Restrict to **subsets** of requirements (! false positive/negatives !)
- **Incremental** approach: Enlarge the subsets when needed

Algorithms

- Incremental rt-consistency checking based on CTL model checking
- Incremental algorithm for *partial-consistency checking* with an SMT solver that can find rt-inconsistencies (Becker 2019)
- Incremental algorithm for *partial rt-consistency* with an SMT solver that can find rt-inconsistencies

We also present a case study where these algorithms are evaluated

Some Related work

- Consistency appears in **contracts for system design** Benveniste et al. 2018

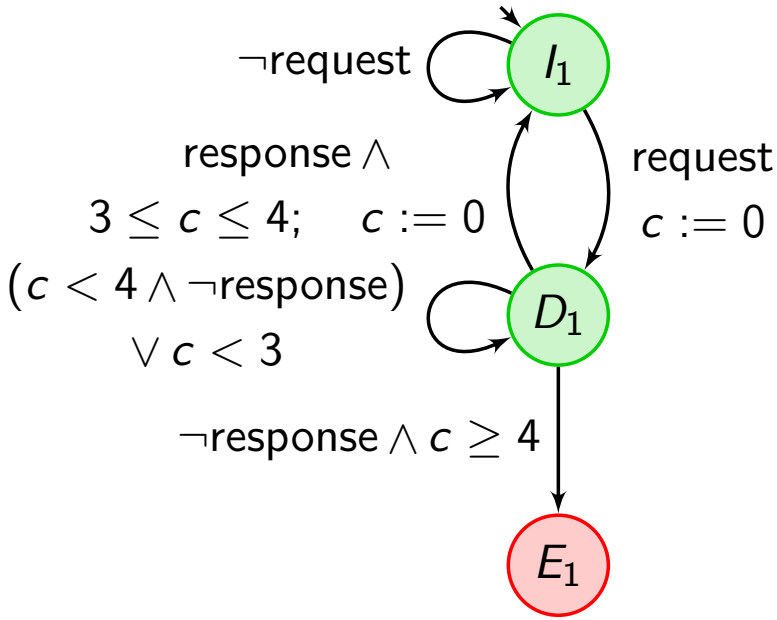
  Existence of an implementation or an execution Aichernig, Hörmaier, Lorber, Nickovíc, Tiran 2017 Ellen, Sieverding, Hungar 2014
- Strong consistency Baumgart et al. 2011
- Specifically **real-time** aspects:
  - **rt-consistency** Post, Hoenicke, Podelski 2011

    Algorithm based on duration calculus and reduction to timed automata model checking. Continuous time.
  - **Simplified Universal Patterns** Bienmüller, Teige, Eggers, Stasch 2016

    Consistency and coverage notions
  - **Partial consistency** Becker 2019

    A sufficient criterion checkable with a SAT/SMT solver that implies other notions of consistency

Requirements as Timed Automata



Complete deterministic timed automata with discrete unit delays

- Some event happens at each time unit
- With safety and co-safety conditions

But we restrict to safety in this talk

Violation / Inevitable Violation

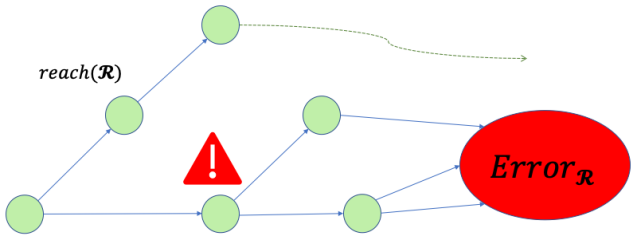
Let $\textbf{Error}_R$ be the **absorbing** set of error states of R

Definition

For a finite trace σ, a TA R:

- σ **fails** $R \triangleq$ the run over σ in R enters $\textbf{Error}_R$.
- σ **l-fails** $R \triangleq$ for all infinite traces σ', $\sigma \cdot \sigma'$ **fails** R.

Remark: **fails** implies **l-fails**, but the inverse does not hold.



