Scalable Redundancy Detection for Real Time Requirements

Lena Funk, joint work with Elisabeth Henkel, Nico Hauff, Vincent Langenfeld, Andreas Podelski

September 5, 2024

University of Freiburg

Why detect redundancy in requirements?

- Requirements are correctness criteria, represent the desired behaviour of a system.

Why detect redundancy in requirements?

- Requirements are correctness criteria, represent the desired behaviour of a system.
- A requirements specification *should* describe a system *correctly*, *completely*, and *concisely*.

Why detect redundancy in requirements?

- Requirements are correctness criteria, represent the desired behaviour of a system.
- A requirements specification *should* describe a system *correctly*, *completely*, and *concisely*.
- Redundancies in a requirements specification can be *intended* or *unintended*.

Why detect redundancy in requirements?

- Requirements are correctness criteria, represent the desired behaviour of a system.
- A requirements specification *should* describe a system *correctly*, *completely*, and *concisely*.
- Redundancies in a requirements specification can be *intended* or *unintended*.
- Either way, *they have to be known*.

What is redundancy?

A requirement is redundant if it can be omitted from the set of requirements without changing the specified system behaviour.

What is redundancy?

A requirement is redundant if it can be omitted from the set of requirements without changing the specified system behaviour.

$$\bigwedge_{\{r_i \in \mathcal{R} | i \neq j\}} r_i \models r_j$$

How do we detect redundancy?

$$\bigwedge_{\{r_i \in \mathcal{R} \mid i \neq j\}} r_i \models r_j$$

How do we detect redundancy?

$$\bigwedge_{\{r_i \in \mathcal{R} \mid i \neq j\}} r_i \models r_j$$

$$\bigcap_{\{\mathcal{A}_i \in \mathcal{R} \mid i \neq j\}} \mathcal{L}(\mathcal{A}_i) \subseteq \mathcal{L}(\mathcal{A}_j)$$

How do we detect redundancy?

$$\bigwedge_{\{r_i \in \mathcal{R} \mid i \neq j\}} r_i \models r_j$$

$$\bigcap_{\{\mathcal{A}_i \in \mathcal{R} \mid i \neq j\}} \mathcal{L}(\mathcal{A}_i) \subseteq \mathcal{L}(\mathcal{A}_j)$$

$$\bigcap_{\{\mathcal{A}_i \in \mathcal{R} \mid i \neq j\}} \mathcal{L}(\mathcal{A}_i) \cap \overline{\mathcal{L}(\mathcal{A}_j)} = \emptyset$$

How do we detect redundancy?

$$\bigwedge_{\{r_i \in \mathcal{R} \mid i \neq j\}} r_i \models r_j$$

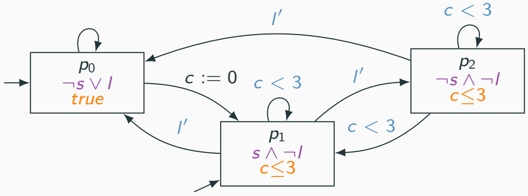
$$\bigcap_{\{\mathcal{A}_i \in \mathcal{R} \mid i \neq j\}} \mathcal{L}(\mathcal{A}_i) \subseteq \mathcal{L}(\mathcal{A}_j)$$

$$\bigcap_{\{\mathcal{A}_i \in \mathcal{R} \mid i \neq j\}} \mathcal{L}(\mathcal{A}_i) \cap \overline{\mathcal{L}(\mathcal{A}_j)} = \emptyset$$

$$\neg \exists \pi \bullet \mathcal{P}(\mathcal{A}_0, \ldots, \overline{\mathcal{A}_j}, \ldots, \mathcal{A}_n) \ni \pi$$

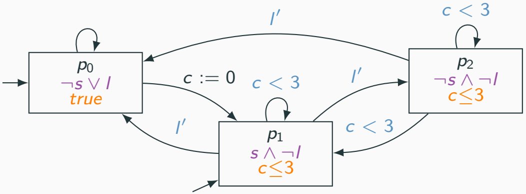
Complementing Phase Event Automata

$\mathtt{r}_1$: If *sensor* holds, then *light* holds after at most 3 time units.



