ICECCS 2013

18th July 2013
Singapore

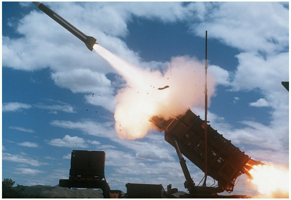
Observer Patterns for Real-Time Systems

Étienne André

Laboratoire d'Informatique de Paris Nord
Université Paris 13, Sorbonne Paris Cité, France

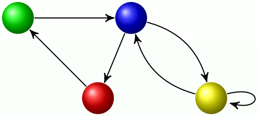
Context: Verifying Complex Timed Systems (1/2)

- Need for early bug detection
  - Bugs discovered when final testing: expensive
  - $\leadsto$ Need for a thorough modeling and verification phase



Context: Verifying Complex Timed Systems (2/2)

- Use formal methods

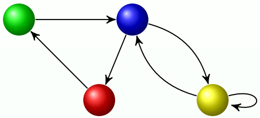


A model of the system

A property to be satisfied

Context: Verifying Complex Timed Systems (2/2)

- Use formal methods



$\overset{?}{\models}$

 is unreachable

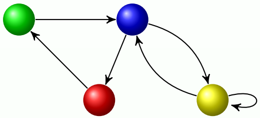
A model of the system

A property to be satisfied

- Question: does the model of the system satisfy the property?

Context: Verifying Complex Timed Systems (2/2)

- Use formal methods



$\overset{?}{\models}$

 is unreachable

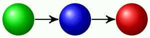
A model of the system

A property to be satisfied

- Question: does the model of the system satisfy the property?

Yes

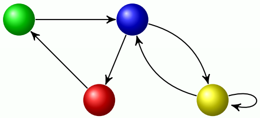
No



Counterexample

Context: Verifying Complex Timed Systems (2/2)

- Use formal methods



$\overset{?}{\models}$

 is unreachable

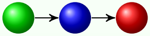
A model of the system

A property to be satisfied

- Question: does the model of the system satisfy the property?

Yes

No



Counterexample

Specifying Properties for Real-Time Systems

- (Timed) temporal logics [Pnueli, 1977, Baier and Katoen, 2008]
  - ☺ Very expressive
  - ☹ Not easy to handle by non-experts
    - EF((EX.P)U(AG.Q))
  - ☹ Expressiveness not often supported in full by tools
  - ☹ Requires complex model checking algorithms
  - ☹ Too expressive?

- Observer-based properties [Aceto et al., 1998b, Aceto et al., 1998a]
  - ☺ Quite expressive
  - ☺ Do not require complex algorithms
  - ☹ Not easy to handle by non-experts
  - ☹ Too expressive?

Outline

Outline

Goal

- Identify and express common properties met in the design and verification of complex real-time systems
- Define an associated formal semantics
- Shall be non-compositional
  - ☺ Avoids mistakes or ill-formed properties
  - ☺ Discards properties never or rarely met in practice
  - ☹ Non-complete

Patterns: Related Works

- Design patterns for software engineering [Gamma et al., 1995]
  - $=$ Non-original, non-compositional, non-complete
  - $\neq$ Can be inserted into freely written code
- Timed automata patterns [Dong et al., 2008]
  - $=$ Specification for real-time concurrent systems
  - $\neq$ Specify the model and not the property
  - $\neq$ Aim at completeness
- Other patterns for real-time systems
  - Dedication to scheduling problems [Khatib et al., 2001]
  - Specification using UML [Mekki et al., 2009]
  - Timed automata patterns generated from RTESMs [Han et al., 2013]

Observers: All is Reachability

- Observer: additional subsystem monitoring the system behavior
  - Synchronization with the system events
  - Definition of bad / good states
- Reduction to pure reachability properties
  Here, we consider 2 main kinds of properties:
  1. Never reach the bad state
  2. Always end in a good state
- Advantages
  - Any tool implementing (non-)reachability can implement such patterns

Our Approach

- A limited set of non-compositional patterns
  - Commonly used in practice
  - ⇝ Discards too complicated properties
- Expressed using an intuitive English-like syntax
  - Partially inspired by CASL-LTL [Reggio et al., 2003, Choppy and Reggio, 2006]
  - Described using a full English sentence
  - ⇝ Easy to understand by non-experts
- Associated with a formal semantics
  - Expressed using observers
  - ⇝ Does not require complex algorithms (only reachability)
    - Implemented in both timed automata [Alur and Dill, 1994] and Stateful Timed CSP [Sun et al., 2013]

Outline

Action Precedence: Acyclic Version (1/2)

Syntax:

if a_2 *then* a_1 *has happened before*

English description:

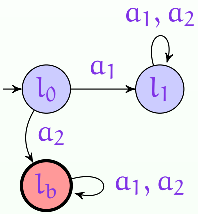
"If a_2 happens at least once, then a_1 has happened before the first occurrence of a_2."