Violation / Inevitable Violation

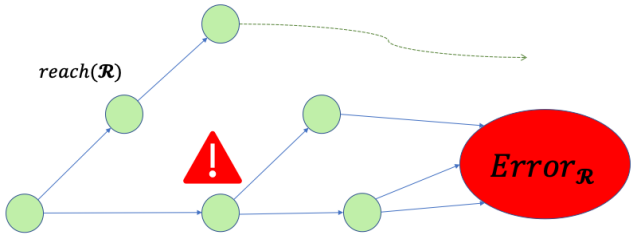
Let $\mathbf{Error}_R$ be the **absorbing** set of error states of R

Definition

For a finite trace σ, a TA R:

- σ **fails** $R \triangleq$ the run over σ in R enters $\mathbf{Error}_R$.
- σ I-fails $R \triangleq$ for all infinite traces σ', $\sigma \cdot \sigma'$ **fails** R.

Remark: **fails** implies I-fails, but the inverse does not hold.



- For a set $\mathcal{R} = \{R_i\}_i$, let $\mathbf{Error}_{\mathcal{R}} \triangleq \bigvee_i \mathbf{Error}_{R_i}$ for the product $\otimes\mathcal{R}$.
- If $\mathcal{R}' \subseteq \mathcal{R}$, σ **fails** (*resp.* I-fails) $\mathcal{R}' \implies \sigma$ **fails** (*resp.* I-fails) $\mathcal{R}$.

Violation / Inevitable Violation

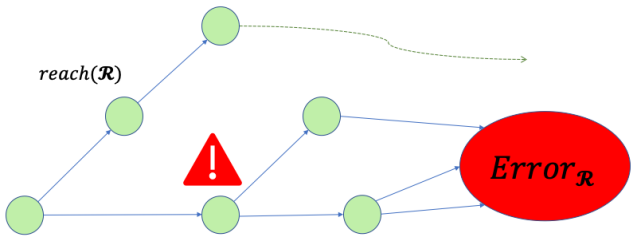
Let $\textbf{Error}_R$ be the **absorbing** set of error states of R

Definition

For a finite trace σ, a TA R:

- σ **fails** $R \triangleq$ the run over σ in R enters $\textbf{Error}_R$.
- σ l-fails $R \triangleq$ for all infinite traces σ', $\sigma \cdot \sigma'$ **fails** R.

Remark: **fails** implies l-fails, but the inverse does not hold.



$\mathcal{R}$ is *rt-consistent* if $\forall\sigma$, σ l-fails $\mathcal{R} \implies \sigma$ **fails** $\mathcal{R}$.

Contributions

Three Incremental Algorithms to find rt-inconsistencies

- **Algorithm 1**: Based on CTL model checking
- Algorithm 2: Based on *partial consistency* and an SMT solver
- **Algorithm 3**: Based on *partial rt-consistency* and an SMT solver

Contributions

Three Incremental Algorithms to find rt-inconsistencies

- **Algorithm 1**: Based on CTL model checking
- Algorithm 2: Based on *partial consistency* and an SMT solver
- **Algorithm 3**: Based on *partial rt-consistency* and an SMT solver

▶ **Incremental**: Consider subsets of requirements, and enlarge them as needed

▶ Algorithms 1 and 3 can be made **complete** when run with the whole set of requirements

▶ But our focus is to find rt-inconsistencies **quickly** rather than proving rt-consistency

Algorithm 1: Incremental RT-Consistency

Post et al. 2011: rt-consistency checking is reduced to model checking **duration calculus**, and the use of a **timed automata model checker**

Algorithm 1: Incremental RT-Consistency

Post et al. 2011: rt-consistency checking is reduced to model checking **duration calculus**, and the use of a **timed automata model checker**

▶ **Our contribution**: Using CTL model checking in discrete time:
$\mathcal{R}$ has a witness to rt-inconsistency if, and only if,

$$\mathcal{R} \models \mathbf{E}(\neg \mathbf{Error}_{\mathcal{R}}\ \mathbf{U}\ (\neg \mathbf{Error}_{\mathcal{R}} \wedge \mathbf{AF}\ \mathbf{Error}_{\mathcal{R}}))$$

$$\iff \mathcal{R} \models \mathbf{E}(\neg \mathbf{Error}_{\mathcal{R}}\ \mathbf{U}\ (\neg \mathbf{Error}_{\mathcal{R}} \wedge \mathbf{AX}\ \mathbf{Error}_{\mathcal{R}})).$$

