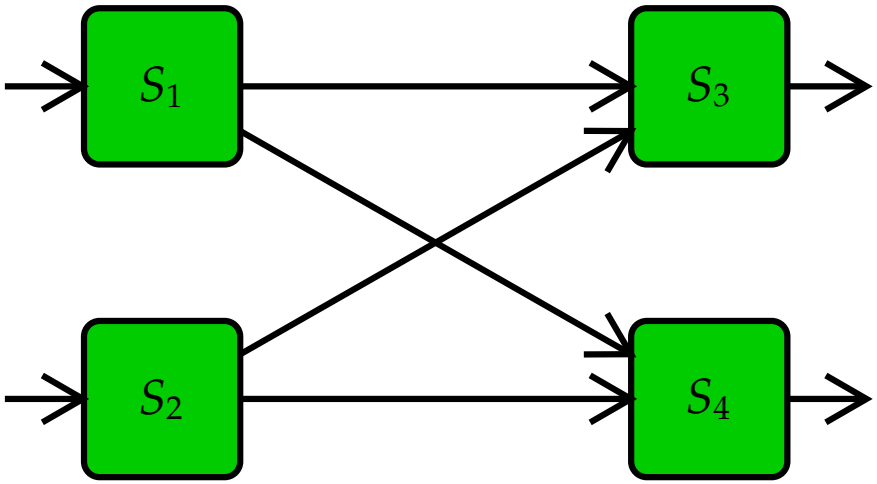
Tutorial on
Runtime Verification and Runtime Reflection

Martin Leucker

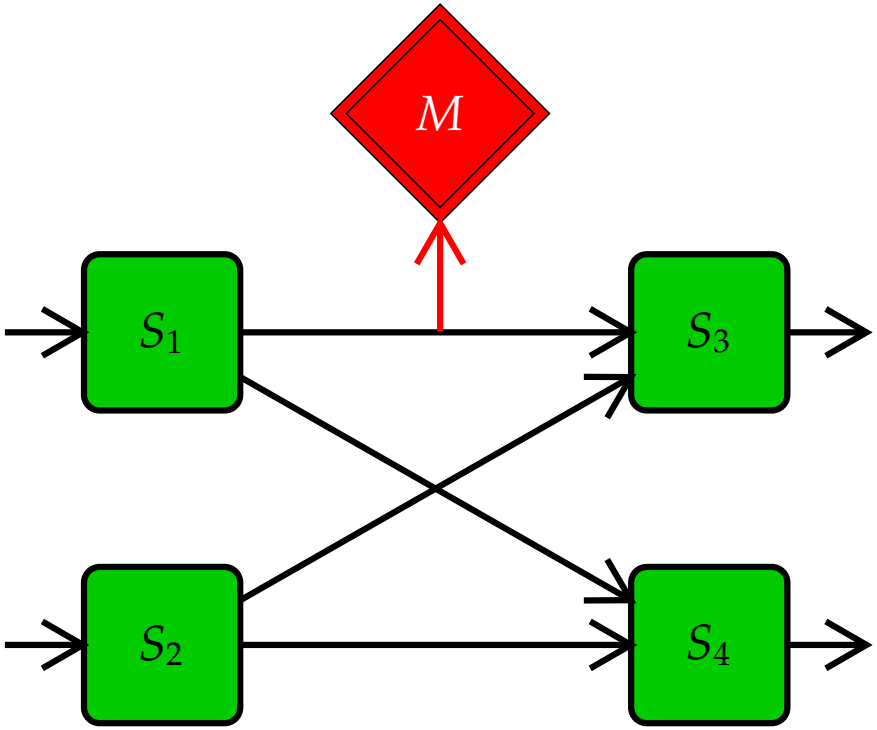
Institute for Software Engineering
Universität zu Lübeck

Dresden, Tuesday 3rd of July 2012

Runtime Verification (RV)

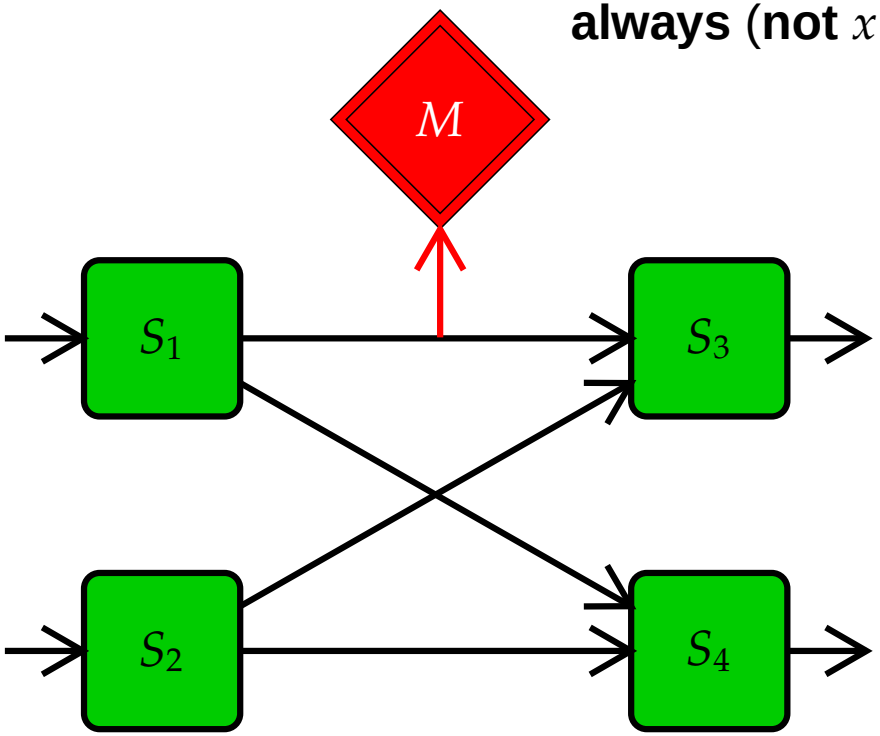


Runtime Verification (RV)



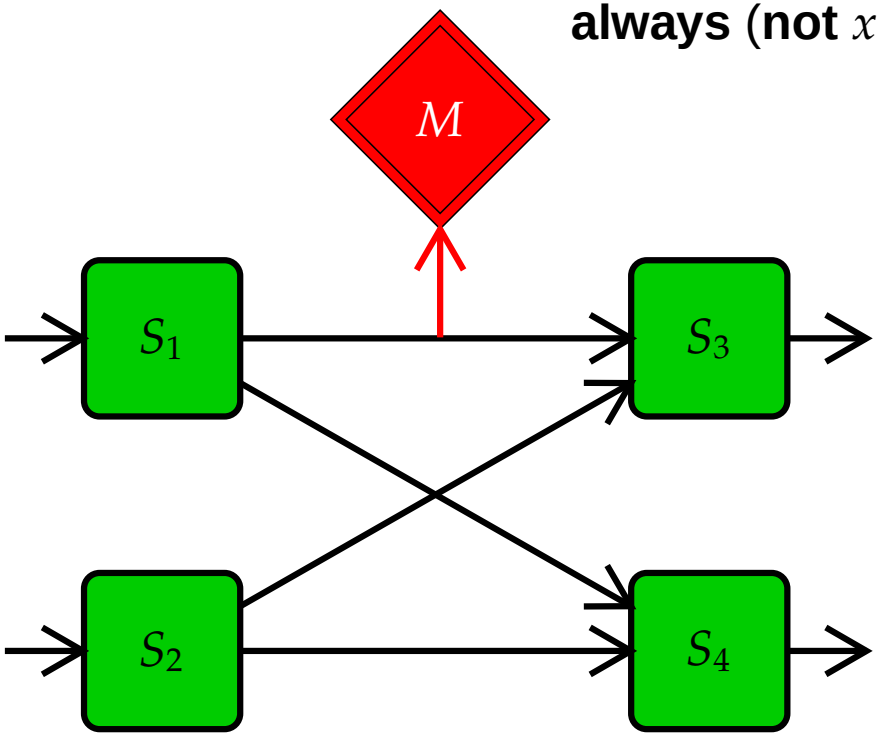
Runtime Verification (RV)

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



Runtime Verification (RV)

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

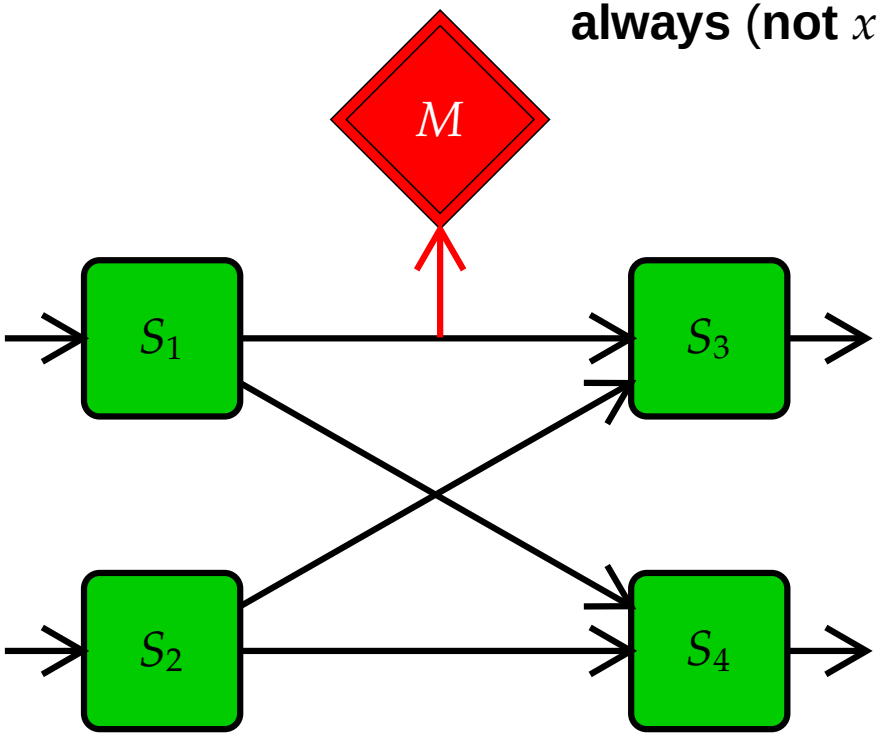


Characterisation

- Verifies (partially) correctness properties based on actual executions

Runtime Verification (RV)

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

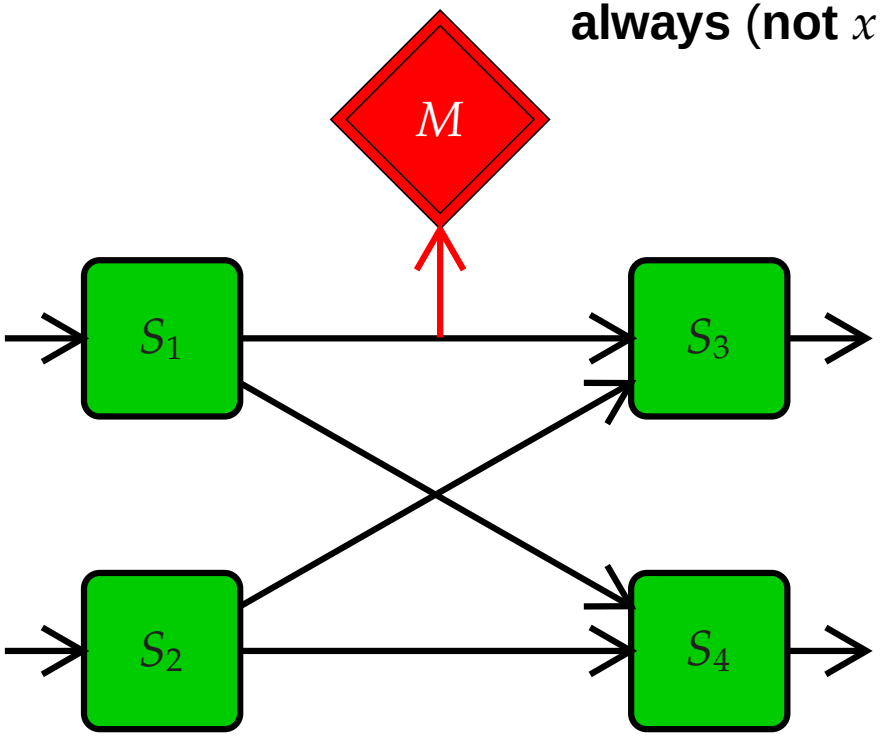


Characterisation

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

Runtime Verification (RV)

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

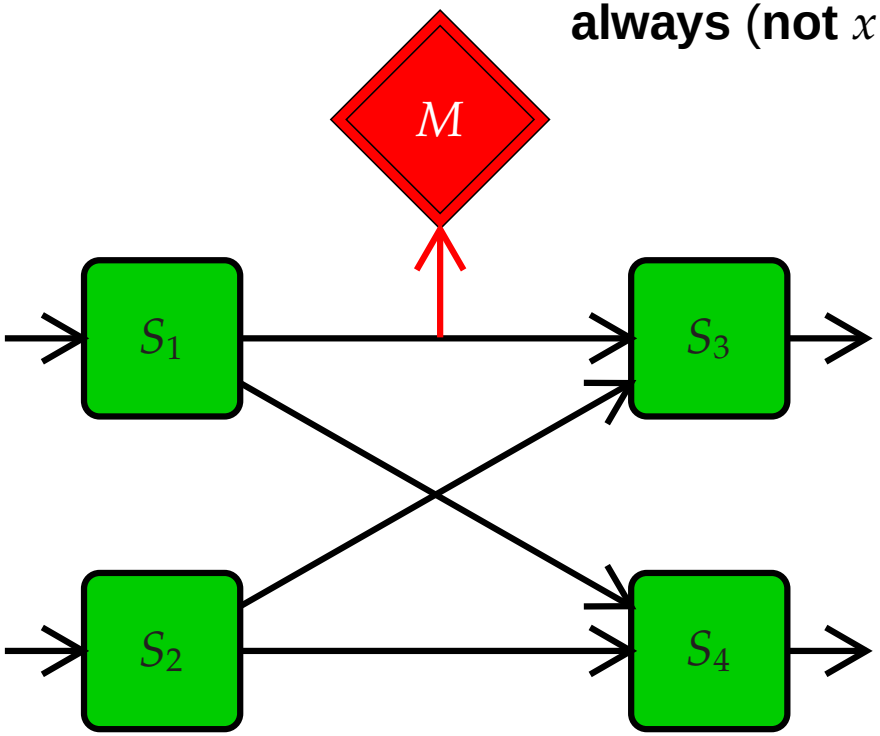


Characterisation

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

Runtime Verification (RV)

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

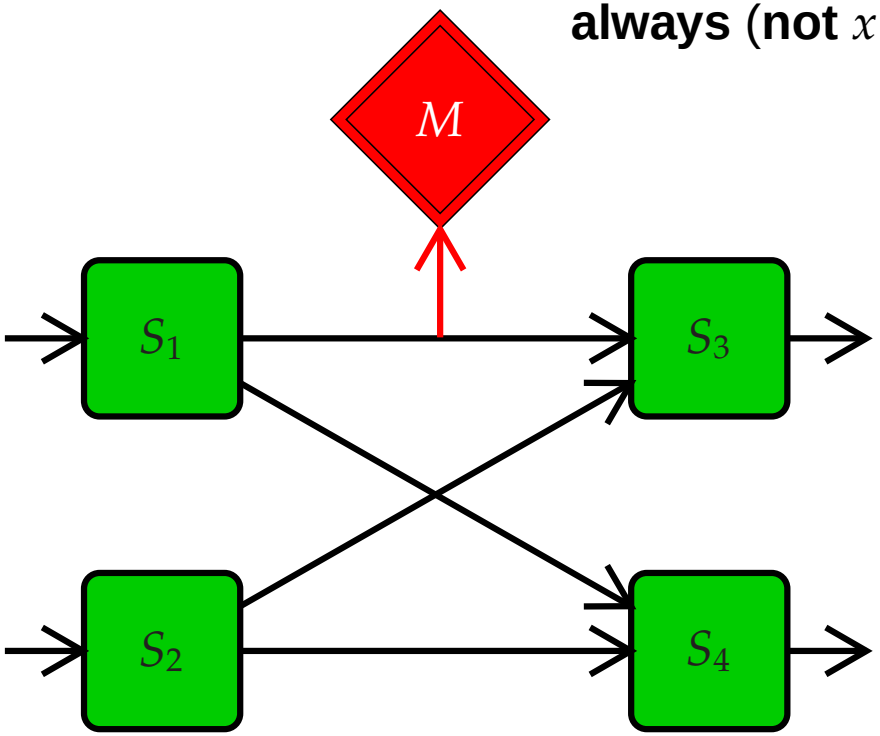


Characterisation

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

Runtime Verification (RV)