Complementing Phase Event Automata

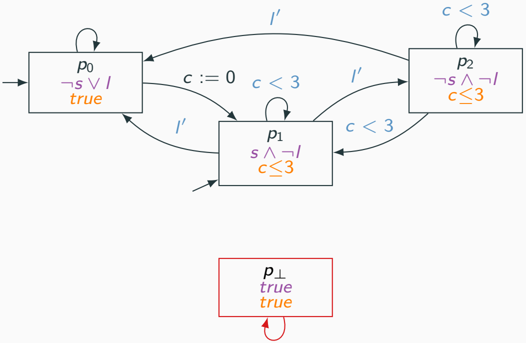
$\mathtt{r}_1$: If *sensor* holds, then *light* holds after at most 3 time units.



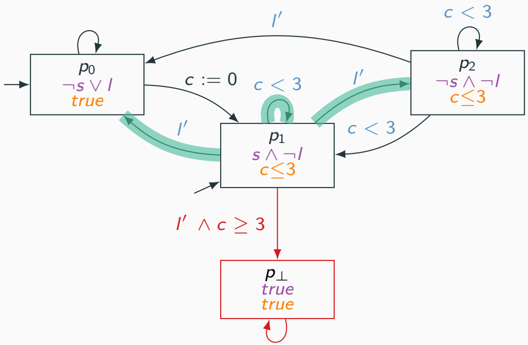
run: sequence of configurations: $(p_0, \beta_0, \gamma_0, t_0), \ldots, (p_n, \beta_n, \gamma_n, t_n)$

Complementing Phase Event Automata

$\mathtt{r}_1$: If *sensor* holds, then *light* holds after at most 3 time units.

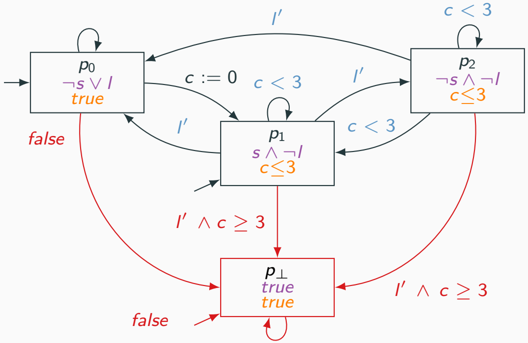


r_1: If *sensor* holds, then *light* holds after at most 3 time units.



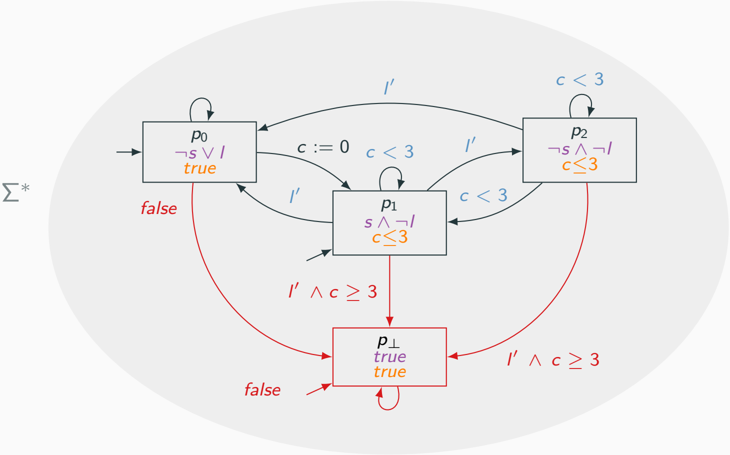
Complementing Phase Event Automata

$\mathtt{r}_1$: If *sensor* holds, then *light* holds after at most 3 time units.