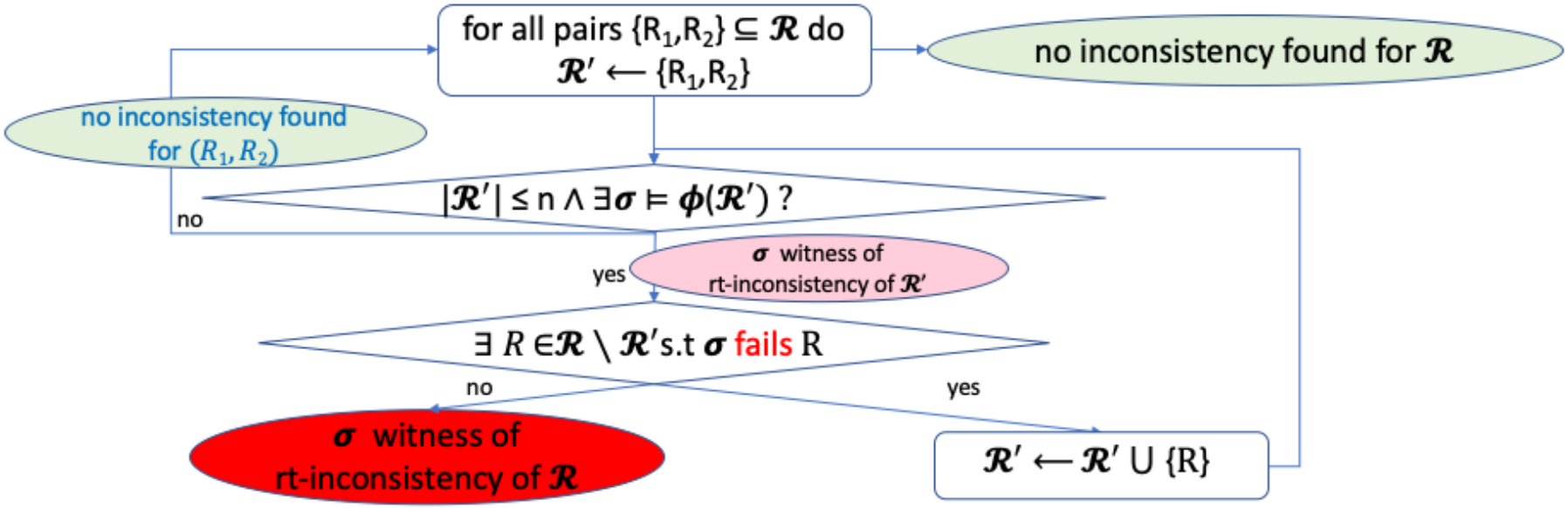
Given any set $\mathcal{R}$, this can be checked with a CTL model checker (e.g. NuSMV)

But it is too costly for large sets $\mathcal{R}$

Algorithm 1: Incremental RT-Consistency

Let $\phi(\mathcal{R}) \leftarrow \mathbf{E}[\neg \mathbf{Error}_{\mathcal{R}}\ \mathbf{U}(\neg \mathbf{Error}_{\mathcal{R}} \wedge \mathbf{AX}\ \mathbf{Error}_{\mathcal{R}})]$