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

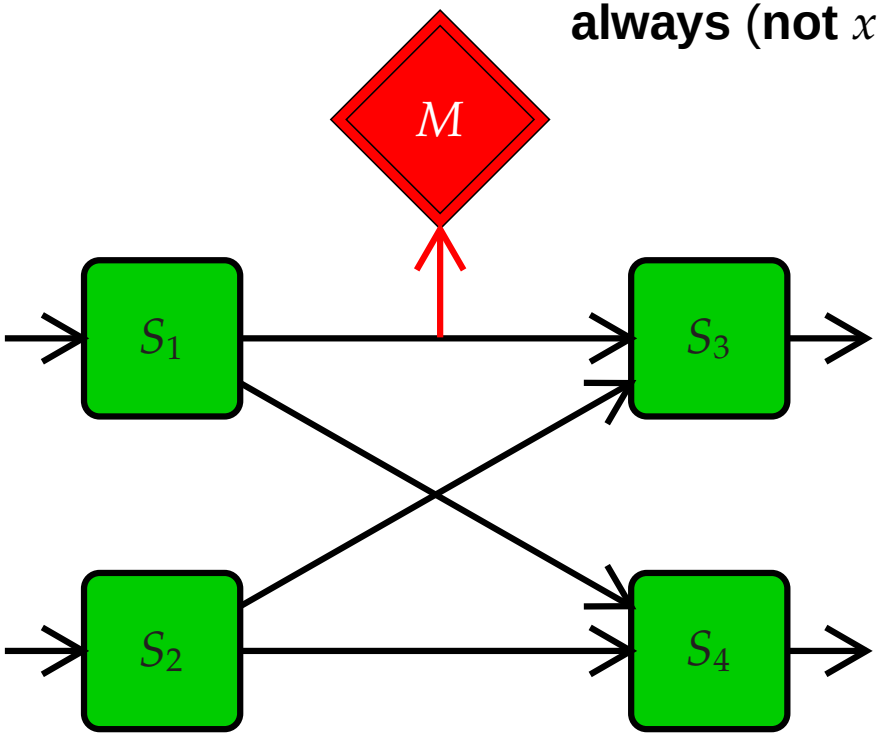


Characterisation

- 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

Presentation outline

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 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)

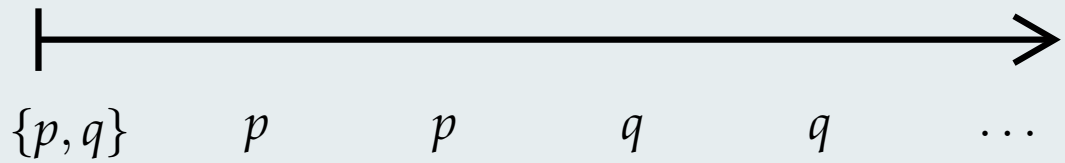
Let p be an atomic proposition from a finite set of atomic propositions AP. The set of LTL formulae, denoted with LTL, is inductively defined by the following grammar:

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

Linear-time Temporal Logic (LTL)

Semantics

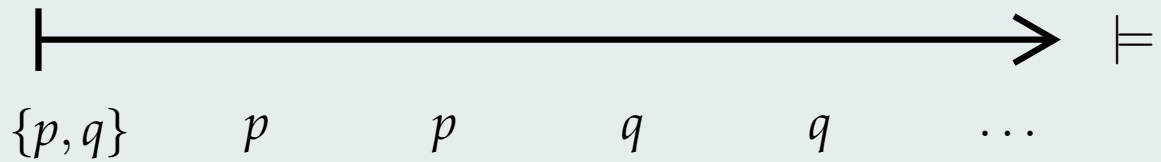
over $w \in (2^{AP})^\omega = \Sigma^\omega$



Linear-time Temporal Logic (LTL)

Semantics

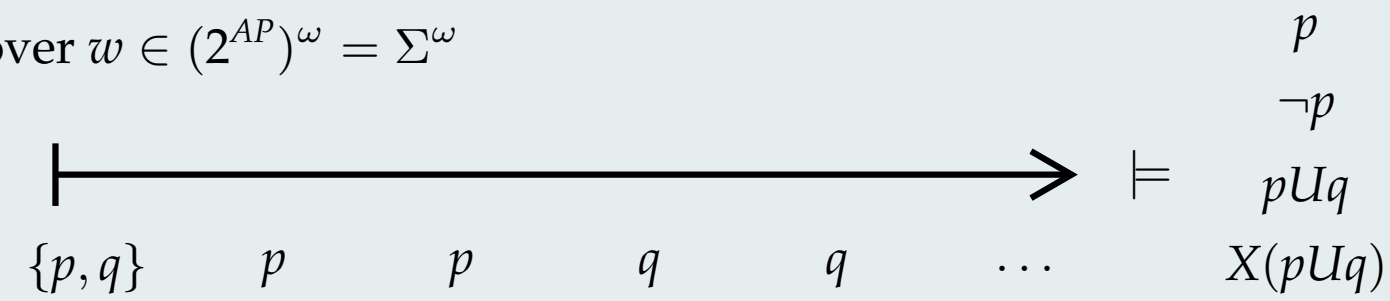
over $w \in (2^{AP})^\omega = \Sigma^\omega$



Linear-time Temporal Logic (LTL)

Semantics

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



Linear-time Temporal Logic (LTL)

Semantics

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

$\{p,q\}$ 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 $w \in (2^{AP})^\omega = \Sigma^\omega$

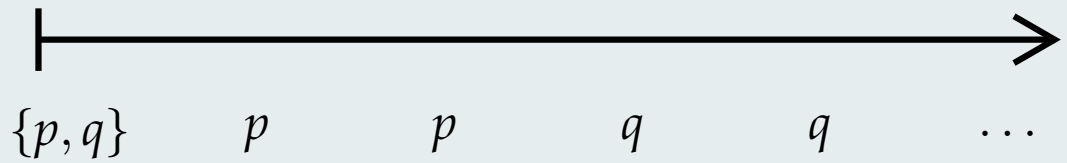
$\models$

p	√
$\neg p$	X
pUq	√
$X(pUq)$	√

Linear-time Temporal Logic (LTL)

Semantics

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



$\models$

p	√
$\neg p$	X
pUq	√
$X(pUq)$	√

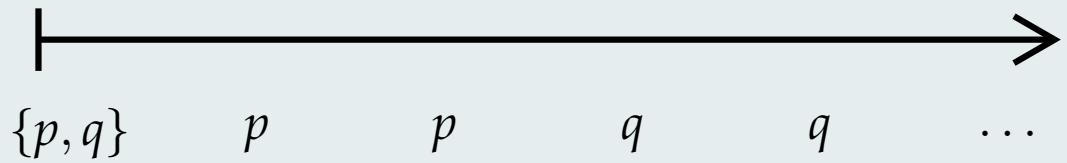
Abbreviation

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

Linear-time Temporal Logic (LTL)

Semantics

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



$\models$

p	√
$\neg p$	X
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 for the working engineer??

Simple??

"LTL is for theoreticians—but for practitioners?"

LTL for the working engineer??

Simple??

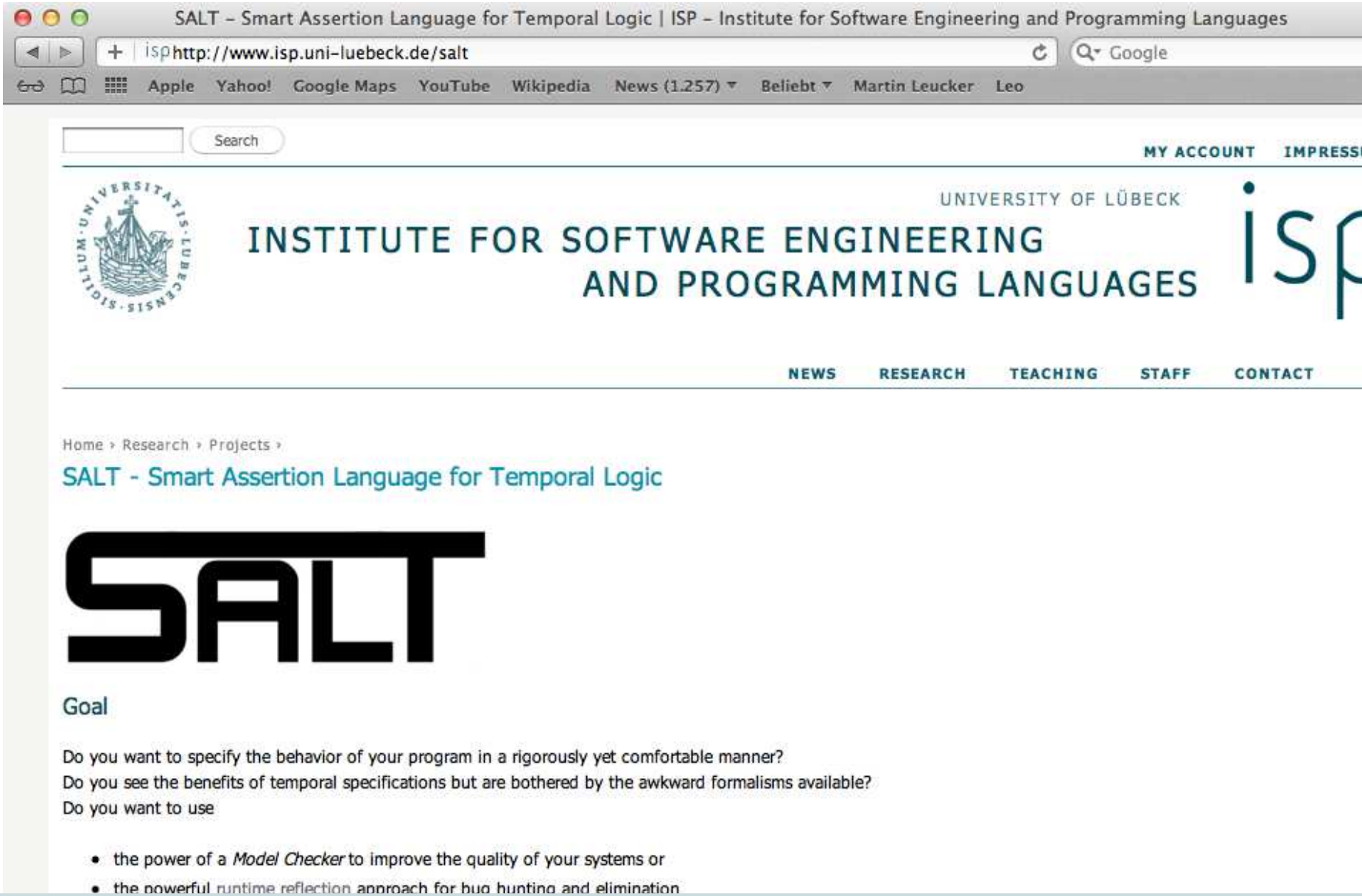
"LTL is for theoreticians—but for practitioners?"

SALT

Structured Assertion Language for Temporal Logic

"Syntactic Sugar for LTL" [Bauer, L., Streit@ICFEM'06]

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



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:

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

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 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

$$[w \models \mathit{true}]_{\mathfrak{L}} = \top \qquad [w \models \mathit{false}]_{\mathfrak{L}} = \bot$$

Boolean combinations

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

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

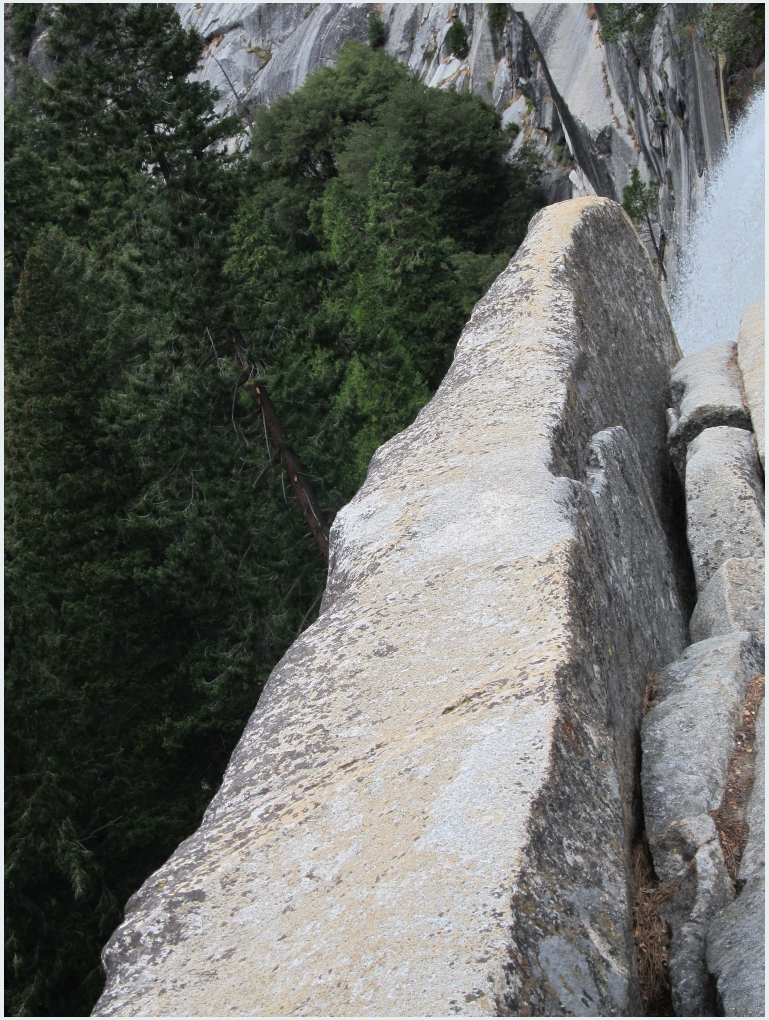
$$[w \models \varphi \; U \; \psi]_{\mathfrak{L}} = \begin{cases} \top & \text{there is a } k,\ 0 \le k < |w| : [w^k \models \psi]_{\mathfrak{L}} = \top \text{ and} \\ & \text{for all } l \text{ with } 0 \le l < k : [w^l \models \varphi] = \top \\ \mathit{TBD} & \text{else} \end{cases}$$

$$[w \models \varphi \; R \; \psi]_{\mathfrak{L}} \equiv \neg(\neg\varphi \; U \; \neg\psi)$$

Outline

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

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

weak next

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

Monitoring LTL on finite words

(Bad) Idea

just compute semantics. . .