- Typical use: avoid false positives for alarms
  - "An alarm must ring only if an intrusion has happened before"
- Does not mean that the intrusion will always lead to an alarm!

Action Precedence: Acyclic Version (2/2)

if a_2 *then* a_1 *has happened before*

Timed automaton observer:



Stateful Timed CSP:

$P_{obs} \doteq (a_2\{v_{bad} := \top\} \rightarrow P_S) \,\square\, (a_1 \rightarrow P_S)$

Action Precedence: Cyclic Version

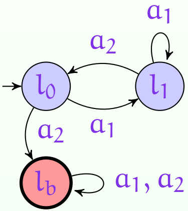
Syntax:

every time a_2 *then* a_1 *has happened before*

English description:

"Every time a_2 happens, then a_1 has happened before, since the last occurrence of a_2 (if any)."

Timed automaton:



Stateful Timed CSP:

$P_{obs} \doteq P_1$

$P_1 \doteq (a_2\{v_{bad} := \top\} \rightarrow P_S) \,\square\, (a_1 \rightarrow P_2)$

$P_2 \doteq (a_1 \rightarrow P_2) \,\square\, (a_2 \rightarrow P_1)$

Action Precedence: Strict Cyclic Version

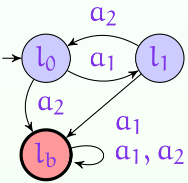
Syntax:

every time a_2 *then* a_1 *has happened exactly once before*

English description:

"Every time a_2 happens, then a_1 has happened before, exactly once since the last occurrence of a_2 (if any)."

Timed automaton:



Stateful Timed CSP:

$$
\begin{aligned}
P_{obs} &\doteq P_1 \\
P_1 &\doteq (a_2\{v_{bad} := \top\} \rightarrow P_S) \,\Box\, (a_1 \rightarrow P_2) \\
P_2 &\doteq (a_1\{v_{bad} := \top\} \rightarrow P_S) \,\Box\, (a_2 \rightarrow P_1)
\end{aligned}
$$

Eventual Response: Acyclic Version (1/2)

Syntax:

if a_1 *then eventually* a_2

English description:

"If a_1 happens, then a_2 eventually happens."

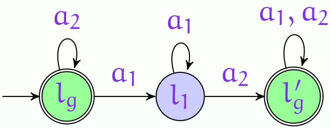
- Typical use:
  - "Every time access is requested, then access is eventually granted"
  - "Every time an intrusion occurs, then the alarm eventually rings"
- Does not require a_1 to happen!
- Typical liveness property[1]

[1] Cf. https://cs.nyu.edu/acsys/beyond-safety/liveness.htm

Eventual Response: Acyclic Version (2/2)

if a_1 *then eventually* a_2

Timed automaton:



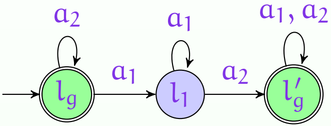
Stateful Timed CSP:

$$P_{obs} \doteq (a_2 \rightarrow P_{obs}) \,\Box\, (a_1\{v_{good} := F\} \rightarrow P_1)$$
$$P_1 \doteq (a_1 \rightarrow P_1) \,\Box\, (a_2\{v_{good} := T\} \rightarrow P_S)$$

Eventual Response: Acyclic Version (2/2)

if a_1 *then eventually* a_2

Timed automaton:



Stateful Timed CSP:

$P_{obs} \doteq (a_2 \rightarrow P_{obs}) \,\square\, (a_1\{v_{good} := F\} \rightarrow P_1)$

$P_1 \quad \doteq (a_1 \rightarrow P_1) \,\square\, (a_2\{v_{good} := T\} \rightarrow P_S)$

Also exists in “cyclic” and “strict cyclic” versions (cf. paper)

Action Before Deadline (1/2)

Syntax:

a no later than d

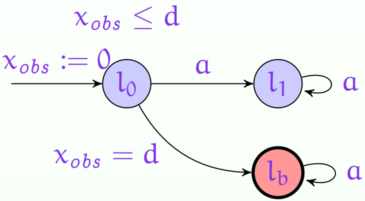
English description:

"a will happen no later than d units of time after the system start."