Input: $\mathcal{R}, n \geq 1$



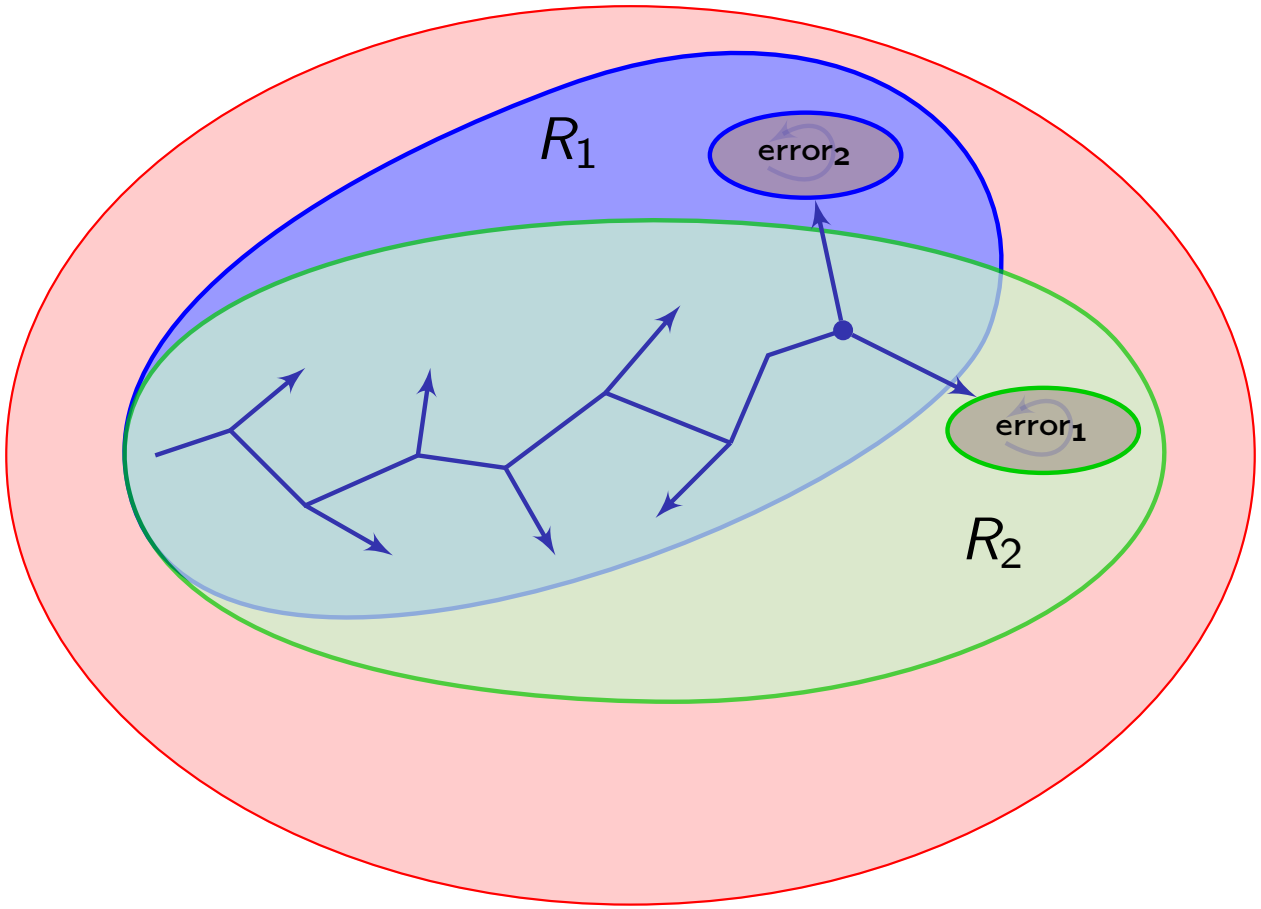
Lifting witnesses of rt-inconsistencies to supersets

Let $\sigma \in \Sigma^*$ be a witness for the rt-inconsistency of $\mathcal{R}' \subseteq \mathcal{R}$.
If $\neg(\sigma$ **fails** $\mathcal{R})$, then σ is a witness for the rt-inconsistency of $\mathcal{R}$.

Any finite trace σ returned by Alg. 1 witnesses the rt-inconsistency of $\mathcal{R}$.