Outline

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

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

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 = (⊤,⊤)
evalFLTL4 false   a = (⊥,⊥)
evalFLTL4 p       a = ((p in a),(p in a))
evalFLTL4 ¬φ      a = let (valPhi,phiRew) = evalFLTL4 φ a
                      in (¬valPhi,¬phiRew)
evalFLTL4 φ ∨ ψ   a = let
                        (valPhi,phiRew) = evalFLTL4 φ a
                        (valPsi,psiRew) = evalFLTL4 ψ a
                      in (valPhi ⊔ valPsi,phiRew ∨ psiRew)
evalFLTL4 φ ∧ ψ   a = let
                        (valPhi,phiRew) = evalFLTL4 φ a
                        (valPsi,psiRew) = evalFLTL4 ψ a
                      in (valPhi ⊓ valPsi,phiRew ∧ psiRew)
evalFLTL4 φ U ψ   a = evalFLTL4 ψ ∨ (φ ∧ X(φ U ψ)) a
evalFLTL4 φ R ψ   a = evalFLTL4 ψ ∧ (φ ∨ X̄(φ R ψ)) a
evalFLTL4 Xφ      a = (⊥^p,φ)
evalFLTL4 X̄φ      a = (⊤^p,φ)
```

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 = (⊤,⊤)
evalFLTL4 false   a = (⊥,⊥)
evalFLTL4 p       a = ((p in a),(p in a))
evalFLTL4 ¬φ      a = let (valPhi,phiRew) = evalFLTL4 φ a
                      in (¬valPhi,¬phiRew)
evalFLTL4 φ ∨ ψ   a = let
                        (valPhi,phiRew) = evalFLTL4 φ a
                        (valPsi,psiRew) = evalFLTL4 ψ a
                      in (valPhi ⊔ valPsi,phiRew ∨ psiRew)
evalFLTL4 φ ∧ ψ   a = let
                        (valPhi,phiRew) = evalFLTL4 φ a
                        (valPsi,psiRew) = evalFLTL4 ψ a
                      in (valPhi ⊓ valPsi,phiRew ∧ psiRew)
evalFLTL4 φ U ψ   a = evalFLTL4 ψ ∨ (φ ∧ X(φ U ψ)) a
evalFLTL4 φ R ψ   a = evalFLTL4 ψ ∧ (φ ∨ X̄(φ R ψ)) a
evalFLTL4 Xφ      a = (⊥^p,φ)
evalFLTL4 X̄φ      a = (⊤^p,φ)
```

Automata-theoretic approach

Further developments possible

- alternating Mealy machines

Automata-theoretic approach

Further developments possible

- alternating Mealy machines
- Moore machines

Automata-theoretic approach

Further developments possible

- alternating Mealy machines
- Moore machines
- alternating machines

Automata-theoretic approach

Further developments possible

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

Automata-theoretic approach

Further developments possible

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

Automata-theoretic approach

Further developments possible

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

Outline

Anticipatory Semantics

Consider possible extensions of the non-completed word

Outline

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 $XXXfalse$

$$\epsilon \quad \models \quad XXXfalse$$

Impartial Anticipation

Impartial

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

Anticipatory

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

$$\epsilon \models XXXfalse$$
$$a \models XXfalse$$

Impartial Anticipation

Impartial

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

Anticipatory

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

$$\begin{aligned} \epsilon &\models XXXfalse \\ a &\models XXfalse \\ aa &\models Xfalse \end{aligned}$$

Impartial Anticipation

Impartial

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

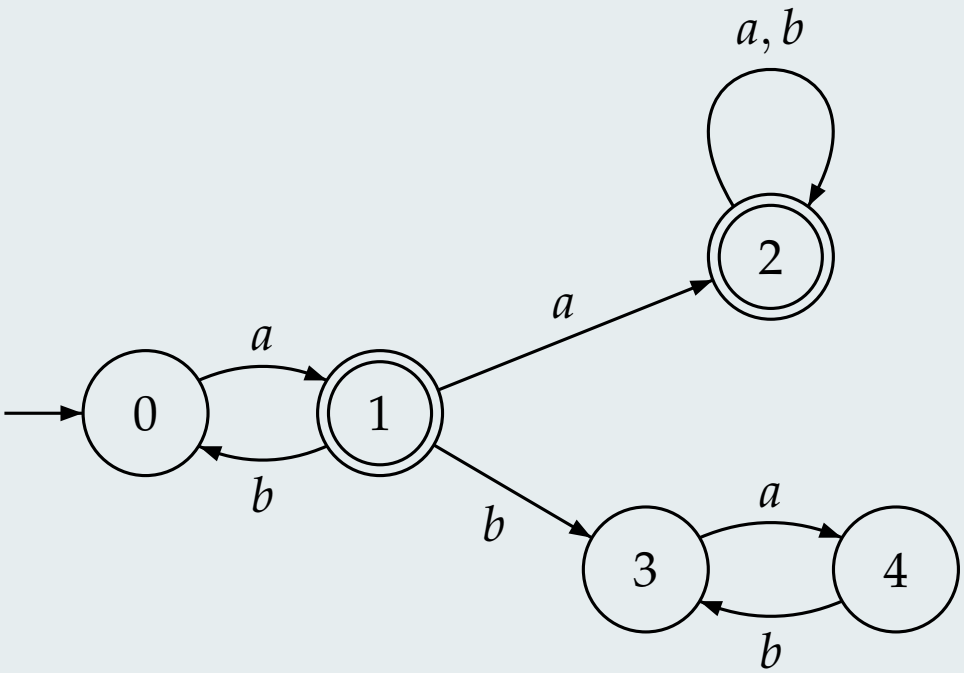
Anticipatory

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

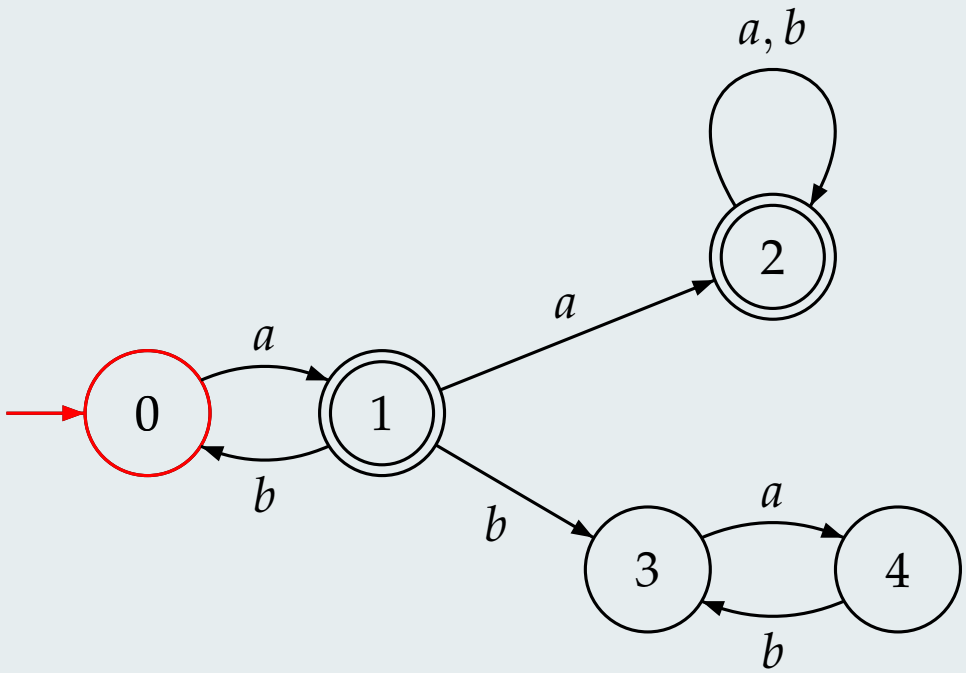
$$\begin{aligned} \epsilon &\models XXXfalse \\ a &\models XXfalse \\ aa &\models Xfalse \\ aaa &\models false \end{aligned}$$

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

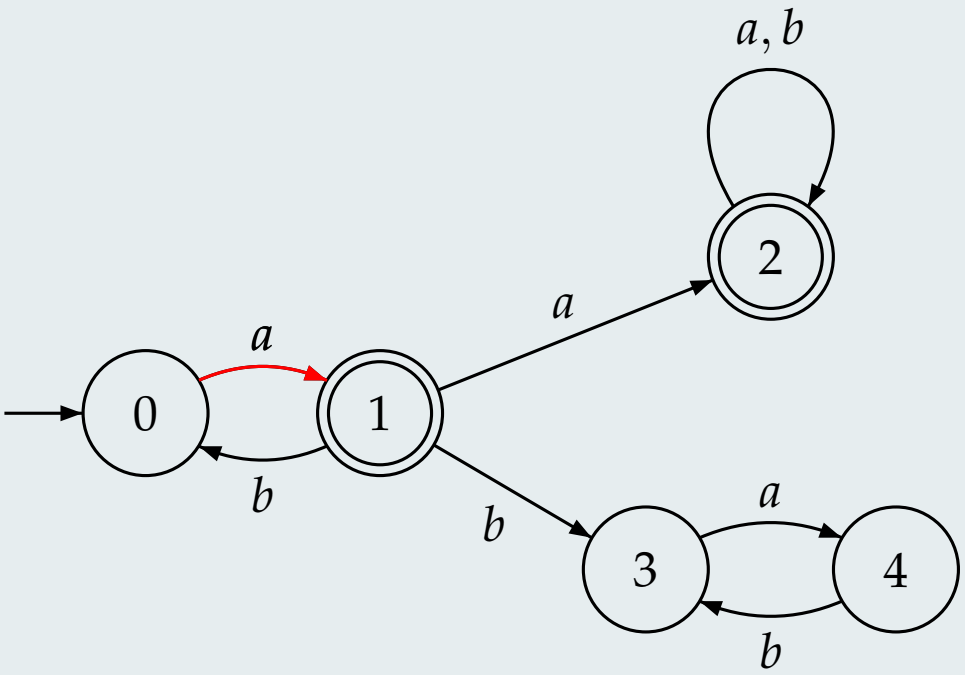
Büchi automata (BA)



Büchi automata (BA)

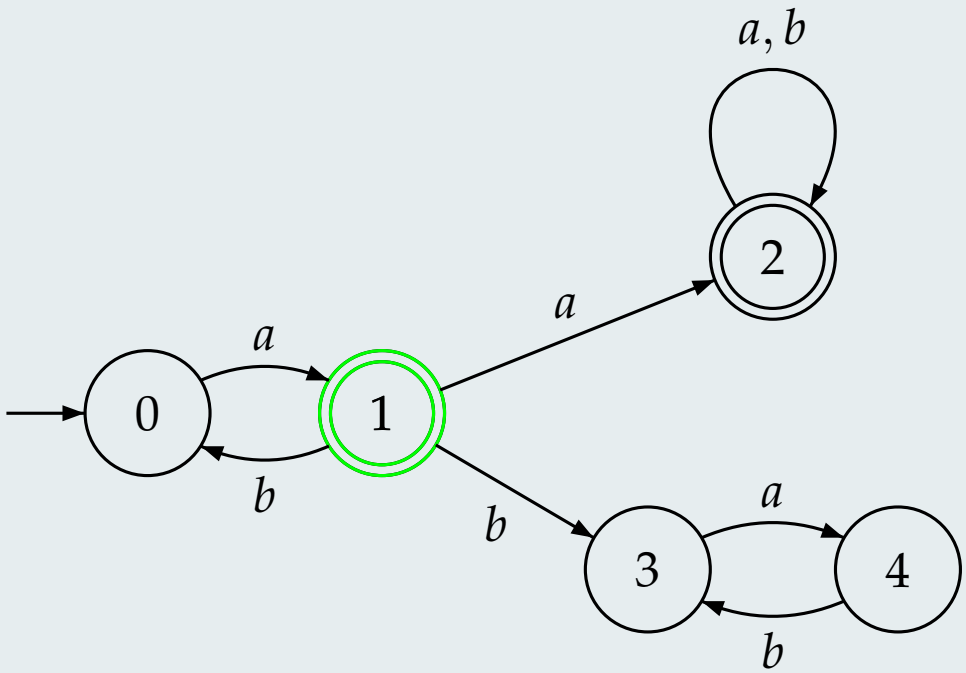


Büchi automata (BA)



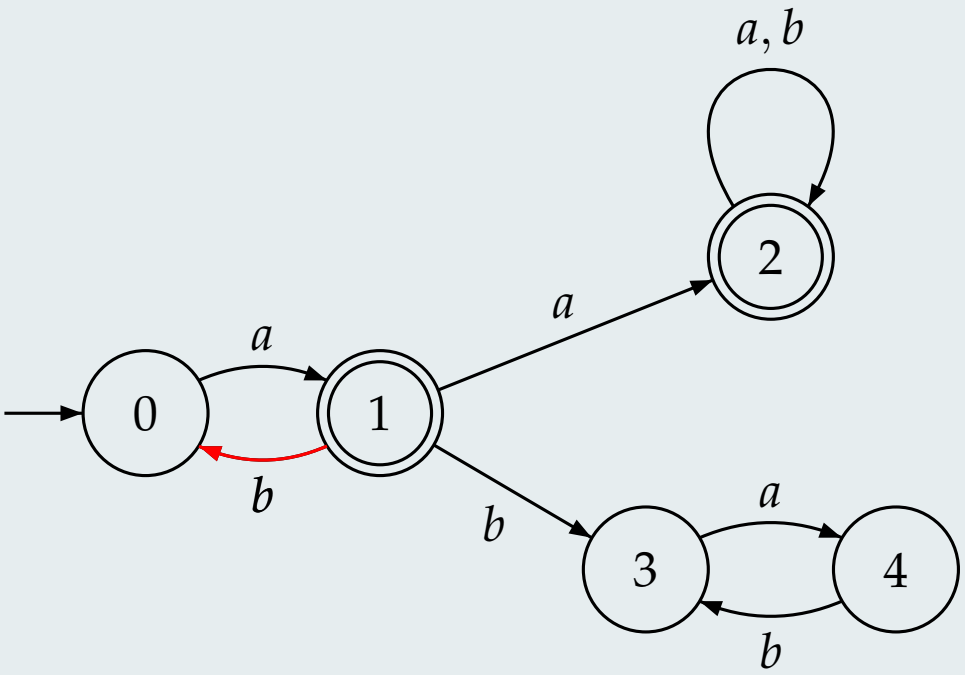
a

Büchi automata (BA)



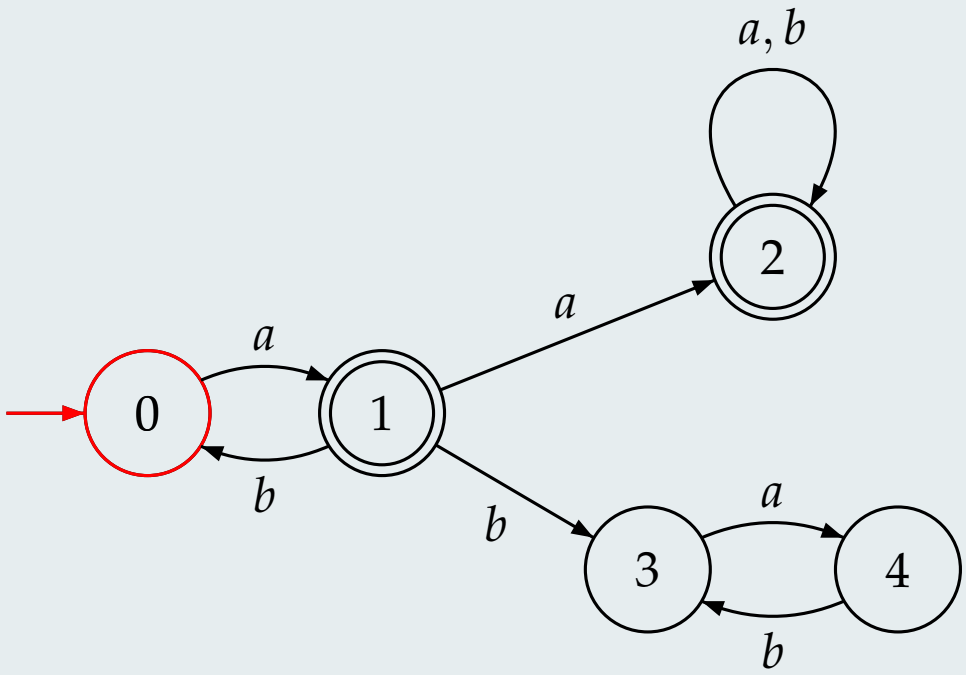
a

Büchi automata (BA)



