Høgskulen på Vestlandet

UNIVERSITÄT ZU LÜBECK
INSTITUTE FOR SOFTWARE ENGINEERING
AND PROGRAMMING LANGUAGES

BeChong Summer School

15. – 19. October 2017, Chongqing, China

Runtime Verification

Volker Stolz
Western Norway University of Applied Sciences, NO

Based on material from Martin Leucker (U. Lübeck, DE)

Chapters of the Course

Høgskulen på Vestlandet

UNIVERSITÄT ZU LÜBECK
INSTITUTE FOR SOFTWARE ENGINEERING
AND PROGRAMMING LANGUAGES

Chapter 1

Fundamentals

Course “Runtime Verification”
M. Leucker & V. Stolz

Chapter 1

Learning Targets of Chapter “Fundamentals”.

1. Understand that software needs to be verified.
2. Understand the underlying principle of testing and runtime verification.
3. Get an idea of the different verification techniques.
4. Know when to use which verification technique.

Høgskulen på Vestlandet

UNIVERSITÄT ZU LÜBECK
INSTITUTE FOR SOFTWARE ENGINEERING
AND PROGRAMMING LANGUAGES

Chapter 1

Outline of Chapter “Fundamentals”.

Section

Motivation

Statistics

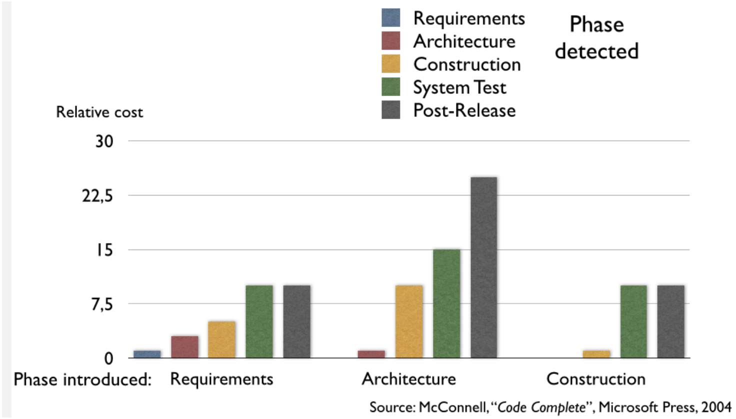
Examples

Some Data

How Important is Testing in IT Projects?

- Software developers spend between 50% and 70% of the time testing and validating code.
- Despite this, reliability is still the main problem of software products (e.g. Microsoft Windows).
- A recent study estimates the cost of bad software for industry in 60 billion dollars per year.
- The cost of a software bug in a critical system can be of millions or even billions.

Cost of Fixing Defects



Ariane V88 Crash (1996)

What Happened

The launcher began to disintegrate at about 39 seconds because of high aerodynamic loads resulting from an angle of attack of more than 20 degrees.

direct cost	500.000.000 €
indirect cost	2.000.000.000 €

Ariane V88 crash (1996)

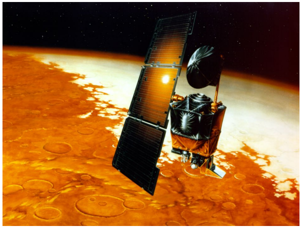
The Cause

```
P_M_DERIVE(T_ALG.E_BH) :=
  UC_16S_EN_16NS (TDB.T_ENTIER_16S
   ((1.0/C_M_LSB_BH) *
    G_M_INFO_DERIVE(T_ALG.E_BH)))
```

- The angle of attack was caused by incorrect altitude data following a software exception.
- The software exception was raised by an overflow in the conversion of a 64-bit floating-point number to a 16-bit signed integer value. The result was an operand error.

Loss of Mars Climate Orbiter (1999)

What Happened



- Mars Climate Orbiter (MCO) was part of the discovery program.
- The NASA robotic space probe was on the way to mars.
- Mars Climate Orbiter went out of radio contact on September 23, 1999.
- Communication was never reestablished.

Loss of Mars Climate Orbiter (1999)

Likely Cause

- Probe was at an altitude of 57 km instead of the calculated 150 to 170 km.
- Probe was destroyed by heat in the mars atmosphere in this too low orbit.
- The cause was a unit mismatch. Navigation software of the probe caluclated thruster performance using the metric unit Newtons while the ground crew entered course correction using the Imperial measure Pound-force (lbf).

Loss of Mars Polar Lander (1998)

What Happened

- Last telemetry was sent on December 3, 1999
- After cruise stage separation and atmospheric entry no further signals were received from the spacecraft. Mars Polar Lander remains lost.

Loss of Mars Polar Lander (1998)

The Likely Cause

- The cause of the communication loss is not known.
- Most likely cause as concluded by the Failure Review Board: Generation of spurious signals when the lander legs were deployed, giving false indication that the spacecraft had landed.
- This resulted in shutting down the engines 40 m above surface.

Power Shutdown of USS Yorktown

- A sailor mistakenly typed 0 in a field of the kitchen inventory application.
- Subsequent division by this field caused an arithmetic exception, which propagated through the system, crashed all LAN consoles and remote terminal units, and lead to power shutdown for about 3 hours.

Pentium Bug (1994)

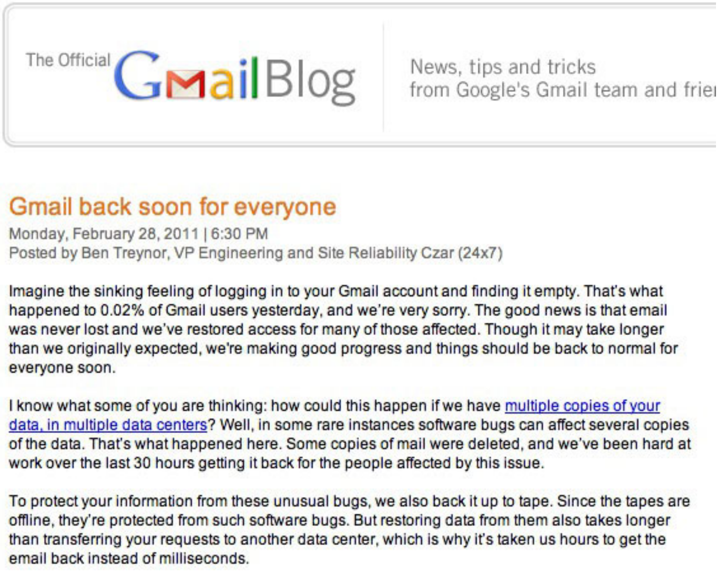
Wrong Calculation

$$4195835 - (4195835/3145727) \cdot 3145727 = 256$$

- Division used a speedy algorithm known as SRT division.
- This algorithm uses a table of 1,066 values (part of the chip's circuitry).
- The cause was the omission of five entries in this table.

Cost (Intel estimate) 500.000.000 $

GMail Data Loss (2011)



The Official GmailBlog

News, tips and tricks
from Google's Gmail team and frie

Gmail back soon for everyone

Monday, February 28, 2011 | 6:30 PM
Posted by Ben Treynor, VP Engineering and Site Reliability Czar (24x7)

Imagine the sinking feeling of logging in to your Gmail account and finding it empty. That's what happened to 0.02% of Gmail users yesterday, and we're very sorry. The good news is that email was never lost and we've restored access for many of those affected. Though it may take longer than we originally expected, we're making good progress and things should be back to normal for everyone soon.

I know what some of you are thinking: how could this happen if we have multiple copies of your data, in multiple data centers? Well, in some rare instances software bugs can affect several copies of the data. That's what happened here. Some copies of mail were deleted, and we've been hard at work over the last 30 hours getting it back for the people affected by this issue.

To protect your information from these unusual bugs, we also back it up to tape. Since the tapes are offline, they're protected from such software bugs. But restoring data from them also takes longer than transferring your requests to another data center, which is why it's taken us hours to get the email back instead of milliseconds.

Høgskulen på Vestlandet

UNIVERSITÄT ZU LÜBECK
INSTITUTE FOR SOFTWARE ENGINEERING AND PROGRAMMING LANGUAGES

Section

Definitions

Chapter 1 “Fundamentals”
Course “Runtime Verification”
M. Leucker & V. Stolz

Testing
The Definition

Definition (Testing)

Testing is the examination of a subset of the behaviours of a program or system.

Testing can detect the presence of errors, but cannot prove their absence.

Myers Definition of Testing

Test Suite, Test Case and Test Run

- Testing is the process of executing a program on a test suite with the intent of finding errors.
- A test suite is a subset of the possible inputs.
- An element of a test suite is called a test case.
- A run of the program on a test case is a test run.

Testing cannot show the absence of errors

- Most programs have infinitely many possible inputs.
- Event programs with only finitely many inputs can still have infinitely many executions.
- A program with finitely many executions can have infinite executions (possibly because of errors!)

Testing is not a correctness proof

Investigating in finitely many executions we cannot prove the absence of errors.

Quotes on Testing

"Testing can show the presence of errors, but never their absence." (E.W. Dijkstra)

"Testing is a destructive process, even a sadistic process." (G.J. Myers)

"Die destruktive Kreativität, das System aufs Kreuz zu legen, und der Sportliche Ehrgeiz, Fehler zu finden, sorgen für gute Testfälle." (J. Siedersleben)

Testing cannot show that software works.

- Testing reveals errors in programs if
  the program does not what it is supposed to do.
- Programs that do what they are supposed to do
  can still contain errors
  if they do more than they are supposed to do.
- Specifications may contain errors as well.
  Testing assumes specifications to be correct.

Validation and Verification

Validation

"Are we building the right product?"
(Does the system meet the client's expectations?)

Verification

"Are we building the product right?"
(Does the system meet its specification?)

Validation and Verification

Validation

"Are we building the right product?"
(Does the system meet the client's expectations?)

Verification

"Are we building the product right?"
(Does the system meet its specification?)

Definition (Verification)

Verification is comparing code with its specification.

IEEE 1012-2004

Standard for Software Verification and Validation

Abstract

Software verification and validation (V&V) processes determine whether the development products of a given activity conform to the requirements of that activity and whether the software satisfies its intended use and user needs. Software V&V life cycle process requirements are specified for different software integrity levels. The scope of V&V processes encompasses software-based systems, computer software, hardware, and interfaces. This standard applies to software being developed, maintained, or reused [legacy, commercial off-the- shelf (COTS), non-developmental items]. The term software also includes firmware, microcode, and documentation. Software V&V processes include analysis, evaluation, review, inspection, assessment, and testing of software products. Keywords: IV&V, software integrity level, software life cycle, V&V, validation, verification

IEEE 1012-2004

Standard for Software Verification and Validation

Software V&V processes consist of the verification process and validation process. The verification process provides objective evidence whether the software and its associated products and processes

- conform to requirements (e.g., for correctness, completeness, consistency, accuracy) for all life cycle activities during each life cycle process (acquisition, supply, development, operation, and maintenance),
- satisfy standards, practices, and conventions during life cycle processes and
- successfully complete each life cycle activity and satisfy all the criteria for initiating succeeding life cycle activities (e.g., building the software correctly).

IEEE 1012-2004

Standard for Software Verification and Validation

The validation process provides evidence whether the software and its associated products and processes

- satisfy system requirements allocated to software at the end of each life cycle activity,
- solve the right problem (e.g., correctly model physical laws, implement business rules, use the proper system assumptions) and
- satisfy intended use and user needs.

Verification or Validation?

Testing and Runtime Verification are verification techniques.

Both do not work if the specification is wrong.

Runtime Verification
The Definition

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 for checking whether a run of a system under scrutiny satisfies or violates a given correctness property.

Run

Definition (Run)

A run of a system is a possibly infinite sequence of the system's states. Formally, a run may be considered as a possibly infinite *word* or *trace*.

- Runs are formed by current variable assignments,
- or as the sequence of actions a system is emitting or performing.

Execution

Definition (Execution)

An *execution* of a system is a *finite prefix* of a run and, formally, it is a finite trace.

- In verification, we check whether a run of a system adhere to given correctness properties.
- RV is primarily used on executions.
- A monitor checks whether an execution meets a correctness property.

Adding Monitors to a System

Definition (Monitor)

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

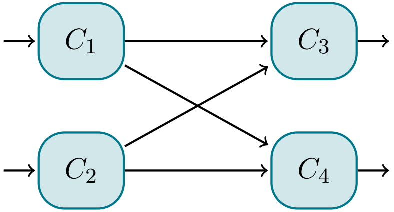


Figure: Monitor M checks correctness of components C_i.

Adding Monitors to a System

Definition (Monitor)

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

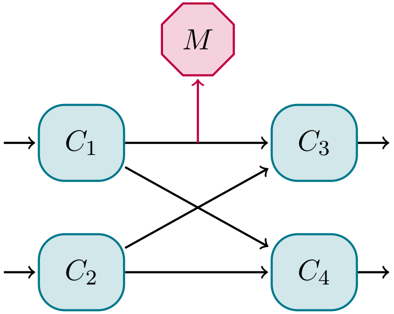


Figure: Monitor M checks correctness of components C_i.

Monitors can Check Relations of Values

- A monitor can use more than one input value.
- A monitor can check the relations of multiple values.

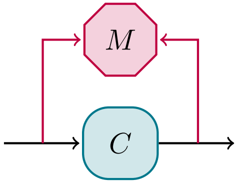


Figure: Monitor M checks data input/output relation of component C.

Høgskulen på Vestlandet

UNIVERSITÄT ZU LÜBECK
INSTITUTE FOR SOFTWARE ENGINEERING
AND PROGRAMMING LANGUAGES

Section

Classification

- Verification Techniques
- Course Topics

Testing

Testing

$i_0/o_0, i_1/o_1, \ldots \to S$

Does system S satisfy the input/output sequence $i_0/o_0, i_1/o_1, \ldots$?

Input/output sequence is written manually.

Main research topic is input/output sequence generation.

Current execution must be correct.

Testing vs. Oracle Testing

Testing	Oracle Testing
$i_0/o_0, i_1/o_1, \ldots \to S$	$i_0, i_1, \ldots \to S \times M$
Does system S satisfy the input/output sequence $i_0/o_0, i_1/o_1, \ldots$?	Does system S satisfy the test oracle M on input sequence $i_0, i_1, \ldots$?
Input/output sequence is written manually.	Input sequence and test oracle are written manually.
Main research topic is input/output sequence generation.	Main research topic is input sequence generation.
Current execution must be correct.	Current execution must be correct.

Oracle Testing vs. Runtime Verification

Oracle Testing	Runtime Verification
$i_0, i_1, \ldots \to S \times M$	$i_0, i_1, \ldots \to S \times M_\varphi$
Does system S satisfy the test oracle M on input sequence $i_0, i_1, \ldots$?	Does system S satisfy the generated monitor M_φ on input sequence $i_0, i_1, \ldots$?
Input sequence and test oracle are written manually.	Monitor is synthesized from correctness property.
Main research topic is input sequence generation.	Main research topic is monitor generation.
Current execution must be correct.	Current execution must be correct.

Runtime Verification vs. Model Checking

Runtime Verification	Model Checking
$i_0, i_1, \ldots \to S \times M_\varphi$	$S \models \varphi$ via $\mathcal{L}(S) \subseteq \mathcal{L}(\varphi)$
Does system S satisfy the generated monitor M_φ on input sequence $i_0, i_1, \ldots$?	Is system S a model of the correctness property φ? Are all runs of the system $\mathcal{L}(S)$ a subset of all correct runs $\mathcal{L}(\varphi)$?
Monitor is synthesized from correctness property.	Automatic proof using manually created system model.
Main research topic is monitor generation.	Main research topic are algorithms for proving model relation.
Current execution must be correct.	All runs must be correct.

Model Checking vs. Theorem Proofing

Model Checking	Theorem Proving
$S \models \varphi$ via $\mathcal{L}(S) \subseteq \mathcal{L}(\varphi)$	$S \models \varphi$ via $S \vdash \varphi$
Is system S a model of the correctness property φ? Are all runs of the system $\mathcal{L}(S)$ a subset of all correct runs $\mathcal{L}(\varphi)$?	Is system S a model of the correctness property φ? Can we find a proof that property φ derives from system S?
Automatic proof using manually created system model.	Proof is done manually by deriving property from the system.
Main research topic are algorithms for proving model relation.	Main research topic is proof generation using a calculus.
All runs must be correct.	All runs must be correct.

Conclusion of the Comparison

SST	RV	MC	TP
$I/O \to S$ resp. $I \to S \times M$	$I \to S \times M_\varphi$	$S \models \varphi$ via $\mathcal{L}(S) \subseteq \mathcal{L}(\varphi)$	$S \models \varphi$ via $S \vdash \varphi$
sequence generation	monitor synthesis	proof algorithms	manual proofs
current execution	current execution	all runs	all runs

- input sequence $I = i_0, i_1, \ldots$
- output sequence $O = o_0, o_1, \ldots$
- system S
- test oracle resp. monitor M
- correctness property φ

Topics Covered in Course “Testing and Runtime Verification”

- Definitions of testing, verification and validation
- Static testing vs. dynamic testing
- Manual vs. automated testing
- Black box vs. white box testing
- Coverage criteria
- Test case generation
- Temporal logic and multi-valued semantics
- Finite (ω-)automata
- Monitor synthesis
  - Automata constructions and -analysis
  - Formula rewriting
- Runtime monitoring, monitor integration
- Runtime verification frameworks
- Conformance testing

Conclusion

1. Testing is the examination of a subset of the behaviours of a program or system. Testing can detect the presence of errors, but cannot prove their absence.
2. Runtime verification deals with verification techniques that allow checking whether an execution of a system under scrutiny satisfies or violates a given correctness property.
3. Testing and Runtime Verification are verification techniques. They do not validate the given specification.
4. One of the main challenges for testing is how to generate a proper input/ output sequence.
5. One of the main challenges for runtime verification is the synthesis of efficient monitors from logical specifications.

Høgskulen på Vestlandet

UNIVERSITÄT ZU LÜBECK
INSTITUTE FOR SOFTWARE ENGINEERING AND PROGRAMMING LANGUAGES

Chapter 2

Recall Runtime Verification in More Depth

Course “Runtime Verification”
M. Leucker & V. Stolz

Chapter 2

Learning Targets of Chapter "Recall Runtime Verification in More Depth".

1. Recall the underlying principle of runtime verification.
2. Get to know to applications of runtime verification.
3. See different frameworks for runtime verification.

Chapter 2

Outline of Chapter “Recall Runtime Verification in More Depth”.

Høgskulen på Vestlandet

UNIVERSITÄT ZU LÜBECK
INSTITUTE FOR SOFTWARE ENGINEERING
AND PROGRAMMING LANGUAGES

Section

Runtime Verification

Recall
Word Problem
Good Monitors?

