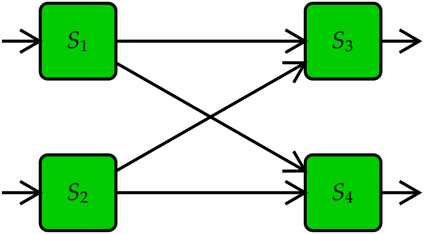
Technical Talk on Runtime Verification

Martin Leucker

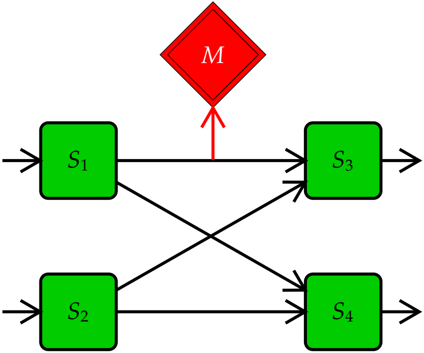
Institute for Software Engineering
Universität zu Lübeck

Marseille, Monday 3rd of December 2012

Runtime Verification (RV)

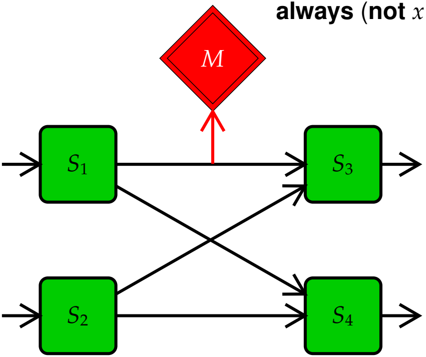


Runtime Verification (RV)

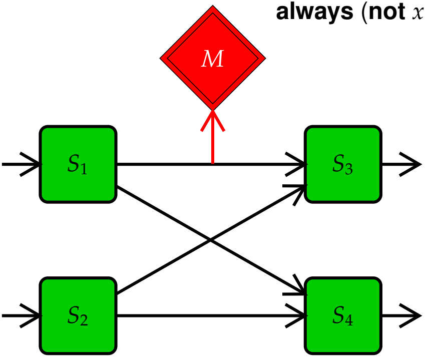


Runtime Verification (RV)

always (**not** $x > 0$ **implies next** $x > 0$)



Runtime Verification (RV)



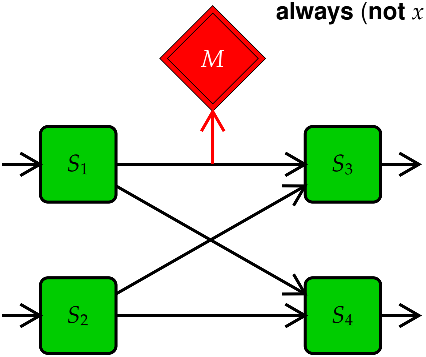
always (**not** $x > 0$ **implies next** $x > 0$)

Characterisation

- Verifies (partially) correctness properties based on actual executions

Runtime Verification (RV)

always (**not** $x > 0$ **implies next** $x > 0$)

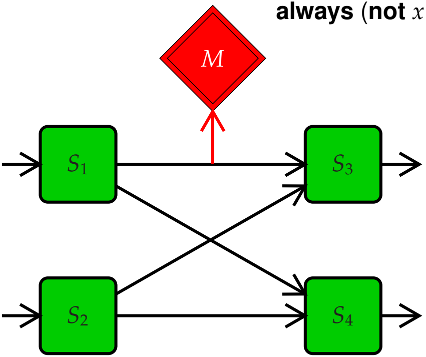


Characterisation

- Verifies (partially) correctness properties based on actual executions
- Simple verification technique

Runtime Verification (RV)

always (**not** $x > 0$ **implies next** $x > 0$)

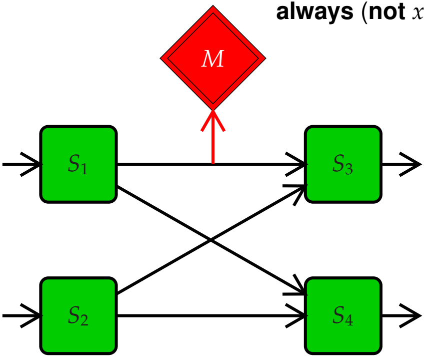


Characterisation

- Verifies (partially) correctness properties based on actual executions
- Simple verification technique
- Complementing

Runtime Verification (RV)

always (**not** $x > 0$ **implies next** $x > 0$)

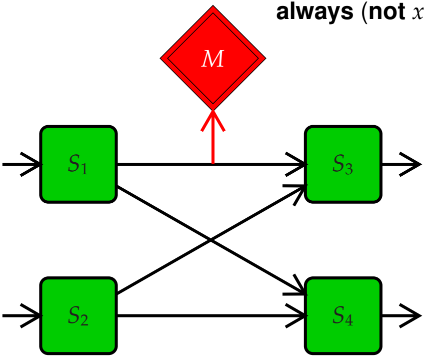


Characterisation

- Verifies (partially) correctness properties based on actual executions
- Simple verification technique
- Complementing
  - Model Checking

Runtime Verification (RV)

always (**not** $x > 0$ **implies next** $x > 0$)

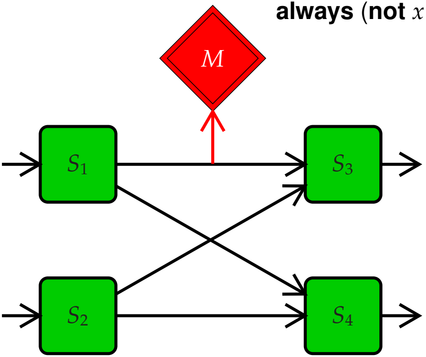


Characterisation

- Verifies (partially) correctness properties based on actual executions
- Simple verification technique
- Complementing
  - Model Checking
  - Testing

Runtime Verification (RV)

always (**not** $x > 0$ **implies next** $x > 0$)



Characterisation

- Verifies (partially) correctness properties based on actual executions
- Simple verification technique
- Complementing
  - Model Checking
  - Testing
- Formal: $w \in \mathcal{L}(\varphi)$

Model Checking

- Specification of System

Model Checking

- Specification of System
  - as formula φ of linear-time temporal logic (LTL)

Model Checking

- Specification of System
  - as formula φ of linear-time temporal logic (LTL)
  - with models $\mathcal{L}(\varphi)$

Model Checking

- Specification of System
  - as formula φ of linear-time temporal logic (LTL)
  - with models $\mathcal{L}(\varphi)$
- Model of System

Model Checking

- Specification of System
  - as formula φ of linear-time temporal logic (LTL)
  - with models $\mathcal{L}(\varphi)$
- Model of System
  - as transition system S with runs $\mathcal{L}(S)$

Model Checking

- Specification of System
  - as formula φ of linear-time temporal logic (LTL)
  - with models $\mathcal{L}(\varphi)$
- Model of System
  - as transition system S with runs $\mathcal{L}(S)$
- Model Checking Problem:
  Do all runs of the system satisfy the specification

Model Checking

- Specification of System
  - as formula φ of linear-time temporal logic (LTL)
  - with models $\mathcal{L}(\varphi)$
- Model of System
  - as transition system S with runs $\mathcal{L}(S)$
- Model Checking Problem:
  Do all runs of the system satisfy the specification
  - $\mathcal{L}(S) \subseteq \mathcal{L}(\varphi)$

Model Checking versus RV

- Model Checking: infinite words

Model Checking versus RV

- Model Checking: infinite words
- Runtime Verification: finite words

Model Checking versus RV

- Model Checking: infinite words
- Runtime Verification: finite words
  - yet continuously expanding words

Model Checking versus RV

- Model Checking: infinite words
- Runtime Verification: finite words
  - yet continuously expanding words
- In RV: Complexity of monitor generation is of less importance than complexity of the monitor

Model Checking versus RV

- Model Checking: infinite words
- Runtime Verification: finite words
  - yet continuously expanding words
- In RV: Complexity of monitor generation is of less importance than complexity of the monitor
- Model Checking: White-Box-Systems

Model Checking versus RV

- Model Checking: infinite words
- Runtime Verification: finite words
  - yet continuously expanding words
- In RV: Complexity of monitor generation is of less importance than complexity of the monitor
- Model Checking: White-Box-Systems
- Runtime Verification: also Black-Box-Systems

Testing

Testing: Input/Output Sequence

- incomplete verification technique

Testing

Testing: Input/Output Sequence

- incomplete verification technique
- test case: finite sequence of input/output actions

Testing

Testing: Input/Output Sequence

- incomplete verification technique
- test case: finite sequence of input/output actions
- test suite: finite set of test cases

Testing

Testing: Input/Output Sequence

- incomplete verification technique
- test case: finite sequence of input/output actions
- test suite: finite set of test cases
- test execution: send inputs to the system and check whether the actual output is as expected

Testing

Testing: Input/Output Sequence

- incomplete verification technique
- test case: finite sequence of input/output actions
- test suite: finite set of test cases
- test execution: send inputs to the system and check whether the actual output is as expected

Testing

Testing: Input/Output Sequence

- incomplete verification technique
- test case: finite sequence of input/output actions
- test suite: finite set of test cases
- test execution: send inputs to the system and check whether the actual output is as expected

Testing: with Oracle

- test case: finite sequence of input actions

Testing

Testing: Input/Output Sequence

- incomplete verification technique
- test case: finite sequence of input/output actions
- test suite: finite set of test cases
- test execution: send inputs to the system and check whether the actual output is as expected

Testing: with Oracle

- test case: finite sequence of input actions
- test oracle: monitor

Testing

Testing: Input/Output Sequence

- incomplete verification technique
- test case: finite sequence of input/output actions
- test suite: finite set of test cases
- test execution: send inputs to the system and check whether the actual output is as expected

Testing: with Oracle

- test case: finite sequence of input actions
- test oracle: monitor
- test execution: send test cases, let oracle report violations

Testing

Testing: Input/Output Sequence

- incomplete verification technique
- test case: finite sequence of input/output actions
- test suite: finite set of test cases
- test execution: send inputs to the system and check whether the actual output is as expected

Testing: with Oracle

- test case: finite sequence of input actions
- test oracle: monitor
- test execution: send test cases, let oracle report violations
- similar to runtime verification

Testing versus RV

- Test oracle manual

Testing versus RV

- Test oracle manual
- RV monitor from high-level specification (LTL)

Testing versus RV

- Test oracle manual
- RV monitor from high-level specification (LTL)
- Testing:

  How to find good test suites?

Testing versus RV

- Test oracle manual
- RV monitor from high-level specification (LTL)
- Testing:

  How to find good test suites?
- Runtime Verification:

  How to generate good monitors?

Outline

- Runtime Verification
- Runtime Verification for LTL
  - LTL over Finite, Completed Words
  - LTL over Finite, Non-Completed Words: Impartiality
  - LTL over Non-Completed Words: Anticipation
  - LTL over Infinite Words: With Anticipation
  - Generalisations: LTL with modulo Constraints
  - Monitorable Properties
  - LTL with a Predictive Semantics
  - LTL wrap-up
- Extensions
- Monitoring Systems/Logging
- Steering
- Diagnosis
  - Ideas
  - RV and Diagnosis
- Conclusion

Presentation outline

Runtime Verification

Runtime Verification for LTL

- LTL over Finite, Completed Words
- LTL over Finite, Non-Completed Words: Impartiality
- LTL over Non-Completed Words: Anticipation
- LTL over Infinite Words: With Anticipation
- Generalisations: LTL with modulo Constraints
- Monitorable Properties
- LTL with a Predictive Semantics
- LTL wrap-up

Extensions

Monitoring Systems/Logging

Steering

Diagnosis

- Ideas
- RV and Diagnosis

Conclusion

Runtime Verification

Definition (Runtime Verification)

Runtime verification is the discipline of computer science that deals with the study, development, and application of those verification techniques that allow checking whether a *run* of a system under scrutiny (SUS) satisfies or violates a given correctness property.

Its distinguishing research effort lies in *synthesizing monitors from high level specifications.*

Runtime Verification

Definition (Runtime Verification)

Runtime verification is the discipline of computer science that deals with the study, development, and application of those verification techniques that allow checking whether a *run* of a system under scrutiny (SUS) satisfies or violates a given correctness property.

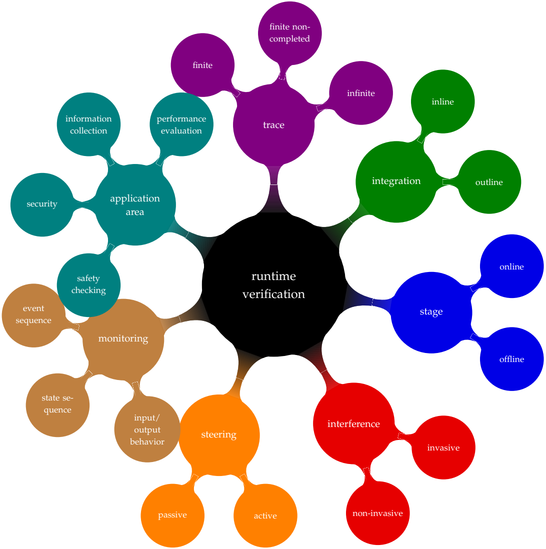
Its distinguishing research effort lies in *synthesizing monitors from high level specifications.*

Definition (Monitor)

A monitor is a device that reads a finite trace and yields a certain verdict.

A verdict is typically a truth value from some truth domain.