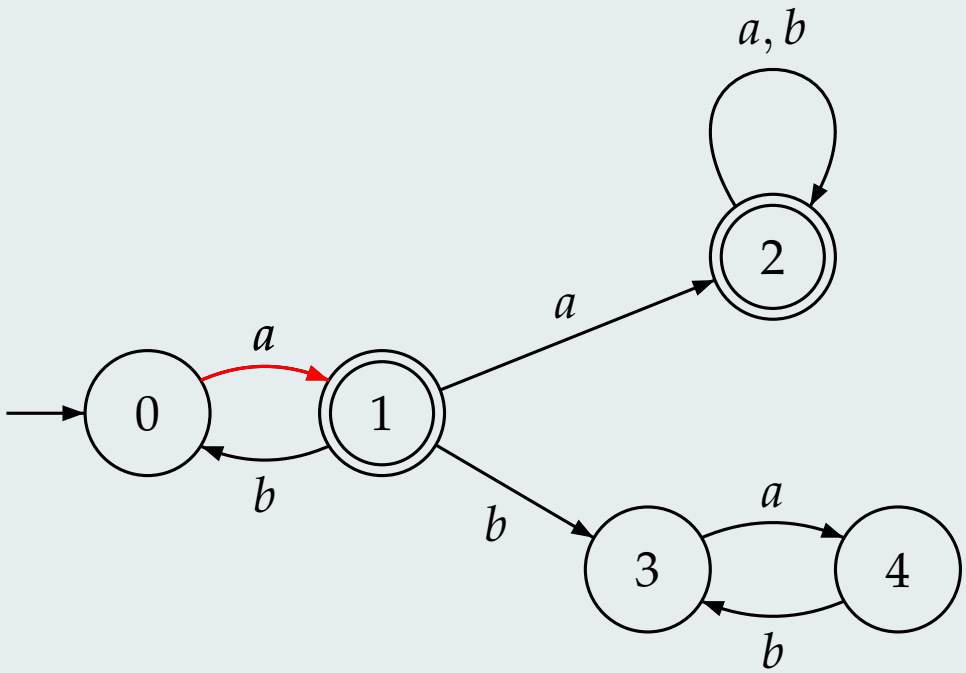
a b

Büchi automata (BA)



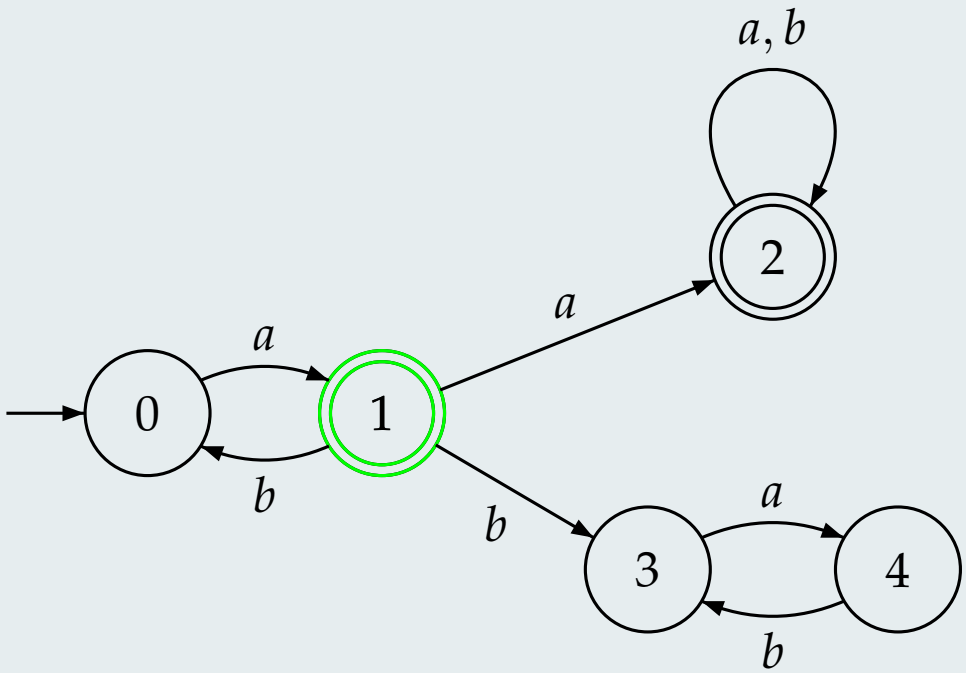
$a\ b$

Büchi automata (BA)



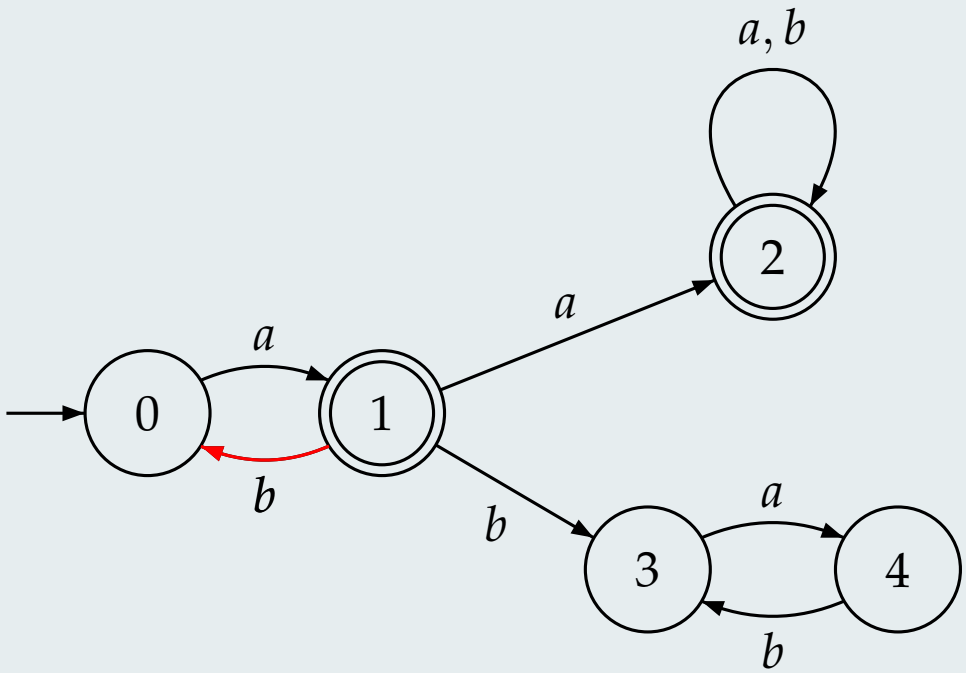
a b a

Büchi automata (BA)



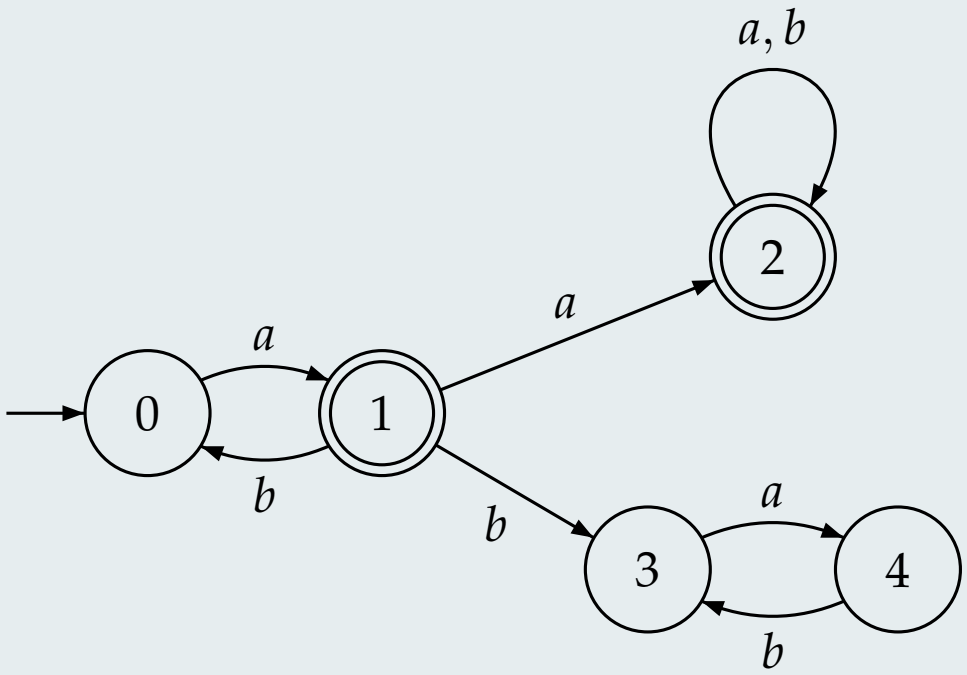
a b a

Büchi automata (BA)



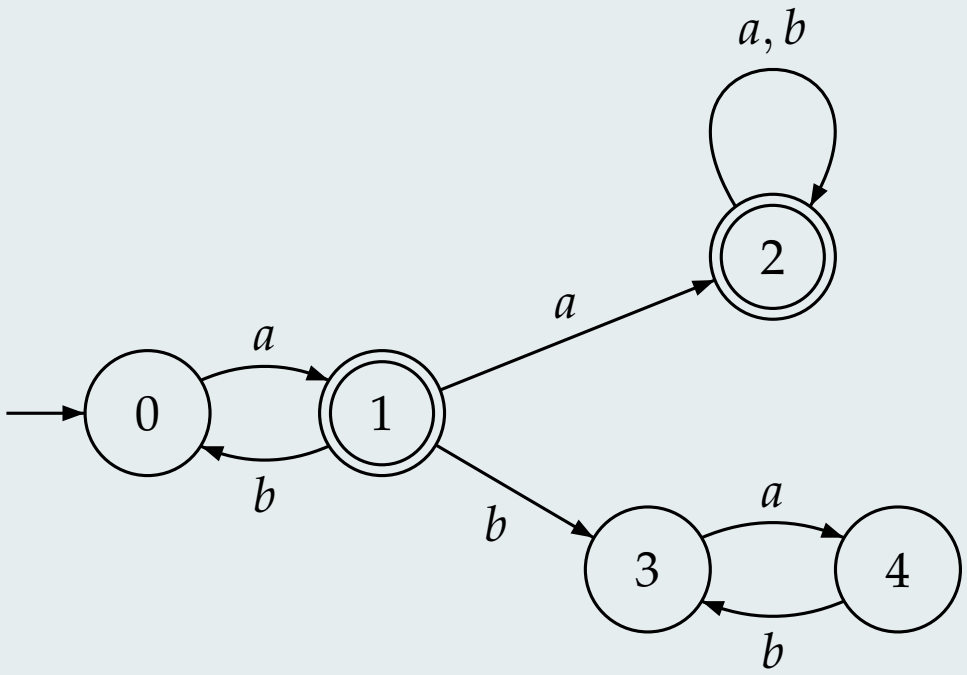
a b a b

Büchi automata (BA)



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

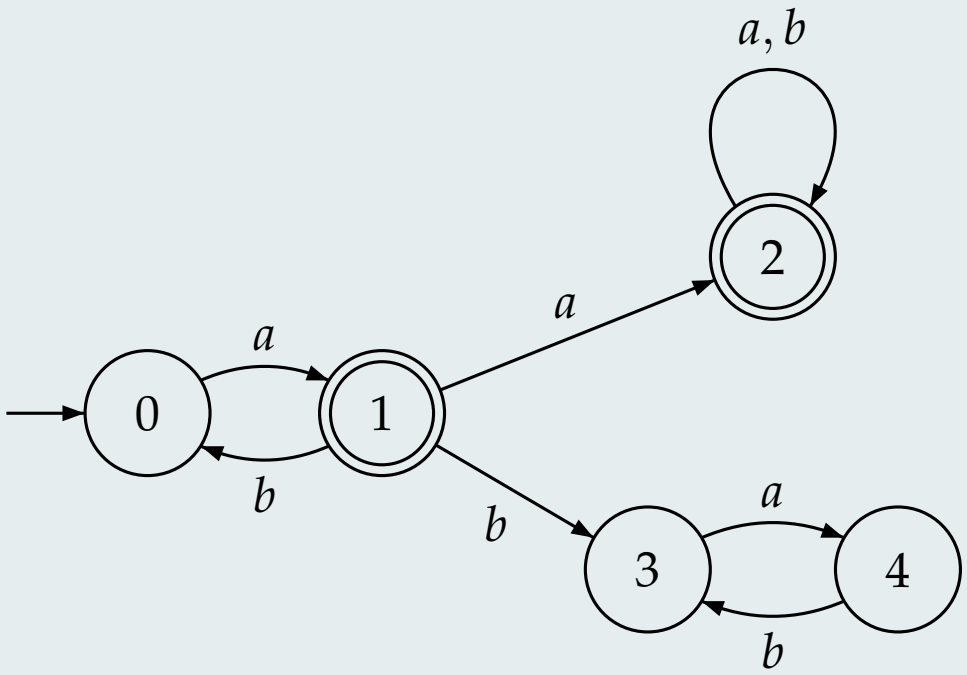
Büchi automata (BA)



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

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

Büchi automata (BA)



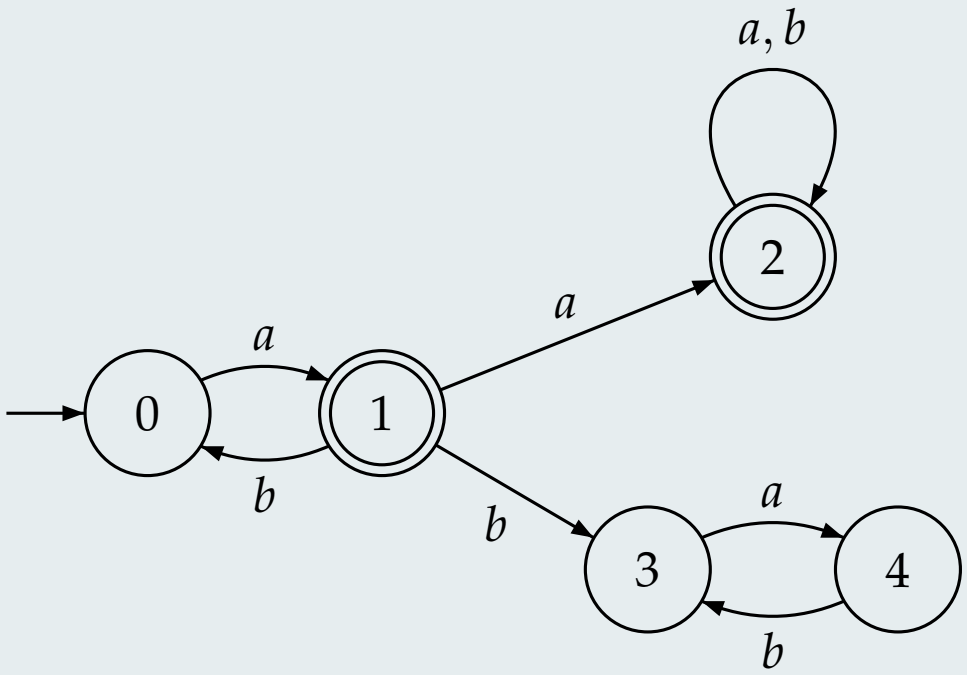
$a\ b\ a\ b\ \ldots$

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

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

Büchi automata (BA)

Emptiness test:



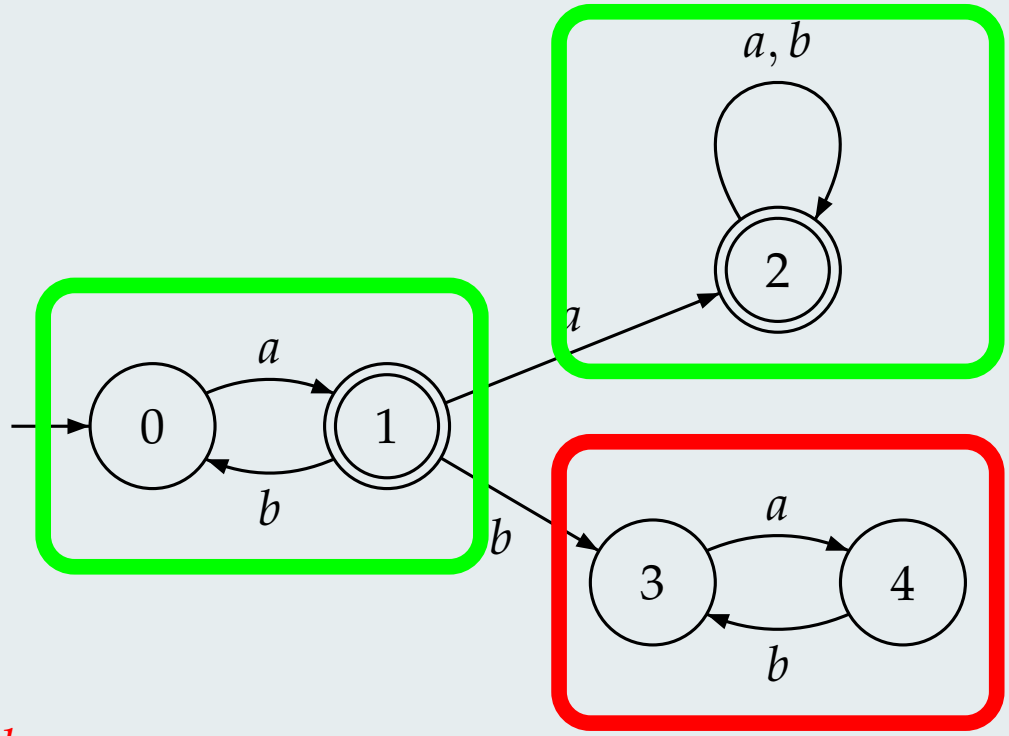
$a\ b\ a\ b\ \ldots$

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

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

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]

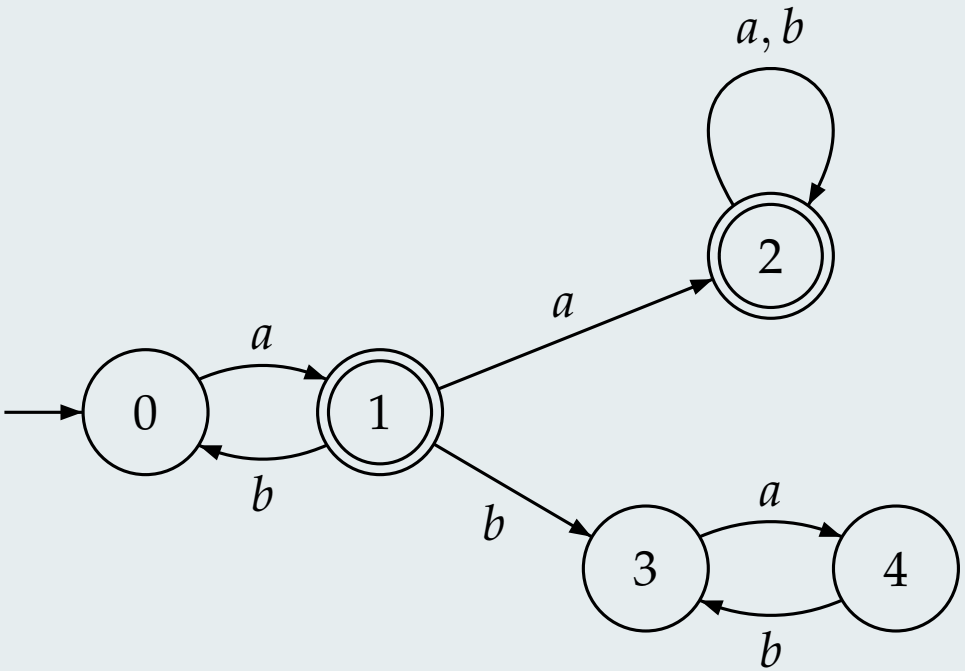
- Translation of an LTL formula φ into Büchi automata $\mathcal{A}_\varphi$ with

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

- Complexity: Exponential in the length of φ

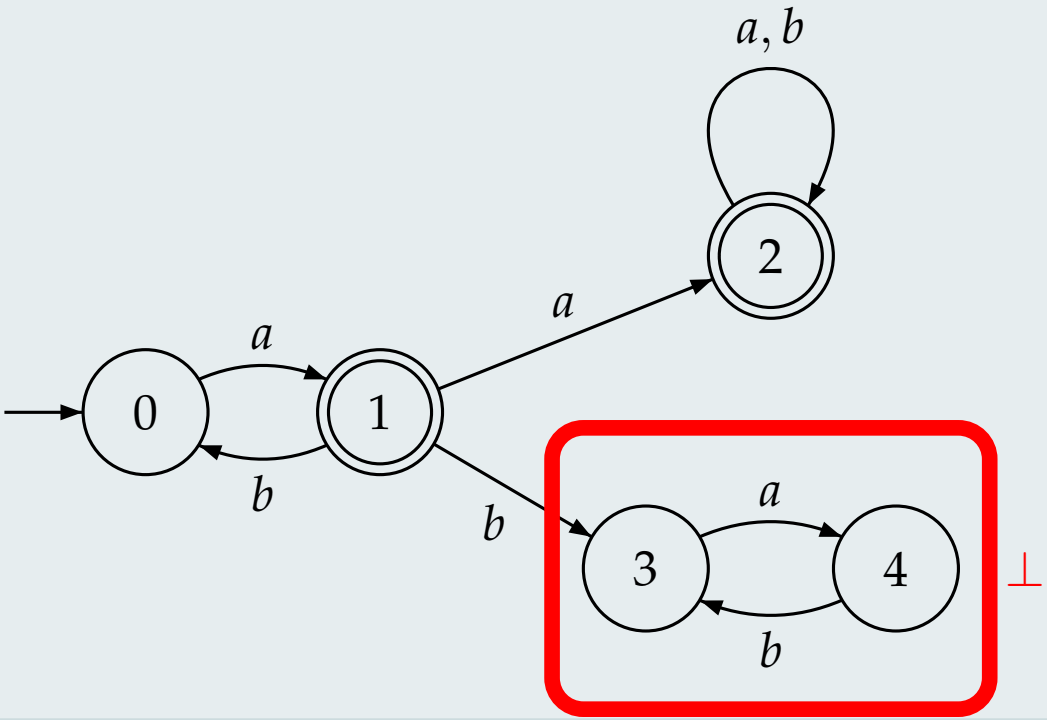
Monitor construction – Idea I

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



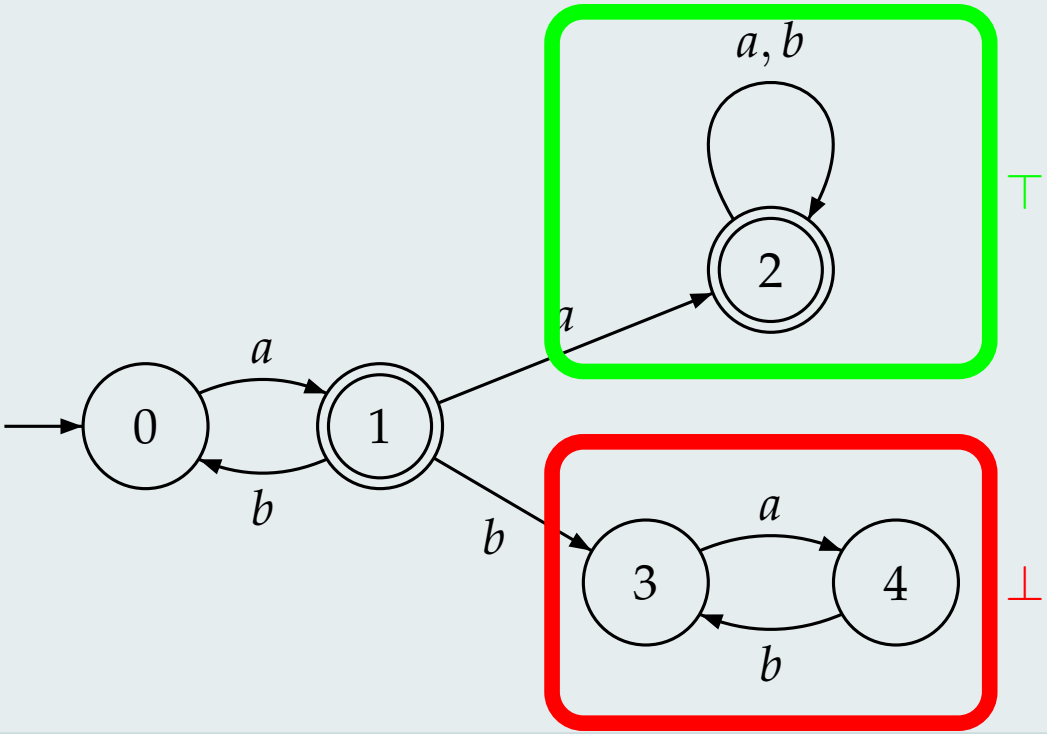
Monitor construction – Idea I

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



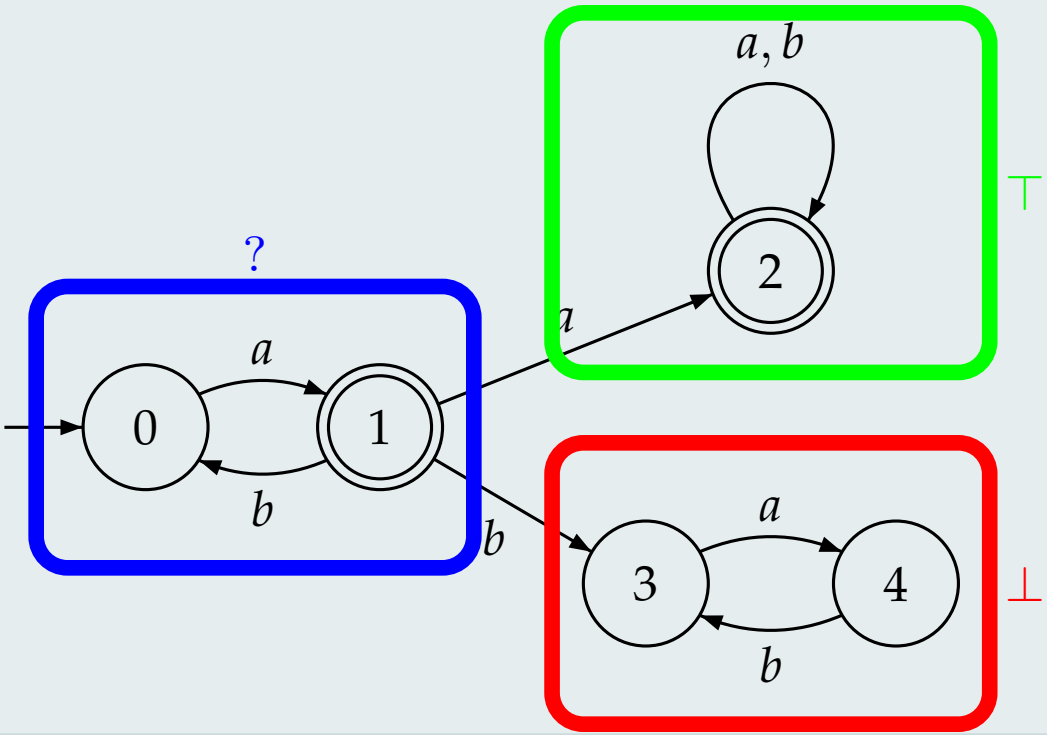
Monitor construction – Idea I

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

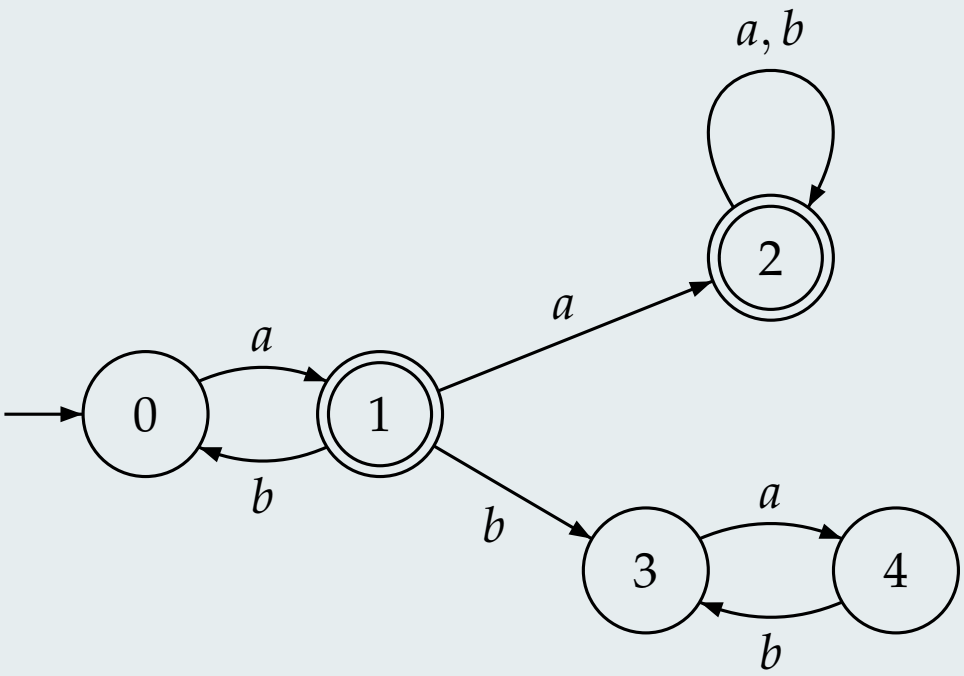


Monitor construction – Idea I

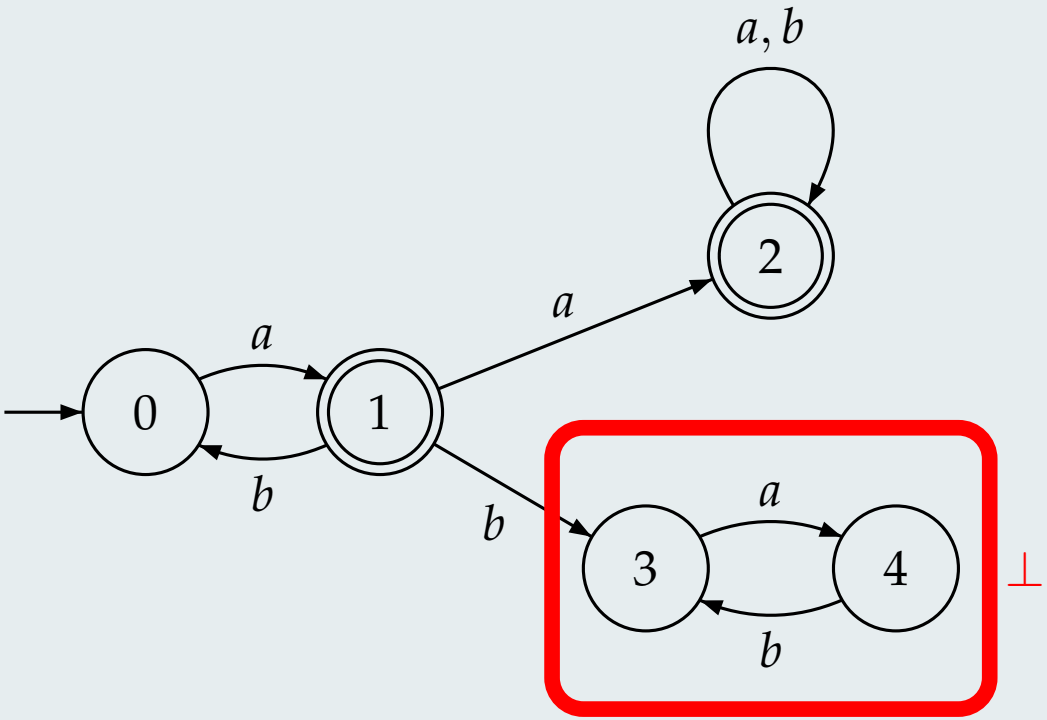
$$[u \models \varphi] = \begin{cases} \top & \text{if } \forall \sigma \in \Sigma^\omega : u\sigma \models \varphi \\ \bot & \text{if } \forall \sigma \in \Sigma^\omega : u\sigma \not\models \varphi \\ ? & \text{else} \end{cases}$$