Chapter 2 “Recall Runtime Verification in More Depth”
Course “Runtime Verification”
M. Leucker & V. Stolz

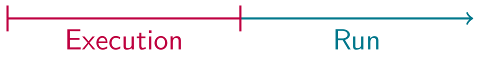
Runtime Verification (Recall)

Verification technique that allow for checking whether a run of a system under scrutiny satisfies or violates a given correctness property.

Run and Execution (Recall)

- Run: possibly infinite sequence of the system's states.
- Run formally: possibly infinite word or trace.

Run and Execution (Recall)

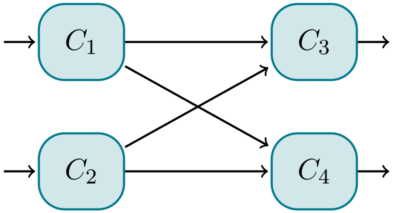


- Run: possibly infinite sequence of the system's states.
- Run formally: possibly infinite word or trace.

- Execution: finite prefix of a run.
- Execution formally: finite word or trace.
- RV is primarly used on executions.

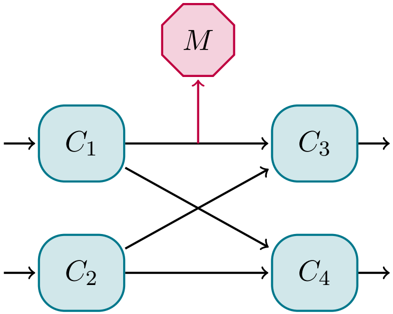
Adding Monitors to a System (Recall)

- A monitor checks whether an execution meets a correctness property.
- A monitor is a device that reads a finite trace and yields a certain verdict.



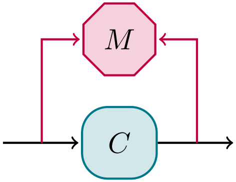
Adding Monitors to a System (Recall)

- A monitor checks whether an execution meets a correctness property.
- A monitor is a device that reads a finite trace and yields a certain verdict.



Monitors can Check Relations of Values (Recall)

- A monitor can use more than one input value.
- A monitor can check the relations of multiple values.



RV and the Word Problem

A simple monitor outputs

- yes if the execution satisfies the correctness property,
- no if not.

- Let $[\![\varphi]\!]$ denote the set of valid executions given by property φ.
- Then runtime verification answers the word problem $w \in [\![\varphi]\!]$.
- The word problem can be decided with lower complexity compared to the subset problem.

How Does a *Good Monitor* Look?

Impartiality and *Anticipation*

Definition (Impartiality)

Impartiality requires that a finite trace is not evaluated to $true$ or, respectively $false$, if there still exists a (possibly infinite) continuation leading to another verdict.

Definition (Anticipation)

Anticipation requires that once every (possibly infinite) continuation of a finite trace leads to the same verdict, then the finite trace evaluates to this very same verdict.

A monitor for RV should adhere to both maxims!

Høgskulen på Vestlandet

UNIVERSITÄT ZU LÜBECK
INSTITUTE FOR SOFTWARE ENGINEERING
AND PROGRAMMING LANGUAGES

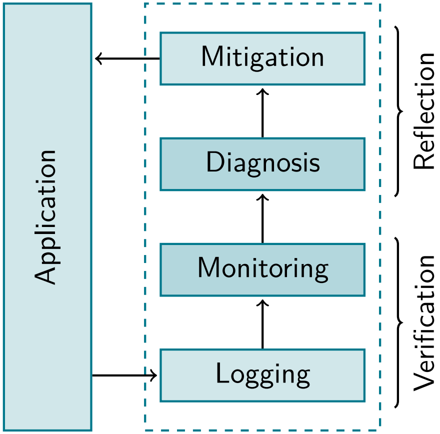
Section

Applications

Runtime Reflection

Runtime reflection (RR) is an architecture pattern for the development of reliable systems.

- A *monitoring layer* is enriched with
- a *diagnosis layer* and a subsequent
- *mitigation layer*.



Runtime Reflection

Logging—Recording of System Events

The logging layer

- observes system events and
- provides them for the monitoring layer.

Realization

- Add code annotations within the system to build or
- use separated stand-alone loggers.

Runtime Reflection

Monitoring—Fault Detection

The monitoring layer

- is implemented using runtime verification techniques,
- consists of a number of monitors,
- detects the presence of faults in the system and
- raises an alarm for the diagnosis layer in case of faults.

Runtime Reflection

Diagnosis—Failure Identification

The diagnosis layer

- collects the verdicts of the monitors and
- deduces an explanation for the current system state solely based upon the results of the monitors and general information on the system.

Runtime Reflection

Mitigation—Reconfiguration

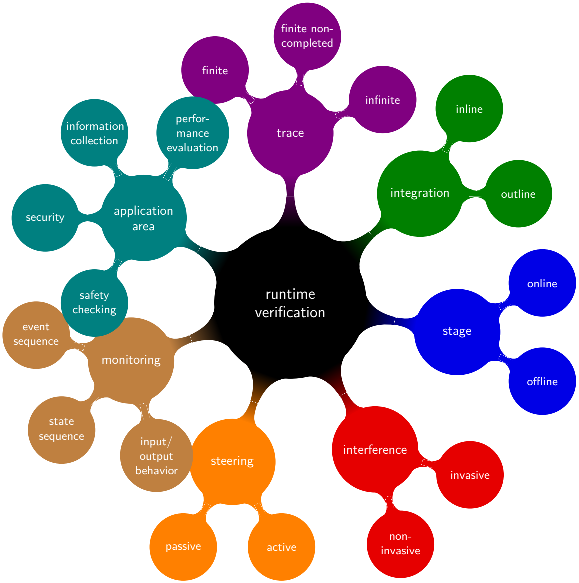
The reconfiguration layer

- mitigates the failure, if possible,
- or else may store detailed diagnosis information for off-line treatment.

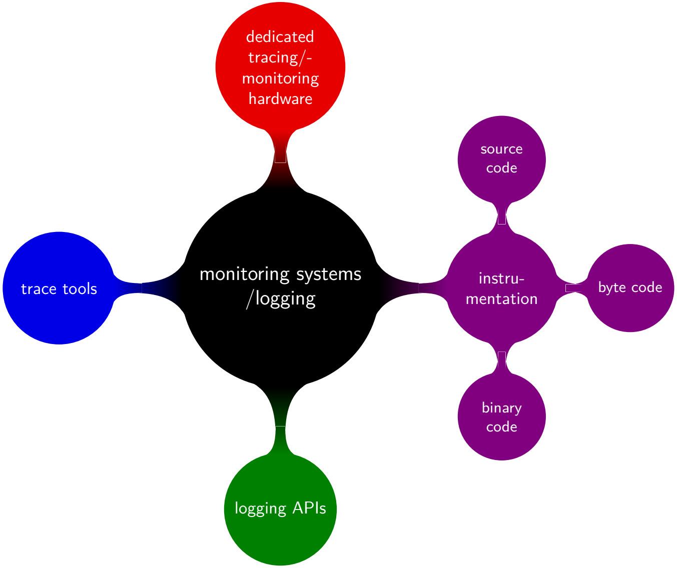
When to Use RV?

- The verification verdict is often referring to a model of the real system. Runtime verification may then be used to easily check the actual execution of the system. Thus, runtime verification may act as a partner to theorem proving and model checking.
- Often, some information is available only at runtime. In such cases, runtime verification is an alternative to theorem proving and model checking.
- The behavior of an application may depend heavily on the environment of the target system. In this scenario, runtime verification adds on formal correctness proofs by model checking and theorem proving.
- In the case of systems where security is important, it is useful also to monitor behavior or properties that have been statically proved or tested.

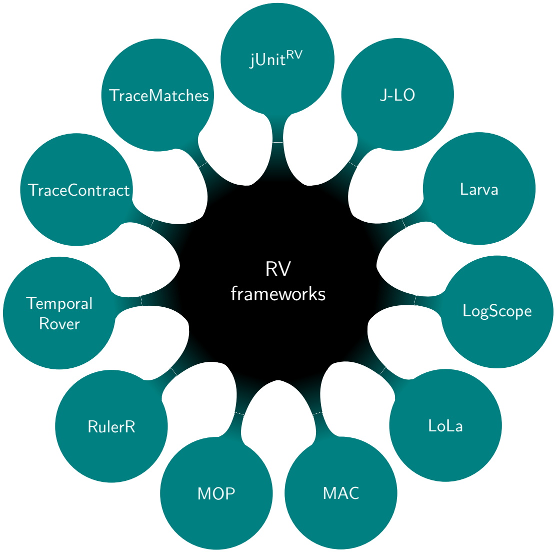
Taxonomy


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
perfor-mance evaluation
security
safety checking

Monitoring Systems/Logging: Overview



Monitoring Systems/Logging: Overview



Conclusion

1. Runtime verification deals with verification techniques that allow checking whether an execution of a system under scrutiny satisfies or violates a given correctness property.
2. A Monitor checks whether an execution meets a correctness property.
3. One of its main technical challenges is the synthesis of efficient monitors from logical specifications.

Høgskulen på Vestlandet

UNIVERSITÄT ZU LÜBECK
INSTITUTE FOR SOFTWARE ENGINEERING
AND PROGRAMMING LANGUAGES

Chapter 3

Specification Languages on Words

Course “Runtime Verification”
M. Leucker & V. Stolz

Høgskulen på Vestlandet

UNIVERSITÄT ZU LÜBECK
INSTITUTE FOR SOFTWARE ENGINEERING
AND PROGRAMMING LANGUAGES

Chapter 3

Learning Targets of Chapter “Specification Languages on Words”.

1. Understand that RV specifies shape of words.
2. Recall the idea of regular expressions and understand their limitations for practical specifications.
3. Get an idea about temporal logics.
4. Understand the difference of regular expressions and temporal logics.
5. Understand how to specify properties in LTL.

Chapter 3

Outline of Chapter “Specification Languages on Words”.

Høgskulen på Vestlandet

UNIVERSITÄT ZU LÜBECK
INSTITUTE FOR SOFTWARE ENGINEERING
AND PROGRAMMING LANGUAGES

Section

Runs Are Words

- States of the System
- Executions Are Words

Chapter 3 "Specification Languages on Words"
Course "Runtime Verification"
M. Leucker & V. Stolz

Recap

We want to monitor the execution of a system.

We have already seen that

- A run of a system is a possibly infinite sequence of the system's states.
- An execution of a system is a finite prefix of a run.

Observations

- We describe the execution of a system in a discrete way.
- The system is in exactly one state at a time.
- In the next step the system is in the next state.

Atomic Propositions

- An atomic proposition is an indivisible bit.
- We consider a fixed set of finitely many such bits.
- In every state every atomic proposition is either true or false.
- In other words:
  In every state of the execution some atomic propositions hold.

Example

- Variable `count` is greater than 5.
- Memory for a variable `data` is allocated.
- Memory for `data` is free.
- The file handle `logfile` points to an opened file.

States

- Let AP be a fixed finite non empty set of atomic propositions.
- $\Sigma = 2^{\mathrm{AP}}$ is the power set of these.
- A state can be seen as an element $a \in \Sigma$.

Executions Are Like Linear Paths

Languages over Alphabets

Let Σ be an alphabet and $n \in \mathbb{N}$.
We then use the following notation:

Notation	Meaning
Σ^*	set of all finite words over Σ
Σ^n	all words in Σ^* of length n
$\Sigma^{\leq n}$	all words in Σ^* of length at most n
$\Sigma^{\geq n}$	all words in Σ^* of length at least n
Σ^+	$= \Sigma^{\geq 1}$
Σ^ω	set of all infinite words over Σ
Σ^∞	$= \Sigma^* \cup \Sigma^\omega$

Executions Are Words

- A state can be seen as an element $a \in \Sigma$.
- Now a run is an infinite word $w \in \Sigma^\omega$
- and an execution a finite prefix $w \in \Sigma^*$.

Runtime verification is about checking if an execution is correct, so we need to specify the set of correct executions as a language $L \subseteq \Sigma^*$. Therefore a correctness property is a language L.

Section

Regular Expressions

Chapter 3 “Specification Languages on Words”
Course “Runtime Verification”
M. Leucker & V. Stolz

Regular Expressions: The Idea

- Use a bottom up construction to construct a complex language by combining simpler languages together.
- Start with languages containing only one word of length 1.
- Use the common operations on languages to combine these into complexer languages.

Operations on Languages

Let $L_1 \subseteq \Sigma^*$ and $L_2 \subseteq \Sigma^*$ be two languages. We than have

intersection $L_1 \cap L_2 = \{w \in \Sigma^* \mid w \in L_1 \wedge w \in L_2\}$

union $L_1 \cup L_2 = \{w \in \Sigma^* \mid w \in L_1 \vee w \in L_2\}$

complement $\overline{L} = \{w \in \Sigma^* \mid w \notin L\}$

concatenation $L_1 \circ L_2 = \{uv \in \Sigma^* \mid u \in L_1 \wedge v \in L_2\}$

Kleene star $L^* = \{u_1 u_2 \dots u_n \in \Sigma^* \mid \forall i : u_i \in L\}$

Regular Expressions

Regular expressions use only the operations union, concatenation and Kleene star. These are enough to build all regular languages.

- Every symbol $a \in \Sigma$ is a regular expression.
- The empty word ε describes the empty word.
- Concatenation is expressed by concatenating regular expressions.
- Union is expressed by the $\mid$ operator combining two regular expressions.
- Kleene star is expressed by the $\star$ operator at the end of a regular expression.

Examples

Examples

Let $\Sigma = \{0, 1\}$ be the finite alphabet.

- `(0|1)*` specifies all words $w \in \Sigma^*$.
- `1*0*` specifies all words $w \in \Sigma^*$ that do not contain the string `01`.
- `((0|1)1)*` specifies all words $w \in \Sigma^*$ of even length where every second letter is `1`.
- `((0|1)1)*(0|1|`ε`)` specifies all words $w \in \Sigma^*$ where every second letter is `1`.

Syntax of Regular Expressions

Definition (Syntax of regular expressions)

Let $x \in \Sigma$ be a symbol from a given alphabet. The syntax of regular expressions is inductively defined by the following grammar:

$$\varphi ::= \varepsilon \mid x \mid \varphi\varphi \mid (\varphi \mid \varphi) \mid (\varphi)\star$$

Semantics of Regular Expression

Definition (Semantics of Regular Expressions)

Let $w, u_i \in \Sigma^*$ be words over the given alphabet, $x \in \Sigma$ be an element of the alphabet and R, R' regular expressions. Then the semantics of a regular expression is inductively defined as relation $\models$ of a non empty word and a regular expression as follows.

$$\begin{aligned}
\varepsilon &\models \varepsilon \\
x &\models x \\
w &\models RR' && \text{iff } \exists u_1, u_2 : w = u_1 u_2 \\
& && \text{and } u_1 \models R \text{ and } u_2 \models R' \\
w &\models (R \mid R') && \text{iff } w \models R \text{ or } w \models R' \\
w &\models (R)\star && \text{iff } \exists u_1, \ldots, u_n : w = u_1 \ldots u_n \\
& && \text{and } \forall i \in \{1, \ldots, n\} : u_i \models R.
\end{aligned}$$

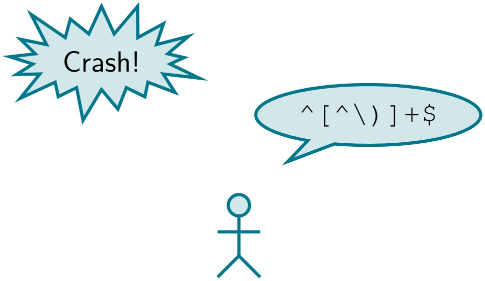
Expressiveness of Regular Expressions

Regular expressions describe regular languages:

- Every language described by a regular expression is a regular language.
  Proof: Structural induction on the syntax of regular expressions.
- Every regular language can be described using a regular expression.
  Proof: Standard translation of deterministic finite automata into regular expressions.

Limitations

Regular expressions sometimes look more like a swear word in a comic book than a specification of anything.



Specifying Correctness Properties

- Specifications must be easy to understand:
  Specification must be correct—otherwise verification makes no sense at all.
- We need kind of negation:
  It is often easier to specify the behaviour we do not want.

Høgskulen på Vestlandet

UNIVERSITÄT ZU LÜBECK
INSTITUTE FOR SOFTWARE ENGINEERING
AND PROGRAMMING LANGUAGES

Section

Linear Temporal Logic (LTL)

Propositional Logic
Temporal Logic

Chapter 3 “Specification Languages on Words”
Course “Runtime Verification”
M. Leucker & V. Stolz

Another Idea

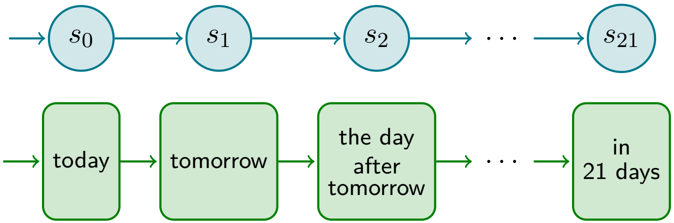
- A state of a system is a set of atomic propositions that hold in this state.
- An execution of a system is a finite sequence of such states.

Let's use operators of

- propositional logic to describe properties of one state.
- temporal logic to describe the relationship of states.
- propositional logic to combine this.

A Simple Analogy

- A state is like a day.
- The initial state is like today.
- The next state is like tomorrow.



Remember

- A day is a state in the execution.
- A day is a letter in the word over $\Sigma = 2^{\mathrm{AP}}$.

Propositional Logic

Using propositional logic without temporal operators we describe only the first state (today).

Example

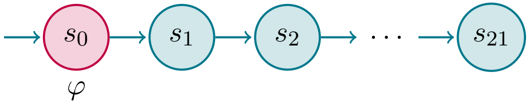
Consider $\mathrm{AP} = \{p, q, r, s\}$ and an initial state s_0 of an execution w in which p and r holds. We then have

$$\begin{tikzcd} {} \arrow[r] & s_0 \arrow[r] & s_1 \arrow[r] & s_2 \arrow[r] & \cdots \arrow[r] & s_{21} \\ & \{p, r\} & & & & \end{tikzcd}$$

$$
\begin{aligned}
w &\models \mathrm{true} & \qquad w &\not\models \mathrm{false} \\
w &\models p & \qquad w &\models p \land r \lor q \\
w &\models \neg\, q \land \neg\, s & \qquad w &\not\models q.
\end{aligned}
$$

Formula: φ

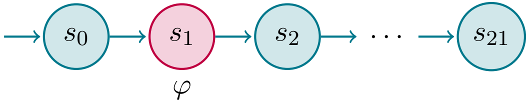
The formula φ holds for an execution if φ holds in the first state s_0 of that execution.



$$\begin{tikzcd}
{} \arrow[r] & s_0 \arrow[r] & s_1 \arrow[r] & s_2 \arrow[r] & \cdots \arrow[r] & s_{21}
\end{tikzcd}$$

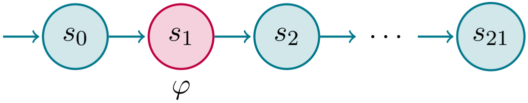
Next: $\mathrm{X}\,\varphi$

The formula $\mathrm{X}\,\varphi$ holds in state s_i if φ holds in state s_{i+1}.
If there is no state s_{i+1} then $\mathrm{X}\,\varphi$ never holds.