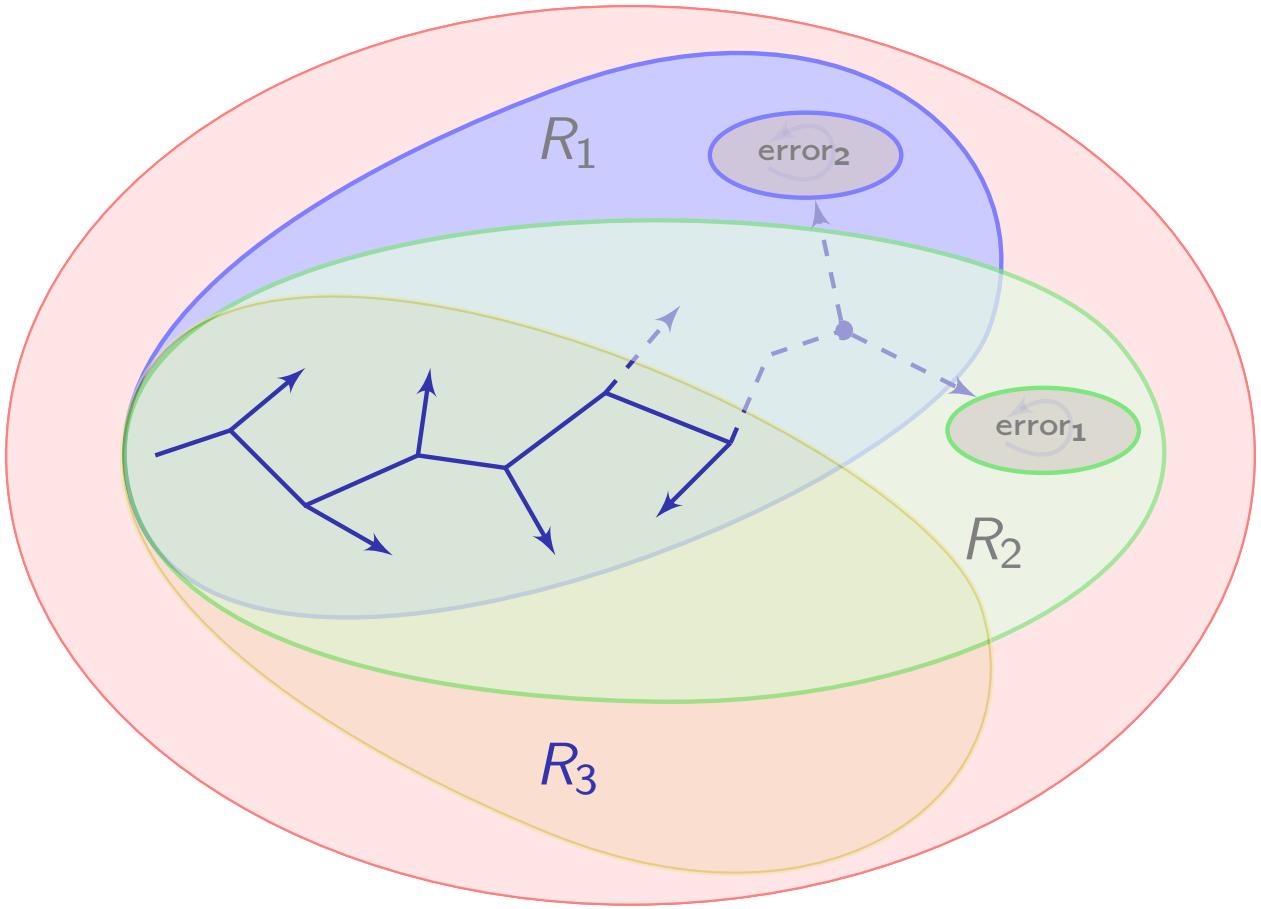
Algorithm 1: Incremental RT-Consistency

In the algorithm: If σ fails R, then we add $\mathcal{R}' \leftarrow \mathcal{R}' \cup \{R\}$
Explanation:



Algorithm 1: Incremental RT-Consistency

In the algorithm: If σ fails R, then we add $\mathcal{R}' \leftarrow \mathcal{R}' \cup \{R\}$
Explanation:



Algorithm 3: Partial rt-consistency

Idea

Check immediate conflicts (AX **Error**$_{\mathcal{S}}$) in **subsets** $\mathcal{S} \subseteq \mathcal{R}$, $|\mathcal{S}| \leq n$, at **bounded depth** α.

