Methods and Tool Support for Refinement, Model Transformation and Verification of Network Systems

Zhiming Liu
Southwest University, China

Volker Stolz
Western Norway University of Applied Sciences, Norway
Preface

Bergen, March 2019

This compendium contains the lecture notes used in the “BeChong” meetings held as part of the SIU-funded bilateral project “Methods and Tool Support for Refinement, Model Transformation and Verification of Network Systems” (UTF-2016-short-term/10049) between the Western Norway University of Applied Sciences (HVL), Bergen, Norway, and Southwest University (SWU), Chongqing, China. The Department of Informatics at the University of Bergen and the Guizhou Academy of Sciences, Guiyang, China participated as network partners.

As part of the project, two seminars have been held (15.–19. Oct. 2017 at SWU, 26. Feb.—2. Mar. 2018 at HVL), and smaller-scale exchange visits have taken place between the partners. The seminars saw participation by students on different levels (SWU: Bachelor/Master/PhD, HVL: Master/PhD students). For the undergraduate- and graduate students, the summer school gave them an insight into the academic, international environment, and find out more about possibilities as a PhD student. The PhD students received important feedback on their work, and the events have been a good networking opportunity, which will both be important for their future research.

Staff members lecturing in the schools have been able to combine their active research with lectures in software engineering. Such a combination is usually not possibly in graduate and under-graduate teaching, and we have used the opportunity to prepare our material on an adequate level for the students as opposed to the usual level of presentation at international conferences.

Zhiming Liu (SWU) and Volker Stolz (HVL) as respective coordinators would like to thank SIU for their support and funding to make these seminars possible. They are grateful to Violet Ka I Pun (HVL) for her help in organising the exchanges, collating this document, and final reporting, as well as her help as translator for both parties during the exchange visits. Furthermore, we would like to express our thanks to Prof. Martin Leucker (University of Lübeck, Germany) and Prof. Zongyang Qiu (Peking University, China) for their contribution to the seminars. We hope that the material presented here will be useful in future lectures at SWU and HVL.

Zhiming Liu
Volker Stolz
## List of Lecturers

<table>
<thead>
<tr>
<th>Name</th>
<th>Institution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peng Cheng</td>
<td>Southwest University, CN</td>
</tr>
<tr>
<td>Svetlana Jakšić</td>
<td>Western Norway University of Applied Sciences, NO</td>
</tr>
<tr>
<td>Martin Leucker</td>
<td>University of Lübeck, Germany</td>
</tr>
<tr>
<td>Dan Li</td>
<td>Western Norway University of Applied Sciences, NO</td>
</tr>
<tr>
<td></td>
<td>Guizhou Academy of Sciences, CN</td>
</tr>
<tr>
<td>Hong Lai</td>
<td>Southwest University, CN</td>
</tr>
<tr>
<td>Bo Liu</td>
<td>Southwest University, CN</td>
</tr>
<tr>
<td>Zhiming Liu</td>
<td>Southwest University, CN</td>
</tr>
<tr>
<td>Lars Michael Kristensen</td>
<td>Western Norway University of Applied Sciences, NO</td>
</tr>
<tr>
<td>Yngve Lamo</td>
<td>Western Norway University of Applied Sciences, NO</td>
</tr>
<tr>
<td>Violet Ka I Pun</td>
<td>Western Norway University of Applied Sciences, NO</td>
</tr>
<tr>
<td></td>
<td>University of Oslo, NO</td>
</tr>
<tr>
<td>Zongyan Qiu</td>
<td>Peking University, CN</td>
</tr>
<tr>
<td>Adrian Rutle</td>
<td>Western Norway University of Applied Sciences, NO</td>
</tr>
<tr>
<td>Volker Stolz</td>
<td>Western Norway University of Applied Sciences, NO</td>
</tr>
<tr>
<td></td>
<td>University of Oslo, NO</td>
</tr>
<tr>
<td>Rui Wang</td>
<td>Western Norway University of Applied Sciences, NO</td>
</tr>
<tr>
<td>Xia Zeng</td>
<td>Southwest University, CN</td>
</tr>
<tr>
<td>Hengjun Zhao</td>
<td>Southwest University, CN</td>
</tr>
</tbody>
</table>
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preface</td>
<td>i</td>
</tr>
<tr>
<td>List of lecturers</td>
<td>iii</td>
</tr>
<tr>
<td>I    BeChong Summer School</td>
<td>1</td>
</tr>
<tr>
<td>Lecture 1</td>
<td></td>
</tr>
<tr>
<td>Runtime Verification</td>
<td>7</td>
</tr>
<tr>
<td>Lecture 2</td>
<td></td>
</tr>
<tr>
<td>A Quantum Secret Sharing Scheme using Orbital Angular Momentum onto Multiple Spin States based on Fibonacci Compression Encoding</td>
<td>71</td>
</tr>
<tr>
<td>Lecture 3</td>
<td></td>
</tr>
<tr>
<td>Multilevel Modelling Tooling with MultiEcore</td>
<td>89</td>
</tr>
<tr>
<td>Lecture 4</td>
<td></td>
</tr>
<tr>
<td>CPN Models for Fault-tolerant Distributed Systems</td>
<td>123</td>
</tr>
<tr>
<td>Lecture 5</td>
<td></td>
</tr>
<tr>
<td>Formal Analysis of Service-level Agreements for Cloud services with ABS</td>
<td>137</td>
</tr>
<tr>
<td>Lecture 6</td>
<td></td>
</tr>
<tr>
<td>Contract-Based Model of Evolving Architecture for Intelligent CPS</td>
<td>171</td>
</tr>
<tr>
<td>Lecture 7</td>
<td></td>
</tr>
<tr>
<td>Service Computing</td>
<td>193</td>
</tr>
<tr>
<td>Lecture 8</td>
<td></td>
</tr>
<tr>
<td>Coloured Petri Nets: Modelling and Validation of Concurrent Systems</td>
<td>211</td>
</tr>
<tr>
<td>Lecture 9</td>
<td></td>
</tr>
<tr>
<td>Model Transformations &amp; QVT</td>
<td>235</td>
</tr>
</tbody>
</table>
# TABLE OF CONTENTS

Lecture 10  
Introduction to Association Analysis ........................................... 281

Lecture 11  
Optimal Switching Controller Synthesis by Symbolic Computation – A Case Study ........................................ 305

II  BeChong Spring Meeting .......................................................... 337

Lecture 12  
Relational Semantics and Unifying Logics of Programming ............. 343

Lecture 13  
Hardware-assisted Data race Detection with COEMS .................... 359

Lecture 14  
Linking Modelling Theories for Intelligent CPS Design ................. 373

Lecture 15  
Transforming Coloured Petri Net Models into Code for TinyOS  
– A Case Study of the RPL Protocol ........................................ 397

Lecture 16  
Model-based Test-Case Generation with Coloured Petri Nets ........ 409

Lecture 17  
Program Analysis with LLVM ..................................................... 421

Lecture 18  
Model Transformation Evolution .................................................. 455

Lecture 19  
Invariant Generation and Safety Verification for Hybrid Systems .......... 469

Lecture 20  
A Formal Model of Parallel Execution on Multicore Architectures with Multilevel Caches ........................................ 495

Lecture 21  
Temporal Logics ........................................................................ 511

Lecture 22  
QoS-aware Service Composition and Recommendation ............... 535
Part I

BeChong Summer School

Chongqing, China

16 Oct. – 19 Oct. 2017
BeChong Summer School on “Methods and Tool Support for Refinement, Model Transformation and Verification of Network Systems”, Chongqing, China, Oct. 15-19 2017

Organisers: Assoc. Prof. Volker Stolz (HVL),
Prof. Zhiming LIU (RISE, Southwest University (SWU), Chongqing, China)

Location: Southwest University, Chongqing, China

Supported by the SIU project “Methods and Tool Support for Refinement, Model Transformation and Verification of Network Systems”.