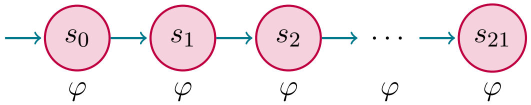
Weak Next: $\overline{\mathrm{X}}\,\varphi$

The formula $\overline{\mathrm{X}}\,\varphi$ holds in state s_i if φ holds in state s_{i+1}.
If there is no state s_{i+1} then $\overline{\mathrm{X}}\,\varphi$ always holds.



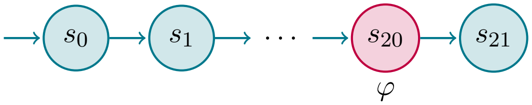
Globally: $\mathrm{G}\,\varphi$

The formula $\mathrm{G}\,\varphi$ holds in state s_i if φ holds in all states s_j for $j \geq i$.



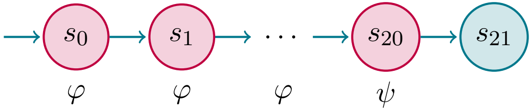
Finally: $\mathrm{F}\,\varphi$

The formula $\mathrm{F}\,\varphi$ holds in state s_i if there is a state s_j for $j \geq i$ in which φ holds.


$$\begin{tikzcd}
{} \arrow[r] & s_0 \arrow[r] & s_1 \arrow[r] & \cdots \arrow[r] & \underset{\varphi}{s_{20}} \arrow[r] & s_{21}
\end{tikzcd}$$

Until: $\varphi \,\mathrm{U}\, \psi$

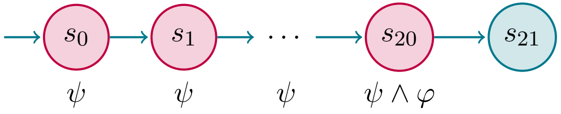
The formula $\varphi \,\mathrm{U}\, \psi$ holds in state s_i if there is a state s_j for $j \geq i$ in which ψ holds and φ holds in all states s_k for $i \leq k < j$.



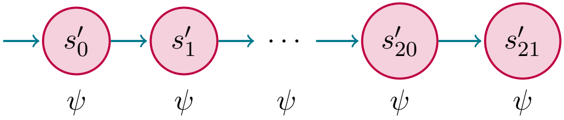
Notice that a state in which φ holds is not required in all cases!

Release: $\varphi \,\mathrm{R}\, \psi$

The formula $\varphi \,\mathrm{R}\, \psi$ holds in state s_i if there is a state s_j for $j \geq i$ in which φ holds and ψ holds in all states s_k for $i \leq k \leq j$.
If there is no such state s_j then the $\varphi \,\mathrm{R}\, \psi$ holds if ψ holds in all states s_k for $k \geq i$.



or



Conclusion

1. The execution of a system is a word over the alphabet $\Sigma = 2^{\mathrm{AP}}$ where AP is the set of atomic propositions.
2. A correctness property is a language describing a set of executions.
3. Regular expressions describe regular languages and could be used to describe regular correctness properties.
4. Linear Temporal Logic (LTL) describes a subset of regular languages but is much better suited to describe correctness properties for runtime verification: Negation and Conjunction of LTL allows often to express correctness properties in a simple manner.

Høgskulen på Vestlandet

UNIVERSITÄT ZU LÜBECK
INSTITUTE FOR SOFTWARE ENGINEERING
AND PROGRAMMING LANGUAGES

Chapter 4

LTL on Finite Words

Course “Runtime Verification”
M. Leucker & V. Stolz

Høgskulen på Vestlandet

UNIVERSITÄT ZU LÜBECK
INSTITUTE FOR SOFTWARE ENGINEERING AND PROGRAMMING LANGUAGES

Chapter 4

Learning Targets of Chapter "LTL on Finite Words".

1. Learn about LTL.
2. Understand the LTL syntax.
3. Understand the LTL semantics on finite words: FLTL.
4. See how RV can be implemented using FLTL and learn about monitors for finite, terminated traces.

Høgskulen på Vestlandet

UNIVERSITÄT ZU LÜBECK
INSTITUTE FOR SOFTWARE ENGINEERING
AND PROGRAMMING LANGUAGES

Chapter 4

Outline of Chapter “LTL on Finite Words”.

Section

LTL Syntax

LTL
SALT

Chapter 4 "LTL on Finite Words"
Course "Runtime Verification"
M. Leucker & V. Stolz

Recall: Specify Correctness Properties

Observing Executions

Recall: Specify Correctness Properties

Observing Executions

Idea

Specify correctness properties in Linear Temporal Logic (LTL).

Recall: Specify Correctness Properties

Observing Executions

Idea

Specify correctness properties in Linear Temporal Logic (LTL).

Commercial

Specify correctness properties in Regular Linear Temporal Logic (RLTL).

Syntax of LTL Formulae

Definition (Syntax of LTL Formulae)

Let $p \in \mathrm{AP}$ be an atomic proposition from a finite set of atomic propositions AP. The set of LTL formulae is inductively defined by the following grammar:

$$\begin{array}{rcl} \varphi & ::= & \mathrm{true} \mid p \quad \mid \varphi \vee \varphi \mid \mathrm{X}\,\varphi \mid \varphi\,\mathrm{U}\,\varphi \mid \mathrm{F}\,\varphi \mid \\ & & \mathrm{false} \mid \neg p \mid \varphi \wedge \varphi \mid \overline{\mathrm{X}}\,\varphi \mid \varphi\,\mathrm{R}\,\varphi \mid \mathrm{G}\,\varphi \mid \\ & & \neg\,\varphi \end{array}$$

Order of Operations

The operator precedence is needed to determine an unambiguous derivation of an LTL formula if braces are left out in nested expressions. The higher the rank of an operator is the later it is derivated.

Braces only need to be added if an operator of lower or same rank should be derivated later than the current one.

Example (operator precedence of arithmetic)

1. exponential operator: $\bullet^{\bullet}$
2. multiplicative operators: $\cdot, /$
3. additive operators: $+, -$

Order of Operations

Definition (operator precedence of LTL)

1. negation operator: $\neg$
2. unary temporal operators: $\mathrm{X}, \overline{\mathrm{X}}, \mathrm{G}, \mathrm{F}$
3. binary temporal logic operators: U, R
4. conjunction operator: $\wedge$
5. disjunction operator: $\vee$

Example

$$\begin{aligned} & \mathrm{G}\ \neg x\ \vee\ \ \neg x\ \mathrm{U}\ \mathrm{G}\, y\ \ \wedge z \\ \equiv\ & \mathrm{G}\,(\neg x) \vee \Big(\big((\neg x)\, \mathrm{U}\, (\mathrm{G}\, y) \big) \wedge z \Big) \end{aligned}$$

LTL for the Working Engineer?

Simple?

LTL is for theoreticians—but for practitioners?

LTL for the Working Engineer?

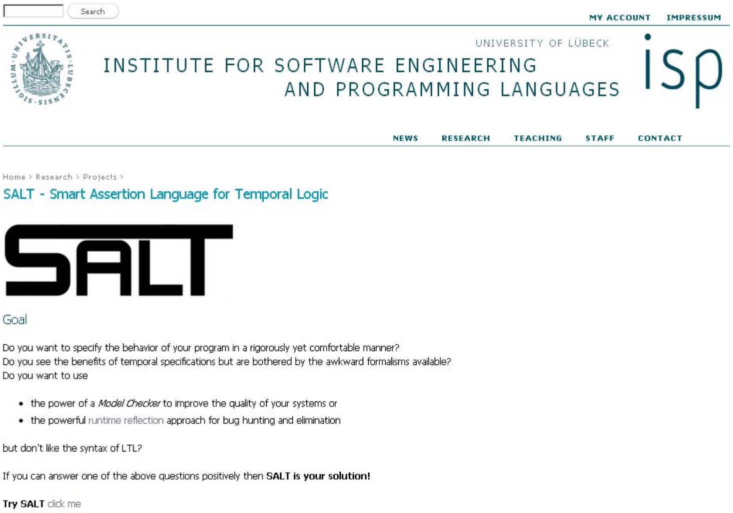
Simple?

LTL is for theoreticians—but for practitioners?

SALT

Structured Assertion Language for Temporal Logic
$\Rightarrow$ Syntactic Sugar for LTL

www.isp.uni-luebeck.de/salt


Search
MY ACCOUNT
IMPRESSUM
UNIVERSITY OF LÜBECK
INSTITUTE FOR SOFTWARE ENGINEERING AND PROGRAMMING LANGUAGES
isp
NEWS
RESEARCH
TEACHING
STAFF
CONTACT
Home > Research > Projects >

SALT - Smart Assertion Language for Temporal Logic

SALT

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

but don't like the syntax of LTL?

If you can answer one of the above questions positively then **SALT is your solution!**

Try SALT click me

Høgskulen på Vestlandet

UNIVERSITÄT ZU LÜBECK
INSTITUTE FOR SOFTWARE ENGINEERING
AND PROGRAMMING LANGUAGES

Section

FLTL Semantics

Chapter 4 “LTL on Finite Words”
Course “Runtime Verification”
M. Leucker & V. Stolz

Parts of Words

In the formal definition of LTL semantics we denote parts of a word as follows:

Let $w = a_1 a_2 \dots a_n \in \Sigma^n$ be a finite word over the alphabet $\Sigma = 2^{\mathrm{AP}}$ and let $i \in \mathbb{N}$ with $1 \leq i \leq n$ be a position in this word. Then

- $|w| := n$ is the length of the word,
- $w_i = a_i$ is the i-th letter of the word and
- $w^i = a_i a_{i+1} \dots a_n$ is the subword starting with letter i.

FLTL Semantics

Definition (FLTL Semantics)

Let φ, ψ be LTL formulae and let $w \in \Sigma^+$ be a finite word. Then the semantics of φ with respect to w is inductively defined as follows:

$$\begin{array}{ll}
w \models \mathrm{true} & \\
w \models p & \text{iff } p \in w_1 \\
w \models \neg p & \text{iff } p \notin w_1 \\
w \models \neg \varphi & \text{iff } w \not\models \varphi \\
w \models \varphi \vee \psi & \text{iff } w \models \varphi \text{ or } w \models \psi \\
w \models \varphi \wedge \psi & \text{iff } w \models \varphi \text{ and } w \models \psi
\end{array}$$

FLTL Semantics

Definition (FLTL Semantics)

Let φ, ψ be LTL formulae and let $w \in \Sigma^+$ be a finite word. Then the semantics of φ with respect to w is inductively defined as follows:

$$
\begin{aligned}
w \models \mathrm{X}\,\varphi \quad & \text{iff } |w| > 1 \\
& \quad \text{and, for } |w| > 1, w^2 \models \varphi \\
w \models \overline{\mathrm{X}}\,\varphi \quad & \text{iff } |w| = 1 \\
& \quad \text{or, for } |w| > 1, w^2 \models \varphi
\end{aligned}
$$

FLTL Semantics

Definition (FLTL Semantics)

Let φ, ψ be LTL formulae and let $w \in \Sigma^+$ be a finite word. Then the semantics of φ with respect to w is inductively defined as follows:

$$\begin{aligned} w \models \varphi \,\mathrm{U}\, \psi \quad & \text{iff } \exists i, 1 \le i \le |w| : (w^i \models \psi \\ & \qquad \text{and } \forall k, 1 \le k < i : w^k \models \varphi) \\ w \models \varphi \,\mathrm{R}\, \psi \quad & \text{iff } \exists i, 1 \le i \le |w| : (w^i \models \varphi \\ & \qquad \text{and } \forall k, 1 \le k \le i : w^k \models \psi) \\ & \quad \text{or } \forall i, 1 \le i \le |w| : w^i \models \psi \end{aligned}$$

FLTL Semantics

Definition (FLTL Semantics)

Let φ, ψ be LTL formulae and let $w \in \Sigma^+$ be a finite word. Then the semantics of φ with respect to w is inductively defined as follows:

$$w \models \mathrm{F}\,\varphi \qquad \text{iff } \exists i, 1 \leq i \leq |w| : w^i \models \varphi$$
$$w \models \mathrm{G}\,\varphi \qquad \text{iff } \forall i, 1 \leq i \leq |w| : w^i \models \varphi$$

Finally and Globally Examples

Examples (Finally and Globally)

Consider words over the alphabet $\Sigma = 2^{\mathrm{AP}}$ with $\mathrm{AP} = \{p, q\}$.

- $\{p\}\emptyset\{q\}\emptyset \models \mathrm{F}\, q$.

Finally and Globally Examples

Examples (Finally and Globally)

Consider words over the alphabet $\Sigma = 2^{\mathrm{AP}}$ with $\mathrm{AP} = \{p, q\}$.

- $\{p\}\emptyset\{q\}\emptyset \models \mathrm{F}\, q$.
- $\{q\}\{q\}\{p, q\}\{q\}\{q\} \models \mathrm{G}\, q$.

Finally and Globally Examples

Examples (Finally and Globally)

Consider words over the alphabet $\Sigma = 2^{\mathrm{AP}}$ with $\mathrm{AP} = \{p, q\}$.

- $\{p\}\emptyset\{q\}\emptyset \models \mathrm{F}\, q$.
- $\{q\}\{q\}\{p, q\}\{q\}\{q\} \models \mathrm{G}\, q$.
- $\emptyset\{p\}\{p, q\}\emptyset\{q\}\emptyset\{q\} \models \mathrm{G}\,\mathrm{F}\, q$.

Finally and Globally Examples

Examples (Finally and Globally)

Consider words over the alphabet $\Sigma = 2^{\mathrm{AP}}$ with $\mathrm{AP} = \{p, q\}$.

- $\{p\}\emptyset\{q\}\emptyset \models \mathrm{F}\, q$.
- $\{q\}\{q\}\{p,q\}\{q\}\{q\} \models \mathrm{G}\, q$.
- $\emptyset\{p\}\{p,q\}\emptyset\{q\}\emptyset\{q\} \models \mathrm{G}\,\mathrm{F}\, q$.
- $\{p\}\emptyset\{q\}\{p\}\{p,q\}\{p,q\}\{q\} \models \mathrm{F}\,\mathrm{G}\, q$.

Finally and Globally Examples

Examples (Finally and Globally)

Consider words over the alphabet $\Sigma = 2^{\mathrm{AP}}$ with $\mathrm{AP} = \{p, q\}$.

- $\{p\}\emptyset\{q\}\emptyset \models \mathrm{F}\, q$.
- $\{q\}\{q\}\{p,q\}\{q\}\{q\} \models \mathrm{G}\, q$.
- $\emptyset\{p\}\{p,q\}\emptyset\{q\}\emptyset\{q\} \models \mathrm{G}\,\mathrm{F}\, q$.
- $\{p\}\emptyset\{q\}\{p\}\{p,q\}\{p,q\}\{q\} \models \mathrm{F}\,\mathrm{G}\, q$.

- $\mathrm{G}\,\mathrm{F}\,\varphi$ can be read as: For every state (globally) there will be a state in the future (finally) in that φ holds.
- $\mathrm{F}\,\mathrm{G}\,\varphi$ can be read as: There will be a state in the future (finally) that φ holds in every state (globally).

Practical Examples

In the following examples we consider these scopes:

everytime: all states

before ψ: all states before the first state in which ψ holds (if there is such a state)

after ψ: all states after and including the first state in which ψ holds (if there is such a state)

Example (Absence)

The formula φ does not hold

everytime: $\mathrm{G}\,\neg\,\varphi$

before ψ: $(\mathrm{F}\,\psi) \rightarrow (\neg\,\varphi\,\mathrm{U}\,\psi)$

after ψ: $\mathrm{G}(\psi \rightarrow (\mathrm{G}\,\neg\,\varphi))$

Practical Examples

In the following examples we consider these scopes:

everytime:	all states
before ψ:	all states before the first state in which ψ holds (if there is such a state)
after ψ:	all states after and including the first state in which ψ holds (if there is such a state)

Example (Existence)

The formula φ holds in the future

everytime:	$\mathrm{F}\,\varphi$
before ψ:	$\mathrm{G}\,\neg\,\psi \vee \neg\,\psi\,\mathrm{U}(\varphi \wedge \neg\,\psi)$
after ψ:	$\mathrm{G}\,\neg\,\psi \vee \mathrm{F}(\psi \wedge \mathrm{F}\,\varphi)$

Practical Examples

In the following examples we consider these scopes:

everytime: all states

before ψ: all states before the first state in which ψ holds (if there is such a state)

after ψ: all states after and including the first state in which ψ holds (if there is such a state)

Example (Universality)

The formula φ holds

everytime: $\mathrm{G}\,\varphi$

before ψ: $(\mathrm{F}\,\psi) \rightarrow (\varphi\,\mathrm{U}\,\psi)$

after ψ: $\mathrm{G}(\psi \rightarrow \mathrm{G}\,\varphi)$

Equivalences

Definition (Equivalence of Formulae)

Let $\Sigma = 2^{\mathrm{AP}}$ and φ and ψ be LTL formulae over AP. φ and ψ are *equivalent*, denoted by $\varphi \equiv \psi$, iff

$$\forall w \in \Sigma^+ : w \models \varphi \Leftrightarrow w \models \psi.$$

Globally and finally can easily be expressed using until and release:

$$\mathrm{F}\,\varphi \equiv \mathrm{true}\,\mathrm{U}\,\varphi$$
$$\mathrm{G}\,\varphi \equiv \mathrm{false}\,\mathrm{R}\,\varphi$$

De Morgan Rules

The negation can always be moved in front of the atomic propositions using the dual operators:

De Morgan Rules of Propositional Logic

$$\neg(\varphi \lor \psi) \equiv \neg\,\varphi \land \neg\,\psi$$
$$\neg(\varphi \land \psi) \equiv \neg\,\varphi \lor \neg\,\psi$$

De Morgan Rules

The negation can always be moved in front of the atomic propositions using the dual operators:

De Morgan Rules of Temporal Logic

$$
\begin{aligned}
\neg(\varphi \,\mathrm{U}\, \psi) &\equiv \neg\varphi \,\mathrm{R}\, \neg\psi \\
\neg(\varphi \,\mathrm{R}\, \psi) &\equiv \neg\varphi \,\mathrm{U}\, \neg\psi \\
\neg(\mathrm{G}\,\varphi) &\equiv \mathrm{F}\,\neg\varphi \\
\neg(\mathrm{F}\,\varphi) &\equiv \mathrm{G}\,\neg\varphi \\
\neg(\mathrm{X}\,\varphi) &\equiv \overline{\mathrm{X}}\,\neg\varphi \\
\neg(\overline{\mathrm{X}}\,\varphi) &\equiv \mathrm{X}\,\neg\varphi
\end{aligned}
$$

Fixed Point Equations

The following fixed point equations can be used to step-wise unwind until and release:

$$\varphi \,\mathrm{U}\, \psi \equiv \psi \vee (\varphi \wedge \mathrm{X}(\varphi \,\mathrm{U}\, \psi))$$
$$\varphi \,\mathrm{R}\, \psi \equiv \psi \wedge (\varphi \vee \overline{\mathrm{X}}(\varphi \,\mathrm{R}\, \psi))$$

Consequently such fix point equations for globally and finally are special cases of the above ones:

$$\mathrm{G}\, \varphi \equiv \varphi \wedge \overline{\mathrm{X}}(\mathrm{G}\, \varphi)$$
$$\mathrm{F}\, \varphi \equiv \varphi \vee \mathrm{X}(\mathrm{F}\, \varphi)$$

Negation Normal Form (NNF)

Definition (Negation Normal Form (NNF))

An LTL formula φ is in Negation Normal Form (NNF) iff $\neg$ only occurs in front of atomic propositions $p \in \mathrm{AP}$.