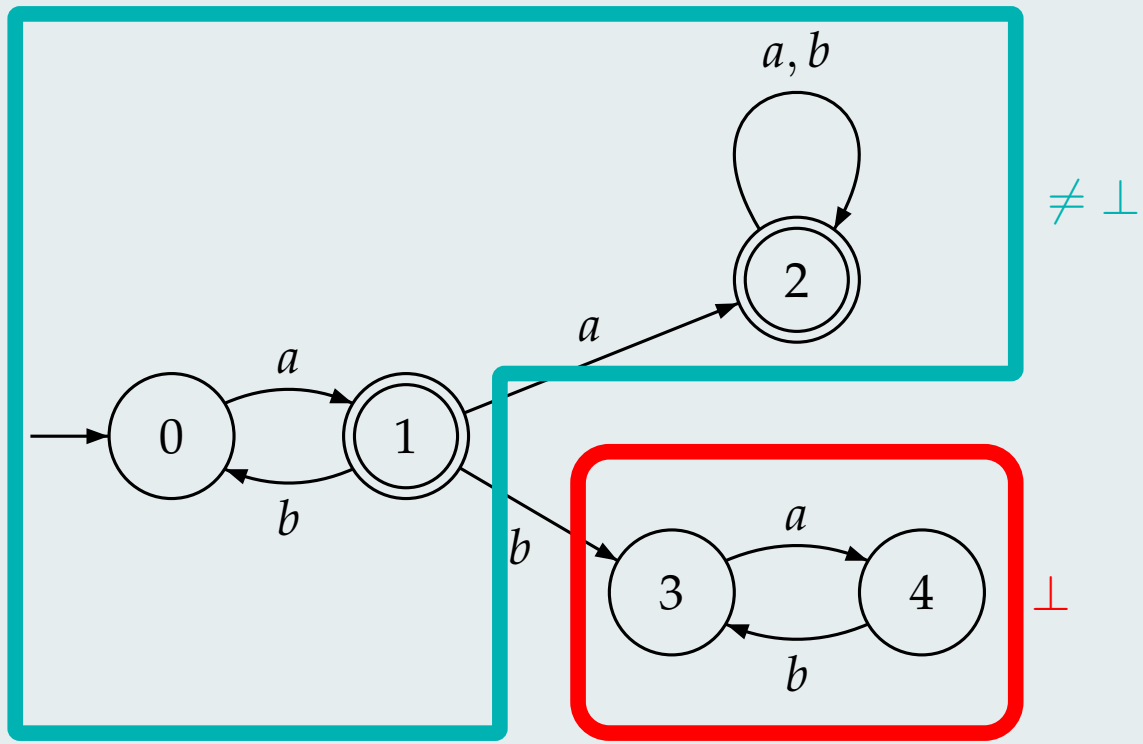
monitor construction – Idea II



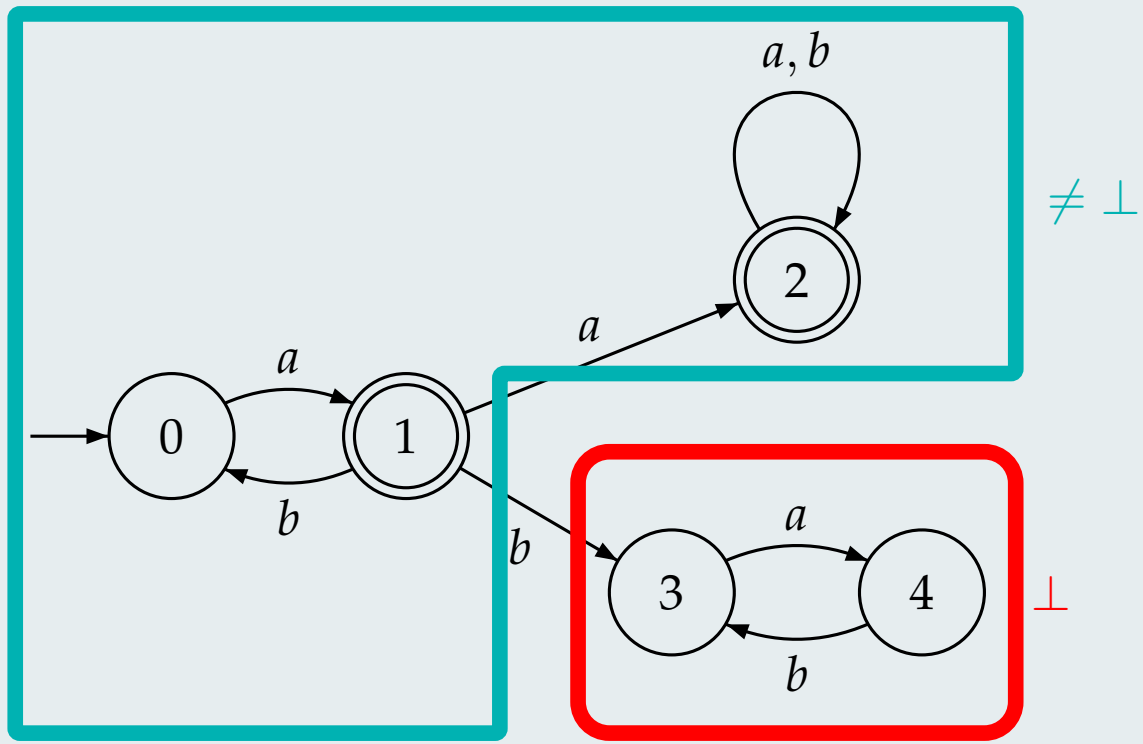
monitor construction – Idea II



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

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

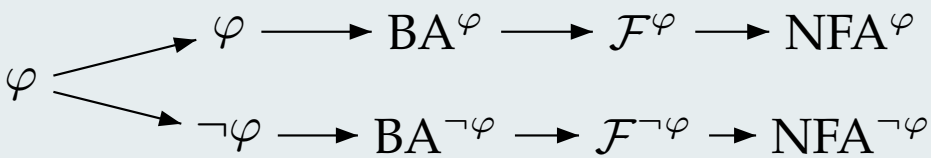
$$\neg\varphi \longrightarrow \mathrm{BA}^{\neg\varphi} \longrightarrow \mathcal{F}^{\neg\varphi} \longrightarrow \mathrm{NFA}^{\neg\varphi}$$

Lemma

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

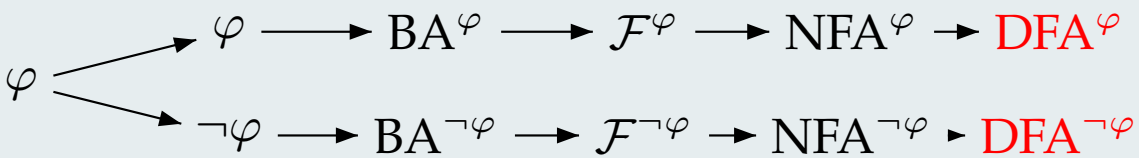
The complete construction

The construction



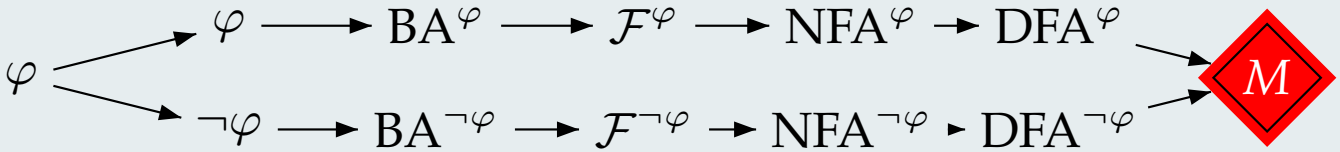
The complete construction

The construction



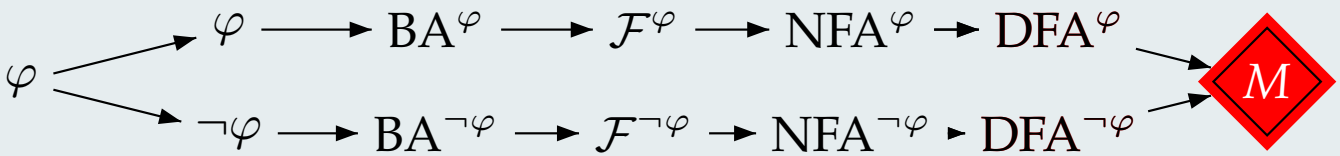
The complete construction

The construction



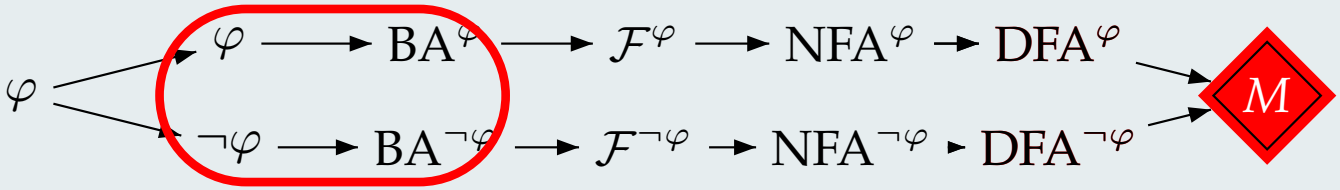
Complexity

The construction



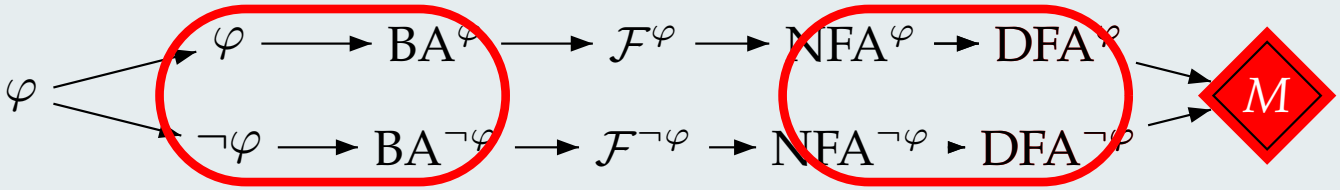
Complexity

The construction



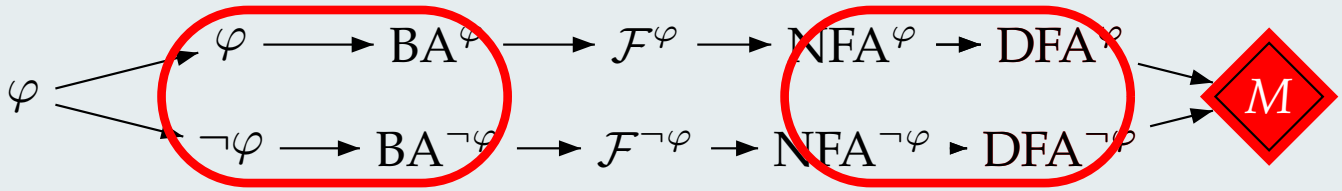
Complexity

The construction



Complexity

The construction

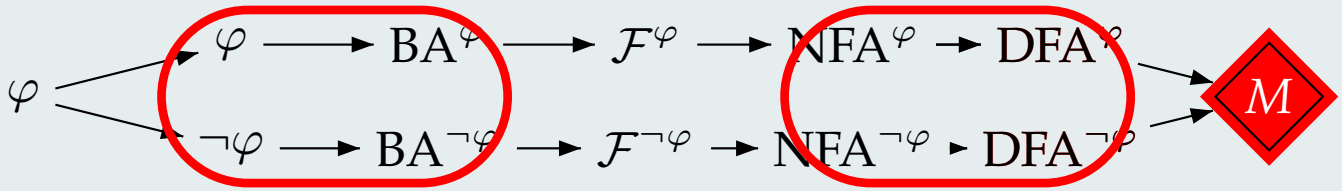


Complexity

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

Complexity

The construction



Complexity

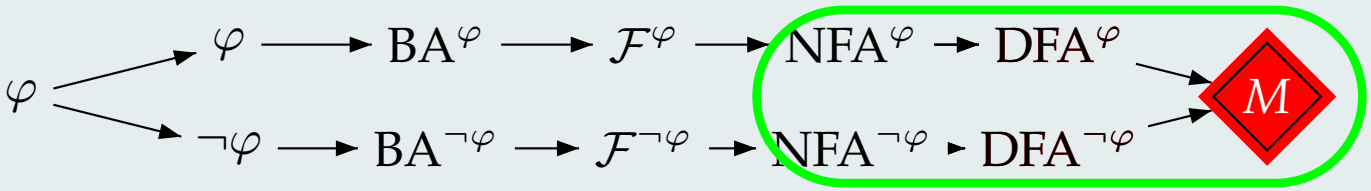
$$|M| \leq 2^{2^{|\varphi|}}$$

Optimal result!

FSM can be minimised (Myhill-Nerode)

On-the-fly Construction

The construction



Outline

Intermediate Summary

Semantics

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

Monitors

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

Presentation outline

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

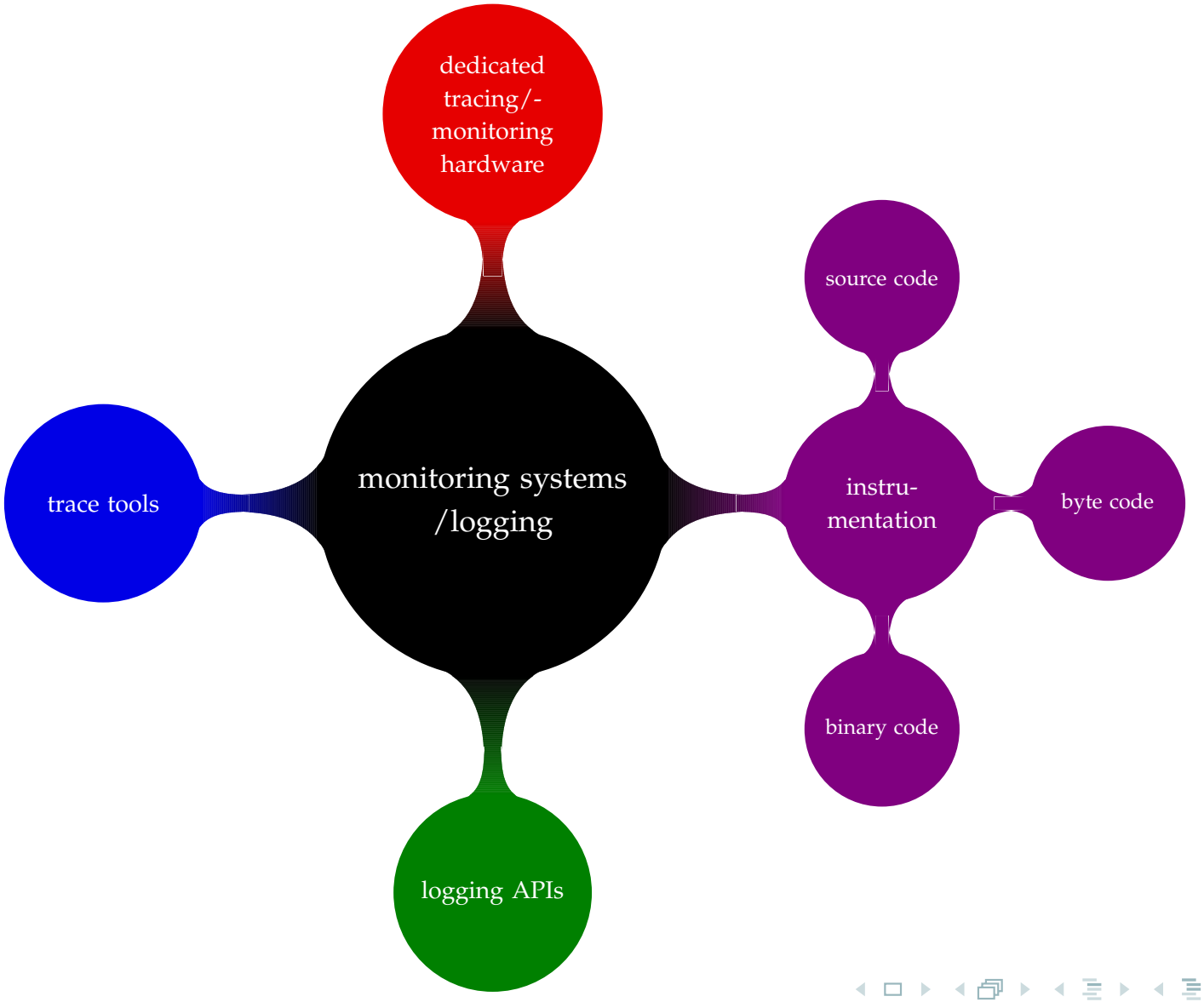
- LOLA
- Eagle (etc.)