Taxonomy


runtime verification
trace
finite
finite non-completed
infinite
integration
inline
outline
stage
online
offline
interference
invasive
non-invasive
steering
passive
active
monitoring
event sequence
state sequence
input/ output behavior
application area
information collection
performance evaluation
security
safety checking

Presentation outline

Runtime Verification

Runtime Verification for LTL

- LTL over Finite, Completed Words
- LTL over Finite, Non-Completed Words: Impartiality
- LTL over Non-Completed Words: Anticipation
- LTL over Infinite Words: With Anticipation
- Generalisations: LTL with modulo Constraints
- Monitorable Properties
- LTL with a Predictive Semantics
- LTL wrap-up

Extensions

Monitoring Systems/Logging

Steering

Diagnosis

- Ideas
- RV and Diagnosis

Conclusion

Runtime Verification for LTL

Observing executions/runs

Runtime Verification for LTL

Observing executions/runs

Idea

Specify correctness properties in LTL

Runtime Verification for LTL

Observing executions/runs

Idea

Specify correctness properties in LTL

Commercial

Specify correctness properties in Regular LTL

Runtime Verification for LTL

Definition (Syntax of LTL formulae)

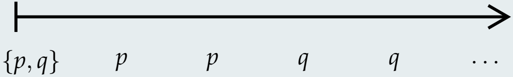
Let p be an atomic proposition from a finite set of atomic propositions AP. The set of LTL formulae, denoted with LTL, is inductively defined by the following grammar:

$$\begin{array}{rcl} \varphi & ::= & \mathit{true} \mid p \mid \varphi \vee \varphi \mid \varphi\, U\, \varphi \mid X\varphi \mid \\ & & \mathit{false} \mid \neg p \mid \varphi \wedge \varphi \mid \varphi\, R\, \varphi \mid \bar{X}\varphi \mid \\ & & \neg\varphi \end{array}$$

Linear-time Temporal Logic (LTL)

Semantics

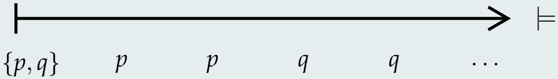
over $w \in (2^{AP})^\omega = \Sigma^\omega$



Linear-time Temporal Logic (LTL)

Semantics

over $w \in (2^{AP})^\omega = \Sigma^\omega$



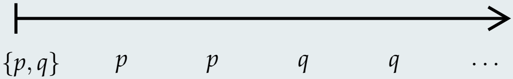
$\models$

$\{p, q\}$ p p q q $\dots$

Linear-time Temporal Logic (LTL)

Semantics

over $w \in (2^{AP})^\omega = \Sigma^\omega$



$\{p,q\}$ p p q q $\cdots$ $\models$

p

$\neg p$

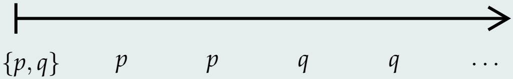
pUq

$X(pUq)$

Linear-time Temporal Logic (LTL)

Semantics

over $w \in (2^{AP})^\omega = \Sigma^\omega$



$\models$

p ✓

$\neg p$

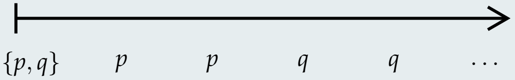
pUq

$X(pUq)$

Linear-time Temporal Logic (LTL)

Semantics

over $w \in (2^{AP})^\omega = \Sigma^\omega$



$\models$

p	✓
$\neg p$	X
pUq	
$X(pUq)$	

Linear-time Temporal Logic (LTL)

Semantics

over $w \in (2^{AP})^\omega = \Sigma^\omega$

p ✓

$\neg p$ X

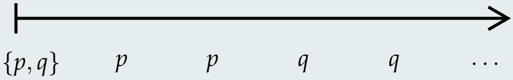
$\{p, q\}$ p p q q $\dots$ $\models$ pUq ✓

$X(pUq)$

Linear-time Temporal Logic (LTL)

Semantics

over $w \in (2^{AP})^\omega = \Sigma^\omega$



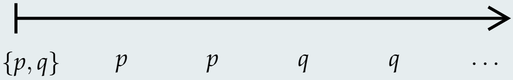
$\models$

p	✓
$\neg p$	X
pUq	✓
$X(pUq)$	✓

Linear-time Temporal Logic (LTL)

Semantics

over $w \in (2^{AP})^\omega = \Sigma^\omega$



$\{p, q\}$ $\quad p \quad p \quad q \quad q \quad \cdots$

$\models$

p	✓
$\neg p$	✗
pUq	✓
$X(pUq)$	✓

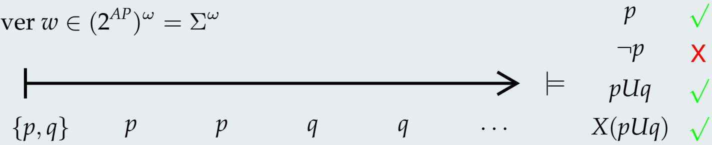
Abbreviation

$F\varphi \equiv trueU\varphi \qquad G\varphi \equiv \neg F \neg \varphi$

Linear-time Temporal Logic (LTL)

Semantics

over $w \in (2^{AP})^\omega = \Sigma^\omega$



$\{p,q\}$ p p q q $\dots$ $\models$

p ✓

$\neg p$ ✗

pUq ✓

$X(pUq)$ ✓

Abbreviation

$F\varphi \equiv trueU\varphi \qquad G\varphi \equiv \neg F \neg \varphi$

Example

$G\neg(critic_1 \wedge critic_2), G(\neg alive \rightarrow Xalive)$

LTL on infinite words

Definition (LTL semantics (traditional))

Semantics of LTL formulae over an infinite word $w = a_0 a_1 \ldots \in \Sigma^\omega$, where $w^i = a_i a_{i+1} \ldots$

$$
\begin{array}{lll}
w \models true & & \\
w \models p & \text{if} & p \in a_0 \\
w \models \neg p & \text{if} & p \notin a_0 \\
w \models \neg\varphi & \text{if} & \text{not } w \models \varphi \\
w \models \varphi \vee \psi & \text{if} & w \models \varphi \text{ or } w \models \psi \\
w \models \varphi \wedge \psi & \text{if} & w \models \varphi \text{ and } w \models \psi \\
w \models X\varphi & \text{if} & w^1 \models \varphi \\
w \models \bar{X}\varphi & \text{if} & w^1 \models \varphi \\
w \models \varphi \; U \; \psi & \text{if} & \text{there is } k \text{ with } 0 \leq k < |w| \text{: } w^k \models \psi \\
 & & \text{and for all } l \text{ with } 0 \leq l < k \; w^l \models \varphi \\
w \models \varphi \; R \; \psi & \text{if} & \text{for all } k \text{ with } 0 \leq k < |w| \text{: (} w^k \models \psi \\
 & & \text{or there is } l \text{ with } 0 \leq l < k \; w^l \models \varphi)
\end{array}
$$

LTL for the working engineer??

Simple??

“LTL is for theoreticians—but for practitioners?”

LTL for the working engineer??

Simple??

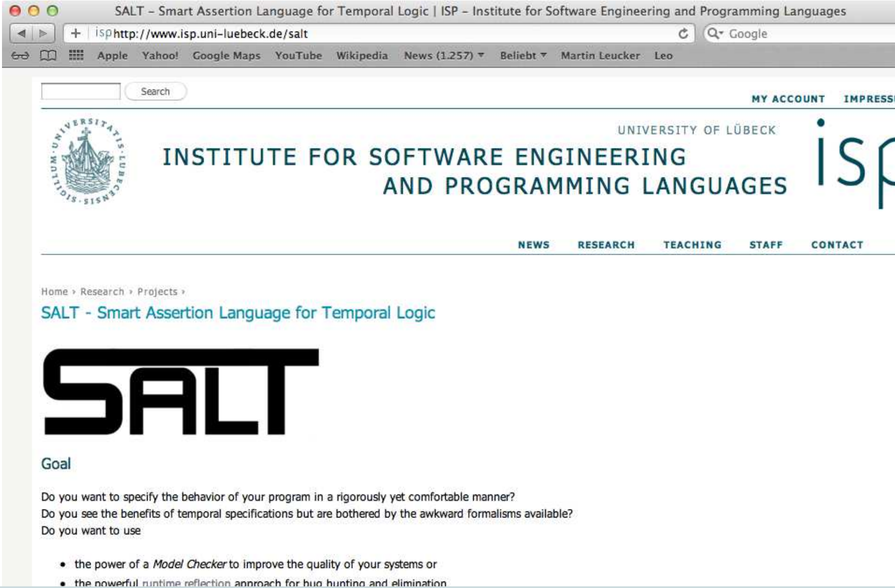
“LTL is for theoreticians—but for practitioners?”

SALT

Structured Assertion Language for Temporal Logic

“Syntactic Sugar for LTL” [Bauer, L., Streit@ICFEM’06]

SALT – `http://www.isp.uni-luebeck.de/salt`


SALT – Smart Assertion Language for Temporal Logic | ISP – Institute for Software Engineering and Programming Languages
isp http://www.isp.uni-luebeck.de/salt
Google
Apple
Yahoo!
Google Maps
YouTube
Wikipedia
News (1.257)
Beliebt
Martin Leucker
Leo
Search
MY ACCOUNT
IMPRESS
UNIVERSITY OF LÜBECK
INSTITUTE FOR SOFTWARE ENGINEERING
AND PROGRAMMING LANGUAGES
isp
NEWS
RESEARCH
TEACHING
STAFF
CONTACT
Home › Research › Projects ›
SALT

SALT - Smart Assertion Language for Temporal Logic

Goal

Do you want to specify the behavior of your program in a rigorously yet comfortable manner?
Do you see the benefits of temporal specifications but are bothered by the awkward formalisms available?
Do you want to use

- the power of a *Model Checker* to improve the quality of your systems or
- the powerful runtime reflection approach for bug hunting and elimination

Runtime Verification for LTL

Idea

Specify correctness properties in LTL

Definition (Syntax of LTL formulae)

Let p be an atomic proposition from a finite set of atomic propositions AP. The set of LTL formulae, denoted with LTL, is inductively defined by the following grammar:

$$
\begin{array}{rcl}
\varphi & ::= & \mathit{true} \mid p \mid \varphi \vee \varphi \mid \varphi \, U \, \varphi \mid X\varphi \mid \\
 & & \mathit{false} \mid \neg p \mid \varphi \wedge \varphi \mid \varphi \, R \, \varphi \mid \bar{X}\varphi \mid \\
 & & \neg\varphi
\end{array}
$$

Truth Domains

Lattice

- A lattice is a partially ordered set $(\mathcal{L}, \sqsubseteq)$ where for each $x, y \in \mathcal{L}$, there exists
  1. a unique greatest lower bound (glb), which is called the meet of x and y, and is denoted with $x \sqcap y$, and
  2. a unique least upper bound (lub), which is called the join of x and y, and is denoted with $x \sqcup y$.
- A lattice is called finite iff $\mathcal{L}$ is finite.
- Every finite lattice has a well-defined unique least element, called bottom, denoted with $\bot$,
- and analogously a greatest element, called top, denoted with $\top$.

Truth Domains (cont.)

Lattice (cont.)

- A lattice is distributive, iff $x \sqcap (y \sqcup z) = (x \sqcap y) \sqcup (x \sqcap z)$, and, dually, $x \sqcup (y \sqcap z) = (x \sqcup y) \sqcap (x \sqcup z)$.
- In a de Morgan lattice, every element x has a unique dual element $\overline{x}$, such that $\overline{\overline{x}} = x$ and $x \sqsubseteq y$ implies $\overline{y} \sqsubseteq \overline{x}$.

Definition (Truth domain)

We call $\mathcal{L}$ a truth domain, if it is a finite distributive de Morgan lattice.

LTL's semantics using truth domains

Definition (LTL semantics (common part))

Semantics of LTL formulae over a finite or infinite word $w = a_0 a_1 \ldots \in \Sigma^\infty$

Boolean constants

$$\begin{aligned} [w \models \mathit{true}]_{\mathfrak{L}} &= \top \\ [w \models \mathit{false}]_{\mathfrak{L}} &= \bot \end{aligned}$$

Boolean combinations

$$\begin{aligned} [w \models \neg\varphi]_{\mathfrak{L}} &= \overline{[w \models \varphi]_{\mathfrak{L}}} \\ [w \models \varphi \vee \psi]_{\mathfrak{L}} &= [w \models \varphi]_{\mathfrak{L}} \sqcup [w \models \psi]_{\mathfrak{L}} \\ [w \models \varphi \wedge \psi]_{\mathfrak{L}} &= [w \models \varphi]_{\mathfrak{L}} \sqcap [w \models \psi]_{\mathfrak{L}} \end{aligned}$$

atomic propositions

$$[w \models p]_{\mathfrak{L}} = \begin{cases} \top & \text{if } p \in a_0 \\ \bot & \text{if } p \notin a_0 \end{cases} \qquad [w \models \neg p]_{\mathfrak{L}} = \begin{cases} \top & \text{if } p \notin a_0 \\ \bot & \text{if } p \in a_0 \end{cases}$$

next X/weak next X TBD

until/release

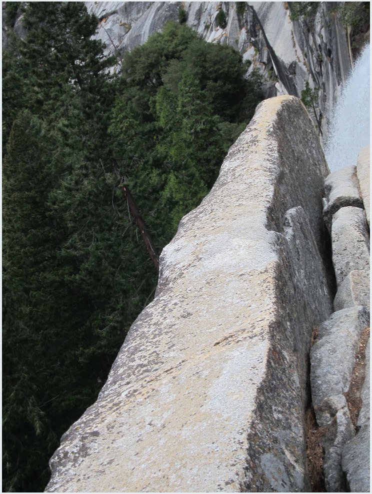
$$\begin{aligned} [w \models \varphi\ U\ \psi]_{\mathfrak{L}} &= \begin{cases} \top & \text{there is a } k,\ 0 \leq k < |w| : [w^k \models \psi]_{\mathfrak{L}} = \top \text{ and} \\ & \text{for all } l \text{ with } 0 \leq l < k : [w^l \models \varphi] = \top \\ \mathit{TBD} & \text{else} \end{cases} \\ \varphi\ R\ \psi &\equiv \neg(\neg\varphi\ U\ \neg\psi) \end{aligned}$$

Outline

- Runtime Verification
- Runtime Verification for LTL
  - LTL over Finite, Completed Words
  - LTL over Finite, Non-Completed Words: Impartiality
  - LTL over Non-Completed Words: Anticipation
  - LTL over Infinite Words: With Anticipation
  - Generalisations: LTL with modulo Constraints
  - Monitorable Properties
  - LTL with a Predictive Semantics
  - LTL wrap-up
- Extensions
- Monitoring Systems/Logging
- Steering
- Diagnosis
  - Ideas
  - RV and Diagnosis
- Conclusion

LTL on finite words

Application area: Specify properties of finite word



LTL on finite words

Definition (FLTL)

Semantics of FLTL formulae over a word $u = a_0 \dots a_{n-1} \in \Sigma^*$

next

$$[u \models X\varphi]_F \quad = \quad \begin{cases} [u^1 \models \varphi]_F & \text{if } u^1 \neq \epsilon \\ \bot & \text{otherwise} \end{cases}$$

weak next

$$[u \models \bar{X}\varphi]_F \quad = \quad \begin{cases} [u^1 \models \varphi]_F & \text{if } u^1 \neq \epsilon \\ \top & \text{otherwise} \end{cases}$$

Monitoring LTL on finite words

(Bad) Idea

just compute semantics. . .