Negation Normal Form (NNF)

Definition (Negation Normal Form (NNF))

An LTL formula φ is in Negation Normal Form (NNF) iff $\neg$ only occurs in front of atomic propositions $p \in \mathrm{AP}$.

Lemma

For every LTL formula there exists an equivalent formula in NNF.

Proof.

Recursively apply De Morgan rules of propositional logic and De Morgan rules of temporal logic. □

Høgskulen på Vestlandet

UNIVERSITÄT ZU LÜBECK
INSTITUTE FOR SOFTWARE ENGINEERING
AND PROGRAMMING LANGUAGES

Section

Monitor Function for FLTL

The Idea

Definition

Chapter 4 “LTL on Finite Words”
Course “Runtime Verification”
M. Leucker & V. Stolz

The Idea

Build up a function that

- takes an LTL formula φ in NNF and a word $w \in \Sigma^+$,
- performs recursion on the structure of φ
- returns true iff $w \models \varphi$.

First Ideas

Let $p \in \mathrm{AP}$ be an atomic proposition and $w \in \Sigma^+$ a word.

We then can evaluate

- true and false.
- $\varphi \vee \psi$ by evaluating φ, evaluating ψ and computing $\varphi \vee \psi$.
- p by checking if $p \in w_1$.
- $\neg p$ by checking if $p \notin w_1$.

Further Ideas

What about next?

We can check if $w \models \mathrm{X}\,\varphi$ holds by omitting

- the first letter of w and
- the next operator

and checking if $w^2 \models \varphi$ holds.

Further Ideas

What about next?

We can check if $w \models \mathrm{X}\,\varphi$ holds by omitting

- the first letter of w and
- the next operator

and checking if $w^2 \models \varphi$ holds.

What about until and release?

Use the already presented fixpoint equations and the above ideas to evaluate conjunction, disjunction and next.

$$\varphi \,\mathrm{U}\, \psi \equiv \psi \vee (\varphi \wedge \mathrm{X}(\varphi \,\mathrm{U}\, \psi))$$
$$\varphi \,\mathrm{R}\, \psi \equiv \psi \wedge (\varphi \vee \overline{\mathrm{X}}(\varphi \,\mathrm{R}\, \psi))$$

evlFLTL

Let $\Sigma = 2^{\mathrm{AP}}$ be the finite alphabet, $p \in \mathrm{AP}$ an atomic proposition, $w \in \Sigma^+$ a finite non-empty word, φ and ψ LTL formulae and $\mathbb{B}_2 = \{\top, \bot\}$.

We then define the function $\mathrm{evlFLTL} : \Sigma^+ \times \mathrm{LTL} \to \mathbb{B}_2$ inductively as follows:

$$\begin{aligned}
\mathrm{evlFLTL}(w, \mathrm{true}) &= \top \\
\mathrm{evlFLTL}(w, \mathrm{false}) &= \bot \\
\mathrm{evlFLTL}(w, \varphi \vee \psi) &= \mathrm{evlFLTL}(w, \varphi) \vee \mathrm{evlFLTL}(w, \psi) \\
\mathrm{evlFLTL}(w, \varphi \wedge \psi) &= \mathrm{evlFLTL}(w, \varphi) \wedge \mathrm{evlFLTL}(w, \psi) \\
\mathrm{evlFLTL}(w, p) &= (p \in w_1) \\
\mathrm{evlFLTL}(w, \neg p) &= (p \notin w_1)
\end{aligned}$$

evlFLTL

Let $\Sigma = 2^{\mathrm{AP}}$ be the finite alphabet, $p \in \mathrm{AP}$ an atomic proposition, $w \in \Sigma^+$ a finite non-empty word, φ and ψ LTL formulae and $\mathbb{B}_2 = \{\top, \bot\}$.

We then define the function $\mathrm{evlFLTL} : \Sigma^+ \times \mathrm{LTL} \to \mathbb{B}_2$ inductively as follows:

$$\begin{aligned}
\mathrm{evlFLTL}(w, \varphi \,\mathrm{U}\, \psi) &= \mathrm{evlFLTL}(w, \psi \vee (\varphi \wedge \mathrm{X}(\varphi \,\mathrm{U}\, \psi)))\\
\mathrm{evlFLTL}(w, \varphi \,\mathrm{R}\, \psi) &= \mathrm{evlFLTL}(w, \psi \wedge (\varphi \vee \overline{\mathrm{X}}(\varphi \,\mathrm{R}\, \psi)))\\
\mathrm{evlFLTL}(w, \mathrm{F}\, \varphi) &= \mathrm{evlFLTL}(w, \varphi \vee \mathrm{X}\,\mathrm{F}\, \varphi)\\
\mathrm{evlFLTL}(w, \mathrm{G}\, \varphi) &= \mathrm{evlFLTL}(w, \varphi \wedge \overline{\mathrm{X}}\,\mathrm{G}\, \varphi)\\
\mathrm{evlFLTL}(w, \mathrm{X}\, \varphi) &= (|w| > 1) \wedge \mathrm{evlFLTL}(w^2, \varphi)\\
\mathrm{evlFLTL}(w, \overline{\mathrm{X}}\, \varphi) &= (|w| = 1) \vee \mathrm{evlFLTL}(w^2, \varphi)
\end{aligned}$$

Conclusion

1. The LTL operations negation, disjunction, until and next are enough to gain the full expressiveness of LTL.
2. If you add the dual operators every LTL formula has a negation normal form (NNF).
3. The fix point equations can be used to step-wise unwind until and release using next and weak next.
4. $\mathrm{evlFLTL}$ is as inductively defined function that answers the question if a given finite non-empty word models a correctness property given as an LTL formula in NNF and can easily be implemented recursively.

Høgskulen på Vestlandet

UNIVERSITÄT ZU LÜBECK
INSTITUTE FOR SOFTWARE ENGINEERING
AND PROGRAMMING LANGUAGES

Chapter 5

Impartial Runtime Verification

Course “Runtime Verification”
M. Leucker & V. Stolz

Chapter 5

Learning Targets of Chapter “Impartial Runtime Verification”.

1. Understand the idea of impartiality and why we want to use impartial evaluation of LTL formulae.
2. Learn the basics of truth domains and lattices.
3. Understand the four-valued LTL semantics on finite words: FLTL_4.
4. See how impartial RV can be implemented using FLTL_4 and learn about automata based monitors for finite, non-completed traces.

Chapter 5

Outline of Chapter “Impartial Runtime Verification”.

Høgskulen på Vestlandet

UNIVERSITÄT ZU LÜBECK
INSTITUTE FOR SOFTWARE ENGINEERING
AND PROGRAMMING LANGUAGES

Section

Truth Domains

Chapter 5 “Impartial Runtime Verification”
Course “Runtime Verification”
M. Leucker & V. Stolz

Words Aren't Terminated— They Are Growing

- In the last chapters we considered finite terminated words.
- A monitor for RV does not get a formula and a finite terminated word.
- A monitor for RV gets a formula and one letter after another.
- With every new system state the monitor gets one more letter.

Examples of Evaluating Growing Words

Consider

- the alphabet $\Sigma = 2^{\mathrm{AP}}$ where $\mathrm{AP} = \{p, q\}$,
- the properties $\varphi = \mathrm{G}\, p$ and $\varphi' = \mathrm{F}\, p$ and
- words w and w' growing with every new state.

Lets watch monitors for RV at work:

$$w = \underbrace{\{p, q\}}_{w \models \varphi}$$

Examples of Evaluating Growing Words

Consider

- the alphabet $\Sigma = 2^{\mathrm{AP}}$ where $\mathrm{AP} = \{p, q\}$,
- the properties $\varphi = \mathrm{G}\, p$ and $\varphi' = \mathrm{F}\, p$ and
- words w and w' growing with every new state.

Lets watch monitors for RV at work:

$$w = \{p,q\}\ \{p\}$$

$$\underbrace{\{p,q\}}_{w \models \varphi}$$

$$\underbrace{\{p,q\}\ \{p\}}_{w \models \varphi}$$

Examples of Evaluating Growing Words

Consider

- the alphabet $\Sigma = 2^{\mathrm{AP}}$ where $\mathrm{AP} = \{p, q\}$,
- the properties $\varphi = \mathrm{G}\, p$ and $\varphi' = \mathrm{F}\, p$ and
- words w and w' growing with every new state.

Lets watch monitors for RV at work:

$$w = \{p,q\}\ \{p\}\ \{q\}$$

$$\underbrace{\{p,q\}}_{w \models \varphi}$$

$$\underbrace{\{p,q\}\ \{p\}}_{w \models \varphi}$$

$$\underbrace{\{p,q\}\ \{p\}\ \{q\}}_{w \not\models \varphi}$$

Examples of Evaluating Growing Words

Consider

- the alphabet $\Sigma = 2^{\mathrm{AP}}$ where $\mathrm{AP} = \{p, q\}$,
- the properties $\varphi = \mathrm{G}\, p$ and $\varphi' = \mathrm{F}\, p$ and
- words w and w' growing with every new state.

Lets watch monitors for RV at work:

$$w = \{p, q\}\ \{p\}\ \{q\}\ \{p\}$$

$$\underbrace{\{p, q\}}_{w \models \varphi}$$

$$\underbrace{\{p, q\}\ \{p\}}_{w \models \varphi}$$

$$\underbrace{\{p, q\}\ \{p\}\ \{q\}}_{w \not\models \varphi}$$

$$\underbrace{\{p, q\}\ \{p\}\ \{q\}\ \{p\}}_{w \not\models \varphi}$$

Examples of Evaluating Growing Words

Consider

- the alphabet $\Sigma = 2^{\mathrm{AP}}$ where $\mathrm{AP} = \{p, q\}$,
- the properties $\varphi = \mathrm{G}\, p$ and $\varphi' = \mathrm{F}\, p$ and
- words w and w' growing with every new state.

Lets watch monitors for RV at work:

$$w = \{p,q\}\ \{p\}\ \{q\}\ \{p\} \qquad\qquad w' = \{q\}$$

$$\underbrace{\{p,q\}}_{w \models \varphi} \qquad\qquad \underbrace{\{q\}}_{w' \not\models \varphi'}$$

$$\underbrace{\{p,q\}\ \{p\}}_{w \models \varphi}$$

$$\underbrace{\{p,q\}\ \{p\}\ \{q\}}_{w \not\models \varphi}$$

$$\underbrace{\{p,q\}\ \{p\}\ \{q\}\ \{p\}}_{w \not\models \varphi}$$

Examples of Evaluating Growing Words

Consider

- the alphabet $\Sigma = 2^{\mathrm{AP}}$ where $\mathrm{AP} = \{p, q\}$,
- the properties $\varphi = \mathrm{G}\, p$ and $\varphi' = \mathrm{F}\, p$ and
- words w and w' growing with every new state.

Lets watch monitors for RV at work:

$w = \{p, q\}\ \{p\}\ \{q\}\ \{p\}$

$w \models \varphi$

$w \models \varphi$

$w \not\models \varphi$

$w \not\models \varphi$

$w' = \{q\}\ \{q\}$

$w' \not\models \varphi'$

$w' \not\models \varphi'$

Examples of Evaluating Growing Words

Consider

- the alphabet $\Sigma = 2^{\mathrm{AP}}$ where $\mathrm{AP} = \{p, q\}$,
- the properties $\varphi = \mathrm{G}\, p$ and $\varphi' = \mathrm{F}\, p$ and
- words w and w' growing with every new state.

Lets watch monitors for RV at work:

$w = \{p,q\}\ \{p\}\ \{q\}\ \{p\}$

$\underbrace{\{p,q\}}$: $w \models \varphi$

$\underbrace{\{p,q\}\ \{p\}}$: $w \models \varphi$

$\underbrace{\{p,q\}\ \{p\}\ \{q\}}$: $w \not\models \varphi$

$\underbrace{\{p,q\}\ \{p\}\ \{q\}\ \{p\}}$: $w \not\models \varphi$

$w' = \{q\}\ \{q\}\ \{p,q\}$

$\underbrace{\{q\}}$: $w' \not\models \varphi'$

$\underbrace{\{q\}\ \{q\}}$: $w' \not\models \varphi'$

$\underbrace{\{q\}\ \{q\}\ \{p,q\}}$: $w' \models \varphi'$

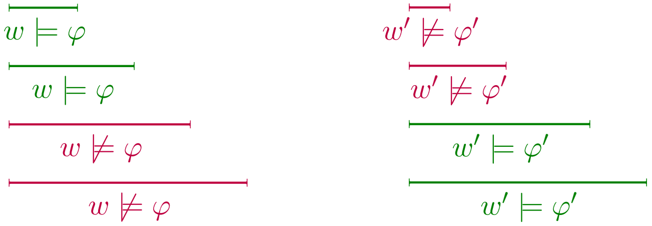
Examples of Evaluating Growing Words

Consider

- the alphabet $\Sigma = 2^{\mathrm{AP}}$ where $\mathrm{AP} = \{p, q\}$,
- the properties $\varphi = \mathrm{G}\, p$ and $\varphi' = \mathrm{F}\, p$ and
- words w and w' growing with every new state.

Lets watch monitors for RV at work:

$w = \{p,q\}\ \{p\}\ \{q\}\ \{p\}$ $\qquad$ $w' = \{q\}\ \{q\}\ \{p,q\}\ \{p\}$



Impartiality

Be Impartial!

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

Definition (Impartiality)

Impartiality requires that a finite trace is not evaluated to $true$ or, respectively $false$, if there still exists an (possibly infinite) continuation leading to another verdict.

Semantic Function

Definition (Semantic Function)

The semantic function

$$\mathrm{sem}_k : \Sigma^+ \times \mathrm{LTL} \to \mathbb{B}_k$$

maps a word $w \in \Sigma^+$ and a an LTL formula φ to a logic value $b \in \mathbb{B}_k$.

We use $[\![w \models \varphi]\!]_k = b$ instead of $\mathrm{sem}_k(w, \varphi) = b$.

Semantic Function for FLTL

- We defined the FLTL semantics as relation $w \models \varphi$ between a word $w \in \Sigma^+$ and an LTL formula φ.
- This can be interpreted as semantic function

$$\mathrm{sem}_2 : \Sigma^+ \times \mathrm{LTL} \to \mathbb{B}_2,$$

$$\mathrm{sem}_2(w, \varphi) = [\![w \models \varphi]\!]_2 := \begin{cases} \top & \text{if } w \models \varphi \\ \bot & \text{else.} \end{cases}$$

Impartiality

Definition (Impartial Semantics)

Let $\Sigma = 2^{\mathrm{AP}}$ be an alphabet, $w \in \Sigma^+$ a word and φ an LTL formula. A semantic function is called impartial iff for all $u \in \Sigma^*$

$$\llbracket w \models \varphi \rrbracket = \top \text{ implies } \llbracket wu \models \varphi \rrbracket = \top$$
$$\llbracket w \models \varphi \rrbracket = \bot \text{ implies } \llbracket wu \models \varphi \rrbracket = \bot.$$

Target

Create monitors which only answer $\top$ or $\bot$ if the result keeps stable for a growing word.

We Need Multiple Values

FLTL semantics are not impartial:

$$w = \{a\}, \varphi = \mathrm{G}\, a \text{ and } wu = \{a\}\{b\}$$

is a counterexample for

$$[\![w \models \varphi]\!] = \top \text{ implies } [\![wu \models \varphi]\!] = \top.$$

We Need Multiple Values

FLTL semantics are not impartial:

$$w = \{a\}, \varphi = \mathrm{G}\, a \text{ and } wu = \{a\}\{b\}$$

is a counterexample for

$$[\![w \models \varphi]\!] = \top \text{ implies } [\![wu \models \varphi]\!] = \top.$$

Impartiality implies multiple values

Every two-valued logic is not impartial.

- Impartiality forbids switching from $\top$ to $\bot$ and vice versa.
- Therefore we need more logic values than $\top$ and $\bot$.

Lattice

Definition (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 $x \sqcup y$.

If the ordering relation $\sqsubseteq$ is obvious we denote the lattice with the set $\mathcal{L}$.

Finite Lattice

Definition (Finite Lattice)

A lattice $(\mathcal{L}, \sqsubseteq)$ is called *finite* iff $\mathcal{L}$ is finite.

Every non-empty finite lattice has two well-defined unique elements:

- A least element, called *bottom*, denoted with $\bot$ and
- a greatest element, called *top*, denoted with $\top$.

Hasse diagram

- Hasse diagrams are used to represent a finite partially ordered set.
- Each element of the set is represented as a vertex in the plane.
- For all $x, y \in \mathcal{L}$ where $x \sqsubseteq y$ but no $z \in \mathcal{L}$ exists where $x \sqsubseteq z \sqsubseteq y$ a line that goes upward from x to y is drawn.