"Bounded model checking" approach to Alg. 1

Algorithm 3: Partial rt-consistency

Idea

Check immediate conflicts (AX $\mathbf{Error}_{\mathcal{S}}$) in **subsets** $\mathcal{S} \subseteq \mathcal{R}$, $|\mathcal{S}| \leq n$, at **bounded depth** α.

"Bounded model checking" approach to Alg. 1

Input: $\mathcal{R}, \alpha, n \geq 1$
$\mathcal{R}$ is *α-bounded n-partially rt-inconsistent* if $\exists \mathcal{S} \subseteq \mathcal{R}$, $|\mathcal{S}| \leq n$, s.t. $\phi_{p,\alpha} = \mathbf{E}(\neg \mathbf{Error}_{\mathcal{S}} \ \mathbf{U}_{\alpha} \ (\neg \mathbf{Error}_{\mathcal{S}} \wedge \mathbf{AX}\ \mathbf{Error}_{\mathcal{S}}))$ holds.

Algorithm 3: Partial rt-consistency

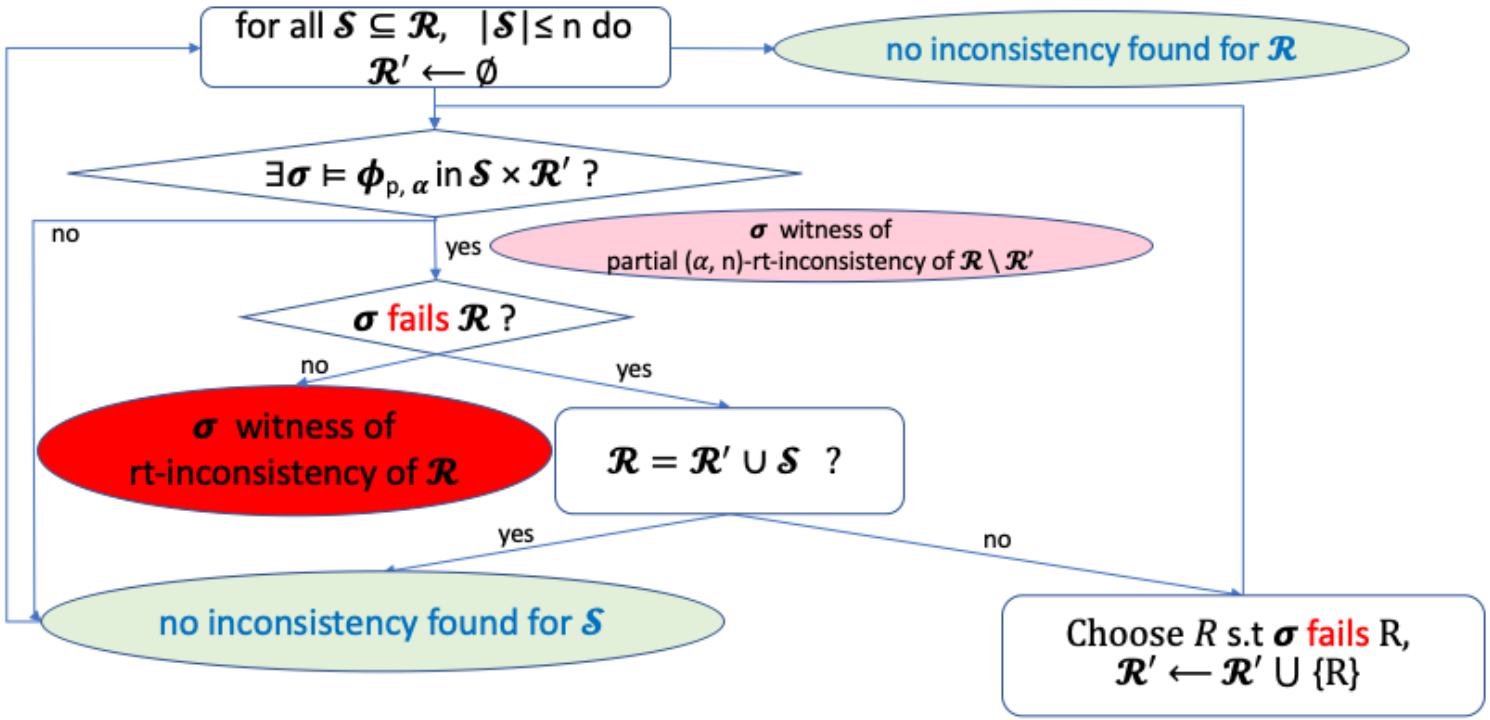
Idea

Check immediate conflicts (AX $\mathbf{Error}_{\mathcal{S}}$) in **subsets** $\mathcal{S} \subseteq \mathcal{R}$, $|\mathcal{S}| \leq n$, at **bounded depth** α.

"Bounded model checking" approach to Alg. 1

Input: $\mathcal{R}, \alpha, n \geq 1$
$\mathcal{R}$ is *α-bounded n-partially rt-inconsistent* $\setminus \mathcal{R}'$ if $\exists \mathcal{S} \subseteq \mathcal{R}$, $|\mathcal{S}| \leq n$, s.t. $\phi_{p,\alpha} = \mathbf{E}(\neg \mathbf{Error}_{\mathcal{S} \cup \mathcal{R}'}\ \mathbf{U}_{\alpha}\ (\neg \mathbf{Error}_{\mathcal{S} \cup \mathcal{R}'} \wedge \mathbf{AX}\ \mathbf{Error}_{\mathcal{S}}))$ holds.