Outline

- Runtime Verification
- Runtime Verification for LTL
  - LTL over Finite, Completed Words
  - LTL over Finite, Non-Completed Words: Impartiality
  - LTL over Non-Completed Words: Anticipation
  - LTL over Infinite Words: With Anticipation
  - Generalisations: LTL with modulo Constraints
  - Monitorable Properties
  - LTL with a Predictive Semantics
  - LTL wrap-up
- Extensions
- Monitoring Systems/Logging
- Steering
- Diagnosis
  - Ideas
  - RV and Diagnosis
- Conclusion

LTL on finite, but not completed words

Application area: Specify properties of finite but expanding word

LTL on finite, but not completed words

Be Impartial!

- go for a final verdict ($\top$ or $\bot$) only if you really know

LTL on finite, but not completed words

Be Impartial!

- go for a final verdict ($\top$ or $\bot$) only if you really know
- be a man: stick to your word

LTL on finite, but not complete words

Impartiality implies multiple values

Every two-valued logic is not impartial.

Definition (FLTL)

Semantics of FLTL formulae over a word $u = a_0 \dots a_{n-1} \in \Sigma^*$

next

$$[u \models X\varphi]_F = \begin{cases} [u^1 \models \varphi]_F & \text{if } u^1 \neq \epsilon \\ \perp^p & \text{otherwise} \end{cases}$$

weak next

$$[u \models \bar{X}\varphi]_F = \begin{cases} [u^1 \models \varphi]_F & \text{if } u^1 \neq \epsilon \\ \top^p & \text{otherwise} \end{cases}$$

Monitoring LTL on finite but expanding words

Left-to-right!

Monitoring LTL on finite but expanding words

Rewriting

Idea: Use rewriting of formula

Evaluating FLTL4 for each subsequent letter

- evaluate atomic propositions
- evaluate next-formulas
- that's it thanks to

$$\varphi\ U\ \psi \equiv \psi \vee (\varphi \wedge X\varphi\ U\ \psi)$$

  and

$$\varphi\ R\ \psi \equiv \psi \wedge (\varphi \vee \bar{X}\varphi\ R\ \psi)$$

- and remember what to evaluate for the next letter

Evaluating FLTL4 for each subsequent letter

Pseudo Code

```
evalFLTL4 $true$      a = ($\top$,$\top$)
evalFLTL4 $false$     a = ($\bot$,$\bot$)
evalFLTL4 p           a = ((p in a),(p in a))
evalFLTL4 $\neg\varphi$         a = let (valPhi,phiRew) = evalFLTL4 $\varphi$ a
                          in ($\overline{\texttt{valPhi}}$,¬phiRew)
evalFLTL4 $\varphi \vee \psi$   a = let
                            (valPhi,phiRew) = evalFLTL4 $\varphi$ a
                            (valPsi,psiRew) = evalFLTL4 $\psi$ a
                          in (valPhi $\sqcup$ valPsi,phiRew $\vee$ psiRew)
evalFLTL4 $\varphi \wedge \psi$   a = let
                            (valPhi,phiRew) = evalFLTL4 $\varphi$ a
                            (valPsi,psiRew) = evalFLTL4 $\psi$ a
                          in (valPhi $\sqcap$ valPsi,phiRew $\wedge$ psiRew)
evalFLTL4 $\varphi\, U\, \psi$   a = evalFLTL4 $\psi \vee (\varphi \wedge X(\varphi\, U\, \psi))$ a
evalFLTL4 $\varphi\, R\, \psi$   a = evalFLTL4 $\psi \wedge (\varphi \vee \bar{X}(\varphi\, R\, \psi))$ a
evalFLTL4 $X\varphi$        a = ($\bot^p$,$\varphi$)
evalFLTL4 $\bar{X}\varphi$        a = ($\top^p$,$\varphi$)
```

Monitoring LTL on finite but expanding words

Automata-theoretic approach

- Synthesize automaton
- Monitoring = stepping through automaton

Rewriting vs. automata

Rewriting function defines transition function

```
evalFLTL4 $true$       a = ($\top$,$\top$)
evalFLTL4 $false$      a = ($\bot$,$\bot$)
evalFLTL4 p            a = ((p in a),(p in a))
evalFLTL4 $\neg\varphi$         a = let (valPhi,phiRew) = evalFLTL4 $\varphi$ a
                           in ($\overline{\text{valPhi}}$,$\neg$phiRew)
evalFLTL4 $\varphi \vee \psi$    a = let
                             (valPhi,phiRew) = evalFLTL4 $\varphi$ a
                             (valPsi,psiRew) = evalFLTL4 $\psi$ a
                           in (valPhi $\sqcup$ valPsi,phiRew $\vee$ psiRew)
evalFLTL4 $\varphi \wedge \psi$    a = let
                             (valPhi,phiRew) = evalFLTL4 $\varphi$ a
                             (valPsi,psiRew) = evalFLTL4 $\psi$ a
                           in (valPhi $\sqcap$ valPsi,phiRew $\wedge$ psiRew)
evalFLTL4 $\varphi\ U\ \psi$    a = evalFLTL4 $\psi \vee (\varphi \wedge X(\varphi\ U\ \psi))$ a
evalFLTL4 $\varphi\ R\ \psi$    a = evalFLTL4 $\psi \wedge (\varphi \vee \bar{X}(\varphi\ R\ \psi))$ a
evalFLTL4 $X\varphi$        a = ($\bot^p$,$\varphi$)
evalFLTL4 $\bar{X}\varphi$        a = ($\top^p$,$\varphi$)
```

Automata-theoretic approach

The roadmap

- alternating Mealy machines

Automata-theoretic approach

The roadmap

- alternating Mealy machines
- Moore machines

Automata-theoretic approach

The roadmap

- alternating Mealy machines
- Moore machines
- alternating machines

Automata-theoretic approach

The roadmap

- alternating Mealy machines
- Moore machines
- alternating machines
- non-deterministic machines

Automata-theoretic approach

The roadmap

- alternating Mealy machines
- Moore machines
- alternating machines
- non-deterministic machines
- deterministic machines

Automata-theoretic approach

The roadmap

- alternating Mealy machines
- Moore machines
- alternating machines
- non-deterministic machines
- deterministic machines
- state sequence for an input word

Supporting alternating finite-state machines

Definition (Alternating Mealy Machine)

A alternating Mealy machine is a tupel $\mathcal{M} = (Q, \Sigma, \Gamma, q_0, \delta)$ where

- Q is a finite set of states,
- Σ is the input alphabet,
- Γ is a finite, distributive lattice, the output lattice,
- $q_0 \in Q$ is the initial state and
- $\delta : Q \times \Sigma \to B^+(\Gamma \times Q)$ is the transition function

Supporting alternating finite-state machines

Definition (Alternating Mealy Machine)

A alternating Mealy machine is a tupel $\mathcal{M} = (Q, \Sigma, \Gamma, q_0, \delta)$ where

- Q is a finite set of states,
- Σ is the input alphabet,
- Γ is a finite, distributive lattice, the output lattice,
- $q_0 \in Q$ is the initial state and
- $\delta : Q \times \Sigma \to B^+(\Gamma \times Q)$ is the transition function

Convention

Understand $\delta : Q \times \Sigma \to B^+(\Gamma \times Q)$ as a function $\delta : Q \times \Sigma \to \Gamma \times B^+(Q)$

Supporting alternating finite-state machines

Definition (Run of an Alternating Mealy Machine)

A run of an alternating Mealy machine $\mathcal{M} = (Q, \Sigma, \Gamma, q_0, \delta)$ on a finite word $u = a_0 \dots a_{n-1} \in \Sigma^+$ is a sequence $t_0 \overset{(a_0,b_0)}{\to} t_1 \overset{(a_1,b_1)}{\to} \dots t_{n-1} \overset{(a_{n-1},b_{n-1})}{\to} t_n$ such that

- $t_0 = q_0$ and
- $(t_i, b_{i-1}) = \hat{\delta}(t_{i-1}, a_{i-1})$

where $\hat{\delta}$ is inductively defined as follows

- $\hat{\delta}(q, a) = \delta(q, a)$,
- $\hat{\delta}(q \vee q', a) = (\hat{\delta}(q, a)|_1 \sqcup \hat{\delta}(q', a)|_1, \hat{\delta}(q, a)|_2 \vee \hat{\delta}(q', a)|_2)$, and
- $\hat{\delta}(q \wedge q', a) = (\hat{\delta}(q, a)|_1 \sqcap \hat{\delta}(q', a)|_1, \hat{\delta}(q, a)|_2 \wedge \hat{\delta}(q', a)|_2)$

The output of the run is b_{n-1}.

Transition function of an alternating Mealy machine

Transition function $\delta_4^a : Q \times \Sigma \to B^+(\Gamma \times Q)$

$$
\begin{array}{lcl}
\delta_4^a(\mathit{true}, a) & = & (\top, \mathit{true}) \\
\delta_4^a(\mathit{false}, a) & = & (\bot, \mathit{false}) \\
\delta_4^a(p, a) & = & (p \in a, [p \in a]) \\
\delta_4^a(\varphi \vee \psi, a) & = & \delta_4^a(\varphi, a) \vee \delta_4^a(\psi, a) \\
\delta_4^a(\varphi \wedge \psi, a) & = & \delta_4^a(\varphi, a) \wedge \delta_4^a(\psi, a) \\
\delta_4^a(\varphi\, U\, \psi, a) & = & \delta_4^a(\psi \vee (\varphi \wedge X(\varphi\, U\, \psi)), a) \\
 & = & \delta_4^a(\psi, a) \vee (\delta_4^a(\varphi, a) \wedge (\varphi\, U\, \psi)) \\
\delta_4^a(\varphi\, R\ \ \psi, a) & = & \delta_4^a(\psi \wedge (\varphi \vee \bar{X}(\varphi\, R\, \psi)), a) \\
 & = & \delta_4^a(\psi, a) \wedge (\delta_4^a(\varphi, a) \vee (\varphi\, R\, \psi)) \\
\delta_4^a(X\varphi, a) & = & (\bot^p, \varphi) \\
\delta_4^a(\bar{X}\varphi, a) & = & (\top^p, \varphi)
\end{array}
$$

Outline

- Runtime Verification
- Runtime Verification for LTL
    - LTL over Finite, Completed Words
    - LTL over Finite, Non-Completed Words: Impartiality
    - LTL over Non-Completed Words: Anticipation
    - LTL over Infinite Words: With Anticipation
    - Generalisations: LTL with modulo Constraints
    - Monitorable Properties
    - LTL with a Predictive Semantics
    - LTL wrap-up
- Extensions
- Monitoring Systems/Logging
- Steering
- Diagnosis
    - Ideas
    - RV and Diagnosis
- Conclusion

Anticipatory Semantics

Consider possible extensions of the non-completed word

Outline

- Runtime Verification
- Runtime Verification for LTL
  - LTL over Finite, Completed Words
  - LTL over Finite, Non-Completed Words: Impartiality
  - LTL over Non-Completed Words: Anticipation
  - LTL over Infinite Words: With Anticipation
  - Generalisations: LTL with modulo Constraints
  - Monitorable Properties
  - LTL with a Predictive Semantics
  - LTL wrap-up
- Extensions
- Monitoring Systems/Logging
- Steering
- Diagnosis
  - Ideas
  - RV and Diagnosis
- Conclusion

LTL for RV [BLS@FSTTCS'06]

Basic idea

- LTL over infinite words is commonly used for specifying correctness properties
- finite words in RV:
  prefixes of infinite, so-far unknown words
- re-use existing semantics

LTL for RV [BLS@FSTTCS'06]

Basic idea

- LTL over infinite words is commonly used for specifying correctness properties
- finite words in RV:
  prefixes of infinite, so-far unknown words
- re-use existing semantics

3-valued semantics for LTL over finite words

$$[u \models \varphi] = \begin{cases} \top & \text{if } \forall \sigma \in \Sigma^\omega : u\sigma \models \varphi \\ \bot & \text{if } \forall \sigma \in \Sigma^\omega : u\sigma \not\models \varphi \\ ? & \text{else} \end{cases}$$

Impartial Anticipation

Impartial

- Stay with $\top$ and $\bot$

Impartial Anticipation

Impartial

- Stay with $\top$ and $\bot$

Anticipatory

- Go for $\top$ or $\bot$
- Consider $XXX\mathit{false}$

$$\epsilon \quad \models \quad XXX\mathit{false}$$

Impartial Anticipation

Impartial

- Stay with $\top$ and $\bot$

Anticipatory

- Go for $\top$ or $\bot$
- Consider $XXXfalse$

$$\begin{array}{ccc} \epsilon & \models & XXXfalse \\ a & \models & XXfalse \end{array}$$

Impartial Anticipation

Impartial

- Stay with $\top$ and $\bot$

Anticipatory

- Go for $\top$ or $\bot$
- Consider $XXX\mathit{false}$

$$\begin{array}{lcl} \epsilon & \models & XXX\mathit{false} \\ a & \models & XX\mathit{false} \\ aa & \models & X\mathit{false} \end{array}$$

Impartial Anticipation

Impartial

- Stay with $\top$ and $\bot$

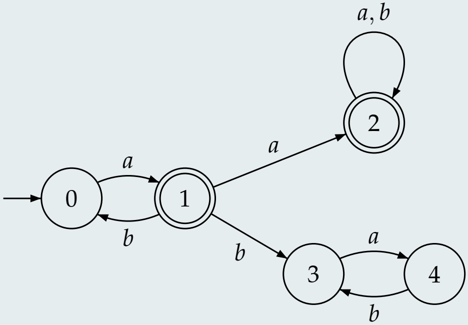
Anticipatory

- Go for $\top$ or $\bot$
- Consider $XXX\mathit{false}$

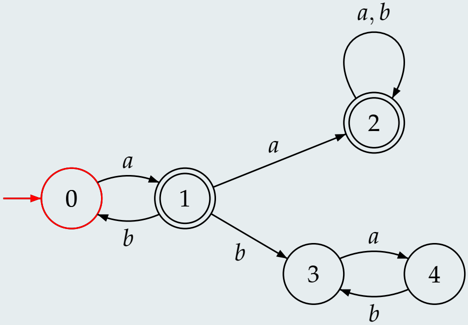
$$\begin{array}{lcl} \epsilon & \models & XXX\mathit{false} \\ a & \models & XX\mathit{false} \\ aa & \models & X\mathit{false} \\ aaa & \models & \mathit{false} \end{array}$$

$$[\epsilon \models XXX\mathit{false}] = \begin{cases} \top & \text{if } \forall \sigma \in \Sigma^\omega : \epsilon\sigma \models XXX\mathit{false} \\ \bot & \text{if } \forall \sigma \in \Sigma^\omega : \epsilon\sigma \not\models XXX\mathit{false} \\ ? & \text{else} \end{cases}$$