Guest speakers:
- Prof. Zongyang Qiu (Peking University, China)

Further lecturers:
- Dr. Peng Chen (SWU)
- Assoc.Prof. Hong Lai (HVL)
- Dr. Dan Li (HVL/Guizhou Academy of Sciences)
- Dr. Bo Liu (SWU)
- Prof. Zhiming Liu (RISE/SWU)
- Assoc.Prof. Violet Ka I Pun (UIB/UIO)
- Assoc.Prof. Adrian Rutle (HVL)
- Assoc.Prof. Volker Stolz (HVL)
- Mr. Rui Wang (HVL)
- Dr. Hengjun Zhao (SWU)

Topics:
- Model driven software engineering with formal diagrammatic models (A. Rutle)
- Software engineering with model transformations (D. Li)
- Runtime Verification (V. Stolz)
- Towards a Contract-Based Model of Software Architecture in CPS (Z. Liu)
- Combining executable specifications with runtime verification for Lego robots (R. Nøse/R. Wang/A. Rutle/V. Stolz)
- Service Computing (B. Liu)
- Formal analysis of Service-level Agreements for Cloud services with ABS (V. Ka I Pun)
- A Mini-Course on B-Method (Z. Qiu)
- Data mining – association analysis (P. Cheng)
- Modelling of fault-tolerant algorithms with Coloured Petri Nets (R. Wang)
- Optimal Switching Controller Synthesis by Symbolic Computation: A Case Study (H. Zhao)
## Agenda:

### Monday (16 Oct)

<table>
<thead>
<tr>
<th>Time</th>
<th>Session</th>
</tr>
</thead>
</table>
| 09:00-09:30 | Opening Session  
(1) Welcome Speech by Zhiming Liu from School of Computer and Information Science  
(2) The BeChong Summer School Briefing by Volker Stolz  
(3) Photo Taking |
| 09:30-10:15 | Runtime Verification (1)  
Volker Stolz (HVL) |
| 10:15-10:30 | Coffee break |
| 10:30-12:00 | A quantum secret sharing scheme using orbital angular momentum onto multiple spin states based on Fibonacci compression encoding  
Hong Lai (SWU) |
| 12:00-14:00 | Lunch |
| 14:00-14:45 | Multilevel Modeling (1)  
Adrian Rutle (HVL) |
| 14:45-15:30 | CPN models for Fault-tolerant Distributed Systems (1)  
Rui Wang (HVL) |
| 15:30-15:45 | Coffee break |
| 15:45-17:15 | Formal analysis of Service-level Agreements for Cloud services with ABS  
Violet Ka I Pun (UiB/HVL) |
| 17:15-18:00 | Research meeting HVL/RISE |

### Tuesday (17 Oct)

<table>
<thead>
<tr>
<th>Time</th>
<th>Session</th>
</tr>
</thead>
</table>
| 09:00-10:30 | Towards a Contract-Based Model of Software Architecture in CPS  
Zhiming Liu (SWU) |
| 10:30-10:45 | Coffee break |
| 10:45-12:15 | Service Computing  
Bo Liu (SWU) |
| 12:00-14:00 | Lunch |
| 14:00-14:45 | Multilevel Modeling (2)  
Adrian Rutle (HVL) |
| 14:45-15:30 | CPN models for Fault-tolerant Distributed Systems (2)  
Rui Wang (HVL) |
| 15:30-15:45 | Coffee break |
| 15:45-16:30 | Short presentations  
PhD/Master students |
<p>| 17:15-18:00 | Research meeting HVL/RISE |</p>
<table>
<thead>
<tr>
<th>Time</th>
<th>Event</th>
</tr>
</thead>
</table>
| 09:00-10:30| A Mini-Course on B-Method  
Zongyan Qiu (Peking University, China)  |
| 10:30-10:45| Coffee break                                                         |
| 10:45-12:15| Model Transformation and QVT  
Dan Li (SWU)  |
| 12:15-12:30| Coffee break                                                         |
| 12:30-13:00| Combining executable specifications with runtime verification for  
Lego robots  
Rutle/Stolz/Wang (HVL) |
| 13:00-14:00| Lunch                                                               |

**Wednesday (18 Oct)**

<table>
<thead>
<tr>
<th>Time</th>
<th>Event</th>
</tr>
</thead>
</table>
| 09:00-10:30| Data mining – association analysis  
Peng Cheng (SWU)  |
| 10:30-10:45| Coffee break                                                         |
| 10:45-12:00| Runtime Verification (2)  
Volker Stolz (HVL)  |
| 12:00-14:00| Lunch                                                               |
| 14:00-15:30| Optimal Switching Controller Synthesis by Symbolic Computation: A  
Case Study  
Hengjun Zhao (SWU)  |
| 15:30-16:00| Closing                                                             |

**Thursday (19 Oct)**
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.

Chapter 1
Outline of Chapter “Fundamentals”.

Motivation
Statistics
Examples

Definitions
Testing
Validation and Verification
Runtime Verification

Classification
Verification Techniques
Course Topics
Section

Motivation

Statistics
Examples

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

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

![Cost of Fixing Defects Chart]

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 calculated thruster performance using the metric unit Newtons while the ground crew entered course correction using the Imperial measure Pound-force (lbf).


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.
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**

\[ 4195835 - \left(\frac{4195835}{3145727}\right) \cdot 3145727 = 256 \]

- 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 $
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?)

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).

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.

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 $C$. 

Figure: Monitor $M$ checks correctness of components $C_i$. 

Section
Classification
Verification Techniques
Course Topics

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

Testing vs. Oracle Testing

<table>
<thead>
<tr>
<th>Testing</th>
<th>Oracle Testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i_0/o_0, i_1/o_1, \ldots \rightarrow S$</td>
<td>$i_0, i_1, \ldots \rightarrow S \times M$</td>
</tr>
<tr>
<td>Does system $S$ satisfy the input/output sequence $i_0/o_0, i_1/o_1, \ldots$?</td>
<td>Does system $S$ satisfy the test oracle $M$ on input sequence $i_0, i_1, \ldots$?</td>
</tr>
<tr>
<td>Input/output sequence is written manually.</td>
<td>Input sequence and test oracle are written manually.</td>
</tr>
<tr>
<td>Main research topic is input/output sequence generation.</td>
<td>Main research topic is input sequence generation.</td>
</tr>
<tr>
<td>Current execution must be correct.</td>
<td>Current execution must be correct.</td>
</tr>
</tbody>
</table>
### Oracle Testing vs. Runtime Verification

<table>
<thead>
<tr>
<th>Oracle Testing</th>
<th>Runtime Verification</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i_0,i_1,\ldots \rightarrow S \times M$</td>
<td>$i_0,i_1,\ldots \rightarrow S \times M\varphi$</td>
</tr>
<tr>
<td>Does system $S$ satisfy the test oracle $M$ on input sequence $i_0,i_1,\ldots$?</td>
<td>Does system $S$ satisfy the generated monitor $M\varphi$ on input sequence $i_0,i_1,\ldots$?</td>
</tr>
<tr>
<td>Input sequence and test oracle are written manually.</td>
<td>Monitor is synthesized from correctness property.</td>
</tr>
<tr>
<td>Main research topic is input sequence generation.</td>
<td>Main research topic is monitor generation.</td>
</tr>
<tr>
<td>Current execution must be correct.</td>
<td>Current execution must be correct.</td>
</tr>
</tbody>
</table>

### Runtime Verification vs. Model Checking

<table>
<thead>
<tr>
<th>Runtime Verification</th>
<th>Model Checking</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i_0,i_1,\ldots \rightarrow S \times M\varphi$</td>
<td>$S \models \varphi$ via $\mathcal{L}(S) \subseteq \mathcal{L}(\varphi)$</td>
</tr>
<tr>
<td>Does system $S$ satisfy the generated monitor $M\varphi$ on input sequence $i_0,i_1,\ldots$?</td>
<td>Is system $S$ a model of the correctness property $\varphi$? Are all runs of the system $\mathcal{L}(S)$ a subset of all correct runs $\mathcal{L}(\varphi)$?</td>
</tr>
<tr>
<td>Monitor is synthesized from correctness property.</td>
<td>Automatic proof using manually created system model.</td>
</tr>
<tr>
<td>Main research topic is monitor generation.</td>
<td>Main research topic are algorithms for proving model relation.</td>
</tr>
<tr>
<td>Current execution must be correct.</td>
<td>All runs must be correct.</td>
</tr>
</tbody>
</table>
## Model Checking vs. Theorem Proving

<table>
<thead>
<tr>
<th>Model Checking</th>
<th>Theorem Proving</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S \models \varphi$ via $\mathcal{L}(S) \subseteq \mathcal{L}(\varphi)$</td>
<td>$S \models \varphi$ via $S \vdash \varphi$</td>
</tr>
</tbody>
</table>

- **Is system $S$ a model of the correctness property $\varphi$?** 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 $\varphi$?** Can we find a proof that property $\varphi$ 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

<table>
<thead>
<tr>
<th>SST</th>
<th>RV</th>
<th>MC</th>
<th>TP</th>
</tr>
</thead>
<tbody>
<tr>
<td>$I/O \rightarrow S$</td>
<td>$I \rightarrow S \times M$</td>
<td>$S \models \varphi$ via $\mathcal{L}(S) \subseteq \mathcal{L}(\varphi)$</td>
<td>$S \models \varphi$ via $S \vdash \varphi$</td>
</tr>
</tbody>
</table>

- **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** $\varphi$
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.
Chapter 2
Recall Runtime Verification in More Depth

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

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”.

Runtime Verification
Recall
Word Problem
Good Monitors?

Applications
Runtime Reflection
When to Use RV?
RV Frameworks

Section
Runtime Verification
Recall
Word Problem
Good Monitors?
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 primarily 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.
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 $\varphi$.
- 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!

---

Section

Applications

- Runtime Reflection
- When to Use RV?
- RV Frameworks

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

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.
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.