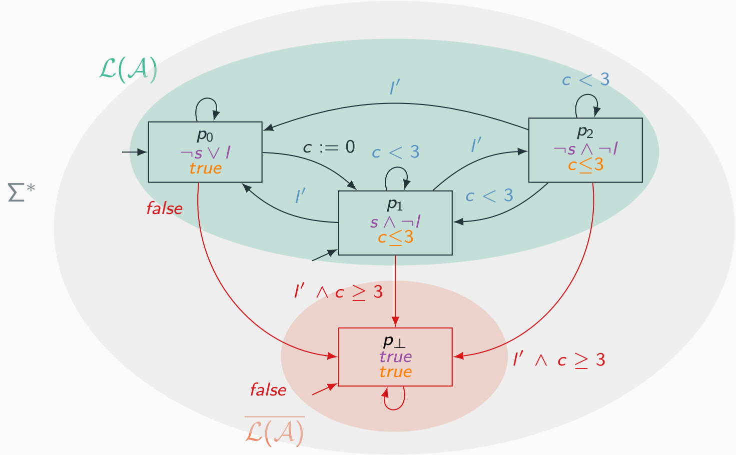
Complementing Phase Event Automata

$\mathtt{r}_1$: If *sensor* holds, then *light* holds after at most 3 time units.



Complementing Phase Event Automata

$\mathtt{r}_1$: If *sensor* holds, then *light* holds after at most 3 time units.



Encoding Redundancy as a Program Analysis Task

- Instead of encoding $\mathcal{P}(\mathcal{A}_0, .., \overline{\mathcal{A}_j}, ..., \mathcal{A}_n)$ for each $\mathtt{r}_j$, we encode $\mathcal{P}_{red} = \mathcal{P}(\mathcal{A}^t_0, .., \mathcal{A}^t_j, ..., \mathcal{A}^t_n)$ only once.
- $\mathcal{P}_{red}$ simulates the execution of $\mathcal{A}_{red} = \mathcal{A}^t_0 || ... || \mathcal{A}^t_j || ... || \mathcal{A}^t_n$.
- A run in $\mathcal{A}_{red}$ that contains a configuration $((p_0, ..., p_j, ..., p_n), \beta, \gamma, t)$, where $p_j = p^j_\perp$, while $p_i \neq p^i_\perp$ for all $i \neq j$, represents system behaviour that violates $\mathtt{r}_j$, but is not prohibited by the rest.
- For each requirement $\mathtt{r}_j$: introduce an error location l^j_{err} to $\mathcal{P}_{red}$ with an annotation expressing the above.

Encoding Redundancy as a Program Analysis Task

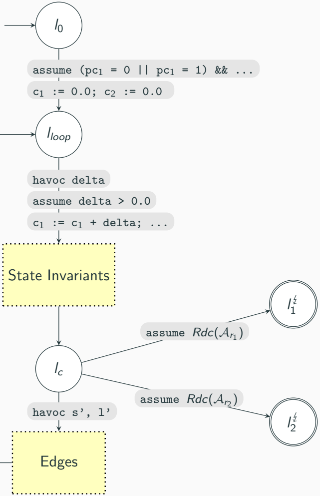
- Instead of encoding $\mathcal{P}(\mathcal{A}_0, .., \overline{\mathcal{A}_j}, ..., \mathcal{A}_n)$ for each $\mathtt{r}_j$, we encode $\mathcal{P}_{red} = \mathcal{P}(\mathcal{A}_0^t, .., \mathcal{A}_j^t, ..., \mathcal{A}_n^t)$ only once.
- $\mathcal{P}_{red}$ simulates the execution of $\mathcal{A}_{red} = \mathcal{A}_0^t || ... || \mathcal{A}_j^t || ... || \mathcal{A}_n^t$.
- A run in $\mathcal{A}_{red}$ that contains a configuration $((p_0, ..., p_j, ..., p_n), \beta, \gamma, t)$, where $p_j = p_\perp^j$, while $p_i \neq p_\perp^i$ for all $i \neq j$, represents system behaviour that violates $\mathtt{r}_j$, but is not prohibited by the rest.
- For each requirement $\mathtt{r}_j$: introduce an error location l_{err}^j to $\mathcal{P}_{red}$ with an annotation expressing the above.

l_{err}^j is reachable if and only if the requirement is not redundant.
l_{err}^j is not reachable if and only if the requirement is redundant