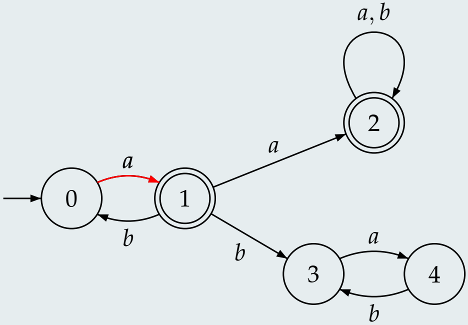
Büchi automata (BA)



Büchi automata (BA)

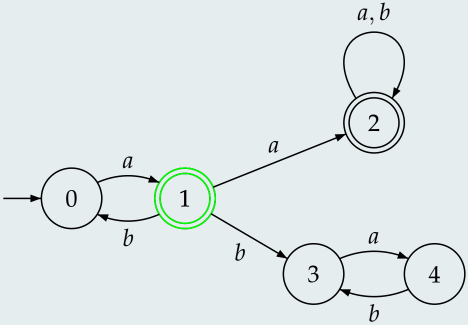


Büchi automata (BA)



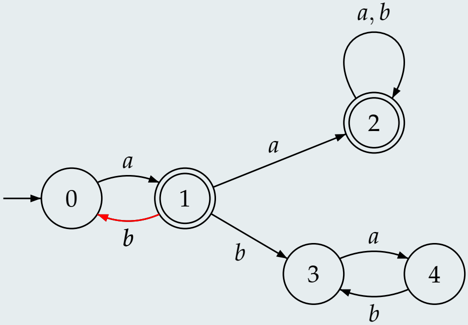
a

Büchi automata (BA)



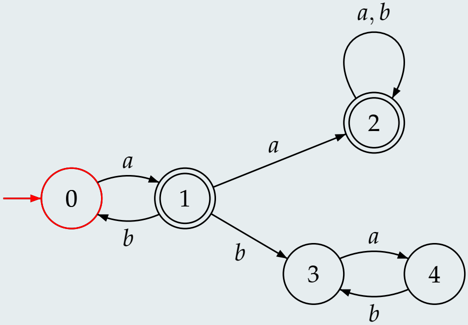
a

Büchi automata (BA)



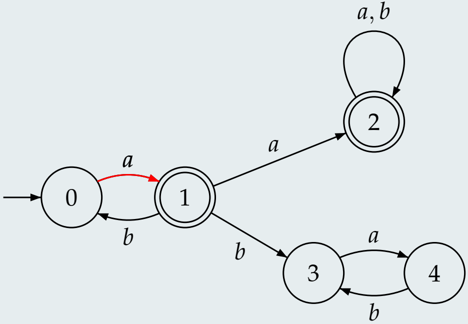
$a\ b$

Büchi automata (BA)



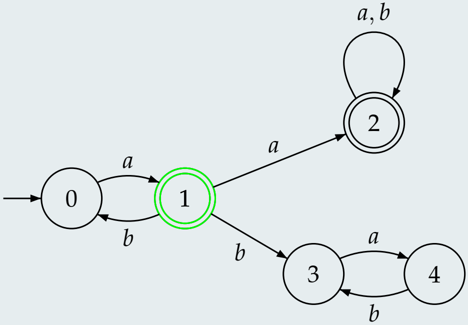
a b

Büchi automata (BA)



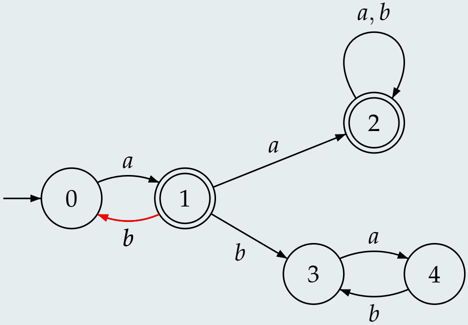
a b a

Büchi automata (BA)



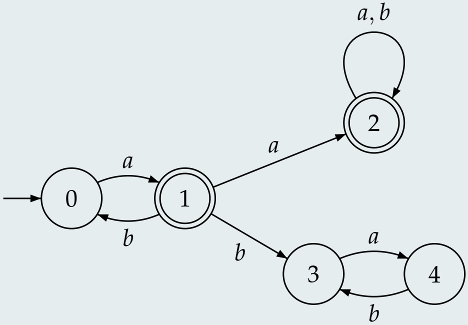
$a\ b\ a$

Büchi automata (BA)



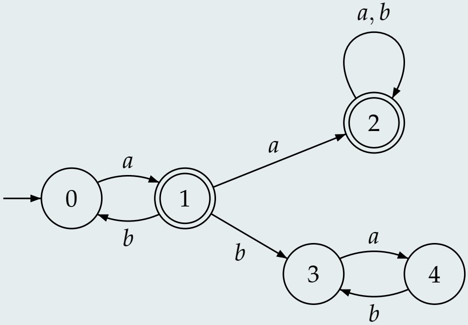
a b a b

Büchi automata (BA)



$a\ b\ a\ b \ldots$

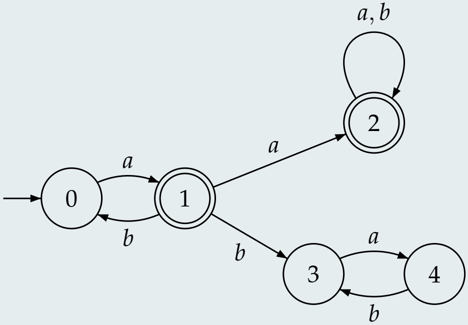
Büchi automata (BA)



$a\ b\ a\ b \dots$

$(ab)^\omega \in \mathcal{L}(\mathcal{A})$

Büchi automata (BA)



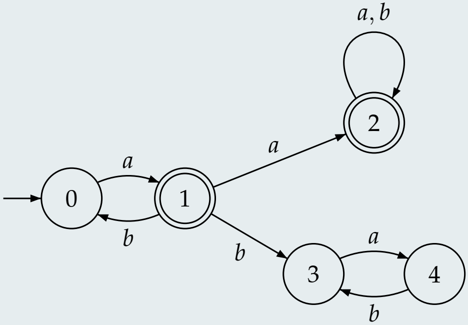
$a\ b\ a\ b\ \ldots$

$(ab)^\omega \in \mathcal{L}(\mathcal{A})$

$(ab)^* aa\{a,b\}^\omega \subseteq \mathcal{L}(\mathcal{A})$

Büchi automata (BA)

Emptiness test:



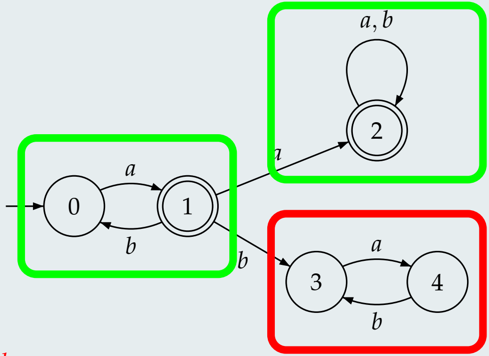
$a\ b\ a\ b\ \ldots$

$(ab)^\omega \in \mathcal{L}(\mathcal{A})$

$(ab)^* aa\{a,b\}^\omega \subseteq \mathcal{L}(\mathcal{A})$

Büchi automata (BA)

Emptiness test: SCCC, Tarjan



$a\ b\ a\ b\ \ldots$

$(ab)^\omega \in \mathcal{L}(\mathcal{A})$

$(ab)^* aa\{a,b\}^\omega \subseteq \mathcal{L}(\mathcal{A})$

LTL to BA

[Vardi & Wolper '86]

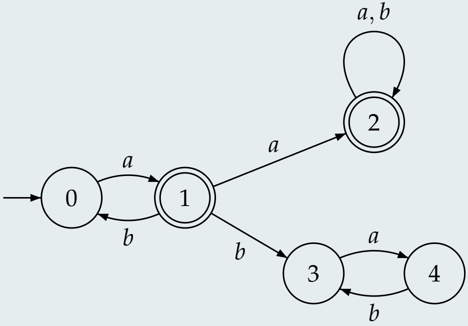
- Translation of an LTL formula φ into Büchi automata $\mathcal{A}_\varphi$ with

$$\mathcal{L}(\mathcal{A}_\varphi) = \mathcal{L}(\varphi)$$

- Complexity: Exponential in the length of φ

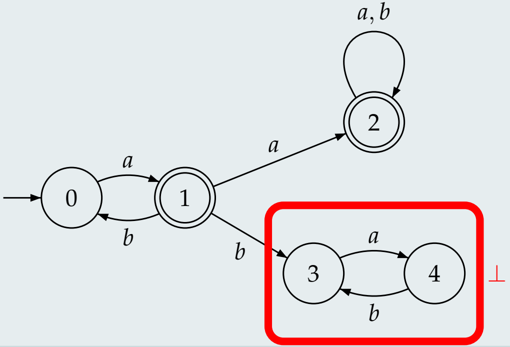
Monitor construction – Idea I

$$[u \models \varphi] = \begin{cases} \top & \text{if } \forall \sigma \in \Sigma^\omega : u\sigma \models \varphi \\ \bot & \text{if } \forall \sigma \in \Sigma^\omega : u\sigma \not\models \varphi \\ ? & \text{else} \end{cases}$$



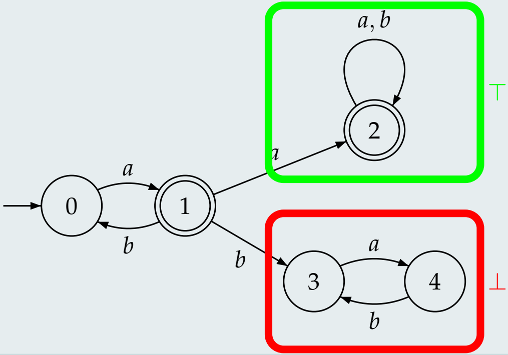
Monitor construction – Idea I

$$[u \models \varphi] = \begin{cases} \top & \text{if } \forall \sigma \in \Sigma^\omega : u\sigma \models \varphi \\ \bot & \text{if } \forall \sigma \in \Sigma^\omega : u\sigma \not\models \varphi \\ ? & \text{else} \end{cases}$$



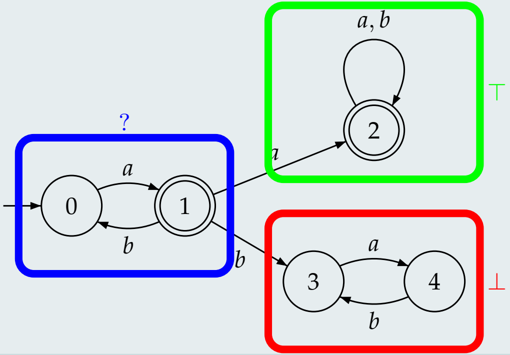
Monitor construction – Idea I

$$[u \models \varphi] = \begin{cases} \top & \text{if } \forall \sigma \in \Sigma^\omega : u\sigma \models \varphi \\ \bot & \text{if } \forall \sigma \in \Sigma^\omega : u\sigma \not\models \varphi \\ ? & \text{else} \end{cases}$$

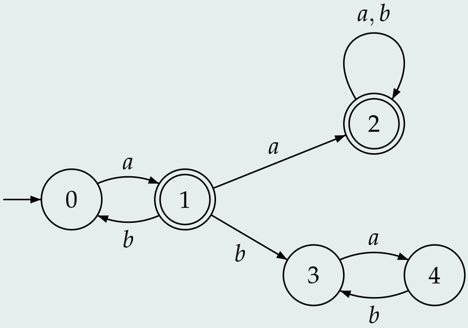


Monitor construction – Idea I

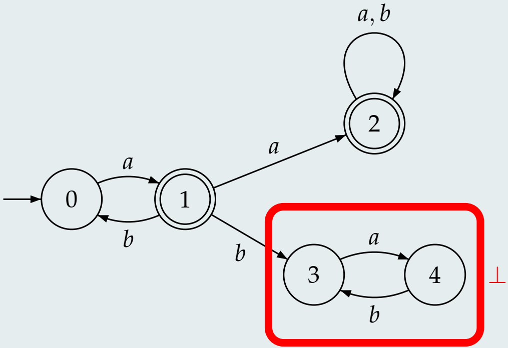
$$[u \models \varphi] = \begin{cases} \top & \text{if } \forall \sigma \in \Sigma^\omega : u\sigma \models \varphi \\ \bot & \text{if } \forall \sigma \in \Sigma^\omega : u\sigma \not\models \varphi \\ ? & \text{else} \end{cases}$$



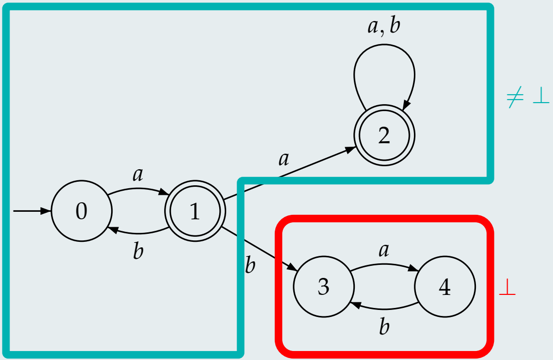
monitor construction – Idea II



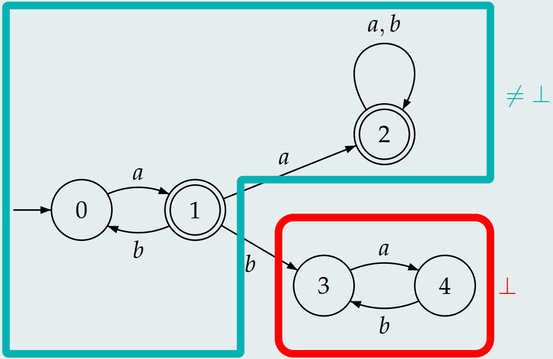
monitor construction – Idea II



monitor construction – Idea II



monitor construction – Idea II



NFA

$\mathcal{F}_\varphi : Q_\varphi \to \{\top, \bot\}$ Emptiness per state

The complete construction

The construction

$$\varphi \longrightarrow \mathrm{BA}^{\varphi} \longrightarrow \mathcal{F}^{\varphi} \longrightarrow \mathrm{NFA}^{\varphi}$$

Lemma

$$[u \models \varphi] = \begin{cases} \top & \\ \bot & \text{if } u \notin \mathcal{L}(\mathrm{NFA}^{\varphi}) \\ ? & \end{cases}$$