Further dimensions

- real-time
- concurrency
- distribution

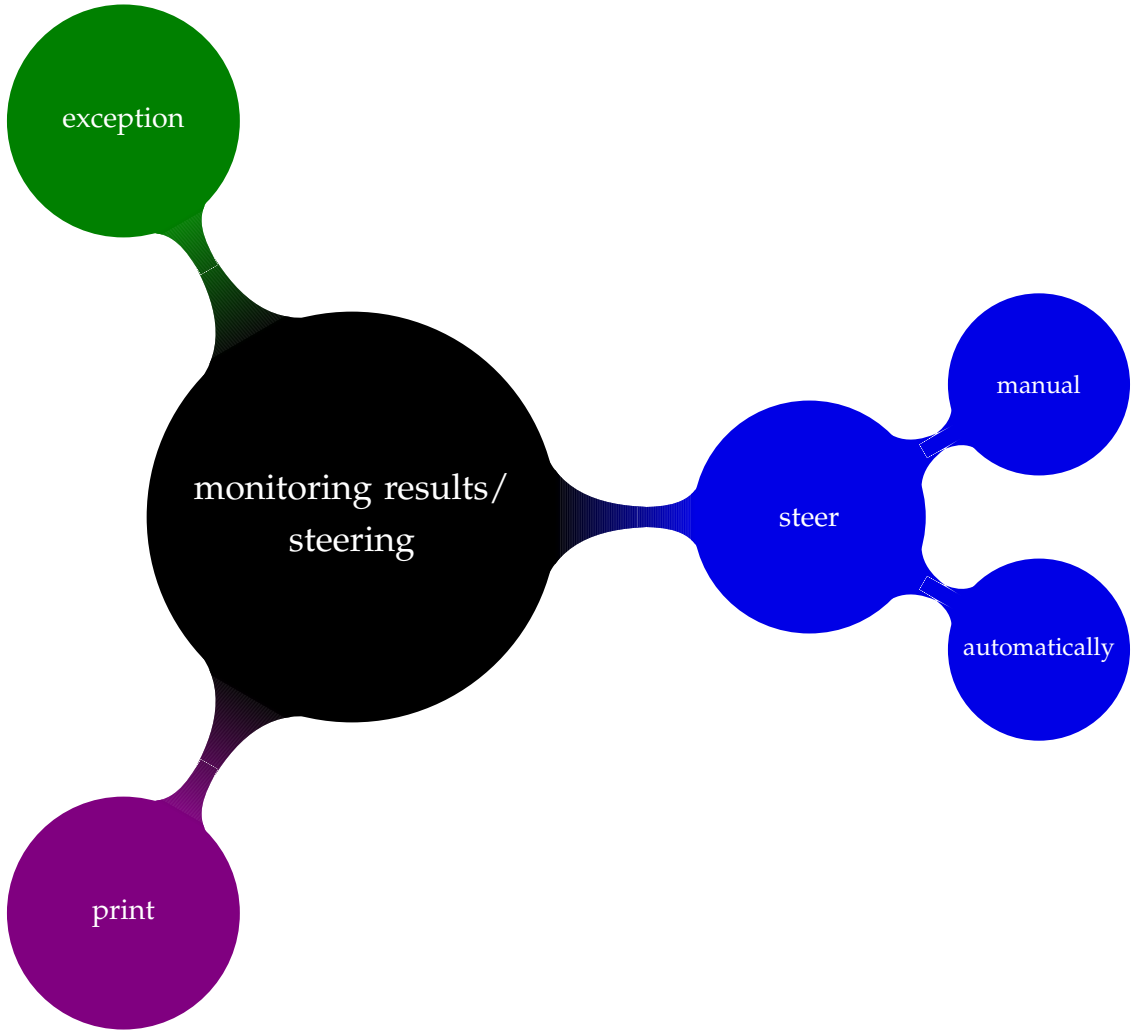
Presentation outline

Monitoring Systems/Logging: Overview



Presentation outline

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

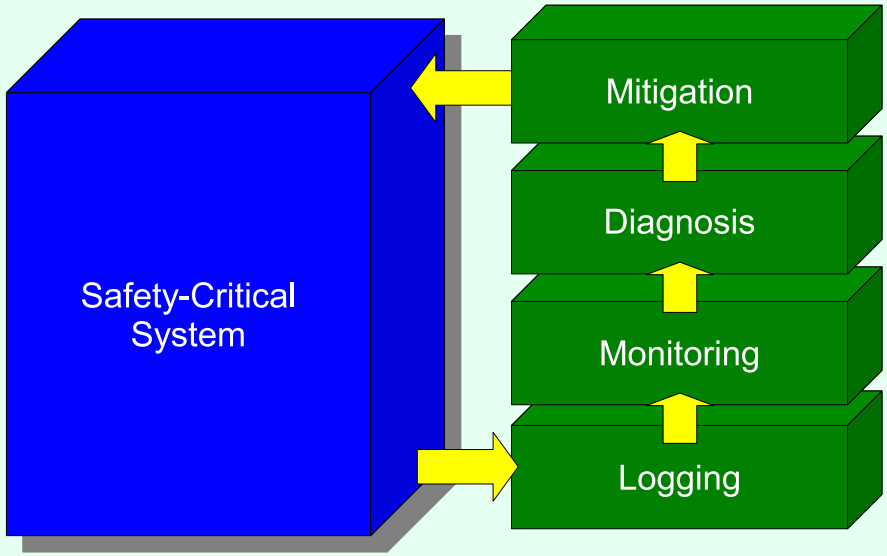
```
class Resource {
/*@
    scope = class
    logic = PTLTL
    {
     Event authenticate: end(exec(*
authenticate()));
     Event use: begin(exec(* access()));
      Formula : use -> <*> authenticate
    }
    violation Handler {
      @this.authenticate();
    }
@*/
void authenticate() {...}
void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

Where → scope = class
How → logic = PTLTL
what → Event authenticate: end(exec(* authenticate()));
What if → violation Handler { @this.authenticate(); }

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

Monitor-based Runtime Reflection

Software Architecture Pattern



Presentation outline

Outline

Diagnosis

Main Ideas

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

Diagnosis

Main Ideas

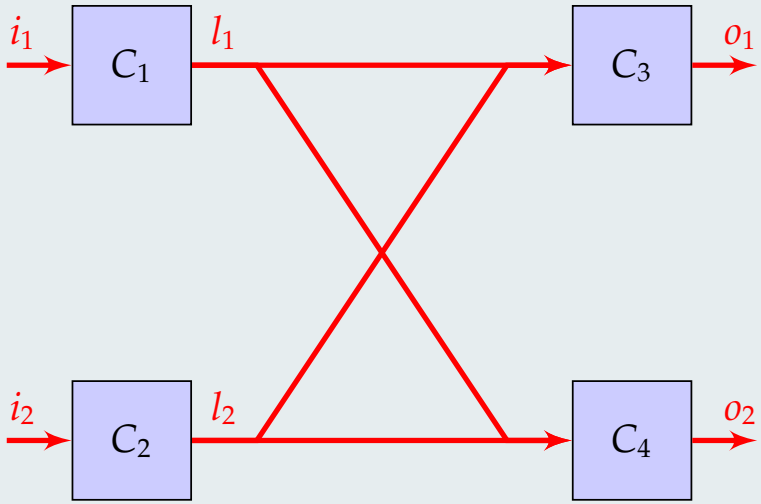
- Knowledge base
- Knowledge
- Explanation of Knowledge with Respect to the Knowledge base

Here

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

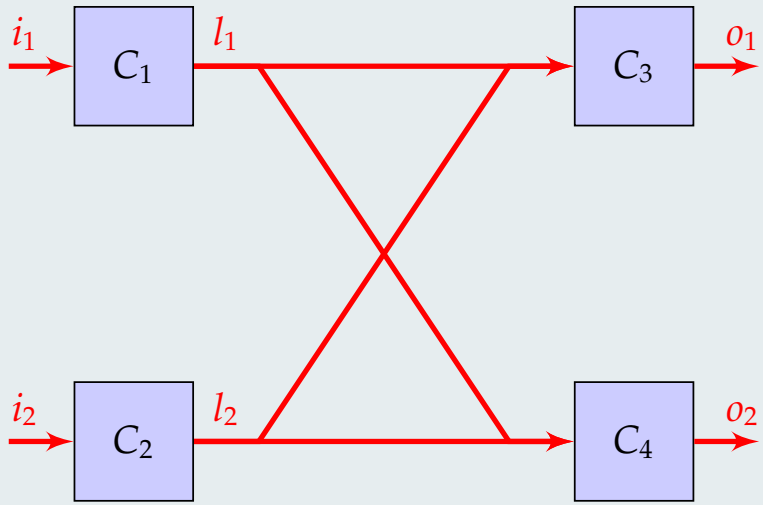
System Description in First-Order Logic

Example



System Description in First-Order Logic

Example

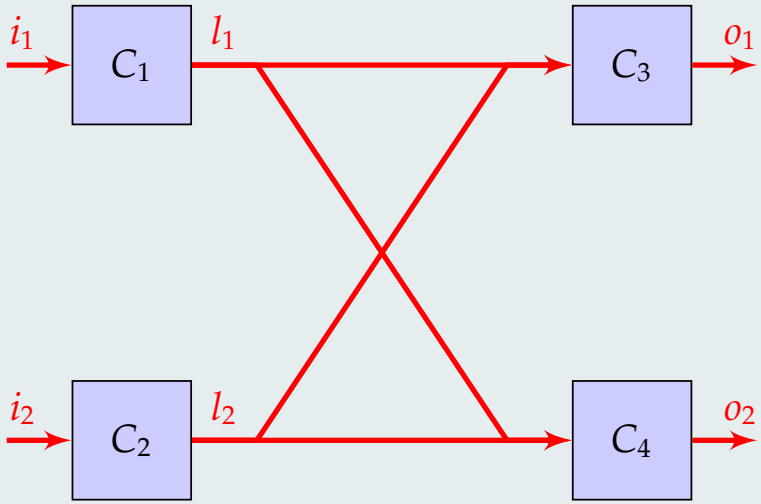


Formally

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

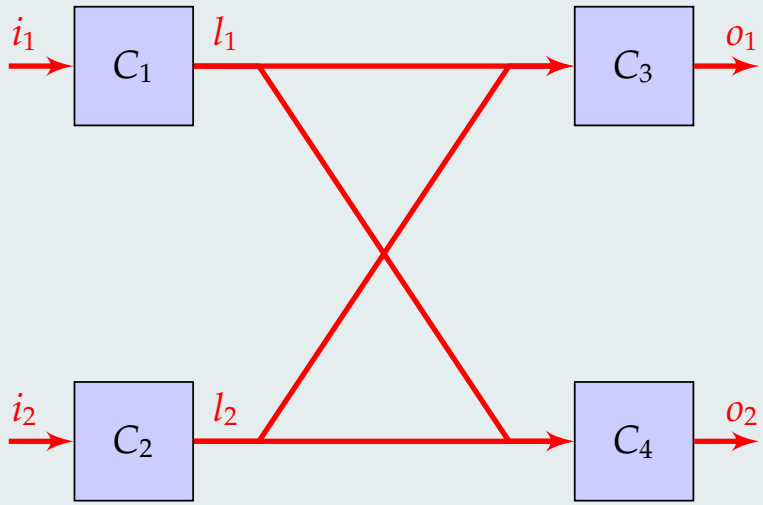
System Description in Propositional Logic

Example



System Description in Propositional Logic

Example

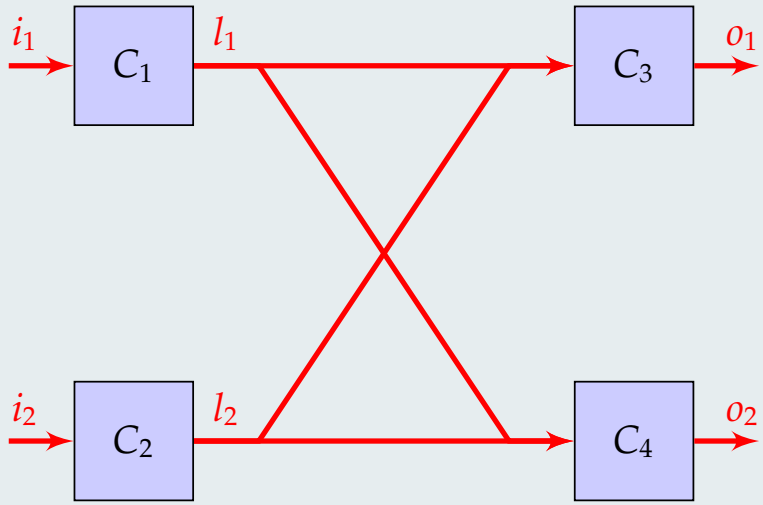


Propositional Logic

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

Observation

Example



Observation

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

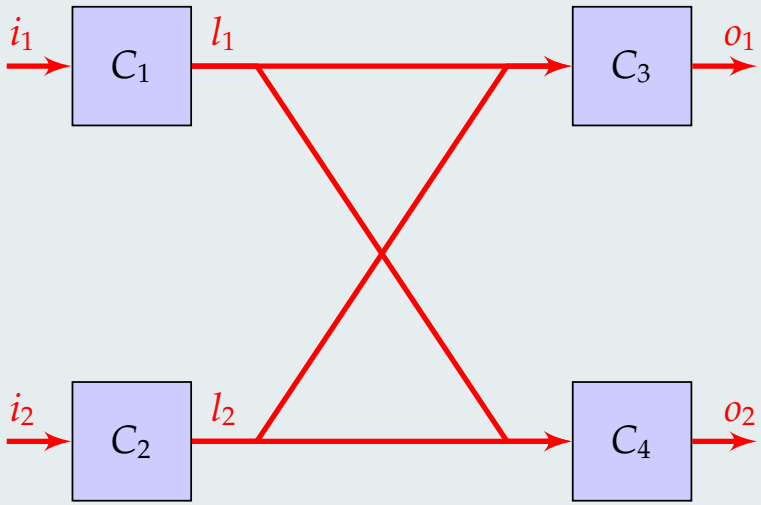
Formally: a formula OBS

Observation

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

Diagnosis

Example

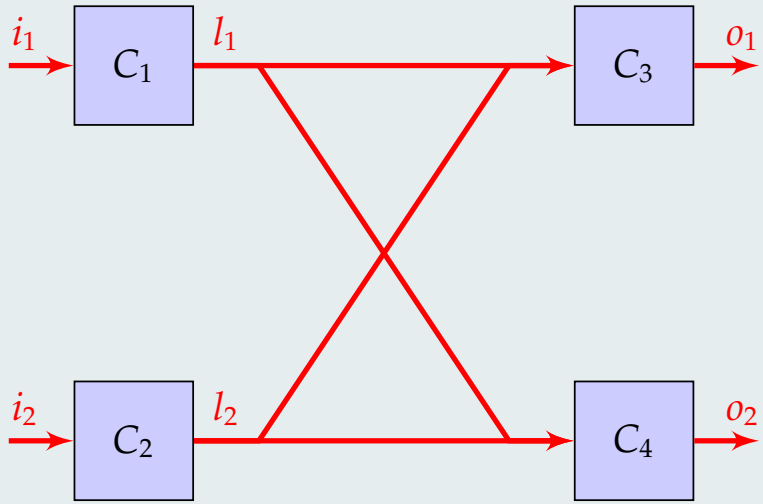


Diagnosis

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

Example

Example



Propositional Logic

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

Observations

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

Example

Propositional Logic

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

Observations

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

CNF

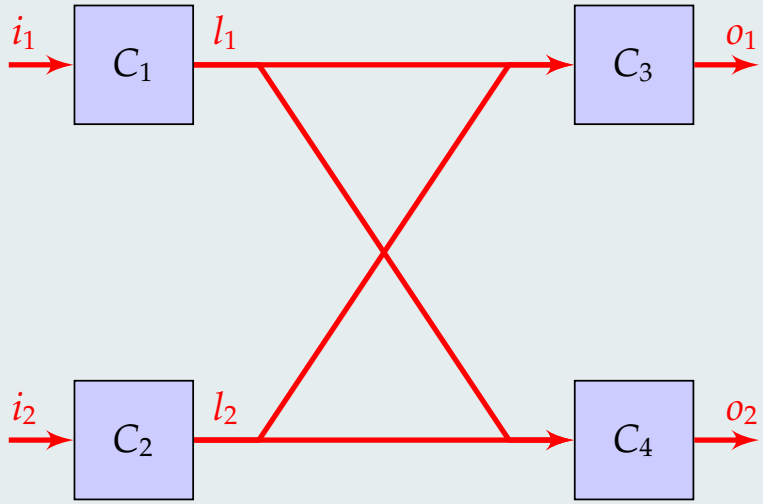
$$\begin{aligned} SD \quad = \quad & \neg i_1 \vee C_1 \vee l_1 \\ \wedge \quad & \neg i_2 \vee C_2 \vee l_2 \\ \wedge \quad & \neg l_1 \vee \neg l_2 \vee C_3 \vee o_1 \\ \wedge \quad & \neg l_1 \vee \neg l_2 \vee C_4 \vee o_2 \end{aligned}$$

SD $\wedge$ Observations

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

Example

Example

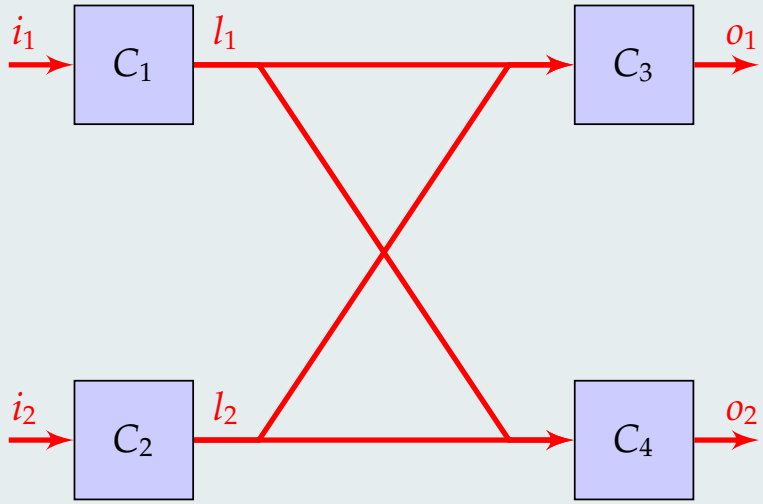


SD ∧ Observations

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

Example

Example

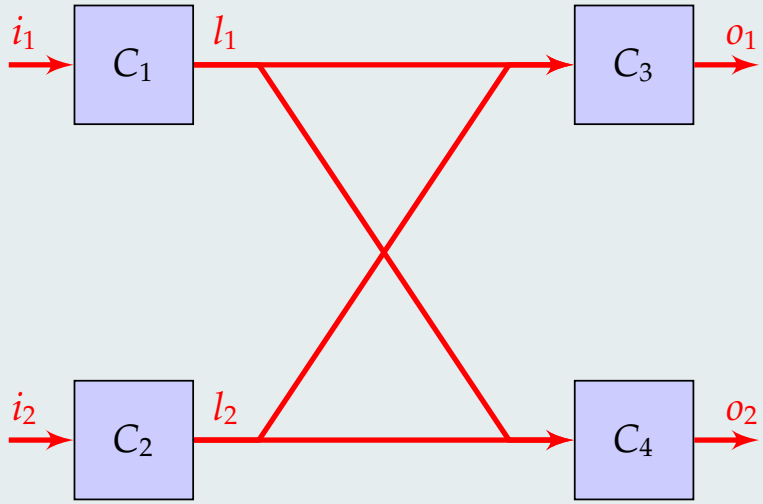


SD ∧ Observations

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

Example

Example

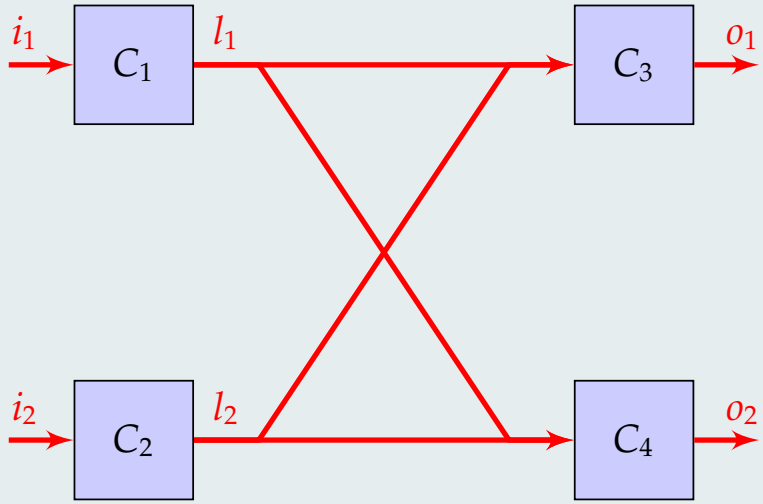