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.
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.
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.
Chapter 3

Specification Languages on Words

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”.

Runs Are Words
- States of the System
- Executions Are Words

Regular Expressions
- The Idea
- Syntax and Semantics
- Limitations

Linear Temporal Logic (LTL)
- Propositional Logic
- Temporal Logic

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 \( \mathsf{AP} \) be a fixed finite non-empty set of atomic propositions.
- \( \Sigma = 2^{\mathsf{AP}} \) 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 $\Sigma$ be an alphabet and $n \in \mathbb{N}$.
We then use the following notation:

<table>
<thead>
<tr>
<th>Notation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Sigma^*$</td>
<td>set of all finite words over $\Sigma$</td>
</tr>
<tr>
<td>$\Sigma^n$</td>
<td>all words in $\Sigma^*$ of length $n$</td>
</tr>
<tr>
<td>$\Sigma \leq n$</td>
<td>all words in $\Sigma^*$ of length at most $n$</td>
</tr>
<tr>
<td>$\Sigma \geq n$</td>
<td>all words in $\Sigma^*$ of length at least $n$</td>
</tr>
<tr>
<td>$\Sigma^+$</td>
<td>$\Sigma_{\geq 1}$</td>
</tr>
<tr>
<td>$\Sigma^\omega$</td>
<td>set of all infinite words over $\Sigma$</td>
</tr>
<tr>
<td>$\Sigma^\infty$</td>
<td>$\Sigma^* \cup \Sigma^\omega$</td>
</tr>
</tbody>
</table>

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

The Idea
Syntax and Semantics
Limitations

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 then have

- **intersection** \( L_1 \cap L_2 = \{ w \in \Sigma^* \mid w \in L_1 \land w \in L_2 \} \)
- **union** \( L_1 \cup L_2 = \{ w \in \Sigma^* \mid w \in L_1 \lor w \in L_2 \} \)
- **complement** \( \overline{L} = \{ w \in \Sigma^* \mid w \notin L \} \)
- **concatenation** \( L_1 \circ L_2 = \{ uv \in \Sigma^* \mid u \in L_1 \land v \in L_2 \} \)
- **Kleene star** \( L^* = \{ u_1 u_2 \ldots u_n \in \Sigma^* \mid \forall i : u_i \in L \} \)

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 \( \varepsilon \) describes the empty word.
- Concatenation is expressed by concatenating regular expressions.
- Union is expressed by the \( | \) operator combining two regular expressions.
- Kleene star is expressed by the \( * \) operator at the end of a regular expression.
Examples

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

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

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:

$$\varphi ::= \varepsilon \mid x \mid \varphi \varphi \mid (\varphi \mid \varphi) \mid (\varphi)^*$$
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.

\[
\begin{align*}
\varepsilon & \models \varepsilon \\
x & \models x \\
w & \models RR' \quad \text{iff } \exists u_1, u_2 : w = u_1u_2 \\
& \quad \text{and } u_1 \models R \text{ and } u_2 \models R' \\
w & \models (R | R') \quad \text{iff } w \models R \text{ or } w \models R' \\
w & \models (R)^* \quad \text{iff } \exists u_1, \ldots, u_n : w = u_1 \ldots u_n \\
& \quad \text{and } \forall i \in \{1, \ldots, n\} : u_i \models R.
\end{align*}
\]

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.
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.

\[ s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \cdots \rightarrow s_{21} \]
\[ \text{today} \rightarrow \text{tomorrow} \rightarrow \text{the day after tomorrow} \rightarrow \cdots \rightarrow \text{in 21 days} \]

Remember

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

Propositional Logic

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

**Example**

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

\[ s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \cdots \rightarrow s_{21} \]
\[ \{ p, r \} \]

\[ w \models \text{true} \quad w \not\models \text{false} \]
\[ w \models p \quad w \models p \land r \lor q \]
\[ w \models \neg q \land \neg s \quad w \not\models q. \]
**Formula:** \( \varphi \)

The formula \( \varphi \) holds for an execution if \( \varphi \) holds in the first state \( s_0 \) of that execution.

\[ s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \ldots \rightarrow s_{21} \]

**Next:** \( X \varphi \)

The formula \( X \varphi \) holds in state \( s_i \) if \( \varphi \) holds in state \( s_{i+1} \). If there is no state \( s_{i+1} \) then \( X \varphi \) never holds.

\[ s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \ldots \rightarrow s_{21} \]
**Weak Next:** $\overline{X} \varphi$

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

- $s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \cdots \rightarrow s_{21}$
  - $\varphi$

**Globally:** $G \varphi$

The formula $G \varphi$ holds in state $s_i$ if $\varphi$ holds in all states $s_j$ for $j \geq i$.

- $s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \cdots \rightarrow s_{21}$
  - $\varphi$
  - $\varphi$
  - $\varphi$
  - $\varphi$
  - $\varphi$
Finally: \( F \varphi \)

The formula \( F \varphi \) holds in state \( s_i \) if there is a state \( s_j \) for \( j \geq i \) in which \( \varphi \) holds.

\[
\begin{array}{c}
\rightarrow s_0 \quad s_1 \quad \cdots \quad s_{20} \quad s_{21} \\
\varphi
\end{array}
\]

Notice that a state in which \( \varphi \) holds is not required in all cases!

Until: \( \varphi U \psi \)

The formula \( \varphi U \psi \) holds in state \( s_i \) if there is a state \( s_j \) for \( j \geq i \) in which \( \psi \) holds and \( \varphi \) holds in all states \( s_k \) for \( i \leq k < j \).

\[
\begin{array}{c}
\rightarrow s_0 \quad s_1 \quad \cdots \quad s_{20} \quad s_{21} \\
\varphi \quad \varphi \quad \varphi \quad \psi
\end{array}
\]

Notice that a state in which \( \varphi \) holds is not required in all cases!
**Release:** $\varphi R \psi$

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

![Diagram](image)

**Conclusion**

1. The execution of a system is a word over the alphabet $\Sigma = 2^{\text{AP}}$ 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.
Chapter 4
LTL on Finite Words

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.
Chapter 4

Outline of Chapter “LTL on Finite Words”.

LTL Syntax
  LTL
  SALT

FLTL Semantics
  Semantics
  Examples and Equivalences
  Negation Normal Form

Monitor Function for FLTL
  The Idea
  Definition
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 $p \in AP$ be an atomic proposition from a finite set of atomic propositions $AP$. The set of LTL formulae is inductively defined by the following grammar:

$$\varphi ::= \text{true} \mid p \mid \varphi \lor \varphi \mid X \varphi \mid \varphi U \varphi \mid F \varphi \mid \text{false} \mid \neg p \mid \varphi \land \varphi \mid X \varphi \mid \varphi R \varphi \mid G \varphi \mid \neg \varphi$$
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: \(\cdot\)^
2. multiplicative operators: \(\cdot, /\)
3. additive operators: \(+, -\)

Definition (operator precedence of LTL)

1. negation operator: \(\neg\)
2. unary temporal operators: \(X, \bar{X}, G, F\)
3. binary temporal logic operators: \(U, R\)
4. conjunction operator: \(\land\)
5. disjunction operator: \(\lor\)

Example

\[ G \neg x \lor \neg x U G y \land z \]
\[ \equiv G (\neg x) \lor ((\neg x) U (G y)) \land z \]
LTL for the Working Engineer?

Simple?
LTL is for theoreticians—but for practitioners?

SALT
Structured Assertion Language for Temporal Logic
⇒ Syntactic Sugar for LTL

www.isp.uni-luebeck.de/salt
Section

FLTL Semantics

Semantics
Examples and Equivalences
Negation Normal Form

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 \( w = a_1a_2 \ldots a_n \in \Sigma^n \) be a finite word over the alphabet \( \Sigma = 2^{AP} \) 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
- \( w^i = a_ia_{i+1} \ldots a_n \) is the subword starting with letter \( i \).
**FLTL Semantics**

**Definition (FLTL Semantics)**

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

- $w \models \text{true}$
- $w \models p$ iff $p \in w_1$
- $w \models \neg p$ iff $p \not\in w_1$
- $w \models \neg \varphi$ iff $w \not\models \varphi$
- $w \models \varphi \lor \psi$ iff $w \models \varphi$ or $w \models \psi$
- $w \models \varphi \land \psi$ iff $w \models \varphi$ and $w \models \psi$

- $w \models X \varphi$ iff $|w| > 1$ and, for $|w| > 1$, $w^2 \models \varphi$

- $w \models \neg X \varphi$ iff $|w| = 1$ or, for $|w| > 1$, $w^2 \models \varphi$
FLTL Semantics

Definition (FLTL Semantics)
Let $\varphi, \psi$ be LTL formulae and let $w \in \Sigma^+$ be a finite word. Then the semantics of $\varphi$ with respect to $w$ is inductively defined as follows:

\[
\begin{align*}
    w \models \varphi \text{ U } \psi & \iff \exists i, 1 \leq i \leq |w| : (w^i \models \psi) \\
    \text{ and } \forall k, 1 \leq k < i : w^k \models \varphi \\
    w \models \varphi \text{ R } \psi & \iff \exists i, 1 \leq i \leq |w| : (w^i \models \varphi) \\
    \text{ and } \forall k, 1 \leq k \leq i : w^k \models \psi \\
    \text{ or } \forall i, 1 \leq i \leq |w| : w^i \models \psi
\end{align*}
\]
Finally and Globally Examples

Examples (Finally and Globally)
Consider words over the alphabet $\Sigma = 2^{AP}$ with $AP = \{p, q\}$:
- $\{p\} \emptyset \{q\} \emptyset \models F q$
- $\{q\} \{p, q\} \{q\} \models G q$
- $\emptyset \{p\} \{p, q\} \emptyset \models G F q$
- $\{p\} \emptyset \{q\} \{p, q\} \{p, q\} \models F G q$

- $GF \varphi$ can be read as: For every state (globally) there will be a state in the future (finally) in that $\varphi$ holds.
- $FG \varphi$ can be read as: There will be a state in the future (finally) that $\varphi$ holds in every state (globally).