The complete construction

The construction

$$\varphi \longrightarrow \mathrm{BA}^{\varphi} \longrightarrow \mathcal{F}^{\varphi} \longrightarrow \mathrm{NFA}^{\varphi}$$

$$\neg\varphi$$

Lemma

$$[u \models \varphi] = \begin{cases} \top & \\ \bot & \text{if } u \notin \mathcal{L}(\mathrm{NFA}^{\varphi}) \\ ? & \end{cases}$$

The complete construction

The construction

$$\begin{tikzcd}
\varphi \arrow[r] & \mathrm{BA}^{\varphi} \arrow[r] & \mathcal{F}^{\varphi} \arrow[r] & \mathrm{NFA}^{\varphi} \\
\neg\varphi \arrow[r] & \mathrm{BA}^{\neg\varphi} \arrow[r] & \mathcal{F}^{\neg\varphi} \arrow[r] & \mathrm{NFA}^{\neg\varphi}
\end{tikzcd}$$

Lemma

$$[u \models \varphi] = \begin{cases} \top & \text{if } u \notin \mathcal{L}(\mathrm{NFA}^{\neg\varphi}) \\ \bot & \text{if } u \notin \mathcal{L}(\mathrm{NFA}^{\varphi}) \\ ? & \text{else} \end{cases}$$

The complete construction

The construction

$$\begin{tikzcd}
 & \varphi \arrow[r] & \mathrm{BA}^{\varphi} \arrow[r] & \mathcal{F}^{\varphi} \arrow[r] & \mathrm{NFA}^{\varphi} \\
\varphi \arrow[ur] \arrow[dr] & & & & \\
 & \neg\varphi \arrow[r] & \mathrm{BA}^{\neg\varphi} \arrow[r] & \mathcal{F}^{\neg\varphi} \arrow[r] & \mathrm{NFA}^{\neg\varphi}
\end{tikzcd}$$

The complete construction

The construction

$$\begin{tikzcd}
& \varphi \arrow[r] & \mathrm{BA}^{\varphi} \arrow[r] & \mathcal{F}^{\varphi} \arrow[r] & \mathrm{NFA}^{\varphi} \arrow[r] & \color{red}{\mathrm{DFA}^{\varphi}} \\
\varphi \arrow[ur] \arrow[dr] & & & & & \\
& \neg\varphi \arrow[r] & \mathrm{BA}^{\neg\varphi} \arrow[r] & \mathcal{F}^{\neg\varphi} \arrow[r] & \mathrm{NFA}^{\neg\varphi} \arrow[r] & \color{red}{\mathrm{DFA}^{\neg\varphi}}
\end{tikzcd}$$

The complete construction

The construction

$$\begin{tikzcd}
 & \varphi \arrow[r] & \mathrm{BA}^{\varphi} \arrow[r] & \mathcal{F}^{\varphi} \arrow[r] & \mathrm{NFA}^{\varphi} \arrow[r] & \mathrm{DFA}^{\varphi} \arrow[dr] & \\
\varphi \arrow[ur] \arrow[dr] & & & & & & M \\
 & \neg\varphi \arrow[r] & \mathrm{BA}^{\neg\varphi} \arrow[r] & \mathcal{F}^{\neg\varphi} \arrow[r] & \mathrm{NFA}^{\neg\varphi} \arrow[r] & \mathrm{DFA}^{\neg\varphi} \arrow[ur] &
\end{tikzcd}$$

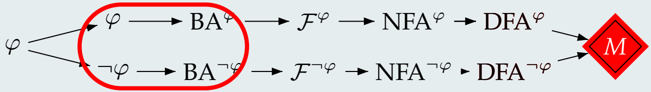
Complexity

The construction

$$\begin{tikzcd}
 & \varphi \arrow[r] & \mathrm{BA}^{\varphi} \arrow[r] & \mathcal{F}^{\varphi} \arrow[r] & \mathrm{NFA}^{\varphi} \arrow[r] & \mathrm{DFA}^{\varphi} \arrow[dr] & \\
\varphi \arrow[ur] \arrow[dr] & & & & & & \boxed{M} \\
 & \neg\varphi \arrow[r] & \mathrm{BA}^{\neg\varphi} \arrow[r] & \mathcal{F}^{\neg\varphi} \arrow[r] & \mathrm{NFA}^{\neg\varphi} \arrow[r] & \mathrm{DFA}^{\neg\varphi} \arrow[ur] &
\end{tikzcd}$$

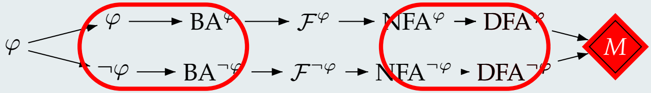
Complexity

The construction



Complexity

The construction



Complexity

The construction

$$\begin{tikzcd}
 & \varphi \arrow[r] & \mathrm{BA}^{\varphi} \arrow[r] & \mathcal{F}^{\varphi} \arrow[r] & \mathrm{NFA}^{\varphi} \arrow[r] & \mathrm{DFA}^{\varphi} \arrow[dr] & \\
\varphi \arrow[ur] \arrow[dr] & & & & & & \boxed{M} \\
 & \neg\varphi \arrow[r] & \mathrm{BA}^{\neg\varphi} \arrow[r] & \mathcal{F}^{\neg\varphi} \arrow[r] & \mathrm{NFA}^{\neg\varphi} \arrow[r] & \mathrm{DFA}^{\neg\varphi} \arrow[ur] &
\end{tikzcd}$$

Complexity

$$|M| \leq 2^{2^{|\varphi|}}$$

Complexity

The construction

$$\begin{tikzcd}
 & \varphi \arrow[r] & \mathrm{BA}^{\varphi} \arrow[r] & \mathcal{F}^{\varphi} \arrow[r] & \mathrm{NFA}^{\varphi} \arrow[r] & \mathrm{DFA}^{\varphi} \arrow[dr] & \\
\varphi \arrow[ur] \arrow[dr] & & & & & & \boxed{M} \\
 & \neg\varphi \arrow[r] & \mathrm{BA}^{\neg\varphi} \arrow[r] & \mathcal{F}^{\neg\varphi} \arrow[r] & \mathrm{NFA}^{\neg\varphi} \arrow[r] & \mathrm{DFA}^{\neg\varphi} \arrow[ur] &
\end{tikzcd}$$

Complexity

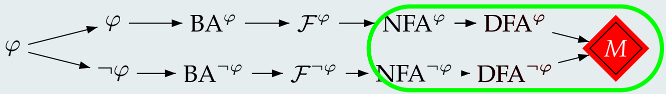
$$|M| \leq 2^{2^{|\varphi|}}$$

Optimal result!

FSM can be minimised (Myhill-Nerode)

On-the-fly Construction

The construction



$$\begin{tikzcd}
 & \varphi \arrow[r] & \mathrm{BA}^{\varphi} \arrow[r] & \mathcal{F}^{\varphi} \arrow[r] & \mathrm{NFA}^{\varphi} \arrow[r] & \mathrm{DFA}^{\varphi} \arrow[dr] & \\
\varphi \arrow[ur] \arrow[dr] & & & & & & M \\
 & \neg\varphi \arrow[r] & \mathrm{BA}^{\neg\varphi} \arrow[r] & \mathcal{F}^{\neg\varphi} \arrow[r] & \mathrm{NFA}^{\neg\varphi} \arrow[r] & \mathrm{DFA}^{\neg\varphi} \arrow[ur] &
\end{tikzcd}$$

Outline

- Runtime Verification
- Runtime Verification for LTL
  - LTL over Finite, Completed Words
  - LTL over Finite, Non-Completed Words: Impartiality
  - LTL over Non-Completed Words: Anticipation
  - LTL over Infinite Words: With Anticipation
  - Generalisations: LTL with modulo Constraints
  - Monitorable Properties
  - LTL with a Predictive Semantics
  - LTL wrap-up
- Extensions
- Monitoring Systems/Logging
- Steering
- Diagnosis
  - Ideas
  - RV and Diagnosis
- Conclusion

Towards richer and more expressive logics [DLS@ATVA'08]

Many linear-time logics

- LTL with Past

Towards richer and more expressive logics [DLS@ATVA'08]

Many linear-time logics

- LTL with Past
- linear-time μ-calculus

Towards richer and more expressive logics [DLS@ATVA'08]

Many linear-time logics

- LTL with Past
- linear-time μ-calculus
- RLTL

Towards richer and more expressive logics [DLS@ATVA'08]

Many linear-time logics

- LTL with Past
- linear-time μ-calculus
- RLTL
- LTL with integer constraints

$$G(fopen_x \to ((x = Xx)\ U\ fclose_x))$$

Linear-time Logic

Definition (Linear-time Logic)

A linear-time logic L defines

- a set F_L of L-formulae and
- a two-valued semantics $\models_L$.

Every L-formula $\varphi \in F_L$ has an associated and possibly infinite alphabet Σ_φ. Moreover, for every formula $\varphi \in F_L$ and every word $\sigma \in \Sigma_\varphi^\omega$, we require

(L1) $\forall \varphi \in F_L \ : \ \neg\varphi \in F_L.$

(L2) $\forall \sigma \in \Sigma_\varphi^\omega \ : \ (\sigma \models_L \varphi \quad \Leftrightarrow \quad \sigma \not\models_L \neg\varphi).$

Anticipation Semantics

Definition (Anticipation Semantics)

Let L be a linear-time logic. We define the anticipation semantics $[\pi \models \varphi]_L$ of an L-formula $\varphi \in F_L$ and a finite word $\pi \in \Sigma_\varphi^*$ with

$$[\pi \models \varphi]_L = \begin{cases} \top & \text{if } \forall \sigma \in \Sigma_\varphi^\omega : \pi\sigma \models_L \varphi \\ \bot & \text{if } \forall \sigma \in \Sigma_\varphi^\omega : \pi\sigma \not\models_L \varphi \\ ? & \text{otherwise} \end{cases}$$

Evaluation using decide

decide

$$[\pi \models \varphi]_L = \begin{cases} \top & \text{if } \mathsf{decide}_{\neg\varphi}(\pi) = \bot \\ \bot & \text{if } \mathsf{decide}_{\varphi}(\pi) = \bot \\ ? & \text{otherwise} \end{cases}$$

where $\mathsf{decide}_\varphi(\pi)$ is defined to return $\top$ for $\varphi \in F_L$ and $\pi \in \Sigma_\varphi$ if $\exists \sigma \in \Sigma_\varphi^\omega \ : \ \pi\sigma \models_L \varphi$ holds, and $\bot$ otherwise.

The automata theoretic approach to SAT

Definition (Satisfiability Check by Automata Abstraction)

Given a linear-time logic L with its formulae F_L, the satisfiability check by automata abstraction proceeds as follows. For formula $\varphi \in F_L$,

1. define alphabet abstraction $\Sigma_\varphi \to \bar{\Sigma}_\varphi$ finite, abstract alphabet
2. define a word abstraction $\alpha(\cdot) : \Sigma_\varphi^\omega \to \bar{\Sigma}_\varphi^\omega$
3. define an automaton construction $\varphi \mapsto \omega$-automaton $\mathcal{A}_\varphi$ over $\bar{\Sigma}_\varphi$ such that for all $\bar{\sigma} \in \bar{\Sigma}_\varphi^\omega$ it holds

   $$\bar{\sigma} \in \mathcal{L}(\mathcal{A}_\varphi) \text{ iff } \exists \sigma \in \Sigma^\omega : \bar{\sigma} = \alpha(\sigma) \text{ and } \sigma \models \varphi$$

   Then

   $$\varphi \text{ satisfiable iff } \mathcal{L}(\mathcal{A}_\varphi) \neq \emptyset \text{ iff non-empty}(\mathcal{A}_\varphi)$$

From finite to infinite

Definition (extrapolate)

$$\text{extrapolate}(\pi) = \left\{ \alpha(\pi\sigma)^{0\ldots i} \mid i+1 = |\pi|, \sigma \in \Sigma^\omega \right\}$$

Definition (Accuracy of Abstract Automata)

accuracy of abstract automata property holds, if, for all $\pi \in \Sigma^*$,

- $(\exists \sigma : \pi\sigma \models_L \varphi) \Rightarrow (\exists \bar{\pi} \exists \bar{\sigma} : \bar{\pi}\bar{\sigma} \in \mathcal{L}(\mathcal{A}_\varphi))$ with $\bar{\pi} \in \text{extrapolate}(\pi)$,
- $(\exists \bar{\sigma} : \bar{\pi}\bar{\sigma} \in \mathcal{L}(\mathcal{A}_\varphi)) \Rightarrow (\exists \pi \exists \sigma : \pi\sigma \models_L \varphi)$ with $\bar{\pi} \in \text{extrapolate}(\pi)$.

Non-incremental version

Theorem (Correctness of decide)

Given a satisfiability check by automata abstraction for a linear-time logic L satisfying the accuracy of automata property, we have

$$\mathsf{decide}(\pi) = \mathsf{non\text{-}empty}\left(\bigcup_{q \in Q_0, \bar{\pi} \in \mathsf{extrapolate}(\pi)} \delta(q, \bar{\pi}) \right)$$

Faithful abstraction

Definition (Forgettable Past and Faithful Abstraction)

Given α of a satisfiability check by automata abstraction. We say that

- α satisfies the forgettable past property, iff

$$\alpha(\pi a \sigma)^{i+1\dots i+1} = \alpha(a\sigma)^{0\dots 0}$$

  for all $\pi \in \Sigma^*$, $|\pi| = i+1$, $a \in \Sigma$, and $\sigma \in \Sigma^\omega$.