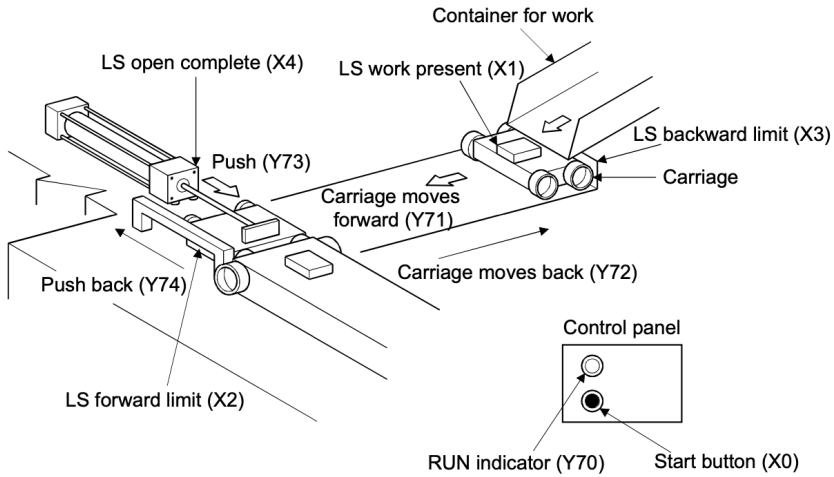
Algorithm 3: Incremental partial rt-consistency checking



Any finite trace σ returned by Alg. 3 witnesses the rt-inconsistency of $\mathcal{R}$.

Experiments: factory automation use case

Algo 1 uses NuSMV for CTL model checking
Algo 2-3 use SMT solver Z3.
$\alpha = 40$ and $n = 2$.



set	**size** T + Bool	**rt-consist.** Algo 1	**partial consist.** Algo 2	**partial rt-consist.** Algo. 3
#1	6 + 9	5 inc. (24s)	4 inc. (36s)	5 inc. (39s)
#2	8 + 10	1 inc. (21s)	✓ (55s)	1 inc. (101s)
#3	8 + 10	✓ (24s)	✓ (61s)	✓ (115s)
#4	10 + 16	✓ (359s)	✓ (85s)	✓ (141s)
#5	12 + 16	✓ (1143s)	✓ (133s)	✓ (227s)
#6	13 + 16	✓ (5311s)	✓ (138s)	✓(232s)

The complete version of Alg. 1 determined that sets 3, 4, 5 are rt-consistent

Conclusion and future work

Conclusion

- Use CTL model checking for rt-consistency
- Incremental algorithms to finding rt-inconsistencies
- Established link between partial consistency notions and rt-consistency

Future work

- Evaluate/improve performance on larger use cases
- Investigate other variants of consistency, trade-off