Practical Examples
In the following examples we consider these scopes:

- **everytime:** all states
- **before $\psi$:** all states before the first state in which $\psi$ holds (if there is such a state)
- **after $\psi$:** all states after and including the first state in which $\psi$ holds (if there is such a state)

**Example (Absence)**
The formula $\varphi$ does not hold
- **everytime:** $G \neg \varphi$
- **before $\psi$:** $(F \psi) \rightarrow (\neg \varphi U \psi)$
- **after $\psi$:** $G(\psi \rightarrow (G \neg \varphi))$
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: F φ
- before ψ: G ¬ψ ∨ ¬ψ U(φ ∧ ¬ψ)
- after ψ: G ¬ψ ∨ F(ψ ∧ F φ)

Example (Universality)

The formula φ holds

- everytime: G φ
- before ψ: (F ψ) →(φ U ψ)
- after ψ: G(ψ → G φ)
### Equivalences

#### Definition (Equivalence of Formulae)

Let $\Sigma = 2^{\text{AP}}$ and $\varphi$ and $\psi$ be LTL formulae over $\text{AP}$. $\varphi$ and $\psi$ 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:

- $F \varphi \equiv \text{true} U \varphi$
- $G \varphi \equiv \text{false} R \varphi$

### 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:

<table>
<thead>
<tr>
<th>De Morgan Rules of Temporal Logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \neg (\varphi \mathcal{U} \psi) \equiv \neg \varphi \mathcal{R} \neg \psi )</td>
</tr>
<tr>
<td>( \neg (\varphi \mathcal{R} \psi) \equiv \neg \varphi \mathcal{U} \neg \psi )</td>
</tr>
<tr>
<td>( \neg (\text{G } \varphi) \equiv \text{F } \neg \varphi )</td>
</tr>
<tr>
<td>( \neg (\text{F } \varphi) \equiv \text{G } \neg \varphi )</td>
</tr>
<tr>
<td>( \neg (\text{X } \varphi) \equiv \text{X } \neg \varphi )</td>
</tr>
<tr>
<td>( \neg (\text{X } \neg \varphi) \equiv \text{X } \neg \varphi )</td>
</tr>
</tbody>
</table>

Fixed Point Equations

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

\[
\varphi \mathcal{U} \psi \equiv \psi \lor (\varphi \land \text{X}(\varphi \mathcal{U} \psi)) \\
\varphi \mathcal{R} \psi \equiv \psi \land (\varphi \lor \text{X}(\varphi \mathcal{R} \psi))
\]

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

\[
\text{G } \varphi \equiv \varphi \land \text{X}(\text{G } \varphi) \\
\text{F } \varphi \equiv \varphi \lor \text{X}(\text{F } \varphi)
\]
Negation Normal Form (NNF)

Definition (Negation Normal Form (NNF))

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

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.

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

First Ideas

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

We then can evaluate

- true and false,
- $\varphi \lor \psi$ by evaluating $\varphi$, evaluating $\psi$
  and computing $\varphi \lor \psi$.
- $p$ 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 \( w \models X \varphi \) 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.

\[
\varphi U \psi \equiv \psi \lor (\varphi \land X(\varphi U \psi)) \\
\varphi R \psi \equiv \psi \land (\varphi \lor X(\varphi R \psi))
\]

\begin{align*}
evl_{FLTL}(w, \text{true}) &= \top \\
evl_{FLTL}(w, \text{false}) &= \bot \\
evl_{FLTL}(w, \varphi \lor \psi) &= evl_{FLTL}(w, \varphi) \lor evl_{FLTL}(w, \psi) \\
evl_{FLTL}(w, \varphi \land \psi) &= evl_{FLTL}(w, \varphi) \land evl_{FLTL}(w, \psi) \\
evl_{FLTL}(w, p) &= (p \in w_1) \\
evl_{FLTL}(w, \neg p) &= (p \notin w_1)
\end{align*}
evlFLTL

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

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

$$
\begin{align*}
\text{evlFLTL}(w, \varphi \lor \psi) &= \text{evlFLTL}(w, \psi \lor (\varphi \land \mathcal{X}(\varphi \lor \psi))) \\
\text{evlFLTL}(w, \varphi \land \psi) &= \text{evlFLTL}(w, \psi \land (\varphi \lor \mathcal{X}(\varphi \land \psi))) \\
\text{evlFLTL}(w, \mathcal{F} \varphi) &= \text{evlFLTL}(w, \varphi \land \mathcal{X} \mathcal{F} \varphi) \\
\text{evlFLTL}(w, \mathcal{G} \varphi) &= \text{evlFLTL}(w, \varphi \land \mathcal{X} \mathcal{G} \varphi) \\
\text{evlFLTL}(w, \mathcal{X} \varphi) &= ((|w| > 1) \land \text{evlFLTL}(w^2, \varphi)) \\
\text{evlFLTL}(w, \mathcal{X} \varphi) &= ((|w| = 1) \lor \text{evlFLTL}(w^2, \varphi))
\end{align*}
$$

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. $\text{evlFLTL}$ 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.
Lecture 2

A Quantum Secret Sharing Scheme using Orbital Angular Momentum onto Multiple Spin States based on Fibonacci Compression Encoding

by Hong Lai
A Quantum Secret Sharing Scheme using Orbital Angular Momentum onto Multiple Spin States based on Fibonacci Compression Encoding

Hong Lai
School of Computer and Information Science,
Southwest University, Chongqing

What is secret sharing?

- a method of distributing a secret amongst a group of people by giving each person a share (or shadow) of the secret.

- Schemes
  - Splitting
  - Threshold
Splitting

- split a secret $S$ amongst $k$ people such that all $k$ people are needed to reconstruct the secret;
- choose $k - 1$ random numbers $r_1, \ldots, r_{k-1}$, and give them to $k - 1$ of the people and give

$$S - \sum_{i=1}^{k-1} r_i \pmod{n}$$

- to the remaining person

Threshold

- split a secret $S$ into $n$ pieces such that any group of people can reconstruct the secret, but a group of less than $k$ people cannot $k$
- called a $(k, n)$ threshold scheme
Why do we want $(k, n)$?

- **security vs. reliability**
  - why not keep the key in one very safe place?
  - why not have multiples copies?

- **safety vs. convenience**
  - copies are convenient but easier to misuse
  - requiring all parties is inconvenient

Combinatorial Approach

- need $\binom{n}{k}$ locks
- each of the $k$ people need $\binom{n-1}{k-1}$ keys
- secret still exists in one place . . .
- very impractical!
Shamir Approach

- invented by Adi Shamir in 1979
- based on $k$ points uniquely determining a polynomial of degree $k-1$
- we divide our secret $S$ into pieces $S_i$ by picking a random $k-1$ degree polynomial in which, and

$$q(x) = a_0 + a_1x + a_2x^2 + \cdots + a_{k-1}x^{k-1}$$

$$a_0 = S \quad S_1 = q(1), S_2 = q(2), \ldots, S_n = q(n)$$

Shamir Approach (continued)

- represent each share as a point $(x_i, q(x_i) = y_i)$
- all arithmetic done modulo a prime number $P$ that is greater than both $S$ and $n$ - why?
- coefficients $a_1, \ldots, a_{k-1}$ of $q(x)$ are randomly chosen from a uniform distribution over the integers in $[0, P)$
Shamir Approach (continued)

- uses Lagrange Interpolation

\[ P(x) = \sum_{i=1}^{k} y_i \prod_{j=1, j \neq i}^{k} \frac{x-x_j}{x_i-x_j} \]