- α is called faithful, iff for all $\pi \in \Sigma^*$, $|\pi| = i+1$, $a \in \Sigma$, $\sigma, \sigma' \in \Sigma^\omega$ for which there is some $\sigma'' \in \Sigma^\omega$ with $\alpha(\pi\sigma)^{0\dots i}\alpha(a\sigma')^{0\dots 0} = \alpha(\sigma'')^{0\dots i+1}$ there also exists a $\sigma''' \in \Sigma^\omega$ with

$$\alpha(\pi\sigma)^{0\dots i}\alpha(a\sigma')^{0\dots 0} = \alpha(\pi a \sigma''')^{0\dots i+1}$$

Incremental version

Theorem (Incremental Emptiness for Extrapolation)

Let $\mathcal{A}$ be a Büchi automaton obtained via a satisfiability check by automata abstraction satisfying the accuracy of automaton abstraction property with a faithful abstraction function having the forgettable past property. Then, for all $\pi \in \Sigma^*$ and $a \in \Sigma$, it holds

$$\mathcal{L}(\mathcal{A}(\mathsf{extrapolate}(\pi a))) = \mathcal{L}(\mathcal{A}(\mathsf{extrapolate}(\pi)\mathsf{extrapolate}(a)))$$

Further logics

Indeed works

- LTL with Past
- linear-time μ-calculus
- RLTL
- *LTL* with integer constraints

Outline

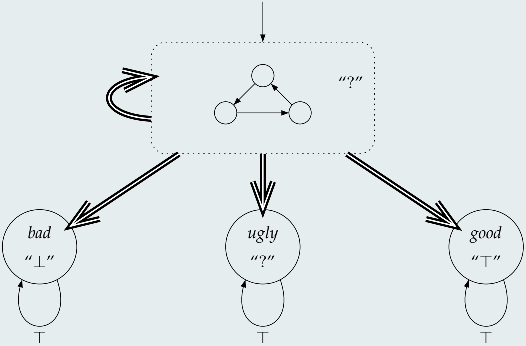
- Runtime Verification
- Runtime Verification for LTL
  - LTL over Finite, Completed Words
  - LTL over Finite, Non-Completed Words: Impartiality
  - LTL over Non-Completed Words: Anticipation
  - LTL over Infinite Words: With Anticipation
  - Generalisations: LTL with modulo Constraints
  - Monitorable Properties
  - LTL with a Predictive Semantics
  - LTL wrap-up
- Extensions
- Monitoring Systems/Logging
- Steering
- Diagnosis
  - Ideas
  - RV and Diagnosis
- Conclusion

Monitorability

When does anticipation help?

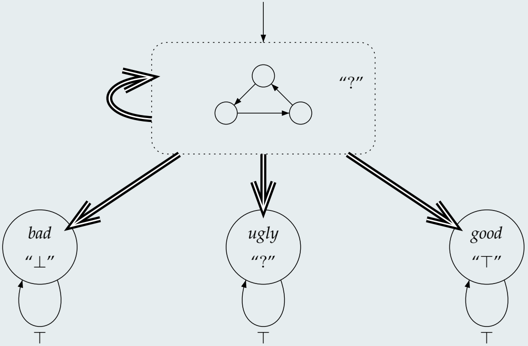
Monitors revisited

Structure of Monitors



Monitors revisited

Structure of Monitors

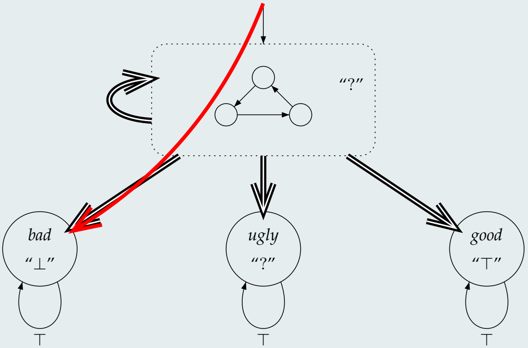


Classification of Prefixes of Words

- Bad prefixes [Kupferman & Vardi'01]

Monitors revisited

Structure of Monitors

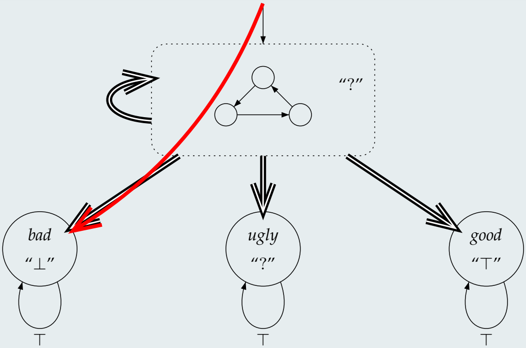


Classification of Prefixes of Words

- Bad prefixes [Kupferman & Vardi'01]

Monitors revisited

Structure of Monitors

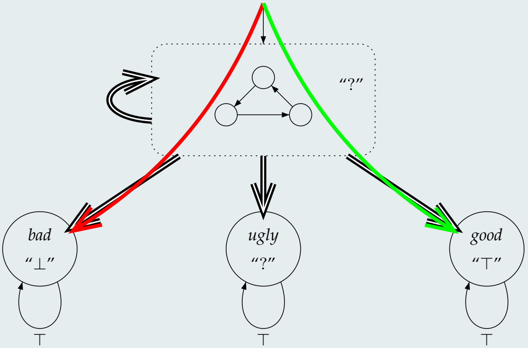


Classification of Prefixes of Words

- Bad prefixes [Kupferman & Vardi'01]
- Good prefixes [Kupferman & Vardi'01]

Monitors revisited

Structure of Monitors

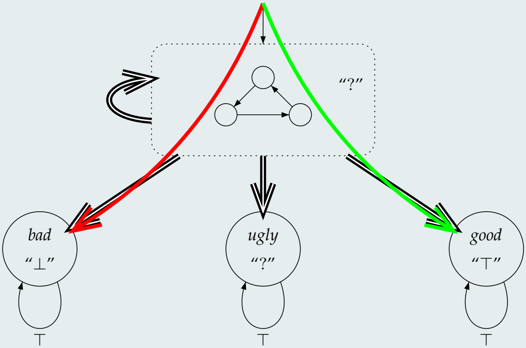


Classification of Prefixes of Words

- Bad prefixes [Kupferman & Vardi'01]
- Good prefixes [Kupferman & Vardi'01]

Monitors revisited

Structure of Monitors

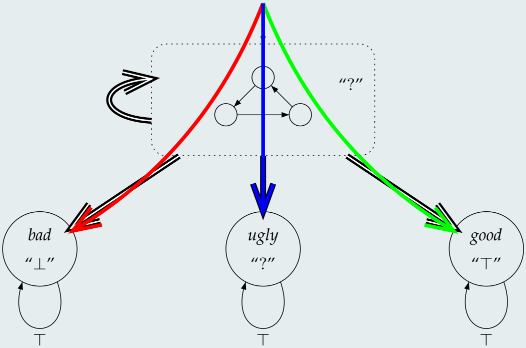


Classification of Prefixes of Words

- Bad prefixes [Kupferman & Vardi'01]
- Good prefixes [Kupferman & Vardi'01]
- Ugly prefixes

Monitors revisited

Structure of Monitors



Classification of Prefixes of Words

- Bad prefixes [Kupferman & Vardi'01]
- Good prefixes [Kupferman & Vardi'01]
- Ugly prefixes

Monitorable

Non-Monitorable [Pnueli & Zaks'07]

φ is non-monitorable after u, if u cannot be extended to a bad oder good prefix.

Monitorable

φ is monitorable if there is no such u.

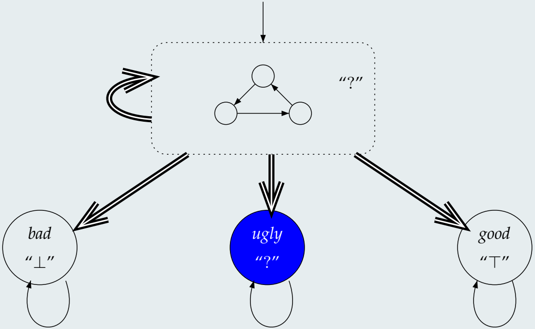
Monitorable

Non-Monitorable [Pnueli & Zaks'07]

φ is non-monitorable after u, if u cannot be extended to a bad oder good prefix.

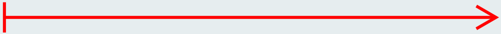
Monitorable

φ is monitorable if there is no such u.



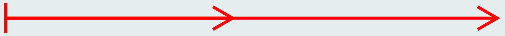
Monitorable Properties

Safety Properties



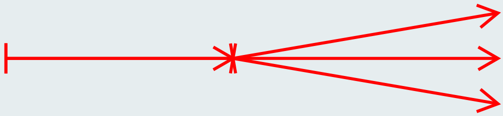
Monitorable Properties

Safety Properties



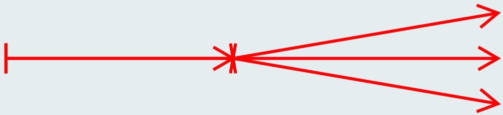
Monitorable Properties

Safety Properties

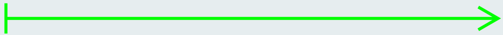


Monitorable Properties

Safety Properties

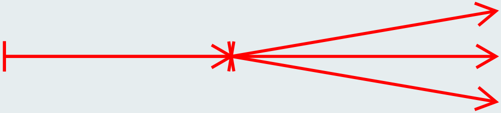


Co-Safety Properties

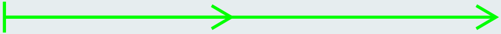


Monitorable Properties

Safety Properties

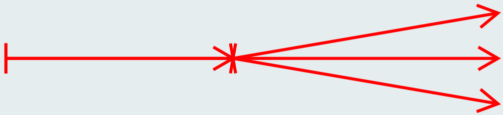


Co-Safety Properties

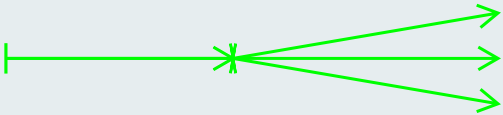


Monitorable Properties

Safety Properties

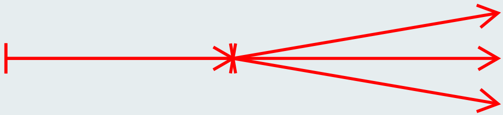


Co-Safety Properties

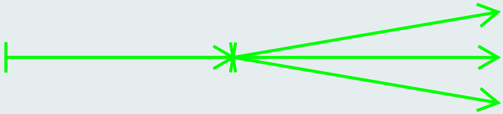


Monitorable Properties

Safety Properties



Co-Safety Properties



Note

Safety and Co-Safety Properties are monitorable

Safety- and Co-Safety-Properties

Theorem

The class of monitorable properties

- comprises safety- and co-safety properties, but
- is strictly larger than their union.

Proof

Consider $((p \vee q)Ur) \vee Gp$

Outline

- Runtime Verification
- Runtime Verification for LTL
  - LTL over Finite, Completed Words
  - LTL over Finite, Non-Completed Words: Impartiality
  - LTL over Non-Completed Words: Anticipation
  - LTL over Infinite Words: With Anticipation
  - Generalisations: LTL with modulo Constraints
  - Monitorable Properties
  - LTL with a Predictive Semantics
  - LTL wrap-up
- Extensions
- Monitoring Systems/Logging
- Steering
- Diagnosis
  - Ideas
  - RV and Diagnosis
- Conclusion

Fusing model checking and runtime verification

LTL with a predictive semantics

Recall anticipatory LTL semantics

The truth value of a LTL$_3$ formula φ wrt. u, denoted by $[u \models \varphi]$, is an element of $\mathbb{B}_3$ defined by

$$[u \models \varphi] = \begin{cases} \top & \text{if } \forall \sigma \in \Sigma^\omega : u\sigma \models \varphi \\ \bot & \text{if } \forall \sigma \in \Sigma^\omega : u\sigma \not\models \varphi \\ ? & \text{otherwise.} \end{cases}$$

Applied to the empty word

Empty word ϵ

$$\begin{array}{ll} & [\epsilon \models \varphi]_{\mathcal{P}} = \top \\ \text{iff} & \forall \sigma \in \Sigma^{\omega} \text{ with } \epsilon\sigma \in \mathcal{P} : \epsilon\sigma \models \varphi \\ \text{iff} & \mathcal{L}(\mathcal{P}) \models \varphi \end{array}$$

RV more difficult than MC?

Then runtime verification implicitly answers model checking

Abstraction

An over-abstraction or and over-approximation of a program $\mathcal{P}$ is a program $\hat{\mathcal{P}}$ such that $\mathcal{L}(\mathcal{P}) \subseteq \mathcal{L}(\hat{\mathcal{P}}) \subseteq \Sigma^\omega$.

Predictive Semantics

Definition (Predictive semantics of LTL)

Let $\mathcal{P}$ be a program and let $\hat{\mathcal{P}}$ be an over-approximation of $\mathcal{P}$. Let $u \in \Sigma^*$ denote a finite trace. The *truth value* of u and an LTL_3 formula φ wrt. $\hat{\mathcal{P}}$, denoted by $[u \models_{\hat{\mathcal{P}}} \varphi]$, is an element of $\mathbb{B}_3$ and defined as follows:

$$[u \models_{\hat{\mathcal{P}}} \varphi] = \begin{cases} \top & \text{if } \forall \sigma \in \Sigma^\omega \text{ with } u\sigma \in \hat{\mathcal{P}} : u\sigma \models \varphi \\ \bot & \text{if } \forall \sigma \in \Sigma^\omega \text{ with } u\sigma \in \hat{\mathcal{P}} : u\sigma \not\models \varphi \\ ? & \text{else} \end{cases}$$

We write $\mathrm{LTL}_{\mathcal{P}}$ whenever we consider LTL formulas with a predictive semantics.

Properties of Predictive Semantics

Let $\hat{\mathcal{P}}$ be an over-approximation of a program $\mathcal{P}$ over Σ, $u \in \Sigma^*$, and $\varphi \in$ LTL.

- Model checking is more precise than RV with the predictive semantics:

$$\mathcal{P} \models \varphi \text{ implies } [u \models_{\hat{\mathcal{P}}} \varphi] \in \{\top, ?\}$$

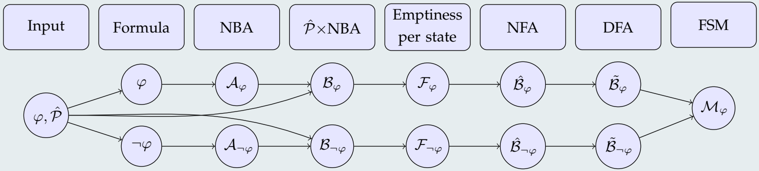
- RV has no false negatives: $[u \models_{\hat{\mathcal{P}}} \varphi] = \bot$ implies $\mathcal{P} \not\models \varphi$
- The predictive semantics of an LTL formula is more precise than LTL$_3$:

$$\begin{array}{lcl} [u \models \varphi] = \top & \text{implies} & [u \models_{\hat{\mathcal{P}}} \varphi] = \top \\ [u \models \varphi] = \bot & \text{implies} & [u \models_{\hat{\mathcal{P}}} \varphi] = \bot \end{array}$$

The reverse directions are in general not true.