Example

Hasse diagram for $\mathbb{B}_2 = \{\bot, \top\}$ with $\bot \sqsubseteq \top$:

$$\begin{tikzcd}
\top \arrow[d, no head] \\
\bot
\end{tikzcd}$$

Example Lattices

$\mathbb{B}_2$:

$$\begin{tikzcd}
\top \arrow[d, dash] \\
\bot
\end{tikzcd}$$

$\mathbb{B}_{2\times 2}$:

$$\begin{tikzcd}
 & \top \arrow[dl, dash] \arrow[dr, dash] & \\
(\top, \bot) \arrow[dr, dash] & & (\bot, \top) \arrow[dl, dash] \\
 & \bot &
\end{tikzcd}$$

$\mathbb{B}_{2\times 2\times 2}$:

$$\begin{tikzcd}
 & \top \arrow[dl, dash] \arrow[d, dash] \arrow[dr, dash] & \\
(\top, \top, \bot) \arrow[d, dash] \arrow[dr, dash] & (\top, \bot, \top) \arrow[dl, dash] \arrow[dr, dash] & (\bot, \top, \top) \arrow[dl, dash] \arrow[d, dash] \\
(\top, \bot, \bot) \arrow[dr, dash] & (\bot, \top, \bot) \arrow[d, dash] & (\bot, \bot, \top) \arrow[dl, dash] \\
 & \bot &
\end{tikzcd}$$

Example Lattices II

$$\begin{tikzcd}[row sep=small]
\mathbb{B}_2: & \top \arrow[d, dash] & & \mathbb{B}_3: & \top \arrow[d, dash] & & \mathbb{B}_4: & \top \arrow[d, dash] \\
& \bot & & & ? \arrow[d, dash] & & & \top^p \arrow[d, dash] \\
& & & & \bot & & & \bot^p \arrow[d, dash] \\
& & & & & & & \bot
\end{tikzcd}$$

Distributive Lattices

Definition (Distributive Lattices)

A lattice $(\mathcal{L}, \sqsubseteq)$ is called a *distributive lattice* iff we have for all elements $x, y, z \in \mathcal{L}$

$$x \sqcap (y \sqcup z) = (x \sqcap y) \sqcup (x \sqcap z) \text{ and}$$
$$x \sqcup (y \sqcap z) = (x \sqcup y) \sqcap (x \sqcup z).$$

De Morgan Lattice

Definition (De Morgan Lattice)

A distributive lattice $(\mathcal{L}, \sqsubseteq)$ is called a *De Morgan lattice* iff every element $x \in \mathcal{L}$ has a unique *dual* element $\overline{x}$, such that

$$\overline{\overline{x}} = x \text{ and } x \sqsubseteq y \text{ implies } \overline{y} \sqsubseteq \overline{x}.$$

Boolean Lattice

Definition (Boolean Lattice)

A De Morgan lattice is called *Boolean lattice* iff for every element x and its dual element $\overline{x}$ we have

$$x \sqcup \overline{x} = \top \text{ and } x \sqcap \overline{x} = \bot.$$

Every Boolean lattice has 2^n elements for some $n \in \mathbb{N}$.

Truth Domain

Definition (Truth Domain)

A *Truth Domain* is a finite De Morgan Lattice.

Examples (Truth Domains)

The following lattices are all Truth Domains:

- $\mathbb{B}_2 = \{\top, \bot\}$ with $\bot \sqsubseteq \top$ and $\overline{\top} = \bot$ and $\overline{\bot} = \top$.
- $\mathbb{B}_3 = \{\top, ?, \bot\}$ with $\bot \sqsubseteq ? \sqsubseteq \top$ and $\overline{\top} = \bot$, $\overline{?} = ?$ and $\overline{\bot} = \top$.
- $\mathbb{B}_4 = \{\top, \top^p, \bot^p, \bot\}$ with $\bot \sqsubseteq \bot^p \sqsubseteq \top^p \sqsubseteq \top$ and $\overline{\top} = \bot$, $\overline{\top^p} = \bot^p$, $\overline{\bot^p} = \top^p$ and $\overline{\bot} = \top$.

Section

Four-Valued LTL Semantics: FLTL_4

- Definition
- Monitor Function

Examples of Impartial LTL Semantics

- We want to create impartial four-valued semantics for LTL on finite, non-completed words
- using the truth domain $(\mathbb{B}_4, \sqsubseteq)$.

Examples (FLTL vs. FLTL$_4$)

The indices 2 and 4 denote FLTL resp. FLTL$_4$.

$$
\begin{array}{rr}
[\![\emptyset \models \mathrm{X}\, a]\!]_2 = \bot & [\![\emptyset \models \mathrm{X}\, a]\!]_4 = \bot^p \\
[\![\emptyset\emptyset \models \mathrm{X}\, a]\!]_2 = \bot & [\![\emptyset\emptyset \models \mathrm{X}\, a]\!]_4 = \bot \\
[\![\emptyset\{a\} \models \mathrm{X}\, a]\!]_2 = \top & [\![\emptyset\{a\} \models \mathrm{X}\, a]\!]_4 = \top \\
[\![\emptyset \models \overline{\mathrm{X}}\, a]\!]_2 = \top & [\![\emptyset \models \overline{\mathrm{X}}\, a]\!]_4 = \top^p \\
[\![\emptyset\emptyset \models \overline{\mathrm{X}}\, a]\!]_2 = \bot & [\![\emptyset\emptyset \models \overline{\mathrm{X}}\, a]\!]_4 = \bot \\
[\![\emptyset\{a\} \models \overline{\mathrm{X}}\, a]\!]_2 = \top & [\![\emptyset\{a\} \models \overline{\mathrm{X}}\, a]\!]_4 = \top
\end{array}
$$

How To Create Impartial LTL Semantics?

At the end of the word

- X evaluates to $\bot^p$ instead of $\bot$ and
- $\overline{\mathrm{X}}$ evaluates to $\top^p$ instead of $\top$.

Idea of $\bullet^p$: Semantics if the word ends here.

Fulfilling the introduced equivalences and fix point equations we get at the end of the word:

- U evaluates to $\bot^p$ instead of $\bot$,
- R evaluates to $\top^p$ instead of $\top$,
- F evaluates to $\bot^p$ instead of $\bot$ and
- G evaluates to $\top^p$ instead of $\top$.

Idea of $\bullet^p$: Semantics if the word ends here or goes on like this forever.

Properties of $(\mathbb{B}_4, \sqsubseteq)$

- $(\mathbb{B}_4, \sqsubseteq)$ is no Boolean Lattice.
- Some equivalences in FLTL do not hold in FLTL$_4$.

For any LTL formula φ using FLTL semantics we have

$$\varphi \vee \neg\varphi \equiv_2 \mathrm{true} \text{ and } \varphi \wedge \neg\varphi \equiv_2 \mathrm{false}\,.$$

Examples (FLTL vs. FLTL$_4$)

For any $w \in \Sigma^+$ and $a \in \Sigma$ we have

$$[\![w \models \mathrm{G}\, a \vee \neg\, \mathrm{G}\, a]\!]_2 = \top \qquad [\![w \models \mathrm{G}\, a \vee \neg\, \mathrm{G}\, a]\!]_4 = \top^p$$
$$[\![w \models \mathrm{F}\, a \wedge \neg\, \mathrm{F}\, a]\!]_2 = \bot \qquad [\![w \models \mathrm{F}\, a \wedge \neg\, \mathrm{F}\, a]\!]_4 = \bot^p$$

FLTL_4 Semantics

Definition (FLTL_4 Semantics)

Let φ, ψ be LTL formulae and let $w \in \Sigma^+$ be a finite word. Then the semantics of φ with respect to w is inductively defined as follows:

$$[\![w \models \mathrm{true}]\!]_4 = \top$$

$$[\![w \models \mathrm{false}]\!]_4 = \bot$$

$$[\![w \models p]\!]_4 = \begin{cases} \top & \text{if } p \in w_1 \\ \bot & \text{if } p \notin w_1 \end{cases}$$

$$[\![w \models \neg p]\!]_4 = \begin{cases} \top & \text{if } p \notin w_1 \\ \bot & \text{if } p \in w_1 \end{cases}$$

FLTL$_4$ Semantics

Definition (FLTL$_4$ Semantics)

Let φ, ψ be LTL formulae and let $w \in \Sigma^+$ be a finite word. Then the semantics of φ with respect to w is inductively defined as follows:

$$\begin{aligned} [\![w \models \neg \varphi]\!]_4 &= \overline{[\![w \models \varphi]\!]_4} \\ [\![w \models \varphi \vee \psi]\!]_4 &= [\![w \models \varphi]\!]_4 \sqcup [\![w \models \psi]\!]_4 \\ [\![w \models \varphi \wedge \psi]\!]_4 &= [\![w \models \varphi]\!]_4 \sqcap [\![w \models \psi]\!]_4 \end{aligned}$$

FLTL$_4$ Semantics

Definition (FLTL$_4$ Semantics)

Let φ, ψ be LTL formulae and let $w \in \Sigma^+$ be a finite word. Then the semantics of φ with respect to w is inductively defined as follows:

$$[\![w \models \mathrm{X}\,\varphi]\!]_4 = \begin{cases} [\![w^2 \models \varphi]\!]_4 & \text{if } |w| > 1 \\ \bot^p & \text{else} \end{cases}$$

$$[\![w \models \overline{\mathrm{X}}\,\varphi]\!]_4 = \begin{cases} [\![w^2 \models \varphi]\!]_4 & \text{if } |w| > 1 \\ \top^p & \text{else} \end{cases}$$

FLTL$_4$ Semantics

Definition (FLTL$_4$ Semantics)

Let φ, ψ be LTL formulae and let $w \in \Sigma^+$ be a finite word. Then the semantics of φ with respect to w is inductively defined as follows:

$$\begin{aligned}
&[\![w \models \varphi \,\mathrm{U}\, \psi]\!]_4 \\
&\quad = \left(\bigsqcup_{1 \leq i \leq |w|} \left([\![w^i \models \psi]\!]_4 \sqcap \bigsqcap_{1 \leq j < i} [\![w^j \models \varphi]\!]_4 \right) \right) \\
&\qquad\qquad \sqcup \left(\bot^p \sqcap \bigsqcap_{1 \leq i \leq |w|} [\![w^i \models \varphi]\!]_4 \right)
\end{aligned}$$

FLTL$_4$ Semantics

Definition (FLTL$_4$ Semantics)

Let φ, ψ be LTL formulae and let $w \in \Sigma^+$ be a finite word. Then the semantics of φ with respect to w is inductively defined as follows:

$$\begin{aligned}
&[\![w \models \varphi \,\mathrm{R}\, \psi]\!]_4 \\
&\quad = \left(\bigsqcup_{1 \le i \le |w|} \left([\![w^i \models \varphi]\!]_4 \sqcap \bigsqcap_{1 \le j \le i} [\![w^j \models \psi]\!]_4 \right) \right) \\
&\qquad\qquad\qquad \sqcup \left(\top^p \sqcap \bigsqcap_{1 \le i \le |w|} [\![w^i \models \psi]\!]_4 \right)
\end{aligned}$$

FLTL$_4$ Semantics

Definition (FLTL$_4$ Semantics)

Let φ, ψ be LTL formulae and let $w \in \Sigma^+$ be a finite word. Then the semantics of φ with respect to w is inductively defined as follows:

$$[\![w \models \mathrm{F}\,\varphi]\!]_4 = \bot^p \sqcup \bigsqcup_{1 \leq i \leq |w|} [\![w^i \models \varphi]\!]_4$$

$$[\![w \models \mathrm{G}\,\varphi]\!]_4 = \top^p \sqcap \bigsqcap_{1 \leq i \leq |w|} [\![w^i \models \varphi]\!]_4$$

Equivalences

Definition (Equivalence of Formulae)

Let $\Sigma = 2^{\mathrm{AP}}$ and φ and ψ be LTL formulae over AP. φ and ψ are *equivalent*, denoted by $\varphi \equiv \psi$, iff

$$\forall w \in \Sigma^+ : [\![w \models \varphi]\!] = [\![w \models \psi]\!].$$

The equivalences described in the previous chapter are still valid using the semantic function of FLTL$_4$.

Monitor Function

Left-to-right!

The Idea

Build up a monitor function for evaluating each subsequent letter of non-completed words.

Such a function

- takes an LTL formula φ in NNF and a letter $a \in \Sigma$,
- performs (not recursive) formula rewriting (progression) and
- returns $[\![a \models \varphi]\!]_4$ and a new LTL formula φ' that the next letter has to fulfill.

The Idea of Progression

- Compute only the semantics of the first letter and let someone else do the rest.
- Rewrite the LTL formula to keep track of what is done and what still needs to be checked.
- Thanks to the impartial semantics we don't need to know the whole word to compute a valid semantics.

Examples

Let $w \in \Sigma^+$ be a word and $a \in \Sigma$ a letter.

- We can compute $[\![w \models \mathrm{X}\, a]\!]_4$ by doing nothing and letting someone else check $[\![w^2 \models a]\!]_4$.
- We can compute $[\![w \models a]\!]_4$ by checking $a \in w_1$. Then the LTL formula is over. This is denoted by true or false as new formula.

Further Ideas

We know how to evaluate

- atomic propositions,
- positive operators of propositional logic $(\wedge, \vee)$ and
- next-formulas.

That's it thanks to

- equivalences for G and F,
- De Morgan rules of propositional and temporal logic for negation $(\neg)$ and
- and fixed point equations for U and R.

$\mathrm{evlFLTL}_4$

Let $\Sigma = 2^{\mathrm{AP}}$ be the finite alphabet, $p \in \mathrm{AP}$ an atomic proposition, $a \in \Sigma$ a letter, and φ and ψ LTL formulae.

We then define the function $\mathrm{evlFLTL}_4 : \Sigma \times \mathrm{LTL} \to \mathbb{B}_4 \times \mathrm{LTL}$ inductively as follows:

$$\mathrm{evlFLTL}_4(a, \mathrm{true}) = (\top, \mathrm{true})$$

$$\mathrm{evlFLTL}_4(a, \mathrm{false}) = (\bot, \mathrm{false})$$

$$\mathrm{evlFLTL}_4(a, p) = \begin{cases} (\top, \mathrm{true}) & \text{if } p \in a \\ (\bot, \mathrm{false}) & \text{else} \end{cases}$$

$$\mathrm{evlFLTL}_4(a, \neg p) = \begin{cases} (\bot, \mathrm{false}) & \text{if } p \in a \\ (\top, \mathrm{true}) & \text{else} \end{cases}$$

evlFLTL_4

Let $\Sigma = 2^{\text{AP}}$ be the finite alphabet, $p \in \text{AP}$ an atomic proposition, $a \in \Sigma$ a letter, and φ and ψ LTL formulae.

We then define the function
$\text{evlFLTL}_4 : \Sigma \times \text{LTL} \to \mathbb{B}_4 \times \text{LTL}$ inductively as follows:

$$\begin{aligned}
\text{evlFLTL}_4(a, \varphi \vee \psi) = (v_\varphi \sqcup v_\psi, \varphi' \vee \psi'),\ &\text{where} \\
(v_\varphi, \varphi') = \text{evlFLTL}_4(a, \varphi)\ &\text{and} \\
(v_\psi, \psi') = \text{evlFLTL}_4(a, \psi)& \\
\text{evlFLTL}_4(a, \varphi \wedge \psi) = (v_\varphi \sqcap v_\psi, \varphi' \wedge \psi'),\ &\text{where} \\
(v_\varphi, \varphi') = \text{evlFLTL}_4(a, \varphi)\ &\text{and} \\
(v_\psi, \psi') = \text{evlFLTL}_4(a, \psi)&
\end{aligned}$$

$\mathrm{evlFLTL}_4$

Let $\Sigma = 2^{\mathrm{AP}}$ be the finite alphabet, $p \in \mathrm{AP}$ an atomic proposition, $a \in \Sigma$ a letter, and φ and ψ LTL formulae.

We then define the function $\mathrm{evlFLTL}_4 : \Sigma \times \mathrm{LTL} \to \mathbb{B}_4 \times \mathrm{LTL}$ inductively as follows:

$$\begin{aligned}
\mathrm{evlFLTL}_4(a, \mathrm{X}\,\varphi) &= (\bot^p, \varphi)\\
\mathrm{evlFLTL}_4(a, \overline{\mathrm{X}}\,\varphi) &= (\top^p, \varphi)\\
\mathrm{evlFLTL}_4(a, \varphi\,\mathrm{U}\,\psi) &= \mathrm{evlFLTL}_4(a, \psi \vee (\varphi \wedge \mathrm{X}(\varphi\,\mathrm{U}\,\psi)))\\
\mathrm{evlFLTL}_4(a, \varphi\,\mathrm{R}\,\psi) &= \mathrm{evlFLTL}_4(a, \psi \wedge (\varphi \vee \overline{\mathrm{X}}(\varphi\,\mathrm{R}\,\psi)))\\
\mathrm{evlFLTL}_4(a, \mathrm{F}\,\varphi) &= \mathrm{evlFLTL}_4(a, \varphi \vee \mathrm{X}\,\mathrm{F}\,\varphi)\\
\mathrm{evlFLTL}_4(a, \mathrm{G}\,\varphi) &= \mathrm{evlFLTL}_4(a, \varphi \wedge \overline{\mathrm{X}}\,\mathrm{G}\,\varphi)
\end{aligned}$$

Examples

Example (Impartial Evaluation of Globally)

Consider $w = \{a\}\{a\}\emptyset$. First letter:

$$\begin{aligned}\mathrm{evlFLTL}_4(\{a\}, \mathrm{G}\, a) &= \mathrm{evlFLTL}_4(\{a\}, a \wedge \overline{\mathrm{X}}\, \mathrm{G}\, a) \\ &= (v_1 \sqcap v_2, \varphi_1 \wedge \varphi_2) \\ &= (\top \sqcap \top^p, \mathrm{true} \wedge \mathrm{G}\, a) \\ &= (\top^p, \mathrm{G}\, a)\end{aligned}$$

where $(v_1, \varphi_1) = \mathrm{evlFLTL}_4(\{a\}, a) = (\top, \mathrm{true})$

$(v_2, \varphi_2) = \mathrm{evlFLTL}_4(\{a\}, \overline{\mathrm{X}}\, \mathrm{G}\, a) = (\top^p, \mathrm{G}\, a)$.

Next letters:

- $\mathrm{evlFLTL}_4(\{a\}, \mathrm{G}\, a) = (\top^p, \mathrm{G}\, a)$
- $\mathrm{evlFLTL}_4(\emptyset, \mathrm{G}\, a) = (\bot, \mathrm{false})$

Examples

Example (Impartial Evaluation of Finally)

Consider $w = \emptyset\emptyset\{a\}$. First letter:

$$\begin{aligned}
\mathrm{evlFLTL}_4(\emptyset, \mathrm{F}\, a) &= \mathrm{evlFLTL}_4(\emptyset, a \vee \mathrm{X}\,\mathrm{F}\, a) \\
&= (v_1 \sqcup v_2, \varphi_1 \vee \varphi_2) \\
&= (\bot \sqcup \bot^p, \mathrm{false} \vee \mathrm{F}\, a) \\
&= (\bot^p, \mathrm{F}\, a)
\end{aligned}$$

where $(v_1, \varphi_1) = \mathrm{evlFLTL}_4(\emptyset, a) = (\bot, \mathrm{false})$

$(v_2, \varphi_2) = \mathrm{evlFLTL}_4(\emptyset, \mathrm{X}\,\mathrm{F}\, a) = (\bot^p, \mathrm{F}\, a)$.