Example 2:
(3, 5) threshold scheme

\[ n = 5 \]
\[ k = 3 \]
\[ S = 7 \]
\[ a_0 = S \]
\[ a_1 = 3 \]
\[ a_2 = 5 \]
\[ p = 11 \]

\[ q(x) = 5x^2 + 3x + 7 \text{ (mod 11)} \]

\[ S_1 = q(1) = 5(1)^2 + 3(1) + 7 \equiv 4 \text{ (mod 11)} \]
\[ S_2 = q(2) = 5(2)^2 + 3(2) + 7 \equiv 0 \text{ (mod 11)} \]
\[ S_3 = q(3) = 5(3)^2 + 3(3) + 7 \equiv 6 \text{ (mod 11)} \]
\[ S_4 = q(4) = 5(4)^2 + 3(4) + 7 \equiv 2 \text{ (mod 11)} \]
\[ S_5 = q(5) = 5(5)^2 + 3(5) + 7 \equiv 4 \text{ (mod 11)} \]

- give each of the \( n \) people a share \( S_i \)
Example 2 (continued)

- Suppose people with shares $S_1 = 4$, $S_2 = 0$, and $S_3 = 4$
- decide to reconstruct the secret

\[
P(x) = [4\frac{(x-2)(x-5)}{(1-2)(1-5)} + 0\frac{(x-1)(x-5)}{(2-1)(2-5)} + 4\frac{(x-1)(x-2)}{(5-1)(5-2)}] \pmod{11}
\]

\[
P(x) = [(x-2)(x-5) + 4(x-1)(x-2)] \pmod{11} = 5x^2 + 3x + 7 \pmod{11}
\]

\[S = P(0) = 7\]

The deadly disadvantage of Shamir secret sharing scheme:

The secret shares can be eavesdropped by Eve without being detected.
Heisenberg's uncertainty principle

\[ (\Delta x) \times (\Delta p) \approx \hbar \]

Position \quad Momentum \quad 10^{-34}

QKD + OTP

Detecting eavesdropping

Quantum cryptography

SUCCESS
China launches world’s first quantum satellite

“Much better than expected”: Chinese ‘hack-proof’ quantum communication satellite put into service
Fibonacci numbers

\[ F_0 = 1 \quad F_{-1} = 0 \]

\[ F_n = F_{n-1} + F_{n-2} \quad \text{for} \quad n > 0 \]

\[ F_n = \frac{1}{\sqrt{5}} \left( \phi^{n+1} - \hat{\phi}^{n+1} \right) \]

\[ \phi = \frac{1 + \sqrt{5}}{2} = 1.618 \quad \hat{\phi} = \frac{1 - \sqrt{5}}{2} = -0.618 \]

\[ 0 \quad 1 \quad 1 \quad 2 \quad 3 \quad 5 \quad 8 \quad 13 \quad 21 \quad 34 \quad 55 \quad 89 \quad 144 \ldots \]

Basis elements of a numeration system

Binary representations of integers

Basis elements: 128 64 32 16 8 4 2 1

\[ 45 = 0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1 \quad 0 \quad 1 \]

Fibonacci: 55 34 21 13 8 5 3 2 1

\[ 45 = 0 \quad 1 \quad 0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0 \]

No adjacent 1’s
Standard Fibonacci

Codeword length: \[
\left\lfloor \log_2 M \right\rfloor \quad \left\lfloor \log_\phi (\sqrt{5M}) - 1 \right\rfloor = \left\lfloor 1.44 \log_2 M - 1 \right\rfloor
\]

Probability of 1-bit: \[
\frac{1}{2} \quad \frac{1}{2} \left(1 - \frac{1}{\sqrt{5}}\right) = 0.276
\]

Avg # of 1-bits: \[
0.5 \log_2 M \quad 0.398 \log_2 M
\]

<table>
<thead>
<tr>
<th>standard</th>
<th>Fib</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
</tr>
<tr>
<td>5</td>
<td>101</td>
</tr>
<tr>
<td>6</td>
<td>110</td>
</tr>
<tr>
<td>7</td>
<td>111</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
</tr>
<tr>
<td>9</td>
<td>1001</td>
</tr>
<tr>
<td>10</td>
<td>1010</td>
</tr>
<tr>
<td>11</td>
<td>1011</td>
</tr>
<tr>
<td>12</td>
<td>1100</td>
</tr>
<tr>
<td>13</td>
<td>1101</td>
</tr>
<tr>
<td>14</td>
<td>1110</td>
</tr>
</tbody>
</table>

**Problem: concatenations**

\[
1 \quad 11 \quad 3 \quad 6
\]

\[
1 \quad 1011 \quad 11 \quad 110
\]

\[
1101111110
\]

\[
0001 \quad 1011 \quad 0011 \quad 0110
\]

\[
1 \quad 10100 \quad 100 \quad 1001
\]

\[
1110100110011001
\]

1 serves as comma
Technical Problem:

code is UD but not instantaneous

11011111110
110 | 11 | 11 | 11 | 10
1101 | 11 | 11 | 110

Reverse codewords

After appending a 1

Get a Prefix code
Usefulness of Fibonacci Codes

- Better compression than standard codes
- More robust than Huffman codes

A QSS SCHEME BASED ON FIBONACCI COMPRESSION ENCODING
A QSS SCHEME BASED ON FIBONACCI

COMPRESSSION ENCODING

Step 1. Compressing the secret. Dealer sets up the secret sharing. She computes the frequency of every block (here, each sentence is considered as a block when the $S$ is the text) of $S$ and represents them in Fibonacci bases such as 34, 21, 13, 8, 5, 3, 2, 1. (see Definition 2).

The compressing process is performed in clearly separated stages:

1. Compute the frequencies of the collection of bits which are encodings of text or integers.
2. Rank the collection of bits by the frequencies.
3. Compute the Fibonacci code of each ranking.
4. Output the ranking as the header of the compressed secret.
5. Reread the input secret, using the code table to generate output to the compressed secret.

Step 2. Quantum encoding. The step aims to convert the binary codes 1s and 0s into single photon's polarizations Vs and Hs. Dealer encodes the frequencies with OAM by assigning a twisted number of a single photon to each Fibonacci value: $1 \rightarrow [1], 2 \rightarrow [2], \ldots, 34 \rightarrow [34]$. That is to say that only 8 different twists are required in terms of Chen and She's work [27] (see Figure 1). Moreover, Dealer encodes every character into binary codes, i.e., 1s and 0s which are represented by single photon's polarizations Vs and Hs. Also, to remove this encoding ambiguity, we append a 1 to every character binary codes, which is used to act as a “comma”, separating consecutive codewords.
Step 3. Distributing key. As shown in Figure 1, sending the quantum shares is achieved by the switching of an array of angularly separated light sources illuminating the static phase mask. After the static modulation, the input planar photons are converted into twisted ones with coinciding propagation direction and each twist number assigned is selectively dependent on their inclinations in front of the phase mask.

Next the encoded twisted photons are sent into the free-space link by an afocal telescope. Dealer’s qudit receiver consists of a similar telescope, the aforementioned OAM sorter, and an array of photodiode detectors (DDDDDDDDDDDD) aligned to monitor the output intensity of each port and decode the message. Dealer can encode the secret message as binary codes (spin), and send quantum shares—the corresponding multiple spin states and publish the determinants of Fibonacci diagonal matrices to participants $P_1, P_2, \ldots, P_m$ through quantum and classical channels respectively.
Example 2. The secret message is “action please”

<table>
<thead>
<tr>
<th>Op</th>
<th>Progress</th>
<th><em>Base</em></th>
<th>Reed–Solomon Representation</th>
<th>Code</th>
<th>Cof</th>
<th>Multiple spin states</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>d</td>
<td>2</td>
<td>d5</td>
<td>01</td>
<td>0</td>
<td>YY</td>
</tr>
<tr>
<td>a</td>
<td>e</td>
<td>3</td>
<td>d6</td>
<td>02</td>
<td>1</td>
<td>VY</td>
</tr>
<tr>
<td>a</td>
<td>f</td>
<td>4</td>
<td>d8</td>
<td>0001</td>
<td>0</td>
<td>0000</td>
</tr>
<tr>
<td>a</td>
<td>f</td>
<td>5</td>
<td>d9</td>
<td>0010</td>
<td>1</td>
<td>0000</td>
</tr>
<tr>
<td>a</td>
<td>f</td>
<td>6</td>
<td>d10</td>
<td>0011</td>
<td>0</td>
<td>0000</td>
</tr>
<tr>
<td>a</td>
<td>f</td>
<td>7</td>
<td>d11</td>
<td>0011</td>
<td>1</td>
<td>0000</td>
</tr>
<tr>
<td>a</td>
<td>f</td>
<td>8</td>
<td>d12</td>
<td>0011</td>
<td>0</td>
<td>0000</td>
</tr>
<tr>
<td>a</td>
<td>f</td>
<td>9</td>
<td>d13</td>
<td>0011</td>
<td>1</td>
<td>0000</td>
</tr>
<tr>
<td>a</td>
<td>f</td>
<td>10</td>
<td>d14</td>
<td>0011</td>
<td>0</td>
<td>0000</td>
</tr>
</tbody>
</table>

Recovering secret phase

According to the secret message “action please”, the final output quantum shares based on output binary strings are as follows.