- Typical use: signal changes in hardware verification
- Similar to the "deadline" pattern in [Dong et al., 2008]

Action Before Deadline (2/2)

a no later than d



Stateful Timed CSP:

$P_{obs} \doteq (a \rightarrow P_S)$ $\texttt{timeout}[d]$

$\big((e_{obs}\{v_{bad} := \top\} \rightarrow P_S) \setminus \{e_{obs}\}\big)$

Time-Bounded Action Precedence

Syntax:
if a_2 *then* a_1 *has happened at most* d *units of time before*

English description:
"If a_2 happens at least once, then a_1 has happened at most d units of time before the first occurrence of a_2."

- Timed extension of the "Action precedence" pattern
- Also exists in "cyclic" and "strict cyclic" versions

Time-Bounded Response

Syntax:

if a_1 *then eventually* a_2 *within* d

English description:

"If a_1 happens, then a_2 will eventually happen within d units of time."

- Typical use:
  - "If an intrusion occurs, then the alarm eventually rings within 2 seconds"
- Also known as time-bounded liveness [Behrmann et al., 2005]
- Timed extension of the "Eventual response" pattern
- Also exists in "cyclic" and "strict cyclic" versions

Sequence

Syntax:

sequence $a_1, \ldots, a_n$

English description:

"Actions $a_1, \ldots, a_n$ must occur in this order."

- Imposes an order, but does require the actions to happen
- Also exists in "cyclic" version

Use of the Patterns in a Set of Case Studies

Pattern	Nb	Applications		
Non-reachability	9		Prtc	
Always end in good state	2	HW	Prtc	
Action precedence	2	HW		
Eventual response	5		Prtc	
Action before deadline	1	HW		
T-bounded action precedence	1			
T-bounded response	3	HW	Prtc	Sched
Sequence	1	HW		
Combinations	3		Prtc	Sched
Others	2		Prtc	

Based on 29 case studies (real-time systems)

Outline

Conclusion

- Observer patterns
  - Common properties
  - Intuitive syntax
  - Reachability-based formal semantics
  - Limited expressiveness
- Extends to parameter synthesis
  - Parameterized version of the patterns
    - E.g.: "if a_1 then eventually a_2 within p", with p a parameter
  - Implemented in IMITATOR [A., Fribourg, Kühne, Soulat, 2012]
- Extends to state- and variables-based properties?
  - So far: action-based
  - No theoretical problem for state and variables
    But what about synchronization?

Perspectives

- What about liveness?
  - Requires Büchi conditions in the observers
- Add a limited dose of compositionality
  - Combination of patterns
    - Example: “[sequence a, b, c] within 10 seconds” (Sequence + new “within” pattern)
  - Propose a bounded depth for well-formedness?
- Semantic equivalence between the implementation of the patterns in timed automata and Stateful Timed CSP respectively
- Reuse some of the most common existing “patterns” [Khatib et al., 2001, Mekki et al., 2009]
- Improve the English description
  - Take compositionality into account

Bibliography

References I

Aceto, L., Bouyer, P., Burgueño, A., and Larsen, K. G. (1998a).
The power of reachability testing for timed automata.
In Arvind, V. and Ramanujam, R., editors, *FSTTCS'98*, volume 1530 of *Lecture Notes in Computer Science*, pages 245–256. Springer.

Aceto, L., Burgueño, A., and Larsen, K. G. (1998b).
Model checking via reachability testing for timed automata.
In *TACAS'98*, volume 1384 of *Lecture Notes in Computer Science*, pages 263–280. Springer.

Alur, R. and Dill, D. L. (1994).
A theory of timed automata.
Theoretical Computer Science, 126(2):183–235.

André, É., Fribourg, L., Kühne, U., and Soulat, R. (2012).
IMITATOR 2.5: A tool for analyzing robustness in scheduling problems.
In *FM'12*, volume 7436 of *Lecture Notes in Computer Science*, pages 33–36. Springer.

Baier, C. and Katoen, J.-P. (2008).
Principles of model checking.
MIT Press.