Next letters:

- $\mathrm{evlFLTL}_4(\emptyset, \mathrm{F}\, a) = (\bot^p, \mathrm{F}\, a)$
- $\mathrm{evlFLTL}_4(\{a\}, \mathrm{F}\, a) = (\top, \mathrm{true})$

Automata-theoretic Approach

Further Topics

- Alternating vs. non-deterministic vs. deterministic machines.

Automata-theoretic Approach

Further Topics

- Alternating vs. non-deterministic vs. deterministic machines.
- Complexity of the translations.

Automata-theoretic Approach

Further Topics

- Alternating vs. non-deterministic vs. deterministic machines.
- Complexity of the translations.
- Size vs. power.

Automata-theoretic Approach

Further Topics

- Alternating vs. non-deterministic vs. deterministic machines.
- Complexity of the translations.
- Size vs. power.
- State sequence for an input word.

Conclusion

1. Every two-valued logic is not impartial. Impartiality implies multiple values.
2. We use the distributive De Morgan lattice $\mathbb{B}_4 = \{\bot, \bot^p, \top^p, \top\}$ in the impartial FLTL_4 semantics.
3. At the end of the word X evaluates to $\bot^p$ and $\overline{\mathrm{X}}$ evaluates to $\top^p$.
4. $\mathrm{evlFLTL}_4$ performs formula rewriting (progression) for an LTL formula and one letter.
5. $\mathrm{evlFLTL}_4$ can be used to describe the transition function of an alternating mealy machine using the subformulae as states.
6. Such an alternating mealy machine can be translated into a deterministic mealy machine using the fact that equivalent positive combinations of states leads to the same next states and output.

Høgskulen på Vestlandet

UNIVERSITÄT ZU LÜBECK
INSTITUTE FOR SOFTWARE ENGINEERING
AND PROGRAMMING LANGUAGES

Chapter 6

Anticipatory LTL Semantics

Course “Runtime Verification”
M. Leucker & V. Stolz

Chapter 6

Learning Targets of Chapter “Anticipatory LTL Semantics”.

1. Understand that LTL semantics can be defined over infinite words as well.
2. Understand the difference of LTL over finite and infinite words.
3. Recall anticipation and understand why impartiality is not enough to build good monitors.
4. Get used to the three-valued sematics for LTL.
5. Understand the concept of safety and co-safety properties and get an idea of monitorable properties.

Chapter 6

Outline of Chapter “Anticipatory LTL Semantics”.

Høgskulen på Vestlandet

UNIVERSITÄT ZU LÜBECK
INSTITUTE FOR SOFTWARE ENGINEERING
AND PROGRAMMING LANGUAGES

Section

LTL on Infinite Words

Semantics
Equivalences and Examples

Chapter 6 “Anticipatory LTL Semantics”
Course “Runtime Verification”
M. Leucker & V. Stolz

Need of LTL on Infinite Words in RV?

Impartiality Say $\top$ or $\bot$ only if you are sure.

Anticipation Say $\top$ or $\bot$ once you can be sure.

We want do define impartial LTL semantics:

- Say $\top$ if every infinite continuation evaluates to $\top$.
- Say $\bot$ if every infinite continuation evaluates to $\bot$.
- Otherwise say $?$.

We Need LTL on Infinite Words

- Impartial LTL semantics will be based on infinite continuations.
- Properties of infinite continuations cannot be expressed using LTL on finite words.

Infinite Words

- An infinite word w is an infinite sequence over the alphabet $\Sigma = 2^{\mathrm{AP}}$.
- w can be interpreted as function $w : \mathbb{N}\backslash\{0\} \to \Sigma$.
- w can be interpreted as concatenation of many finite and one infinite words.

Examples (Infinite Words)

Consider the alphabet $\Sigma = 2^{\mathrm{AP}}$ with $\mathrm{AP} = \{p, q\}$.

- $\{p\}^\omega$ denotes the infinite word where every letter is $\{p\}$ and can be interpreted as $w(i) = \{p\}$ for all $i \geq 1$.
- $\emptyset(\{q\}\{p\})^\omega$ can be interpreted as

$$w(i) = \begin{cases} \emptyset & \text{if } i = 1 \\ \{q\} & \text{if } i \equiv 0 \mod 2 \\ \{p\} & \text{else} \end{cases}$$

Finally and Globally Examples

Examples (Finally and Globally)

Consider infinite words over the alphabet $\Sigma = 2^{\mathrm{AP}}$ with $\mathrm{AP} = \{p, q\}$.

- $\{p\}\emptyset\{q\}\emptyset^\omega \models \mathrm{F}\, q$.

Finally and Globally Examples

Examples (Finally and Globally)

Consider infinite words over the alphabet $\Sigma = 2^{\mathrm{AP}}$ with $\mathrm{AP} = \{p, q\}$.

- $\{p\}\emptyset\{q\}\emptyset^\omega \models \mathrm{F}\, q$.
- $\{q\}\{q\}(\{p, q\}\{q\})^\omega \models \mathrm{G}\, q$.

Finally and Globally Examples

Examples (Finally and Globally)

Consider infinite words over the alphabet $\Sigma = 2^{\mathrm{AP}}$ with $\mathrm{AP} = \{p, q\}$.

- $\{p\}\emptyset\{q\}\emptyset^\omega \models \mathrm{F}\, q$.
- $\{q\}\{q\}(\{p, q\}\{q\})^\omega \models \mathrm{G}\, q$.
- $(\emptyset\{p\}\{p, q\}\emptyset)^\omega \models \mathrm{G}\,\mathrm{F}\, q$.

Finally and Globally Examples

Examples (Finally and Globally)

Consider infinite words over the alphabet $\Sigma = 2^{\mathrm{AP}}$ with $\mathrm{AP} = \{p, q\}$.

- $\{p\}\emptyset\{q\}\emptyset^\omega \models \mathrm{F}\, q$.
- $\{q\}\{q\}(\{p,q\}\{q\})^\omega \models \mathrm{G}\, q$.
- $(\emptyset\{p\}\{p,q\}\emptyset)^\omega \models \mathrm{G}\,\mathrm{F}\, q$.
- $\{p\}\emptyset\{q\}\{p\}(\{p,q\}\{q\})^\omega \models \mathrm{F}\,\mathrm{G}\, q$.

Parts of Infinite Words

In the formal definition of LTL semantics we denote parts of a word as follows:

Let $w = a_1 a_2 a_3 \ldots \in \Sigma^\omega$ be an infinite word over the alphabet $\Sigma = 2^{\mathrm{AP}}$ and let $i \in \mathbb{N}$ with $i \geq 1$ be a position in this word. Then

- $w_i = a_i$ is the i-th letter of the word,
- $w^{(i)} = a_1 a_2 \ldots a_i$ is the prefix of w of length i and
- w^i is the subword of w s. t. $w = w^{(i-1)} w^i$.

LTL Semantics on Infinite Words

Definition (LTL Semantics on Infinite Words)

Let φ, ψ be LTL formulae and let $w \in \Sigma^\omega$ be an infinite word. Then the semantics of φ with respect to w is inductively defined as follows:

$$\begin{array}{ll}
w \models \mathrm{true} & \\
w \models p & \text{iff } p \in w_1 \\
w \models \neg p & \text{iff } p \notin w_1 \\
w \models \neg \varphi & \text{iff } w \not\models \varphi \\
w \models \varphi \vee \psi & \text{iff } w \models \varphi \text{ or } w \models \psi \\
w \models \varphi \wedge \psi & \text{iff } w \models \varphi \text{ and } w \models \psi
\end{array}$$

LTL Semantics on Infinite Words

Definition (LTL Semantics on Infinite Words)

Let φ, ψ be LTL formulae and let $w \in \Sigma^\omega$ be an infinite word. Then the semantics of φ with respect to w is inductively defined as follows:

$$\begin{array}{ll} w \models \mathrm{X}\,\varphi & \text{iff } w^2 \models \varphi \\ w \models \overline{\mathrm{X}}\,\varphi & \text{iff } w^2 \models \varphi \end{array}$$

LTL Semantics on Infinite Words

Definition (LTL Semantics on Infinite Words)

Let φ, ψ be LTL formulae and let $w \in \Sigma^\omega$ be an infinite word. Then the semantics of φ with respect to w is inductively defined as follows:

$$\begin{aligned} w \models \varphi \,\mathrm{U}\, \psi \quad & \text{iff } \exists i \geq 1 : (w^i \models \psi \\ & \qquad \text{and } \forall k, 1 \leq k < i : w^k \models \varphi) \\ w \models \varphi \,\mathrm{R}\, \psi \quad & \text{iff } \exists i \geq 1 : (w^i \models \varphi \\ & \qquad \text{and } \forall k, 1 \leq k \leq i : w^k \models \psi) \\ & \quad \text{or } \forall i \geq 1 : w^i \models \psi \end{aligned}$$

LTL Semantics on Infinite Words

Definition (LTL Semantics on Infinite Words)

Let φ, ψ be LTL formulae and let $w \in \Sigma^\omega$ be an infinite word. Then the semantics of φ with respect to w is inductively defined as follows:

$$
\begin{array}{ll}
w \models \mathrm{F}\,\varphi & \text{iff } \exists i \geq 1 : w^i \models \varphi \\
w \models \mathrm{G}\,\varphi & \text{iff } \forall i \geq 1 : w^i \models \varphi
\end{array}
$$

Semantic Function for LTL on Infinite Words

- We defined the LTL semantics on infinite words as relation $w \models \varphi$ between a word $w \in \Sigma^\omega$ and a LTL formula φ.
- This can be interpreted as semantic function

$$\mathrm{sem}_\omega : \Sigma^\omega \times \mathrm{LTL} \to \mathbb{B}_2,$$

$$\mathrm{sem}_\omega(w, \varphi) = [\![w \models \varphi]\!]_\omega := \begin{cases} \top & \text{if } w \models \varphi \\ \bot & \text{else.} \end{cases}$$

Languages Defined by LTL Formulae

The set of models of an LTL formula φ defines a language $\mathcal{L}(\varphi) \subseteq \Sigma^\omega$ of infinite words over $\Sigma = 2^{\mathrm{AP}}$ as follows:

$$\mathcal{L}(\varphi) = \{w \in \Sigma^\omega \mid [\![w \models \varphi]\!]_\omega = \top\}.$$

Equivalences

Weak Next

Let $w \in \Sigma^\omega$ be an infinite word.

- w has no last character.
- For every position $i \geq 1$ the word $w^i \in \Sigma^\omega$ is infinite.
- X and $\overline{\mathrm{X}}$ have the same semantics.

The De Morgan rules, equivalences for G and F and the fixed point equations for U and R are still valid.

Finite and Infinite Semantics

Examples (Globally and Finally)

- We know
$$[\![\{p\}(\{p\}\{q\})^\omega \models \mathrm{F}\, q]\!]_\omega = \top$$
from
$$[\![\{p\}\{p\}\{q\} \models \mathrm{F}\, q]\!]_4 = \top.$$
- We know
$$[\![\{p\}(\{p\}\{q\})^\omega \models \mathrm{G}\, p]\!]_\omega = \bot$$
from
$$[\![\{p\}\{p\}\{q\} \models \mathrm{G}\, p]\!]_4 = \bot.$$

Finite and Infinite Semantics

Examples (Until)

- We know
$$[\![\{p\}(\{p\}\{q\})^\omega \models p \,\mathrm{U}\, q]\!]_\omega = \top$$
from
$$[\![\{p\}\{p\}\{q\} \models p \,\mathrm{U}\, q]\!]_4 = \top.$$
- We know
$$[\![\{p\}\emptyset\{q\}^\omega \models p \,\mathrm{U}\, q]\!]_\omega = \bot$$
from
$$[\![\{p\}\emptyset \models p \,\mathrm{U}\, q]\!]_4 = \bot.$$

Høgskulen på Vestlandet

UNIVERSITÄT ZU LÜBECK
INSTITUTE FOR SOFTWARE ENGINEERING AND PROGRAMMING LANGUAGES

Section

Anticipatory LTL Semantics: LTL$_3$

Anticipation

Be Anticipatory

- go for a final verdict ($\top$ or $\bot$) once you really know
- do not delay the decision

Definition (Anticipation)

Anticipation requires that once every (possibly infinite) continuation of a finite trace leads to the same verdict, then the finite trace evaluates to this very same verdict.

FLTL$_4$ is Not Anticipatory

Example (Next Operator)

We have

$$[\![\{p\} \models \mathrm{X}\,\mathrm{X}\,\mathrm{false}]\!]_4 = \bot^p$$

$$\begin{aligned}&[\![\{p\}\{p\} \models \mathrm{X}\,\mathrm{X}\,\mathrm{false}]\!]_4\\ &\quad = [\![\{p\} \models \mathrm{X}\,\mathrm{false}]\!]_4 = \bot^p\end{aligned}$$

$$\begin{aligned}&[\![\{p\}\{p\}\{p\} \models \mathrm{X}\,\mathrm{X}\,\mathrm{false}]\!]_4\\ &\quad = [\![\{p\}\{p\} \models \mathrm{X}\,\mathrm{false}]\!]_4\\ &\quad\quad = [\![\{p\} \models \mathrm{false}]\!]_4 = \bot,\end{aligned}$$

but it would be anticipatory to have

$$[\![\{p\} \models \mathrm{X}\,\mathrm{X}\,\mathrm{false}]\!]_3 = \bot.$$

FLTL$_4$ is Not Anticipatory

Example (Globally and Finally Operator)

We have

$$[\![w \models \mathrm{G}\,\mathrm{true}]\!]_4 = [\![w \models \mathrm{false}\,\mathrm{R}\,\mathrm{true}]\!]_4 = \top^p \text{ and}$$
$$[\![w \models \mathrm{F}\,\mathrm{false}]\!]_4 = [\![w \models \mathrm{true}\,\mathrm{U}\,\mathrm{false}]\!]_4 = \bot^p$$

but it would be anticipatory to have

$$[\![w \models \mathrm{G}\,\mathrm{true}]\!]_3 = \top \text{ and}$$
$$[\![w \models \mathrm{F}\,\mathrm{false}]\!]_3 = \bot.$$

Impartial Anticipation

- Define LTL semantics for finite, non-terminated words.
- The set of all infinite continuations of a finite word contains only infinite words.
- Define semantics for finite words based on semantics of these infinite continuations.
- If the semantic function yields the same verdict for all infinite continuations use that verdict.
- Combine $\top^p$ and $\bot^p$ to a common $?$ for the other cases.

Anticipatory Three-Valued LTL Semantics

Definition (LTL$_3$ Semantics)

Let φ be an LTL formula and let $u \in \Sigma^*$ be a finite word. Then the semantics of φ with respect to u is defined as follows:

$$[\![u \models \varphi]\!]_3 = \begin{cases} \top & \text{if } \forall w \in \Sigma^\omega : [\![uw \models \varphi]\!]_\omega = \top \\ \bot & \text{if } \forall w \in \Sigma^\omega : [\![uw \models \varphi]\!]_\omega = \bot \\ ? & \text{else.} \end{cases}$$

Example

Consider $\varphi = \mathrm{G}(p \rightarrow \mathrm{F}\, false)$ and $\emptyset\{q\}\{p\}\emptyset \in \Sigma^*$ for $\Sigma = 2^{\mathrm{AP}}$ and $\mathrm{AP} = \{p, q\}$. We then have

- $[\![\emptyset \models \varphi]\!]_3 = ?$
- $[\![\emptyset\{q\} \models \varphi]\!]_3 = ?$
- $[\![\emptyset\{q\}\{p\} \models \varphi]\!]_3 = \bot$
- $[\![\emptyset\{q\}\{p\}\emptyset \models \varphi]\!]_3 = \bot$

Example

Consider $\varphi = \mathrm{G}(p \rightarrow \mathrm{F}\, false)$ and $\emptyset\{q\}\{p\}\emptyset \in \Sigma^*$ for $\Sigma = 2^{\mathrm{AP}}$ and $\mathrm{AP} = \{p, q\}$. We then have

- $[\![\emptyset \models \varphi]\!]_3 = ?$
- $[\![\emptyset\{q\} \models \varphi]\!]_3 = ?$
- $[\![\emptyset\{q\}\{p\} \models \varphi]\!]_3 = \bot$
- $[\![\emptyset\{q\}\{p\}\emptyset \models \varphi]\!]_3 = \bot$
- $[\![\emptyset\{q\}\{p\}u \models \varphi]\!]_3 = \bot$ for all $u \in \Sigma^*$

Possible Verdicts of LTL Formulae

Consider a word $w \in \Sigma^*$ for $\Sigma = 2^{\mathrm{AP}}$ and propositions $p, q \in \mathrm{AP}$. We then have

- $[\![w \models p \,\mathrm{U}\, q]\!]_3 \in \{\top, ?, \bot\}$
- $[\![w \models p \,\mathrm{R}\, q]\!]_3 \in \{\top, ?, \bot\}$

Possible Verdicts of LTL Formulae

Consider a word $w \in \Sigma^*$ for $\Sigma = 2^{\mathrm{AP}}$ and propositions $p, q \in \mathrm{AP}$. We then have

- $[\![w \models p \, \mathrm{U} \, q]\!]_3 \in \{\top, ?, \bot\}$
- $[\![w \models p \, \mathrm{R} \, q]\!]_3 \in \{\top, ?, \bot\}$
- $[\![w \models \mathrm{F} \, p]\!]_3 \in \{\top, ?\}$
- $[\![w \models \mathrm{G} \, p]\!]_3 \in \{?, \bot\}$

Possible Verdicts of LTL Formulae

Consider a word $w \in \Sigma^*$ for $\Sigma = 2^{\mathrm{AP}}$ and propositions $p, q \in \mathrm{AP}$. We then have

- $[\![w \models p \,\mathrm{U}\, q]\!]_3 \in \{\top, ?, \bot\}$
- $[\![w \models p \,\mathrm{R}\, q]\!]_3 \in \{\top, ?, \bot\}$
- $[\![w \models \mathrm{F}\, p]\!]_3 \in \{\top, ?\}$
- $[\![w \models \mathrm{G}\, p]\!]_3 \in \{?, \bot\}$
- $[\![w \models \mathrm{G}\,\mathrm{F}\, p]\!]_3 = ?$
- $[\![w \models \mathrm{F}\,\mathrm{G}\, p]\!]_3 = ?$

Section

Monitorable Properties

(Co-)Safety

Examples

Monitorability

Chapter 6 “Anticipatory LTL Semantics”
Course “Runtime Verification”
M. Leucker & V. Stolz

Monitorability

When Does Anticipation Help?