Output binary strings:

Output quantum shares–multiple spin states:
- Header/VV/HHV/HVHHVV/VHVV/VHHVV/VVHV/HVHV/VHHVH/VVH/VV/HHHVV
- H/VV.

Dealer sends VV to P_1; HHVV to P_2; ..., HVV to P_{10}. At the same time, Dealer publish det(Q_1^2), det(Q_1^3), ..., det(Q_1^5).
Step 1. Eavesdropping Detection. Upon receiving these multiple spin states, the participants \([P_1, P_2, \ldots, P_n]\) can obtain their qubits. According to Table I, they first verify the determinants of Fibonacci diagonal matrices consisted of the Bank’s Fibonacci basis in terms of Eq. (5). If they are different, they abort the communication and the final output is FAIL. Otherwise, they continue to obtain the matching codes.

Step 2. Secret Recovery. After confirming that the secret shares received by all participants are valid, Denby informs the combiner (which can be any participant) the positions of every character or integer of the collection of bits. Finally, with all participants’s secret shares, the shared secret secret can be recovered in terms of Table I. So, the final output is \(S\).

TABLE II: Performance comparison between standard binary encoding and Fibonacci encoding, where \(M\) is the maximal bytes of codewords length.

<table>
<thead>
<tr>
<th>The way to encode</th>
<th>Standard binary encoding</th>
<th>Fibonacci compression encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>The length of Codeword</td>
<td>([\log_2^M])</td>
<td>([\log_2^M + 1 - 1] = [1.44\log_2^M - 1])</td>
</tr>
<tr>
<td>The Probability of 1-bit guess</td>
<td>(\frac{1}{2})</td>
<td>(\frac{1}{2}(1 - \frac{1}{2}) = 0.276)</td>
</tr>
<tr>
<td>Average codeword length of 1-bit</td>
<td>([0.5\log_2^M])</td>
<td>([0.38\log_2^M])</td>
</tr>
</tbody>
</table>
Thank You!
Lecture 3

Multilevel Modelling Tooling with MultEcore

by Adrian Rutle
Multilevel Modelling
Tooling with MultEcore

Adrian Rutle
Collaboration with Fernando Macías and Volker Stolz
https://www.hvl.no/person/?user=3600543
Chongqing, China, October 16, 2017

Why Modelling?

What software engineers understand
Technical Language

What domain experts understand
Domain Natural Language

What computers understand
Formal Language

Happy zone of understanding
Domain Specific Modelling Language (DSML)
### MOF-based approaches

**Pros**
- High reliability
- Mature (meta)modelling ecosystems
- Good tool coverage

**Cons**
- Mixed abstraction levels
- Synthetic typing relation
- Convoluted

### Multilevel approaches

**Pros**
- Unbounded number of levels
- Deep hierarchies (potency)
- Linguistic extensions

**Cons**
- Lack of clear consensus on the foundations
- No common focus in current multilevel tools
- Technology lock-in
Common realization of MLM: Clabject

Adapted from: Melanee Project – https://melanee2.informatik.uni-mannheim.de/confluence/

**Issues**
- Requires a linguistic metamodel
- Every element needs a linguistic type
- Synthetic typing and flattening of the ontological stack
- Custom tools and representations

Our realization of MLM: MultEcore

**Ontological stack** Does not require linguistic metamodels, synthetic typing relations or flattening

**Supplementary metamodels** Multiple and independent metamodels orthogonal to the ontological stack

**Supplementary typing** Loose and flexible. An element may have none, one or several supplementary types

**Sliding window** Reuse of the two-level cascading technique
Our realization of MLM: MultEcore

**Ontological stack** Does not require linguistic metamodels, synthetic typing relations or flattening

**Supplementary metamodels** Multiple and independent metamodels orthogonal to the ontological stack

**Supplementary typing** Loose and flexible. An element may have none, one or several supplementary types

**Sliding window** Reuse of the two-level cascading technique
Our realization of MLM: MultEcore

**Ontological stack** Does not require linguistic metamodels, synthetic typing relations or flattening

**Supplementary metamodels** Multiple and independent metamodels orthogonal to the ontological stack

**Supplementary typing** Loose and flexible. An element may have none, one or several supplementary types

**Sliding window** Reuse of the two-level cascading technique

---

![Diagram](https://example.com/diagram.png)
MultEcore tool

MEF representation for EMF

mi.mef

mef2xmi

xmi2mef

mef2ecore

ecore2mef

Editor

mi+1.mi

configures

typed by

creates
Applications – Behavioural metamodelling

Part
M1
Machine

cr:creates

m1
M1
p1
P1
m1

META
FROM
TO

PLS example with two-level approach

Adapted from the PLS case study at
http://atenea.lcc.uma.es/index.php/Main_Page/Resources/ReusableObservers
PLS example with two-level approach
Adapted from the PLS case study at http://atenea.lcc.uma.es/index.php/Main_Page/Resources/ReusableObservers

PLS example with multilevel approach (Stool)
Defining semantics for behavioural models

- We use model transformations:
  - Inherently multilevel
- Commonality can be exploited for reusability
  - Across languages (horizontally)
  - Inside the same stack of languages (vertically)
- Generic if defined on higher levels
- Trade-off between genericity may lead to imprecision

Our proposal: Multilevel transformations coupled with a meta-level: precise and reusable

Option 1: Two-level rule

- Create an instance of Handle for each instance of GenHandle
- Create an instance of Head for each instance of GenHead

FROM 
GenHead ghe
 TO 
 cr:creates
GenHead ghe

h

(b) hammer_plant

(d) hammer_config_1
Option 1: Two-level rule

Problems
- Too specific
- Difficult to reuse
  - Each machine needs a new rule
  - Each language or hierarchy needs its set of similar rules
- Leads to proliferation
Option 2: Multilevel rule

Create an instance of Part for each instance of Machine

FROM Machine m | TO Part p cr:creates Machine m
Option 2: Multilevel rule

Problems
- Too generic
- Not precise
  - All machines will create parts
  - All parts can be created directly
  - Any machine can create any part

FROM
<table>
<thead>
<tr>
<th>Machine</th>
<th>m</th>
</tr>
</thead>
<tbody>
<tr>
<td>Part</td>
<td>p</td>
</tr>
<tr>
<td>cr:creates</td>
<td>m</td>
</tr>
</tbody>
</table>

Option 3: Multilevel Coupled rule

- Create an instance of each specific Part for each instance of Machine that generates that Part

FROM
<table>
<thead>
<tr>
<th>META</th>
<th>FROM</th>
<th>TO</th>
</tr>
</thead>
<tbody>
<tr>
<td>m1</td>
<td>P1</td>
<td>cr:creates</td>
</tr>
</tbody>
</table>
Option 3: Multilevel Coupled rule

Advantages

✓ Generic enough
✓ Reusable in different configurations
✓ Reusable in different languages
✓ Precise enough
✓ Only the right machine creates a part
**Comparison**

<table>
<thead>
<tr>
<th>Two-level</th>
<th>Multilevel</th>
<th>Multilevel Coupled</th>
</tr>
</thead>
<tbody>
<tr>
<td>FROM</td>
<td>TO</td>
<td>cr:creates</td>
</tr>
<tr>
<td>L</td>
<td>I</td>
<td>R</td>
</tr>
<tr>
<td>PO</td>
<td>PBC</td>
<td>l</td>
</tr>
<tr>
<td>m</td>
<td>k</td>
<td>...</td>
</tr>
</tbody>
</table>

**Multilevel Coupled Rules – CreatePart**

```plaintext
rule CreatePart {
  meta {
    P1: mm[0]!Part
    M1: mm[0]!Machine
    cr: mm[0]!Machine.creates
    [M1.cr = P1]
  }
  from {
    m1: M1
  }
  to {
    p1: P1
    m1: M1
    cr: cr
    [m1.cr = p1]
  }
}
```
Multilevel Coupled Rules – *SendPartOut*

```java
rule SendPartOut {
    meta {
        P1: mm[0]!Part
        M1: mm[0]!Machine
        C1: mm[0]!Container
        cr: mm[0]!Machine.cr
        out: $mm[0]!Machine.out
        contains: $mm[0]!Container.contains
    }

    from {
        p1: P1
        m1: M1
        c1: C1
        cr: cr
        out: out
    }

    to {
        p1: P1
        m1: M1
        c1: C1
        out: out
        c: contains
    }
}
```