Monitor generation

The procedure for getting $[u \models_{\hat{\mathcal{P}}} \varphi]$ for a given φ and over-approximation $\hat{\mathcal{P}}$



Input

Formula

NBA

$\hat{\mathcal{P}}\times$NBA

Emptiness per state

NFA

DFA

FSM

$\varphi, \hat{\mathcal{P}}$

φ

$\neg\varphi$

$\mathcal{A}_\varphi$

$\mathcal{A}_{\neg\varphi}$

$\mathcal{B}_\varphi$

$\mathcal{B}_{\neg\varphi}$

$\mathcal{F}_\varphi$

$\mathcal{F}_{\neg\varphi}$

$\hat{\mathcal{B}}_\varphi$

$\hat{\mathcal{B}}_{\neg\varphi}$

$\tilde{\mathcal{B}}_\varphi$

$\tilde{\mathcal{B}}_{\neg\varphi}$

$\mathcal{M}_\varphi$

Outline

Runtime Verification

Runtime Verification for LTL

- LTL over Finite, Completed Words
- LTL over Finite, Non-Completed Words: Impartiality
- LTL over Non-Completed Words: Anticipation
- LTL over Infinite Words: With Anticipation
- Generalisations: LTL with modulo Constraints
- Monitorable Properties
- LTL with a Predictive Semantics
- LTL wrap-up

Extensions

Monitoring Systems/Logging

Steering

Diagnosis

- Ideas
- RV and Diagnosis

Conclusion

Intermediate Summary

Semantics

- completed traces
  - two valued semantics
- non-completed traces
  - Impartiality
    - at least three values
  - Anticipation
    - finite traces
    - infinite traces
    - . . .
    - monitorability
  - Prediction

Monitors

- left-to-right
- time versus space trade-off
  - rewriting
  - alternating automata
  - non-deterministic automata
  - deterministic automata

Presentation outline

- Runtime Verification
- Runtime Verification for LTL
  - LTL over Finite, Completed Words
  - LTL over Finite, Non-Completed Words: Impartiality
  - LTL over Non-Completed Words: Anticipation
  - LTL over Infinite Words: With Anticipation
  - Generalisations: LTL with modulo Constraints
  - Monitorable Properties
  - LTL with a Predictive Semantics
  - LTL wrap-up
- Extensions
- Monitoring Systems/Logging
- Steering
- Diagnosis
  - Ideas
  - RV and Diagnosis
- Conclusion

Extensions

LTL is just half of the story

Extensions

LTL with data

- J-LO

Extensions

LTL with data

- J-LO
- MOP (parameterized LTL)

Extensions

LTL with data

- J-LO
- MOP (parameterized LTL)
- RV for LTL with integer constraints

Extensions

LTL with data

- J-LO
- MOP (parameterized LTL)
- RV for LTL with integer constraints

Extensions

LTL with data

- J-LO
- MOP (parameterized LTL)
- RV for LTL with integer constraints

Further “rich” approaches

- LOLA

Extensions

LTL with data

- J-LO
- MOP (parameterized LTL)
- RV for LTL with integer constraints

Further “rich” approaches

- LOLA
- Eagle (etc.)

Extensions

LTL with data

- J-LO
- MOP (parameterized LTL)
- RV for LTL with integer constraints

Further "rich" approaches

- LOLA
- Eagle (etc.)

Extensions

LTL with data

- J-LO
- MOP (parameterized LTL)
- RV for LTL with integer constraints

Further “rich” approaches

- LOLA
- Eagle (etc.)

Further dimensions

- real-time

Extensions

LTL with data

- J-LO
- MOP (parameterized LTL)
- RV for LTL with integer constraints

Further “rich” approaches

- LOLA
- Eagle (etc.)

Further dimensions

- real-time
- concurrency

Extensions

LTL with data

- J-LO
- MOP (parameterized LTL)
- RV for LTL with integer constraints

Further “rich” approaches

- LOLA
- Eagle (etc.)

Further dimensions

- real-time
- concurrency
- distribution

Presentation outline

Runtime Verification

Runtime Verification for LTL

- LTL over Finite, Completed Words
- LTL over Finite, Non-Completed Words: Impartiality
- LTL over Non-Completed Words: Anticipation
- LTL over Infinite Words: With Anticipation
- Generalisations: LTL with modulo Constraints
- Monitorable Properties
- LTL with a Predictive Semantics
- LTL wrap-up

Extensions

Monitoring Systems/Logging

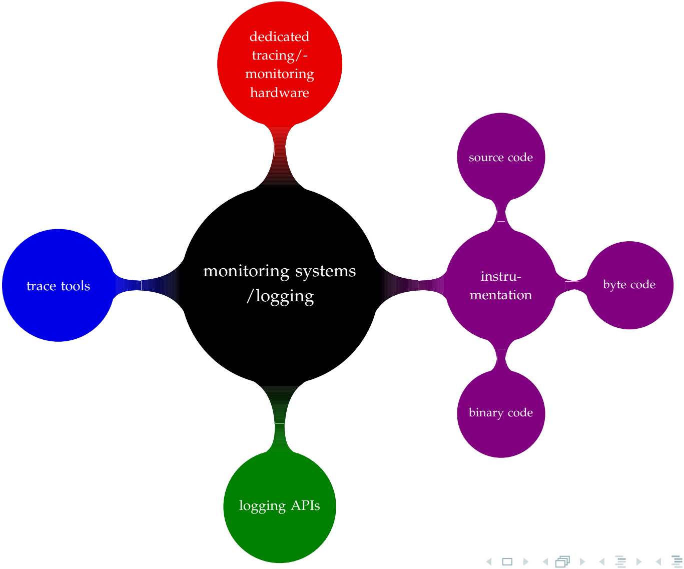
Steering

Diagnosis

- Ideas
- RV and Diagnosis

Conclusion

Monitoring Systems/Logging: Overview



Presentation outline

Runtime Verification

Runtime Verification for LTL

- LTL over Finite, Completed Words
- LTL over Finite, Non-Completed Words: Impartiality
- LTL over Non-Completed Words: Anticipation
- LTL over Infinite Words: With Anticipation
- Generalisations: LTL with modulo Constraints
- Monitorable Properties
- LTL with a Predictive Semantics
- LTL wrap-up

Extensions

Monitoring Systems/Logging

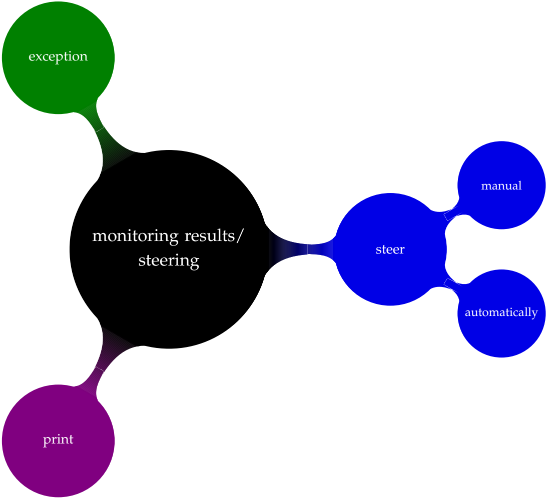
Steering

Diagnosis

- Ideas
- RV and Diagnosis

Conclusion

Monitoring Systems/Logging: Overview



React!

Runtime Verification

Observe—do not react

Realising dynamic systems

- self-healing systems
- adaptive systems, self-organising systems
- . . .

React!

Runtime Verification

Observe—do not react

Realising dynamic systems

- self-healing systems
- adaptive systems, self-organising systems
- ...
- use monitors for observation—then react

jMOP [Rosu et al.]

Java Implementation

```
class Resource {
/*@
     scope = class
     logic = PTLTL
     {
      Event authenticate: end(exec(*
authenticate()));
      Event use: begin(exec(* access()));
      Formula : use -> <*> authenticate
     }
     violation Handler {
       @this.authenticate();
     }
@*/
void authenticate() {...}
void access() {...}
...
}
```

Where

How

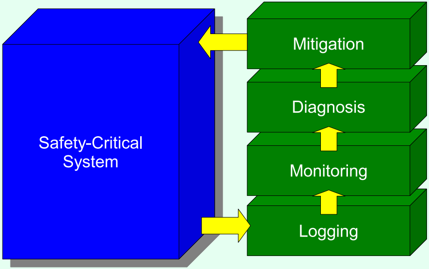
what

What if

Runtime Reflection [Bauer, L., Schallhart@ASWEC'06]

Monitor-based Runtime Reflection

Software Architecture Pattern



Presentation outline

- Runtime Verification
- Runtime Verification for LTL
  - LTL over Finite, Completed Words
  - LTL over Finite, Non-Completed Words: Impartiality
  - LTL over Non-Completed Words: Anticipation
  - LTL over Infinite Words: With Anticipation
  - Generalisations: LTL with modulo Constraints
  - Monitorable Properties
  - LTL with a Predictive Semantics
  - LTL wrap-up
- Extensions
- Monitoring Systems/Logging
- Steering
- Diagnosis
  - Ideas
  - RV and Diagnosis
- Conclusion

Outline

- Runtime Verification
- Runtime Verification for LTL
  - LTL over Finite, Completed Words
  - LTL over Finite, Non-Completed Words: Impartiality
  - LTL over Non-Completed Words: Anticipation
  - LTL over Infinite Words: With Anticipation
  - Generalisations: LTL with modulo Constraints
  - Monitorable Properties
  - LTL with a Predictive Semantics
  - LTL wrap-up
- Extensions
- Monitoring Systems/Logging
- Steering
- Diagnosis
  - Ideas
  - RV and Diagnosis
- Conclusion

Diagnosis

Main Ideas

- Knowledge base
- Knowledge
- Explanation of Knowledge with Respect to the Knowledge base

Diagnosis

Main Ideas

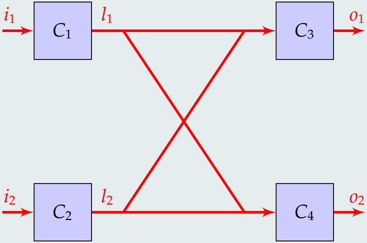
- Knowledge base
- Knowledge
- Explanation of Knowledge with Respect to the Knowledge base

Here

- System description
- Observations
- Diagnosis: Explanation of the Observations with respect to the System description

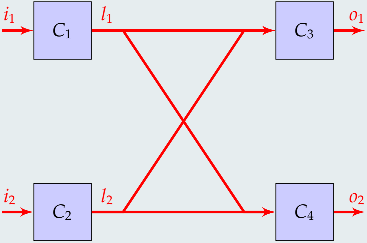
System Description in First-Order Logic

Example



System Description in First-Order Logic

Example

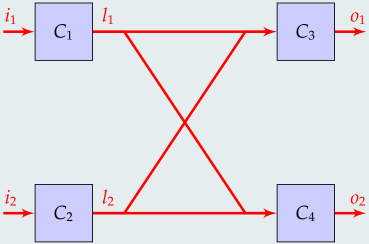


Formally

$$\begin{aligned} SD \quad = \quad & \quad ok(i_1) \land \neg AB(C_1) \rightarrow l_1 = C_1(i_1) \\ \land & \quad ok(i_2) \land \neg AB(C_2) \rightarrow l_2 = C_2(i_2) \\ \land & \quad ok(l_1) \land ok(l_2) \land \neg AB(C_3) \rightarrow o_1 = C_3(l_1, l_2) \\ \land & \quad ok(l_1) \land ok(l_2) \land \neg AB(C_4) \rightarrow o_2 = C_4(l_1, l_2) \end{aligned}$$

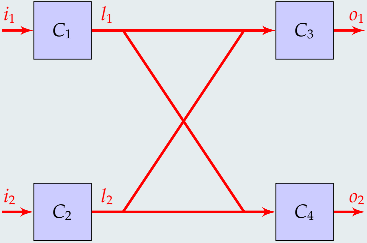
System Description in Propositional Logic

Example



System Description in Propositional Logic

Example

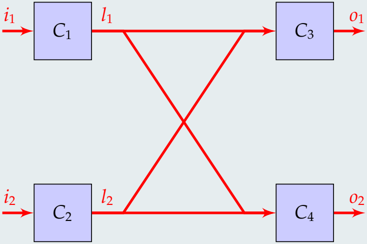


Propositional Logic

$$\begin{aligned} SD \quad = & \quad i_1 \land \neg C_1 \to l_1 \\ \land & \quad i_2 \land \neg C_2 \to l_2 \\ \land & \quad l_1 \land l_2 \land \neg C_3 \to o_1 \\ \land & \quad l_1 \land l_2 \land \neg C_4 \to o_2 \end{aligned}$$

Observation

Example



Observation

(Truth) values for (some of) the propositions involved

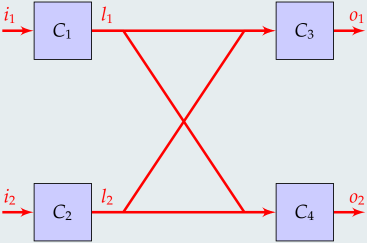
Formally: a formula OBS

Observation

$\neg o_1 \wedge i_1 \wedge i_2 \wedge o_2$

Diagnosis

Example

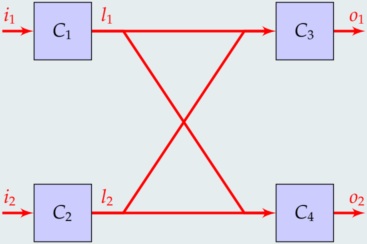


Diagnosis

A minimal set of components such that $SD \wedge OBS \wedge \Delta$ is satisfiable, where Δ encodes the chosen components.