SD ∧ Observations

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

Example

Example

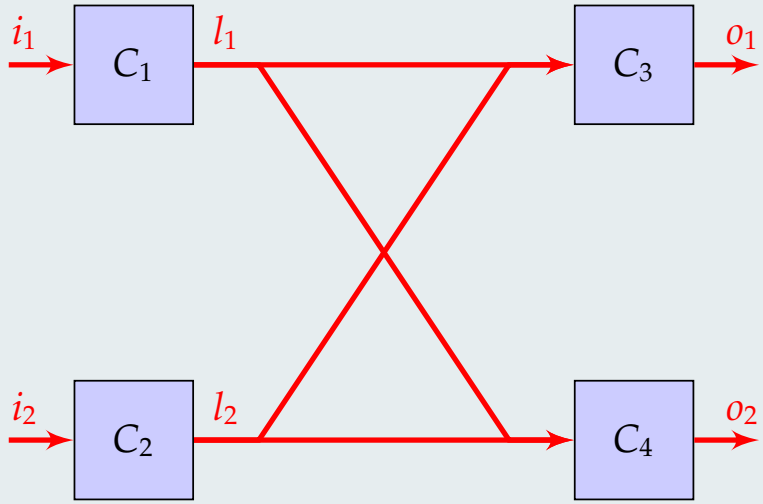


SD ∧ Observations

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

Example

Example

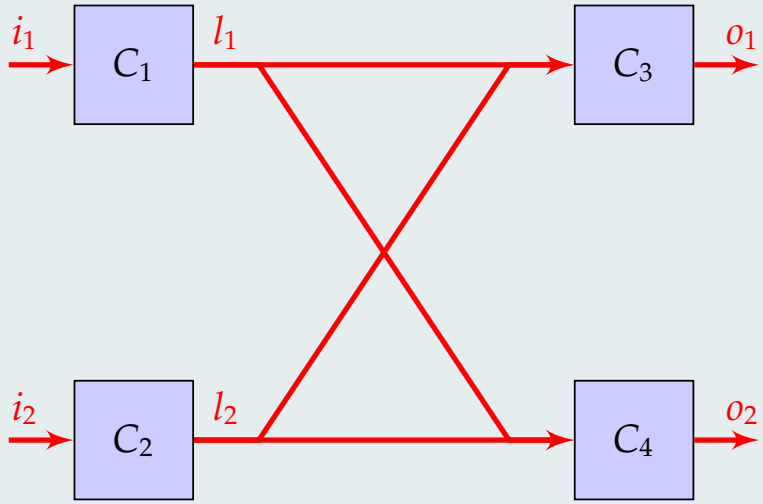


SD ∧ Observations

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

Example

Example

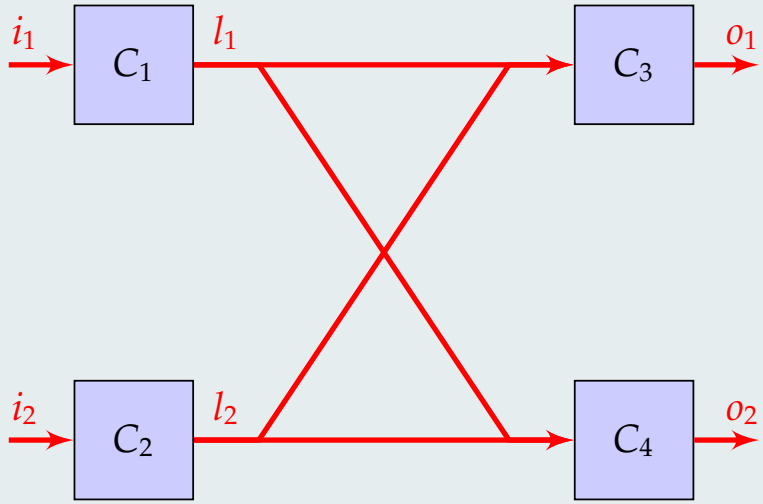


SD ∧ Observations

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

Example

Example



SD ∧ Observations

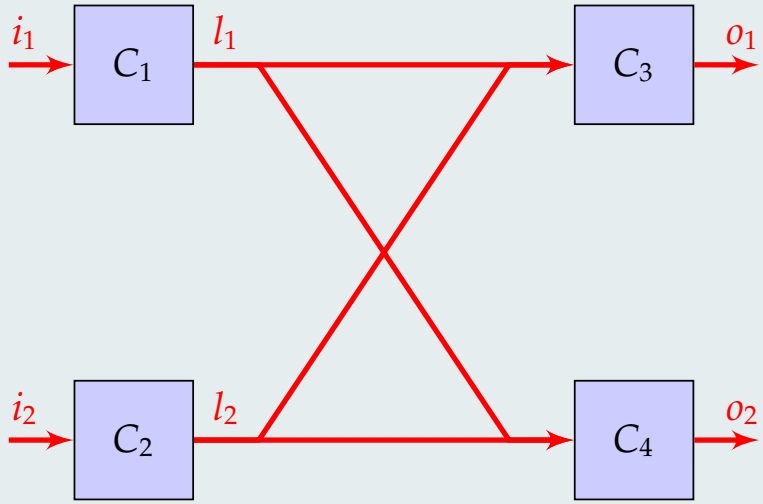
$$
\begin{aligned}
SD \quad = \quad & C_1 \vee l_1 \\
\wedge \quad & C_2 \vee l_2 \\
\wedge \quad & \neg l_1 \vee \neg l_2 \vee C_3 \\
\wedge \quad & \neg o_1 \wedge i_1 \wedge i_2 \wedge o_2
\end{aligned}
$$

Diagnoses

- $\Delta_1 = \{C_1\}$

Example

Example



SD ∧ Observations

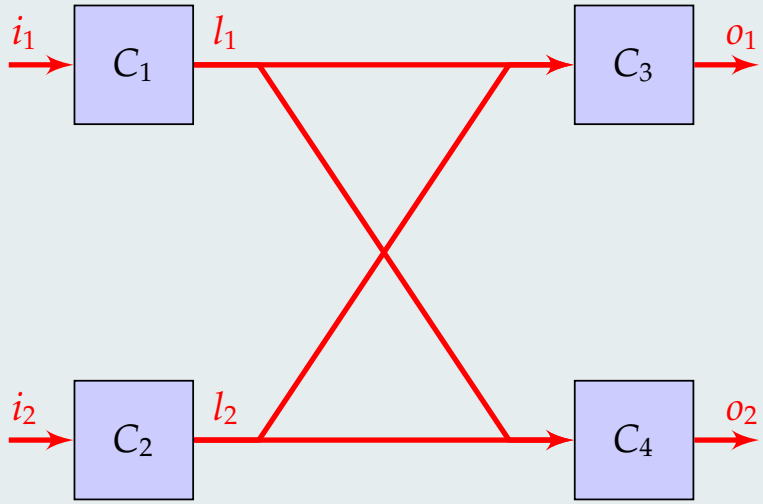
$$\begin{aligned} SD \quad = \quad & C_1 \vee l_1 \\ \wedge \quad & C_2 \vee l_2 \\ \wedge \quad & \neg l_1 \vee \neg l_2 \vee C_3 \\ \wedge \quad & \neg o_1 \wedge i_1 \wedge i_2 \wedge o_2 \end{aligned}$$

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

Monitors yield Obervations

We have. . .

- Monitor reports $\bot \rightsquigarrow$ 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 $\bot \rightsquigarrow$ 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

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 $\bot \rightsquigarrow$ 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)
- Monitor reports $\top \rightsquigarrow$ line is true

Oniscent Monitors

Example

Oniscent Monitors

Example

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

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

Oniscent Monitors

Example

$$SD = i \wedge \neg C_1 \rightarrow l \;\wedge\; l \wedge \neg C_2 \rightarrow o \qquad SD = \neg i \vee C_1 \vee l \;\wedge\; \neg l \vee C_2 \vee o$$

Observation: $i \wedge \neg o$

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

Oniscent Monitors

Example

$$SD = i \wedge \neg C_1 \rightarrow l \;\wedge\; l \wedge \neg C_2 \rightarrow o \qquad SD = \neg i \vee C_1 \vee l \;\wedge\; \neg l \vee C_2 \vee o$$

Observation: $i \wedge \neg o$

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

Diagnoses: C_2 or C_1

Oniscent Monitors

Example

$$SD = i \wedge \neg C_1 \rightarrow l \;\wedge\; l \wedge \neg C_2 \rightarrow o \qquad SD = \neg i \vee C_1 \vee l \;\wedge\; \neg l \vee C_2 \vee o$$

Observation: $i \wedge \neg o$

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

Diagnoses: C_2 or C_1

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

Oniscent Monitors

Example

$$SD = i \wedge \neg C_1 \rightarrow l \;\wedge\; l \wedge \neg C_2 \rightarrow o \qquad SD = \neg i \vee C_1 \vee l \;\wedge\; \neg l \vee C_2 \vee o$$

Observation: $i \wedge \neg o$

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

Diagnoses: C_2 or C_1

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

$\rightsquigarrow$ notion of omniscent diagnoses

Presentation outline

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?