References II

Behrmann, G., Larsen, K. G., and Rasmussen, J. I. (2005).
Beyond liveness: Efficient parameter synthesis for time bounded liveness.
In *FORMATS'05*, volume 3829 of *Lecture Notes in Computer Science*, pages 81–94. Springer.

Choppy, C. and Reggio, G. (2006).
A formally grounded software specification method.
Journal of Logic and Algebraic Programming, 67(1-2):52–86.

Dong, J. S., Hao, P., Qin, S., Sun, J., and Yi, W. (2008).
Timed automata patterns.
IEEE Transactions on Software Engineering, 34(6):844–859.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995).
Design patterns: Elements of reusable object-oriented software.
Addison-Wesley Longman Publishing Co., Inc.

Han, F., Herrmann, P., and Le, H. (2013).
Modeling and verifying real-time properties of reactive systems.
In *ICECCS'13*, pages 14–23. IEEE Computer Society.

References III

Khatib, L., Muscettola, N., and Havelund, K. (2001).
Mapping temporal planning constraints into timed automata.
In *TIME'01*, pages 21–27. IEEE Computer Society.

Mekki, A., Ghazel, M., and Toguyeni, A. (2009).
Validating time-constrained systems using UML statecharts patterns and timed automata observers.
In *VECoS'09*, pages 112–124. British Computer Society.

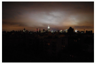
Pnueli, A. (1977).
The temporal logic of programs.
In *FOCS'77*, pages 46–57. IEEE Computer Society.

Reggio, G., Astesiano, E., and Choppy, C. (2003).
CASL-LTL : A CASL extension for dynamic reactive systems version 1.0– summary.
Technical Report of DISI.

Sun, J., Liu, Y., Dong, J. S., Liu, Y., Shi, L., and André, É. (2013).
Modeling and verifying hierarchical real-time systems using Stateful Timed CSP.
ACM Transactions on Software Engineering and Methodology, 22(1):3.1–3.29.

Additional explanation

Explanation for the 4 pictures in the beginning



Allusion to the Northeast blackout (USA, 2003)
Computer bug
Consequences: 11 fatalities, huge cost
(Picture actually from the Sandy Hurricane, 2012)

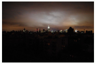
Allusion to any plane crash
(Picture actually from the happy-ending US Airways Flight 1549, 2009)

Allusion to the sinking of the Sleipner A offshore platform (Norway, 1991)
No fatalities
Computer bug: inaccurate finite element analysis modeling
(Picture actually from the Deepwater Horizon Offshore Drilling Platform)

Allusion to the MIM-104 Patriot Missile Failure (Iraq, 1991)
28 fatalities, hundreds of injured
Computer bug: software error (clock drift)
(Picture of an actual MIM-104 Patriot Missile, though not the one of 1991)

Licensing

Source of the graphics used (1/2)



Title: Hurricane Sandy Blackout New York Skyline
Author: David Shankbone
Source: https://commons.wikimedia.org/wiki/File:Hurricane_Sandy_Blackout_New_York_Skyline.JPG
License: CC BY 3.0

Title: Miracle on the Hudson
Author: Janis Krums (cropped by Étienne André)
Source: https://secure.flickr.com/photos/davidwatts1978/3199405401/
License: CC BY 2.0

Title: Deepwater Horizon Offshore Drilling Platform on Fire
Author: ideum
Source: https://secure.flickr.com/photos/ideum/4711481781/
License: CC BY-SA 2.0

Title: DA-SC-88-01663
Author: imcomkorea
Source: https://secure.flickr.com/photos/imcomkorea/3017886760/
License: CC BY-NC-ND 2.0

Source of the graphics used (2/2)

Title: Smiley green alien big eyes (aaah)
Author: LadyofHats
Source: https://commons.wikimedia.org/wiki/File:Smiley_green_alien_big_eyes.svg
License: public domain

Title: Smiley green alien big eyes (cry)
Author: LadyofHats
Source: https://commons.wikimedia.org/wiki/File:Smiley_green_alien_big_eyes.svg
License: public domain

License of this document

This presentation can be published, reused and modified under the terms of the license Creative Commons **Attribution-ShareAlike 3.0 Unported (CC BY-SA 3.0)**

(LaTeX source available on demand)

Author: Étienne André

https://creativecommons.org/licenses/by-sa/3.0/