The Good, The Bad and The Ugly

Definition (Good, Bad and Ugly Prefixes)

Given a language $L \subseteq \Sigma^\omega$ of infinite words over Σ we call a finite word $u \in \Sigma^*$

- a good prefix for L if $\forall w \in \Sigma^\omega : uw \in L$,
- a bad prefix for L if $\forall w \in \Sigma^\omega : uw \notin L$ and
- an ugly prefix for L if $\forall v \in \Sigma^* : uv$ is neither a good prefix nor a bad prefix.

Examples for Good, Bad and Ugly

Examples (The Good, The Bad and The Ugly)

- $\{p\}\{q\}$ is a good prefix for $\mathcal{L}(\mathrm{F}\, q)$.
- $\{p\}\{q\}\{p\}$ is a good prefix for $\mathcal{L}(\mathrm{F}\, q)$.
- $\{p\}\{q\}$ is a bad prefix for $\mathcal{L}(\mathrm{G}\, p)$.
- every $w \in \Sigma^*$ is an ugly prefix for $\mathcal{L}(\mathrm{G}\,\mathrm{F}\, p)$.
- $\{p\}$ is an ugly prefix for $\mathcal{L}(p \to \mathrm{G}\,\mathrm{F}\, p)$.

LTL$_3$ indentifies good/bad prefixes

Given an LTL formula φ and a finite word $u \in \Sigma^*$, then

$$[\![u \models \varphi]\!]_3 = \begin{cases} \top & \text{if } u \text{ is a good prefix for } \mathcal{L}(\varphi) \\ \bot & \text{if } u \text{ is a bad prefix for } \mathcal{L}(\varphi) \\ ? & \text{otherwise} \end{cases}$$

The Idea of Saftey and Co-Safety

Safety Properties assert that nothing bad happens.
Such a property is violated iff something bad happens after finitely many steps.
($\rightarrow$ A bad prefix exists.)

Co-Safety Properties assert that something good happens.
Such a property is fulfilled iff something good happens after finitely many steps.
($\rightarrow$ A good prefix exists.)

(Co-)Safety Languages and Properties

Definition ((Co-)Safety Languages)

A language $L \subseteq \Sigma^\omega$ is called

- a *safety language* if for all $w \notin L$ there is a prefix $u \in \Sigma^*$ of w which is a bad prefix for L.
- a *co-safety language* if for all $w \in L$ there is a prefix $u \in \Sigma^*$ of w which is a good prefix for L.

Definition ((Co-)Safety Properties)

An LTL formula φ is called

- a *safety property* if its set of models $\mathcal{L}(\varphi)$ is a safety language.
- a *co-safety property* if its set of models $\mathcal{L}(\varphi)$ is a co-safety language.

Examples

Consider propositions $p, q \in \mathrm{AP}$.

Formula	Safety	Co-Safety
$\mathrm{G}\, p$		
$\mathrm{F}\, p$		
$\mathrm{X}\, p$		
$\mathrm{G}\,\mathrm{F}\, p$		
$\mathrm{F}\,\mathrm{G}\, p$		
$\mathrm{X}\, p \vee \mathrm{G}\,\mathrm{F}\, p$		
$p \,\mathrm{U}\, q$		
$p \,\mathrm{R}\, q$		

Examples

Consider propositions $p, q \in \mathrm{AP}$.

Formula	Safety	Co-Safety
$\mathrm{G}\,p$	✓	✗
$\mathrm{F}\,p$		
$\mathrm{X}\,p$		
$\mathrm{G}\,\mathrm{F}\,p$		
$\mathrm{F}\,\mathrm{G}\,p$		
$\mathrm{X}\,p \vee \mathrm{G}\,\mathrm{F}\,p$		
$p\,\mathrm{U}\,q$		
$p\,\mathrm{R}\,q$		

Examples

Consider propositions $p, q \in \mathrm{AP}$.

Formula	Safety	Co-Safety
$\mathrm{G}\,p$	✔	✘
$\mathrm{F}\,p$	✘	✔
$\mathrm{X}\,p$		
$\mathrm{G}\,\mathrm{F}\,p$		
$\mathrm{F}\,\mathrm{G}\,p$		
$\mathrm{X}\,p \vee \mathrm{G}\,\mathrm{F}\,p$		
$p\,\mathrm{U}\,q$		
$p\,\mathrm{R}\,q$		

Examples

Consider propositions $p, q \in \mathrm{AP}$.

Formula	Safety	Co-Safety
$\mathrm{G}\, p$	✓	✗
$\mathrm{F}\, p$	✗	✓
$\mathrm{X}\, p$	✓	✓
$\mathrm{G}\,\mathrm{F}\, p$		
$\mathrm{F}\,\mathrm{G}\, p$		
$\mathrm{X}\, p \vee \mathrm{G}\,\mathrm{F}\, p$		
$p\, \mathrm{U}\, q$		
$p\, \mathrm{R}\, q$		

Examples

Consider propositions $p, q \in \mathrm{AP}$.

Formula	Safety	Co-Safety
$\mathrm{G}\, p$	✓	✗
$\mathrm{F}\, p$	✗	✓
$\mathrm{X}\, p$	✓	✓
$\mathrm{G}\,\mathrm{F}\, p$	✗	✗
$\mathrm{F}\,\mathrm{G}\, p$		
$\mathrm{X}\, p \vee \mathrm{G}\,\mathrm{F}\, p$		
$p\,\mathrm{U}\, q$		
$p\,\mathrm{R}\, q$		

Examples

Consider propositions $p, q \in \mathrm{AP}$.

Formula	Safety	Co-Safety
$\mathrm{G}\, p$	✓	✗
$\mathrm{F}\, p$	✗	✓
$\mathrm{X}\, p$	✓	✓
$\mathrm{G}\,\mathrm{F}\, p$	✗	✗
$\mathrm{F}\,\mathrm{G}\, p$	✗	✗
$\mathrm{X}\, p \vee \mathrm{G}\,\mathrm{F}\, p$		
$p \,\mathrm{U}\, q$		
$p \,\mathrm{R}\, q$		

Examples

Consider propositions $p, q \in \mathrm{AP}$.

Formula	Safety	Co-Safety
$\mathrm{G}\, p$	✓	✗
$\mathrm{F}\, p$	✗	✓
$\mathrm{X}\, p$	✓	✓
$\mathrm{G}\,\mathrm{F}\, p$	✗	✗
$\mathrm{F}\,\mathrm{G}\, p$	✗	✗
$\mathrm{X}\, p \vee \mathrm{G}\,\mathrm{F}\, p$	✗	✗
$p\,\mathrm{U}\, q$		
$p\,\mathrm{R}\, q$		

Examples

Consider propositions $p, q \in \mathrm{AP}$.

Formula	Safety	Co-Safety
$\mathrm{G}\,p$	✓	✗
$\mathrm{F}\,p$	✗	✓
$\mathrm{X}\,p$	✓	✓
$\mathrm{G}\,\mathrm{F}\,p$	✗	✗
$\mathrm{F}\,\mathrm{G}\,p$	✗	✗
$\mathrm{X}\,p \vee \mathrm{G}\,\mathrm{F}\,p$	✗	✗
$p\,\mathrm{U}\,q$	✗	✓
$p\,\mathrm{R}\,q$		

Examples

Consider propositions $p, q \in \mathrm{AP}$.

Formula	Safety	Co-Safety
$\mathrm{G}\, p$	✓	✗
$\mathrm{F}\, p$	✗	✓
$\mathrm{X}\, p$	✓	✓
$\mathrm{G}\,\mathrm{F}\, p$	✗	✗
$\mathrm{F}\,\mathrm{G}\, p$	✗	✗
$\mathrm{X}\, p \vee \mathrm{G}\,\mathrm{F}\, p$	✗	✗
$p\, \mathrm{U}\, q$	✗	✓
$p\, \mathrm{R}\, q$	✓	✗

Details on The Examples

- $p \,\mathrm{U}\, q$ is not a safety property, because $\{p\}^\omega \not\models p \,\mathrm{U}\, q$, but there is no bad prefix.
- $p \,\mathrm{U}\, q$ is a co-safety property, because every infinite word $w \in \Sigma^\omega$ with $w \models p \,\mathrm{U}\, q$ must contain the releasing q in a finite prefix.
- $p \,\mathrm{R}\, q$ is not a co-safety property, because $\{q\}^\omega \models p \,\mathrm{R}\, q$, but there is no good prefix.
- $p \,\mathrm{R}\, q$ is a safety property, because every infinite word $w \in \Sigma^\omega$ with $w \not\models p \,\mathrm{R}\, q$ must contain the violating absence of q in a finite prefix.

Monitorability

Definition (Monitorable Languages)

A language $L \subseteq \Sigma^\omega$ is called *monitorable* iff L has no ugly prefix.

Definition (Monitorable Properties)

An LTL formula φ is called *monitorable* iff its set of models $\mathcal{L}(\varphi)$ is monitorable.

Monitorable Properties

Safety Properties

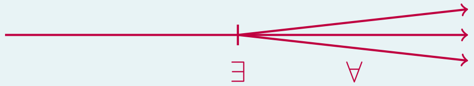
$$\longrightarrow$$

Monitorable Properties

Safety Properties

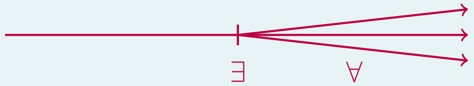
Monitorable Properties

Safety Properties



Monitorable Properties

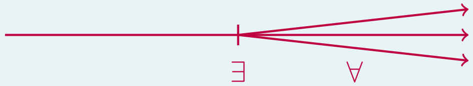
Safety Properties



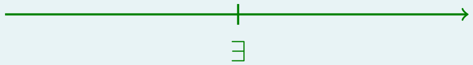
Co-Safety Properties

Monitorable Properties

Safety Properties

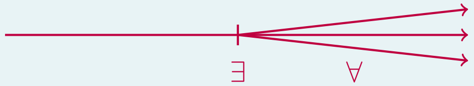


Co-Safety Properties

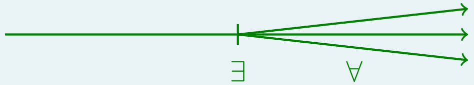


Monitorable Properties

Safety Properties

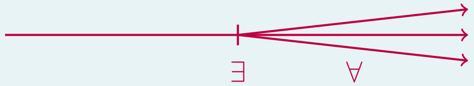


Co-Safety Properties

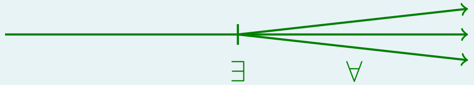


Monitorable Properties

Safety Properties



Co-Safety Properties



Remark

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.*

Safety- and Co-Safety-Properties

Proof.

Consider $\mathrm{AP} = \{p, q, r\}$ and $\varphi = ((p \vee q) \mathbin{\mathrm{U}} r) \vee \mathrm{G}\, p$.

- $\{p\}^\omega \models \varphi$ without good prefix, therefore φ is not a co-safety property.
- $\{q\}^\omega \not\models \varphi$ without bad prefix, therefore φ is not a safety property.
- Every finite word $u \in \Sigma^*$ that is not a bad prefix can become a good prefix by appending $\{r\}$.
- Every finite word $u \in \Sigma^*$ that is not a good prefix can become a bad prefix by appending $\emptyset$.
- No ugly prefix exists as every prefix is either good, bad or can become good or bad by appending $\{r\}$ or $\emptyset$.

$\square$

Safety- and Co-Safety-Properties

Proof by Another Counterexample.

Consider $\mathrm{AP} = \{p, q\}$ and $\varphi = \mathrm{F}\, p \vee \mathrm{G}\, q$.

- $\{q\}^\omega \models \varphi$ without good prefix, therefore φ is not a co-safety property.
- $\emptyset^\omega \not\models \varphi$ without bad prefix, therefore φ is not a safety property.
- Every finite word $u \in \Sigma^*$ that is not a bad prefix can become a good prefix by appending $\{p\}$.
- No ugly prefix exists as every prefix is either bad or can become good by appending $\{p\}$.

□

Conclusion

1. In the semantics for LTL on infinite words there is no difference between X and $\overline{\mathrm{X}}$.
2. The semantics LTL_3 for finite, non-terminated words is defined based on the LTL semantics for infinite words on the lattice $\mathbb{B}_3 = \{\top, ?, \bot\}$.
3. FLTL_4 is not anticipatory, LTL_3 is.
4. $[\![w \models \varphi]\!]_3$ is $\top$ for all good prefixes of $\mathcal{L}(\varphi)$, $\bot$ for all bad prefixes of $\mathcal{L}(\varphi)$ and $?$ for all ugly prefixes of $\mathcal{L}(\varphi)$.
5. The class of monitorable properties comprises safety- and co-safety properties, but is strictly larger than their union.

Høgskulen på Vestlandet

UNIVERSITÄT ZU LÜBECK
INSTITUTE FOR SOFTWARE ENGINEERING
AND PROGRAMMING LANGUAGES

Chapter 7

LTL with a Predictive Semantics

Course “Runtime Verification”
M. Leucker & V. Stolz

Chapter 7

Learning Targets of Chapter “LTL with a Predictive Semantics”.

1. Understand how the underlying program to monitor could be taken into account.
2. Understand how to build a corresponding monitor synthesis procedure.

Chapter 7

Outline of Chapter “LTL with a Predictive Semantics”.

Section

Fusing model checking and runtime verification

LTL with a predictive semantics

Recall anticipatory LTL semantics

The truth value of a LTL$_3$ formula φ with respect to 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

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

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 formula φ with respect to $\hat{\mathcal{P}}$, denoted by $[\![u \models \varphi]\!]_{\hat{\mathcal{P}}} \in \mathbb{B}_4^{¿} = \{\bot, \top, ?, ¿\}$, and defined as follows:

$$[\![u \models \varphi]\!]_{\hat{\mathcal{P}}} = \begin{cases} \top \text{ if} & u \in_\omega \mathcal{L}(\hat{\mathcal{P}}) \wedge \forall w \in \Sigma^\omega : \\ & uw \in \mathcal{L}(\hat{\mathcal{P}}) \Rightarrow [\![uw \models \varphi]\!]_\omega = \top \\ \bot \text{ if} & u \in_\omega \mathcal{L}(\hat{\mathcal{P}}) \wedge \forall w \in \Sigma^\omega : \\ & uw \in \mathcal{L}(\hat{\mathcal{P}}) \Rightarrow [\![uw \models \varphi]\!]_\omega = \bot \\ ? \text{ if} & \exists w, w' \in \Sigma^\omega : uw, uw' \in \mathcal{L}(\hat{\mathcal{P}}) \wedge \\ & [\![uw \models \varphi]\!]_\omega = \top \wedge [\![uw' \models \varphi]\!]_\omega = \bot \\ ¿ \text{ if} & u \notin_\omega \mathcal{L}(\hat{\mathcal{P}}) \end{cases}$$

We use $\mathrm{LTL}_4^{\mathcal{P}}$ to indicate LTL with predictive semantics.

Properties of Predictive Semantics

Remark

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

- Model checking is more precise than RV with the predictive semantics:
  $$\mathcal{P} \models \varphi \quad \text{implies} \quad [\![u \models \varphi]\!]_{\hat{\mathcal{P}}} \in \{\top, ?\}$$
- RV has no false negatives:
  $$[\![u \models \varphi]\!]_{\hat{\mathcal{P}}} = \bot \quad \text{implies} \quad \mathcal{P} \not\models \varphi$$
- The predictive semantics of an LTL formula is more precise than LTL_3:
  $$[\![u \models \varphi]\!]_3 = \top \quad \text{implies} \quad [\![u \models \varphi]\!]_{\hat{\mathcal{P}}} = \top$$
  $$[\![u \models \varphi]\!]_3 = \bot \quad \text{implies} \quad [\![u \models \varphi]\!]_{\hat{\mathcal{P}}} = \bot$$

The reverse directions are in general not true.

Section

Monitoring $\mathrm{LTL}_4^{\mathcal{P}}$

Chapter 7 “LTL with a Predictive Semantics”
Course “Runtime Verification”
M. Leucker & V. Stolz

Conclusion

1. $\mathrm{LTL}_4^{\mathcal{P}}$ only considers extensions of the current word leading to executions of an over-abstraction $\hat{\mathcal{P}}$ of the underlying program $\mathcal{P}$.
2. We introduced the new value ¿ of the output alphabet indicating that the current execution has left the over-abstraction.
3. The use of an over-abstraction is the tradeoff between model checking and runtime verification as the use of the program $\mathcal{P}$ itself would implicitly solve the model checking problem.

Høgskulen på Vestlandet

UNIVERSITÄT ZU LÜBECK
INSTITUTE FOR SOFTWARE ENGINEERING
AND PROGRAMMING LANGUAGES

Chapter 8

Runtime Verification Summary

Course “Runtime Verification”
M. Leucker & V. Stolz

Chapter 8

Learning Targets of Chapter “Runtime Verification Summary”.

1. Understand that FLTL, FLTL_4 and LTL on infinite words share a very similar semantics.
2. Understand that LTL_3 and $\mathrm{LTL}_4^{\mathcal{P}}$ are defined with respect to existing semantics.
3. Understand the difference of propositions and events.

Chapter 8

Outline of Chapter "Runtime Verification Summary".