Example

Example



Propositional Logic

$$\begin{array}{rcl} SD & = & i_1 \wedge \neg C_1 \to l_1 \\ & \wedge & i_2 \wedge \neg C_2 \to l_2 \\ & \wedge & l_1 \wedge l_2 \wedge \neg C_3 \to o_1 \\ & \wedge & l_1 \wedge l_2 \wedge \neg C_4 \to o_2 \end{array}$$

Observations

$\neg o_1 \wedge i_1 \wedge i_2 \wedge o_2$

Example

Propositional Logic

$$
\begin{aligned}
SD \quad = & \quad i_1 \land \neg C_1 \to l_1 \\
\land & \quad i_2 \land \neg C_2 \to l_2 \\
\land & \quad l_1 \land l_2 \land \neg C_3 \to o_1 \\
\land & \quad l_1 \land l_2 \land \neg C_4 \to o_2
\end{aligned}
$$

Observations

$\neg o_1 \land i_1 \land i_2 \land o_2$

CNF

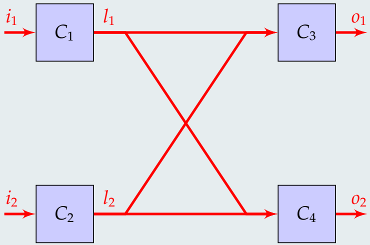
$$
\begin{aligned}
SD \quad = & \quad \neg i_1 \lor C_1 \lor l_1 \\
\land & \quad \neg i_2 \lor C_2 \lor l_2 \\
\land & \quad \neg l_1 \lor \neg l_2 \lor C_3 \lor o_1 \\
\land & \quad \neg l_1 \lor \neg l_2 \lor C_4 \lor o_2
\end{aligned}
$$

SD ∧ Observations

$$
\begin{aligned}
SD \quad = & \quad C_1 \lor l_1 \\
\land & \quad C_2 \lor l_2 \\
\land & \quad \neg l_1 \lor \neg l_2 \lor C_3 \\
\land & \\
\land & \quad \neg o_1 \land i_1 \land i_2 \land o_2
\end{aligned}
$$

Example

Example

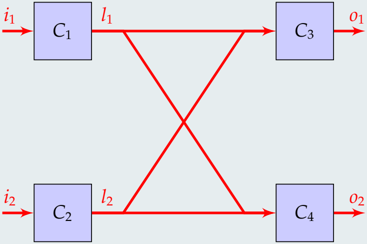


SD ∧ Observations

$$\begin{aligned} SD \quad = \quad & C_1 \vee l_1 \\ \wedge \quad & C_2 \vee l_2 \\ \wedge \quad & \neg l_1 \vee \neg l_2 \vee C_3 \\ \wedge \quad & \neg o_1 \wedge i_1 \wedge i_2 \wedge o_2 \end{aligned}$$

Example

Example

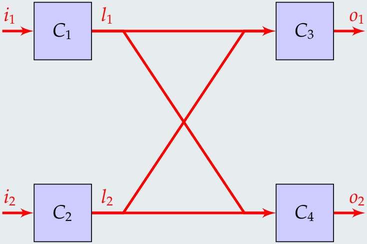


SD ∧ Observations

$$\begin{aligned} SD \quad = & \quad C_1 \vee l_1 \\ \wedge & \quad C_2 \vee l_2 \\ \wedge & \quad \neg l_1 \vee \neg l_2 \vee C_3 \\ \wedge & \quad \neg o_1 \wedge i_1 \wedge i_2 \wedge o_2 \end{aligned}$$

Example

Example

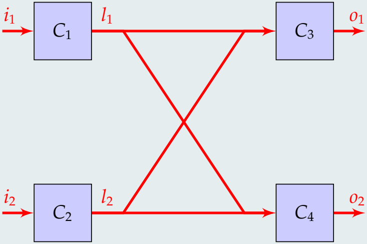


SD ∧ Observations

$$\begin{array}{rcl} SD & = & C_1 \lor l_1 \\ & \land & C_2 \lor l_2 \\ & \land & \neg l_1 \lor \neg l_2 \lor C_3 \\ & \land & \neg o_1 \land i_1 \land i_2 \land o_2 \end{array}$$

Example

Example

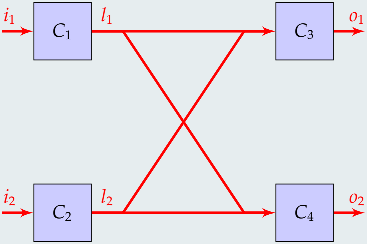


SD ∧ Observations

$$\begin{aligned} SD \quad = \quad & C_1 \vee l_1 \\ \wedge \quad & C_2 \vee l_2 \\ \wedge \quad & \neg l_1 \vee \neg l_2 \vee C_3 \\ \wedge \quad & \neg o_1 \wedge i_1 \wedge i_2 \wedge o_2 \end{aligned}$$

Example

Example

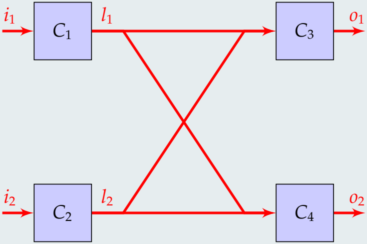


SD ∧ Observations

$$\begin{aligned} SD \quad = & \quad C_1 \vee l_1 \\ \wedge & \quad C_2 \vee l_2 \\ \wedge & \quad \neg l_1 \vee \neg l_2 \vee C_3 \\ \wedge & \quad \neg o_1 \wedge i_1 \wedge i_2 \wedge o_2 \end{aligned}$$

Example

Example

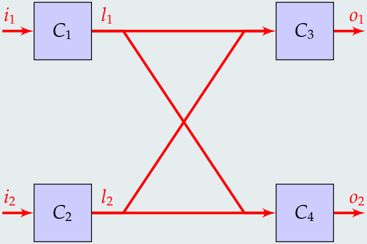


SD ∧ Observations

$$
\begin{aligned}
SD \quad = & \quad C_1 \vee l_1 \\
\wedge & \quad C_2 \vee l_2 \\
\wedge & \quad \neg l_1 \vee \neg l_2 \vee C_3 \\
\wedge & \quad \neg o_1 \wedge i_1 \wedge i_2 \wedge o_2
\end{aligned}
$$

Example

Example



SD ∧ Observations

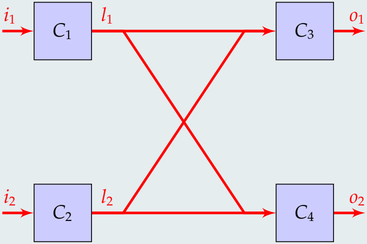
$$\begin{aligned} SD \quad = \quad & C_1 \vee l_1 \\ \wedge \quad & C_2 \vee l_2 \\ \wedge \quad & \neg l_1 \vee \neg l_2 \vee C_3 \\ \wedge \quad & \neg o_1 \wedge i_1 \wedge i_2 \wedge o_2 \end{aligned}$$

Diagnoses

- $\Delta_1 = \{C_1\}$

Example

Example



SD ∧ Observations

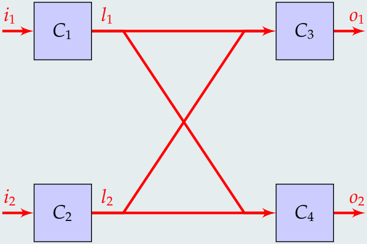
$$\begin{aligned} SD \quad = \quad & C_1 \vee l_1 \\ \wedge \quad & C_2 \vee l_2 \\ \wedge \quad & \neg l_1 \vee \neg l_2 \vee C_3 \\ \wedge \quad & \neg o_1 \wedge i_1 \wedge i_2 \wedge o_2 \end{aligned}$$

Diagnoses

- $\Delta_1 = \{C_1\}$
- $\Delta_2 = \{C_2\}$

Example

Example



SD ∧ Observations

$$\begin{aligned} SD \quad = \quad & C_1 \vee l_1 \\ \wedge \quad & C_2 \vee l_2 \\ \wedge \quad & \neg l_1 \vee \neg l_2 \vee C_3 \\ \wedge \quad & \neg o_1 \wedge i_1 \wedge i_2 \wedge o_2 \end{aligned}$$

Diagnoses

- $\Delta_1 = \{C_1\}$
- $\Delta_2 = \{C_2\}$
- $\Delta_3 = \{C_3\}$

Outline

- Runtime Verification
- Runtime Verification for LTL
  - LTL over Finite, Completed Words
  - LTL over Finite, Non-Completed Words: Impartiality
  - LTL over Non-Completed Words: Anticipation
  - LTL over Infinite Words: With Anticipation
  - Generalisations: LTL with modulo Constraints
  - Monitorable Properties
  - LTL with a Predictive Semantics
  - LTL wrap-up
- Extensions
- Monitoring Systems/Logging
- Steering
- Diagnosis
  - Ideas
  - RV and Diagnosis
- Conclusion

Monitors yield Obervations

We have. . .

- Monitor reports $\perp$ $\rightsquigarrow$ line is false

Monitors yield Obervations

We have. . .

- Monitor reports $\bot \rightsquigarrow$ line is false
- Monitor reports ? $\rightsquigarrow$ line is ? (no assignment)

Monitors yield Obervations

We have. . .

- Monitor reports $\bot \rightsquigarrow$ line is false
- Monitor reports $? \rightsquigarrow$ line is ? (no assignment)
- Monitor reports $\top \rightsquigarrow$ line is ? (no assignment)

Monitors yield Obervations

We have...

- Monitor reports $\bot \rightsquigarrow$ line is false
- Monitor reports ? $\rightsquigarrow$ line is ? (no assignment)
- Monitor reports $\top \rightsquigarrow$ line is ? (no assignment)

Monitors yield Obervations

We have...

- Monitor reports $\bot \rightsquigarrow$ line is false
- Monitor reports ? $\rightsquigarrow$ line is ? (no assignment)
- Monitor reports $\top \rightsquigarrow$ line is ? (no assignment)

Omniscent Monitors

A monitor is called omnicscent if its output $\top$ implies that the results on the monitored output are indeed correct.

Monitors yield Obervations

We have. . .

- Monitor reports $\perp \rightsquigarrow$ line is false
- Monitor reports ? $\rightsquigarrow$ line is ? (no assignment)
- Monitor reports $\top \rightsquigarrow$ line is ? (no assignment)

Omniscent Monitors

A monitor is called omnicscent if its output $\top$ implies that the results on the monitored output are indeed correct.

For Omniscent Monitors

- Monitor reports $\perp \rightsquigarrow$ line is false

Monitors yield Obervations

We have...

- Monitor reports $\bot \rightsquigarrow$ line is false
- Monitor reports ? $\rightsquigarrow$ line is ? (no assignment)
- Monitor reports $\top \rightsquigarrow$ line is ? (no assignment)

Omniscent Monitors

A monitor is called omnicscent if its output $\top$ implies that the results on the monitored output are indeed correct.

For Omniscent Monitors

- Monitor reports $\bot \rightsquigarrow$ line is false
- Monitor reports ? $\rightsquigarrow$ line is ? (no assignment)

Monitors yield Obervations

We have. . .

- Monitor reports $\perp$ $\rightsquigarrow$ line is false
- Monitor reports ? $\rightsquigarrow$ line is ? (no assignment)
- Monitor reports $\top$ $\rightsquigarrow$ line is ? (no assignment)

Omniscent Monitors

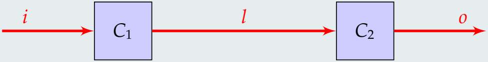
A monitor is called omnicscent if its output $\top$ implies that the results on the monitored output are indeed correct.

For Omniscent Monitors

- Monitor reports $\perp$ $\rightsquigarrow$ line is false
- Monitor reports ? $\rightsquigarrow$ line is ? (no assignment)
- Monitor reports $\top$ $\rightsquigarrow$ line is true

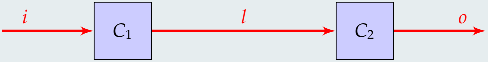
Oniscent Monitors

Example



Oniscent Monitors

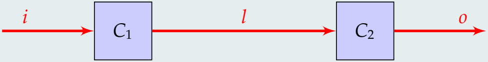
Example



$$\begin{aligned} SD &= && i \wedge \neg C_1 \rightarrow l \\ & \wedge && l \wedge \neg C_2 \rightarrow o \end{aligned} \qquad \begin{aligned} SD &= && \neg i \vee C_1 \vee l \\ & \wedge && \neg l \vee C_2 \vee o \end{aligned}$$

Oniscent Monitors

Example



$$SD = i \wedge \neg C_1 \rightarrow l \quad \wedge \quad l \wedge \neg C_2 \rightarrow o$$

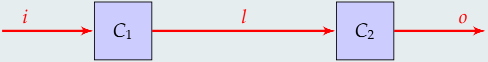
$$SD = \neg i \vee C_1 \vee l \quad \wedge \quad \neg l \vee C_2 \vee o$$

Observation: $i \wedge \neg o$

$$SD = C_1 \vee l \quad \wedge \quad \neg l \vee C_2$$

Oniscent Monitors

Example



$$
\begin{aligned}
SD &= && i \wedge \neg C_1 \rightarrow l & \qquad SD &= && \neg i \vee C_1 \vee l \\
& \wedge && l \wedge \neg C_2 \rightarrow o & & \wedge && \neg l \vee C_2 \vee o
\end{aligned}
$$

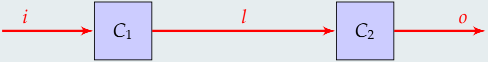
Observation: $i \wedge \neg o$

$$
\begin{aligned}
SD &= && C_1 \vee l \\
& \wedge && \neg l \vee C_2
\end{aligned}
$$

Diagnoses: C_2 or C_1

Oniscent Monitors

Example



$$\begin{aligned} SD &= & i \land \neg C_1 \to l \\ & \land & l \land \neg C_2 \to o \end{aligned} \qquad \begin{aligned} SD &= & \neg i \lor C_1 \lor l \\ & \land & \neg l \lor C_2 \lor o \end{aligned}$$

Observation: $i \land \neg o$

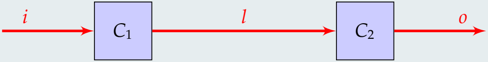
$$\begin{aligned} SD &= & C_1 \lor l \\ & \land & \neg l \lor C_2 \end{aligned}$$

Diagnoses: C_2 or C_1

If additionally l known to be correct, only C_2 diagnosed.

Oniscent Monitors

Example



$$\begin{aligned} SD \quad &= \quad i \wedge \neg C_1 \rightarrow l \\ &\wedge \quad l \wedge \neg C_2 \rightarrow o \end{aligned} \qquad \begin{aligned} SD \quad &= \quad \neg i \vee C_1 \vee l \\ &\wedge \quad \neg l \vee C_2 \vee o \end{aligned}$$

Observation: $i \wedge \neg o$

$$\begin{aligned} SD \quad &= \quad C_1 \vee l \\ &\wedge \quad \neg l \vee C_2 \end{aligned}$$

Diagnoses: C_2 or C_1

If additionally l known to be correct, only C_2 diagnosed.

$\rightsquigarrow$ notion of omniscent diagnoses

Presentation outline

- Runtime Verification
- Runtime Verification for LTL
  - LTL over Finite, Completed Words
  - LTL over Finite, Non-Completed Words: Impartiality
  - LTL over Non-Completed Words: Anticipation
  - LTL over Infinite Words: With Anticipation
  - Generalisations: LTL with modulo Constraints
  - Monitorable Properties
  - LTL with a Predictive Semantics
  - LTL wrap-up
- Extensions
- Monitoring Systems/Logging
- Steering
- Diagnosis
  - Ideas
  - RV and Diagnosis
- **Conclusion**

Conclusion

Summary

- RV for Failure detection
  - various, multi-valued approaches
  - various existing systems
  - does generally identifies failure detection and identification
- Diagonis for Failure identification?

Future work

What is the *right* combination?

That’s it!

Thanks! - Comments?

That's it!

Thanks! - Comments?