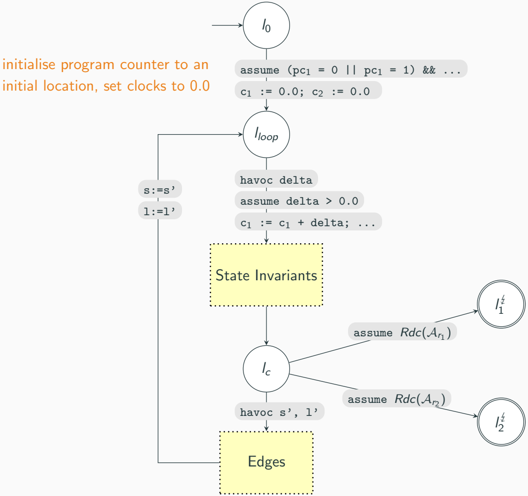
Toy Example

$\mathtt{r}_1$: If *sensor* holds, then *light* holds after at most 3 time units.
$\mathtt{r}_2$: If *sensor* holds, then *light* holds after at most 5 time units.

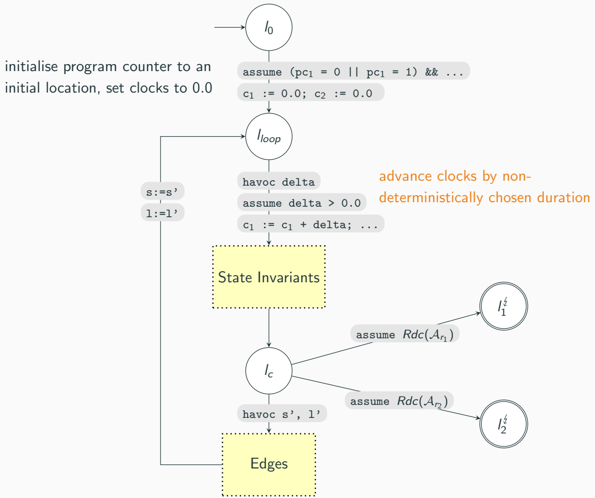
Resulting Boogie Program (for our toy example)



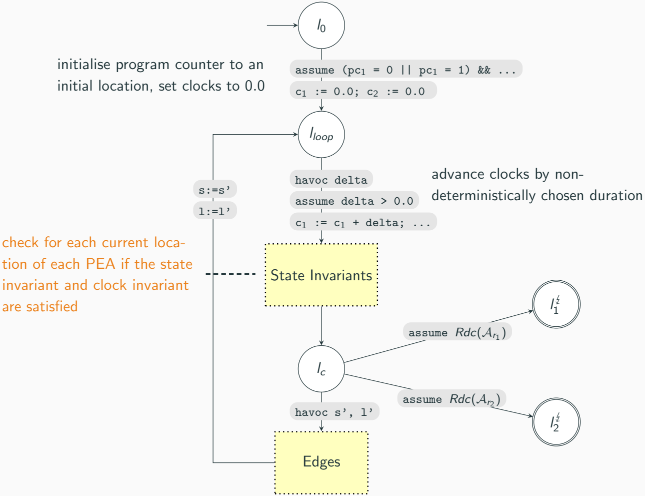
Resulting Boogie Program (for our toy example)



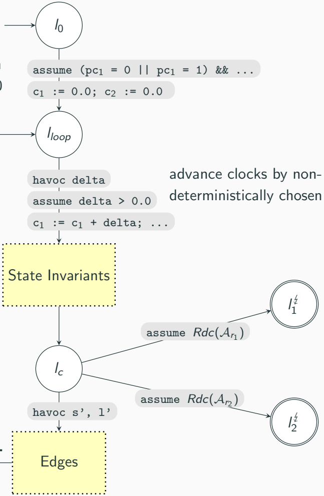
Resulting Boogie Program (for our toy example)



Resulting Boogie Program (for our toy example)



Resulting Boogie Program (for our toy example)