Høgskulen på Vestlandet

UNIVERSITÄT ZU LÜBECK
INSTITUTE FOR SOFTWARE ENGINEERING AND PROGRAMMING LANGUAGES

Section

Impartial RV

Common LTL Semantics

RV on Finite, Terminated Executions

Impartial RV on Finite, Non-Terminated Words

LTL Semantics

We know the following LTL semantics

FLTL	LTL on finite, completed words
FLTL_4	impartial LTL on finite, non-completed words
LTL	LTL on infinite words
LTL_3	anticipatory LTL on finite, non-completed words
$\mathrm{LTL}_4^{\mathcal{P}}$	anticipatory LTL on finite, non-completed words with respect to an over-abstraction of a program $\mathcal{P}$

- FLTL, FLTL_4 and LTL have very similar semantics with a big common part
- LTL_3 and $\mathrm{LTL}_4^{\mathcal{P}}$ are defined based on LTL

The Common Parts of LTL Semantics

Let AP be a finite set of atomic propositions, $\Sigma = 2^{\mathrm{AP}}$, $p \in \mathrm{AP}$, φ, ψ LTL formulae, and $w \in \Sigma^\infty$ a (finite or infinite) word. The the common part of the LTL semantics FLTL, FLTL_4 and LTL (indicated by $\mathfrak{L}$) of an LTL formula with respect to w is inductively defined as follows:

Boolean Constants

$$\llbracket w \models \mathrm{true} \rrbracket_{\mathfrak{L}} = \top$$
$$\llbracket w \models \mathrm{false} \rrbracket_{\mathfrak{L}} = \bot$$

Boolean Combinations

$$\llbracket w \models \neg\, \varphi \rrbracket_{\mathfrak{L}} = \overline{\llbracket w \models \varphi \rrbracket_{\mathfrak{L}}}$$
$$\llbracket w \models \varphi \vee \psi \rrbracket_{\mathfrak{L}} = \llbracket w \models \varphi \rrbracket_{\mathfrak{L}} \sqcup \llbracket w \models \psi \rrbracket_{\mathfrak{L}}$$
$$\llbracket w \models \varphi \wedge \psi \rrbracket_{\mathfrak{L}} = \llbracket w \models \varphi \rrbracket_{\mathfrak{L}} \sqcap \llbracket w \models \psi \rrbracket_{\mathfrak{L}}$$

The Common Parts of LTL Semantics

Let AP be a finite set of atomic propositions, $\Sigma = 2^{\mathrm{AP}}$, $p \in \mathrm{AP}$, φ, ψ LTL formulae, and $w \in \Sigma^\infty$ a (finite or infinite) word. The the common part of the LTL semantics FLTL, FLTL_4 and LTL (indicated by $\mathfrak{L}$) of an LTL formula with respect to w is inductively defined as follows:

Atomic Propositions

$$[\![w \models p]\!]_{\mathfrak{L}} = \begin{cases} \top & \text{if } p \in w_1 \\ \bot & \text{if } p \notin w_1 \end{cases}$$

Local Temporal Operators

$$[\![w \models \mathrm{X}\,\varphi]\!]_{\mathfrak{L}} = \text{defined dependent of } \mathfrak{L} \text{ later}$$
$$[\![w \models \overline{\mathrm{X}}\,\varphi]\!]_{\mathfrak{L}} = \text{defined dependent of } \mathfrak{L} \text{ later}$$

The Common Parts of LTL Semantics

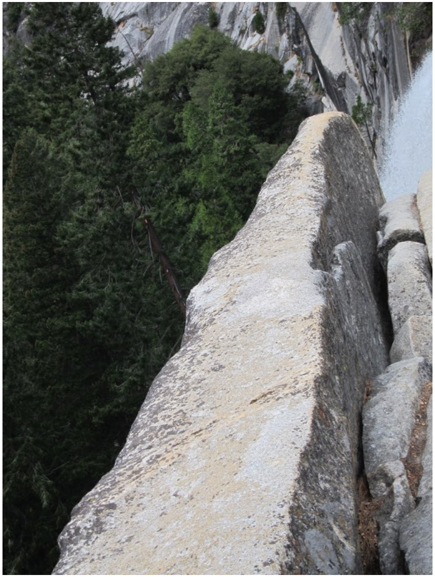
Let AP be a finite set of atomic propositions, $\Sigma = 2^{\mathrm{AP}}$, $p \in \mathrm{AP}$, φ, ψ LTL formulae, and $w \in \Sigma^\infty$ a (finite or infinite) word. The the common part of the LTL semantics FLTL, FLTL_4 and LTL (indicated by $\mathfrak{L}$) of an LTL formula with respect to w is inductively defined as follows:

Fixed Point Operators

$$[\![w \models \varphi \,\mathrm{U}\, \psi]\!]_{\mathfrak{L}} = \begin{cases} \top \text{ if } \exists i, 1 \leq i \leq |w| : ([\![w^i \models \psi]\!]_{\mathfrak{L}} = \top \\ \qquad \text{and } \forall k, 1 \leq k < i : [\![w^k \models \varphi]\!]_{\mathfrak{L}} = \top) \\ \text{defined dependent of } \mathfrak{L} \text{ later else} \end{cases}$$

$$\begin{aligned} [\![w \models \varphi \,\mathrm{R}\, \psi]\!]_{\mathfrak{L}} &= \overline{[\![w \models \neg\varphi \,\mathrm{U}\, \neg\psi]\!]_{\mathfrak{L}}} \\ [\![w \models \mathrm{F}\,\varphi]\!]_{\mathfrak{L}} &= [\![w \models \mathrm{true}\,\mathrm{U}\,\varphi]\!]_{\mathfrak{L}} \\ [\![w \models \mathrm{G}\,\varphi]\!]_{\mathfrak{L}} &= [\![w \models \mathrm{false}\,\mathrm{R}\,\varphi]\!]_{\mathfrak{L}} \end{aligned}$$

LTL on Finite, Terminated Words



FLTL: LTL on Finite, Completed Words

Let φ be an LTL formual. We then define the semantics of FLTL by extending the common LTL semantics of an LTL formula with respect to $w \in \Sigma^*$ as follows:

Local Temporal Operators

$$[\![w \models \mathrm{X}\,\varphi]\!]_2 = \begin{cases} [\![w^2 \models \varphi]\!]_2 & \text{if } |w| > 1 \\ \bot & \text{else} \end{cases}$$

$$[\![w \models \overline{\mathrm{X}}\,\varphi]\!]_2 = \begin{cases} [\![w^2 \models \varphi]\!]_2 & \text{if } |w| > 1 \\ \top & \text{else} \end{cases}$$

FLTL: LTL on Finite, Completed Words

Let φ be an LTL formual. We then define the semantics of FLTL by extending the common LTL semantics of an LTL formula with respect to $w \in \Sigma^*$ as follows:

Fixed Point Operator Until

$$[\![w \models \varphi \,\mathrm{U}\, \psi]\!]_2 = \begin{cases} \top \text{ if } \exists i, 1 \leq i \leq |w| : ([\![w^i \models \psi]\!]_2 = \top \\ \qquad \text{and } \forall k, 1 \leq k < i : [\![w^k \models \varphi]\!]_2 = \top) \\ \bot \text{ else} \end{cases}$$

Monitor Function For FLTL

The function

$$\mathrm{evlFLTL} : \Sigma^{+} \times \mathrm{LTL} \to \mathbb{B}_2$$

takes a finite completed word $w \in \Sigma^{+}$ and
an LTL formula φ and
returns $[\![w \models \varphi]\!]_2$.

$\mathrm{evlFLTL}$ evaluates recursively

- boolean constants, combinations and atomic propisitions directly,
- the next operator by omitting the first letter of the word and
- the fixed point operators using their fixed point equations.

FLTL_4: LTL on Finite, Non-Completed Words

Let φ be an LTL formual. We then define the semantics of FLTL_4 by extending the common LTL semantics of an LTL formula with respect to $w \in \Sigma^*$ as follows:

Local Temporal Operators

$$[\![w \models \mathrm{X}\,\varphi]\!]_4 = \begin{cases} [\![w^2 \models \varphi]\!]_4 & \text{if } |w| > 1 \\ \bot^p & \text{else} \end{cases}$$

$$[\![w \models \overline{\mathrm{X}}\,\varphi]\!]_4 = \begin{cases} [\![w^2 \models \varphi]\!]_4 & \text{if } |w| > 1 \\ \top^p & \text{else} \end{cases}$$

FLTL_4: LTL on Finite, Non-Completed Words

Let φ be an LTL formual. We then define the semantics of FLTL_4 by extending the common LTL semantics of an LTL formula with respect to $w \in \Sigma^*$ as follows:

Fixed Point Operator Until

$$\begin{aligned}
&[\![w \models \varphi \,\mathrm{U}\, \psi]\!]_4 \\
&\quad = \left(\bigsqcup_{1 \leq i \leq |w|} \left([\![w^i \models \psi]\!]_4 \sqcap \bigsqcap_{1 \leq j < i} [\![w^j \models \varphi]\!]_4 \right) \right) \\
&\qquad\qquad\qquad \sqcup \left(\bot^p \sqcap \bigsqcap_{1 \leq i \leq |w|} [\![w^i \models \varphi]\!]_4 \right)
\end{aligned}$$

Monitor Function for FLTL_4

The function

$$\mathrm{evlFLTL}_4 : \Sigma \times \mathrm{LTL} \to \mathbb{B}_4 \times \mathrm{LTL}$$

takes a letter $a \in \Sigma$ of a finite non-completed word and
an LTL formula φ and
returns $[\![a \models \varphi]\!]_4$ and a new LTL formula φ'.

$\mathrm{evlFLTL}_4$

- is based on the ideas of $\mathrm{evlFLTL}$, but
- performs (not recursive) formula rewriting (progression) and
- can be used as transition function of an AMM.

Monitor For FLTL_4

The monitor AMM $\mathcal{M}_\varphi = (\Sigma, Q, q_0, \Gamma, \delta)$ of the LTL formula φ consists of

- the input alphabet $\Sigma = 2^{\mathrm{AP}}$,
- the states Q containing all subformulae of φ,
- the initial state $q_0 = \varphi$,
- the output alphabet $\Gamma = \mathcal{B}_4 = \{\bot, \bot^p, \top^p, \top\}$ and
- the transition function $\delta = \mathrm{evlFLTL}_4$,
  where boolean combinations are interpreted over $B^+(Q)$.

Such an AMM can be translated into an MM using conjunctive or disjunctive normal forms as new states.

Høgskulen på Vestlandet

UNIVERSITÄT ZU LÜBECK
INSTITUTE FOR SOFTWARE ENGINEERING
AND PROGRAMMING LANGUAGES

Section

Anticipatory RV

LTL on Infinite Words
Anticipatory RV on Finite, Non-Terminated Words

Chapter 8 “Runtime Verification Summary”
Course “Runtime Verification”
M. Leucker & V. Stolz

LTL: LTL on Infinite Words

Let φ be an LTL formual. We then define the semantics LTL by extending the common LTL semantics of an LTL formula with respect to $w \in \Sigma^\omega$ as follows:

Local Temporal Operators

$$[\![w \models \mathrm{X}\,\varphi]\!]_\omega = [\![w \models \overline{\mathrm{X}}\,\varphi]\!]_\omega = [\![w^2 \models \varphi]\!]_\omega$$

LTL: LTL on Infinite Words

Let φ be an LTL formual. We then define the semantics LTL by extending the common LTL semantics of an LTL formula with respect to $w \in \Sigma^\omega$ as follows:

Fixed Point Operator Until

$$[\![w \models \varphi \,\mathrm{U}\, \psi]\!]_\omega = \begin{cases} \top \text{ if } \exists i, 1 \leq i : ([\![w^i \models \psi]\!]_\omega = \top \\ \qquad \text{and } \forall k, 1 \leq k < i : [\![w^k \models \varphi]\!]_\omega = \top) \\ \bot \text{ else} \end{cases}$$

Monitor For LTL

The monitor ABA $\mathcal{A}^{\varphi} = (\Sigma, Q, q_0, \delta, F)$ of the LTL formula φ consists of

- the input alphabet $\Sigma = 2^{\mathrm{AP}}$,
- the states Q containing all subformulae of φ,
- the initial state $q_0 = \varphi$,
- the transition function δ, that performs progression like $\mathrm{evlFLTL}_4$, and
- the set F of accepting states, that contains all subformulae searching a greatest fixpoint.

Such an ABA can be translated into an BA using a power set construction where every state consists of two sets of states: All states from paths where we already saw an accepting states and states from paths where we still need to see an accepting state.

LTL_3: LTL on Finite, Non-Completed Words

Let φ be an LTL formual. We then define the semantics LTL_3 of an LTL formula with respect to $w \in \Sigma^*$ based on the LTL semantics as follows:

$$[\![u \models \varphi]\!]_3 = \begin{cases} \top & \text{if } \forall \sigma \in \Sigma^\omega : [\![w\sigma \models \varphi]\!]_\omega = \top \\ \bot & \text{if } \forall \sigma \in \Sigma^\omega : [\![w\sigma \models \varphi]\!]_\omega = \bot \\ ? & \text{else.} \end{cases}$$

Monitor For LTL_3

The monitor FSM $\mathcal{M}_\varphi = \tilde{\mathcal{A}}^\varphi \times \tilde{\mathcal{A}}^{\neg\varphi}$ of the LTL formula φ consists of

- the DFA $\tilde{\mathcal{A}}^\varphi$ computed from the LTL monitor $\mathcal{A}^\varphi = (\Sigma, Q^\varphi, \varphi, \delta^\varphi, F^\varphi)$ via the emptiness per state function,
- the DFA $\tilde{\mathcal{A}}^{\neg\varphi}$ computed from the LTL monitor $\mathcal{A}^{\neg\varphi} = (\Sigma, Q^{\neg\varphi}, \neg\varphi, \delta^{\neg\varphi}, F^{\neg\varphi})$ via the emptiness per state function and
- the labeling function $\lambda : Q \to \mathbb{B}_3$ that prints
  - $\top$ if $\tilde{\mathcal{A}}^\varphi$ is in a rejecting state
  - $\perp$ if $\tilde{\mathcal{A}}^{\neg\varphi}$ is in a rejecting state
  - $?$ if both are in accepting states.

$\mathrm{LTL}_4^{\mathcal{P}}$: Predictive LTL on Finite, Non-Completed Words

Let φ be an LTL formual and $\mathcal{P}$ a program. We then define the semantics $\mathrm{LTL}_3^{\mathcal{P}}$ of an LTL formula with respect to $w \in \Sigma^*$ and and an over-approximation $\hat{\mathcal{P}}$ of $\mathcal{P}$ based on the LTL semantics as follows:

$$[\![u \models \varphi]\!]_{\hat{\mathcal{P}}} = \begin{cases} \top \text{ if} & u \in_\omega \mathcal{L}(\hat{\mathcal{P}}) \wedge \forall w \in \Sigma^\omega : \\ & uw \in \mathcal{L}(\hat{\mathcal{P}}) \Rightarrow [\![uw \models \varphi]\!]_\omega = \top \\ \bot \text{ if} & u \in_\omega \mathcal{L}(\hat{\mathcal{P}}) \wedge \forall w \in \Sigma^\omega : \\ & uw \in \mathcal{L}(\hat{\mathcal{P}}) \Rightarrow [\![uw \models \varphi]\!]_\omega = \bot \\ ? \text{ if} & \exists w, w' \in \Sigma^\omega : uw, uw' \in \mathcal{L}(\hat{\mathcal{P}}) \wedge \\ & [\![uw \models \varphi]\!]_\omega = \top \wedge [\![uw' \models \varphi]\!]_\omega = \bot \\ ¿ \text{ if} & u \notin_\omega \mathcal{L}(\hat{\mathcal{P}}) \end{cases}$$

Høgskulen på Vestlandet

UNIVERSITÄT ZU LÜBECK
INSTITUTE FOR SOFTWARE ENGINEERING
AND PROGRAMMING LANGUAGES

Section

Other LTL Semantics

LTL With Propositions

So far we always used

- a set AP of atomic propositions and
- an alphabet $\Sigma = 2^{\mathrm{AP}}$.

An LTL formula consisting of an atomic proposition $p \in \mathrm{AP}$ gets evaluated with respect to a word $w \in \Sigma^\infty$ as follows:

$$[\![w \models p]\!]_{\mathfrak{L}} = \begin{cases} \top & \text{if } p \in w_1 \\ \bot & \text{if } p \notin w_1 \end{cases}$$

LTL With Events

We now consider

- a set EV of events and
- an alphabet $\Sigma = \mathrm{EV}$.

An LTL formula consisting of an event $e \in \mathrm{EV}$ then gets evaluated with respect to a word $w \in \Sigma^\infty$ as follows:

$$[\![w \models e]\!]_{\mathfrak{L}} = \begin{cases} \top & \text{if } e = w_1 \\ \bot & \text{if } e \neq w_1 \end{cases}$$

Propositions vs. Events

Propositions

- A state consist of a set of propisitions.
- A word $w \in (2^{\mathrm{AP}})^\omega$ is a sequence of states.
- The formula $p \wedge q$ requires that p and q hold in the current state and therefore can be fulfilled.

Events

- A state consist of one event.
- A word $w \in (\mathrm{EV})^\omega$ is a sequence of events.
- The formula $p \wedge q$ requires that the current state is p and q and therefore cannot be fulfilled for $p \neq q$!

LTL With Past

Sometimes it makes things easier to look back

Consider the property "Every alarm is due to a fault" expressed in LTL as follows:

$$\mathrm{fault}\,\mathrm{R}(\neg\,\mathrm{alarm} \vee \mathrm{fault})$$

Using the past operator once (finally in past) O this can be expressed more intuitive as follows:

$$\mathrm{G}(\mathrm{alarm} \rightarrow \mathrm{O}\,\mathrm{fault})$$

- Monitor Generation for LTL with past uses two-way automata.
- LTL with past is kind of syntactic sugar as it is not more expressive than future LTL.

Regular LTL

Adding the Power of Regular Expressions to the Elegance of LTL

Consider the property "p holds in every other state" for a proposition $p \in \mathrm{AP}$ expressed as language as follows:

$$(\Sigma \circ \{p\})^*$$

- LTL can express exactly the star-free languages.
- This property cannot be expressed in LTL.

The property "φ holds in every other state" for an RLTL formula φ can be expressed as

$$\varphi | (\Sigma \circ \Sigma) \rangle \emptyset$$

using the ternary weak power operator $\bullet | \bullet \rangle \bullet$ with a delay of two states expressed as language $\Sigma \circ \Sigma$.

Parameterized LTL

Consider the LTL formula $\mathrm{G}(\texttt{close} \rightarrow \mathrm{X}\,\mathrm{G}(\neg\,\texttt{write}))$ that prohibits writing to the closed resource.

Such formula would fail on an execution with two (or more) resources `c1` and `c2`:

```
open(c1); open(c2);
write(c2);
close(c2);
write(c1); // fail
close(c1);
```

We could solve this problem by allowing free variables in LTL formulas. For example $\mathrm{G}(\texttt{close}(c) \rightarrow \mathrm{X}\,\mathrm{G}(\neg\,\texttt{write}(c)))$ is parametric in c.

LTL With Modulo Constraints

Consider a set VAR of integer variables. We then define LTL semantics with respect to infinite sequences of valuations for VAR taking their values in $\mathbb{Z}/n\mathbb{Z}$.

Example

$$\varphi := \mathrm{G}(\mathrm{X}\, x = x)$$

requires that $x \in \mathrm{VAR}$ evaluates in every state to the same value as in the next state (in all states).

Models of such LTL formulas are words $w \in (\mathbb{Z}/n\mathbb{Z})^\omega$:

- $(12)^\omega \models \varphi$,
  because x is always 12.
- $(12; 13)^\omega \not\models \varphi$,
  because x alternates between 12 and 13.

Conclusion

1. FLTL, FLTL_4 and LTL share common semantics for boolean constants, boolean combinations, atomic propositions and fixed point operators defined using dualization or simplification.
2. We only need to define the semantics of next, weak next and until to specify the semantics of FLTL, FLTL_4 and LTL using the common semantics.
3. LTL_3 and $\mathrm{LTL}_4^{\mathcal{P}}$ are defined based on LTL.
4. Sometimes it make sense to define LTL semantics using events as states instead of sets of propositions.
5. There are very many extensions of LTL and runtime verification not covered (and many not even mentioned) in this course.