Multilevel Coupled Rules – *TransferPart*

```java
rule TransferPart {
    meta {
        P1: mm[0]!Part
        Conveyor: $mm[1]!Conveyor
        Tray: $mm[1]!Tray
        contains: $mm[0]!Container.contains
        cout: $mm[1]!Conveyor.cout
    }

    from {
        p1: P1
        t1: Tray
        c1: Conveyor
        cout: cout
    }

    to {
        p1: P1
        t1: Tray
        c1: Conveyor
        cout: cout
        co2: contains
    }
}
```
rule Assemble {
  meta {
    P1, P2, P3: mm[0]!Part
    C1, C2: mm[0]!Container
    M1: mm[0]!Machine
    c: $mm[0]!Container.contains
    in: $mm[0]!Machine.in
    out: $mm[0]!Machine.out
    has1, has2: mm[0]!Part.has
    [P3.has1 = P1]
    [P3.has2 = P2]
  }
  from {
    p1: P1, p2: P2
    c1: C1, c2: C2
    m1: M1
    i: in, o: out, co1: c, co2: c
    [c1.co1 = p1]
    [c1.co2 = p2]
    [m1.i = c1]
    [m1.out = c2]
  }
  to {
    p3: P3
    c1: C1, c2: C2
    m1: M1
    i: in, o: out, co3: c
    [c2.co3 = p3]
    [m1.i = c1] [m1.o = c2]
  }
}
Robolang example

Initial
Task
1-1
GoForward
Task
1-1
GoBack
Task
1-1
TurnLeft
Task
1-1
TurnRight

Border
Input
1-1
Obstacle
Input
1-1
Timeout
Input
1-1

in1@1-1
out1@1-1
in@2
out@2
in2@1-1
out2@1-1
in@2
out@2
in3@1-1
out3@1-1
in@2
out@2
in4@1-1
out4@1-1
in@2
out@2
in5@1-1
out5@1-1
in@2
out@2
in6@1-1
out6@1-1
in@2
out@2
in7@1-1
out7@1-1
in@2
out@2
Robolang example
Robolang Multilevel Coupled Rules – Start

rule Start {
  meta {
    I: mm[0]!Initial
    X: mm[0]!Task
    T: mm[0]!Transition
    in: mm[0]!Transition.in
    out: mm[0]!Transition.out
    [T.in = I]
    [T.out = X]
  }
  from {
    i: i
  }
  to {
    x: X
    t: T
    in: in
    out: out
    [t.in = i]
    [t.out = x]
  }
}

Robolang Multilevel Coupled Rules – FireTransition

rule FireTransition {
  meta {
    X: mm[0]!Task
    Y: mm[0]!Task
    I: mm[0]!Input
    T: mm[0]!Transition
    inputs: mm[0]!Transition.inputs
    in: mm[0]!Transition.in
    out: mm[0]!Transition.out
    [T.inputs = I]
    [T.in = X]
    [T.out = Y]
  }
  from {
    x: X
    i: i
  }
  to {
    x: X
    y: Y
    i: i
    t: T
    in: in
    out: out
    inputs: inputs
    [t.inputs = I]
    [t.in = x]
    [t.out = y]
  }
}
rule DeleteTask {
    meta {
        X: mm[0]!Task
        Y: mm[0]!Task
        T: mm[0]!Transition
        in: mm[0]!Transition.in
        out: mm[0]!Transition.out
        [T.in = X]
        [T.out = Y]
    }
    from {
        x: X
        y: Y
        t: T
        in: in
        out: out
        [t.in = x]
        [t.out = y]
    }
    to {
        y: Y
    }
}

rule InsertInput {
    meta {
        I: mm[0]!Input
    }
    from {
    }
    to {
        i: I
    }
}
rule InsertEffectiveInput {
    meta {
        X: m0!Task
        Y: m0!Task
        I: m0!Input
        T: m0!Transition
        inputs: m0!Transition.inputs
        in: m0!Transition.in
        out: m0!Transition.out
        [T.inputs = I]
        [T.in = X]
        [T.out = Y]
    }
    from {
        x: X
    }
    to {
        x: X
        i: I
    }
}

rule DeleteInput {
    meta {
        I: m0!Input
    }
    from {
        i: I
    }
    to {
    }
}
Why Runtime Verification?

- DSML do not shield the software from design errors
- Runtime Verification checks the execution of real system
  - Consider environmental influences
  - React to failures
- Testing is seldom exhaustive
- Model Checking can not always guarantee the correctness of executing system

Runtime Verification

- Use runtime monitors to observe the run of systems
- Check whether the current execution of such systems violates given correctness properties
- Such correctness properties can be formulated in linear-time temporal logic, LTL
Multilevel Metamodelling

Example Scenario

\[ G(\text{obs} \rightarrow \chi(\neg \text{obs} \cup \text{to})) \]

- Generate Python code for the robot and monitor
Example Execution

Example Execution II

\[ \mathcal{G}(\text{obs} \rightarrow \lambda(\neg \text{obs} \leftarrow \text{to})) \]

\[ \mathcal{G}(\text{obs} \rightarrow \lambda(\neg \text{obs} \leftarrow \text{to})) \]

\[ \neg \text{obs} \leftarrow \text{to} \]
Model Transformation Rules

Current Snapshot \( S_n \) → Environment Rules → Intermediate Snapshot \( S_{i,n} \) → Behavior Rules → Next Snapshot \( S_{n+1} \)

Current State \( q_n \) → Monitor Rules → Next State \( q_{n+1} \)

Example Distributed Scenario

Romba
- GoFwd
- GoBack
- TurnLeft
- Timeout
- Border
- Found
- Obstacle
- FarObstacle

Spinner
- TurnRight
- TurnLeft
- GoFwd
- Happy
- GoBack
- Obstacle
- Found
- FarObstacle
- Bump
- Timeout
Formalisation

For $j \geq k > i \geq 0$

Typing morphism from graph $G_k$ at $k$ to graph $G_i$ at $i$ is defined as

$$T_{k,i} : D(T_{k,i}) \rightarrow G_i$$

Direct mappings $x \in G_k \mapsto ty(x) \in G_{k - df(x)} \mapsto ty^2(x) \in G_{k - df^2(x)}$

The type of $x \in G_k$ is $ty(x) = t \in G_{k - df(x)}$ where $df(x)$ is the difference in "abstraction" levels between $t$ and $x$, given by the potency on $t$.

$ty^2(x) = ty(ty(x))$, $ty^3(x) = ty(ty^2(x)) = ty(ty(ty(x)))$, ...

$df^2(x) = df(x) + df(ty(x))$, $df^3(x) = df^2(x) + df(ty^2(x))$, ...

Formalisation

Diagram:

- Level $i$ with $G_i$
- Level $j$ with $G_j$
- $D(T_{j,i})$
- $\tau_{j,i}$
- $\tau_{j,i}$

Lines:
- $\tau_{j,i}$ from $G_i$ to $G_j$
- $\tau_{j,i}$ from $G_j$ to $G_i$
Formalisation

- Metamodelling hierarchy is based on typed graphs and graph homomorphisms.
- Potencies for deep characterization supported by allowing the typing relation to “jump” over levels.

For $j \geq k > i \geq 0$

Typing morphism from graph $G_k$ at $k$ to graph $G_i$ at $i$ is defined as

$$T_{k,i} : D(T_{k,i}) \rightarrow G_i$$

Direct mappings $x \in G_k \mapsto ty(x) \in G_{k-df(x)} \mapsto ty^2(x) \in G_{k-df^2(x)}$

Conclusions

- MultEcore, an alternative framework for multilevel modelling
- Applied to behavioural metamodelling and RV
- Tool as an Eclipse plugin, bypassing EMF’s two-level limitation
  - Small learning curve
  - Mature ecosystem and toolset

Future Work

- New multilevel functionalities: navigation of typing relations
- Creation of a hierarchy of behavioural models
- Multilevel Coupled Model Transformations (*Stay tuned!*)

http://prosjekt.hib.no/ict/multecore/
Conclusions

- Coupled model transformations exploit the advantages of multilevel metamodelling:
  - Reusability
  - Genericity
  - Precission
- Application to behavioural metamodelling natural and straightforward

Future Work

- Full formalization
- Expand horizontal and vertical flexibility
- Implement tool support

Conclusion and outlook

- Present a metamodel that captures a wider range of aspects of the robots
  - Sensors
  - Motors
  - Communication
- Integrate runtime verification into the whole software engineering process
  - Design
  - Simulation
  - Code generation for the robot and monitor
- Allow the design of distributed systems in the future
  - Distribution can be modeled through replication of existing instances on the modeling level
Lecture 4

CPN Models for Fault-tolerant Distributed Systems

by Rui Wang
CPN models for Fault-tolerant Distributed Systems
- Single-decree Paxos

Rui Wang
rwa@hvl.no

1Department of Computing, Mathematics, and Physics,
Western Norway University of Applied Sciences, Norway

The BeChong Summer School, Oct. 2017

Motivation

- Cloud-based services and distributed systems
  - Correctness
  - Availability
  - Fault-tolerance

- Distributed replication protocols, such as Paxos, can make it possible to handle the consensus problem and ensure fault-tolerance

- Challenges
  - Difficult to understand
  - Design, implementation and testing are challenging and error-prone

Overview

- Distributed replication protocols
  - Consensus problem
  - Fault-tolerance

- Coloured Petri Nets (CPNs)
  - Modeling and verifying models of complex distributed systems
  - Explore the behaviors of such systems and analyze their properties

Research Goal

Investigate the use of Coloured Petri Nets (CPNs) for modeling a complex distributed replication protocol: Paxos.