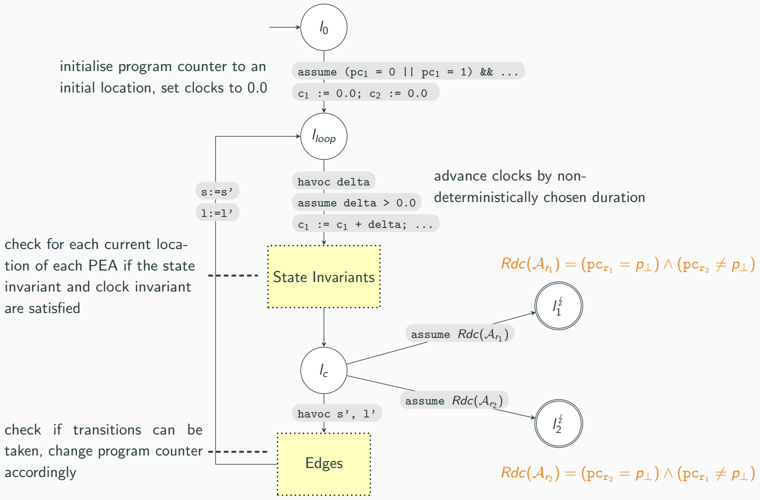
initialise program counter to an initial location, set clocks to 0.0

advance clocks by non-deterministically chosen duration

check for each current location of each PEA if the state invariant and clock invariant are satisfied

check if transitions can be taken, change program counter accordingly

Resulting Boogie Program (for our toy example)



Evaluation

	Requirements			Redundancy			
ID	R	RT	V	No	Yes	TO	T (min)
dev-01	26	21	27	26	0	0	0.7
dev-02	50	47	53	49	1	0	5.8
dev-03	52	11	34	51	0	1	15.9
dev-04	58	53	53	57	1	0	7.2
dev-05	68	64	89	64	2	2	39.7
abz	83	52	52	78	5	0	23.3
dev-06	100	95	101	99	0	1	21.6
dev-07	107	80	172	107	0	0	3.5
dev-08	263	234	239	235	7	21	375.5
dev-09	407	358	326	396	4	7	464.1
dev-10	699	543	1003	684	7	8	819.2

- Implemented as part of Ultimate ReqAnalyzer
- 15 min timeout per requirement; AMD Ryzen 5 5600 6-Core CPU with 3.5 GHz and 30 GB RAM

Conclusion

Recap

- Classical approach to redundancy
- Encoded as program analysis task
- Scales well on real requirements sets

Conclusion

Recap

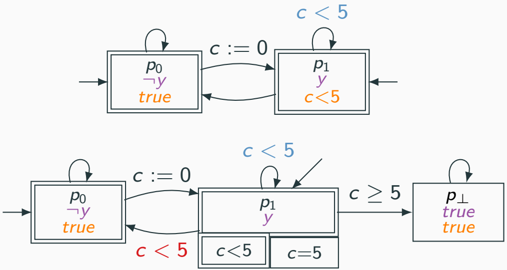
- Classical approach to redundancy
- Encoded as program analysis task
- Scales well on real requirements sets

Future Work

- Extract explanations to support interpretation of redundancy analysis results
- Minimisation of Phase Event Automata
- Upcoming journal submission: Redundancy vs. Vacuity

Formalization

Strict Clock Invariants



Deep Dive

Non-sink transitions of the totalized PEA:

$$E^t(p) := \begin{cases} E(p) & \text{if } I(p) = I^t(p), \\ \{(p, g^t, X, p') \mid (p, g, X, p') \in E \wedge g^t = (g \wedge \bigwedge_{(c_i < t_i) \in I(p)} c_i < t_i)\} & \text{otherwise.} \end{cases}$$

Guards for the sink transitions:

$$g_\perp(p) := p_\perp \neg \bigvee_{(p,g,X,p') \in E^t} \left(g \wedge s'(p') \wedge \bigwedge_{\{\delta_c \mid \delta_c \in I_<(p') \wedge c \notin X\}} \delta_c \right)$$

$$g_\perp^{in} := \neg \bigvee_{(g,p) \in E_0} (g \wedge s'(p))$$

Sink transitions of the totalized PEA:

$$E_\perp := \bigcup_{p \in P} (p, g_\perp(p), \emptyset, p_\perp) \cup \{(p_\perp, true, \emptyset, p_\perp)\}$$