---

Outline

- Quorum-based distributed systems and the Gorums framework
- Single-decree Paxos
- CPN model for single-decree Paxos
- State space analysis
- Conclusions and future work

---

**Quorums**

- Abstraction for ensuring consistency and fault-tolerance in distributed systems through replication

- **A quorum system**: a collection of subsets of nodes, called quorums, where every pair of subsets intersect\(^3\)

- Example: Read and write quorum system with majority quorum

![Quorum System Diagram](image)

---


---

**Gorums Framework\(^4\)**

- A software framework implemented in the Go programming language

- Ease the implementation effort of Quorum-based distributed systems

- Build advanced distributed algorithms and storages, which rely on a quorum system to achieve fault-tolerance

---

Gorums Abstraction

- **Quorum calls**
  invoke RPCs on set of processes and collect responses

- **Quorum functions**
  process responses to determine if a quorum has been obtained

Single-decree Paxos

- The most basic of the Paxos family
- Agree on a common value among distributed participants
- All participants only need to decide on one single consensus value
Paxos Roles

- **Proposer**: propose value for consensus
- **Acceptor**: choose the consensus value
- **Learner**: learn the consensus value
- **Leader**: the proposer that obtained promise by Acceptors

Quorums and Quorum calls for Paxos

- Express the safety properties of Paxos by ensuring at least some surviving processes
- Abstraction for ensuring consistency and fault-tolerance in distributed systems through replication
- For the Paxos protocol, Quorums can be defined as subsets of Acceptors
- **Quorum calls**\(^5\) can be used by Paxos roles to communicate each other

Two Communication Phases of Single-decree Paxos

- **Phase One**: Prepare(crnd) and Promise(crnd,(vrnd,vval))

- **Phase Two**: Accept(crnd,cval) and Learn(crnd,cval)
A Successful Communication Round

Crash and Partition
Leader Change

s1 Leader

s2 Prepare

s3 Promise

Partition

Leader

Accept

Promise

Learn

Leader Change

s1 Leader

s2 Leader

s3 Prepare

Promise

Promise

Prepare

Leader
Leader Change

- Leader
- Prepare
- Partition
- Promise
- Ignored

Round Number

- Added round number crnd in messages to identify the leader
- Round numbers are assigned:
  - S1: 1, 4, 7, ...
  - S2: 2, 5, 8, ...
  - S3: 3, 6, 9, ...
CPN Model for Single-decree Paxos

State Space Analysis

- A state space is a directed graph
  - A node: each reachable marking (state)
  - An arc: each occurring binding element
- State spaces can be used to investigate the behavioral properties of the model
State Space Analysis

地道

Statistics

State Space
Nodes: 5819
Arcs: 23724
Secs: 6
Status: Full

Scc Graph
Nodes: 5819
Arcs: 23724
Secs: 0

Properties analysis

Home Properties

Home Markings
None

Liveness Properties

Dead Markings
[5818, 5819]

Dead Transition Instances
InitProposer'Increase_Round_LFD 1
InitProposer'Increase_Round_PFD 1
Learner_Failure_Detector'Timeout_Change_leader 1
Proposer_Failure_Detector'Replace_Leader 1
Proposer_Failure_Detector'Timeout_Change_leader 1
Quorum_Calls'Prepare_Value_For_Same_Lader 1

Live Transition Instances
None

Fairness Properties

No infinite occurrence sequences.
Conclusions

- Completed construction of a CPN model for the single-decree Paxos
  - Understanding of Paxos protocol
  - Foundation for building Multi-decree Paxos and other distributed protocols

- State space exploration was applied to analyze a successful round of two communication phases of the model

Future Work

- The constructed model can serve as a testing model for the implementation of the single-decree Paxos
  - Perform model-based testing
  - Investigate testing approaches for such complex distributed protocols

- Improve current CPN model for the state space exploration and consider more complex scenarios of state space analysis
  - Failures
  - Retransmission
  - Leader changing
Lecture 5

Formal Analysis of Service-level Agreements for Cloud services with ABS

by Violet Ka I Pun
Formal analysis of Service-level Agreements for Cloud services with ABS

Violet Ka I Pun
(& many other people)

BeChong Summer School
Southwest University, Chongqing, China
16 – 19 October 2017

What are Formal Methods?
**Timeliness**

Formal methods are gaining traction

Formal methods increasingly applied to complex systems

- Traditional usage limited to **critical systems**
- **Enabler**: Huge progress in automation
- Traditionally heavyweight, breakthrough in **lightweight FMs**
- **Big success stories** in 2015: Amazon, Facebook, TimSort

**Correctness**

Violet Ka I Pun  SLA for Cloud with ABS
What are Formal Methods?

“Mathematically based techniques for the specification, development and verification of software”

Automated and tool-supported verification methods

- **Simulation** and visualization enables quick model exploration
- **Systematic testing** and automated **test case generation**
- **Automated theorem proving** can produce a formal proof given a system description, logical axioms, and inference rules
- **Model checking** can verify some properties by an exhaustive search through all states reachable during system execution
- **Abstract interpretation** verifies an over-approximation of a behavioural property of the program

The level of automation depends on the complexity of the model/program and on the complexity of the property
Programs & their models

Working at the code level
- Program based analysis
- **Complexity**: Must deal with the whole program
- **State space explosion!**

Working with abstractions
- Analyze models
- **Abstractions**: reduce state space, focus on relevant aspects
- How does the model relate to the real world?

Design space for analysis techniques
- Will my analysis overlook errors?
- Will my analysis report false errors?
- How easy is it to perform the analysis?
- How incremental is the analysis?
Short Introduction to ABS

What ABS Is All About

Consequences of design time decisions often realized only at runtime

- Modern SW development often model-/feature-driven
- Most modeling languages do not address behaviour rigorously
- Mismatch among artefacts from analysis and coding phases
- "Built-in" disconnect between analysts and implementors
- Complicating factors: product variability, concurrency
Design Languages: Mind the Gap!

- Design-oriented, architectural, structural
  - UML, FDL, etc.
- Implementation level
  - JML, SPEC#, etc.
- Minimalistic foundational
  - \( \pi \)-calculus, ambient c., etc.

Realistic

Abstract

Abstract Behavioural Specification
ABS

+ executability

+ verifiability

+ usability

Implementation level
JML, SPEC#, etc.

Minimalistic foundational
\( \pi \)-calculus, ambient c., etc.

The Abstract Behavioral Modeling Paradigm

A tool-supported formal modeling language for building highly adaptable and trustworthy software

Main ingredients

- Executable, formal modeling language for adaptable software:
  Abstract Behavioral Specification (ABS) language
- Tool suite for ABS/executable code analysis & development:
  Analytic functional/behavioral verification, resource analysis, feature consistency, RAC, types, TCG, visualization
  Generative code generation, model mining, monitor inlining, . . .
  Develop methods in tandem with ABS to ensure scalability
- Framework integrating tool architecture and ABS language
Motivation

Why formal?
- Informal notations can’t describe software behavior with rigor:
  - concurrency, compositionality, correctness, security, resources, . . .
- Formalization ⇒ more advanced tools
  - more complex products
  - higher automation: cost-efficiency

Why adaptable?
- Feature-rich software, deployment scenarios
- Changing requirements (technology/market)
- Evolution of software in unanticipated directions
- Language-supported adaptability is a key to successful reuse
- Reuse of designs, code and verification effort

Main Design Goals of ABS

ABS is designed with analysis/code generation tools in mind
- Expressivity carefully traded off with analysability
  - enables incremental/compositional static and dynamic analyses
- State-of-art programming language concepts
  - ADTs + functions + objects
  - type-safety, data race-freeness by design
  - modules, components
  - pluggable type systems, annotations
- Layered concurrency model
  Upper tier: asynchronous, no shared state, actor-based
  Lower tier: synchronous, shared state, cooperative multitasking
- Modeling of variability/resources with first-class language support
  - feature models, delta-oriented programming
  - deployment, abstract resources in the cloud
- Not just (static) analysis, but also code + test generation
<table>
<thead>
<tr>
<th>Feature Modeling Languages</th>
<th>Real-Time ABS Deployment Components</th>
<th>Runtime Components</th>
</tr>
</thead>
<tbody>
<tr>
<td>Behavioral Interface Specs</td>
<td>Local Contracts, Assertions</td>
<td>Syntactic Modules</td>
</tr>
<tr>
<td>Asynchronous Actor-Based Communication</td>
<td>Concurrent Object Groups (COGs)</td>
<td>Imperative Language</td>
</tr>
<tr>
<td>Object Model</td>
<td>Pure Functional Programs</td>
<td>Algebraic (Parametric) Data Types</td>
</tr>
</tbody>
</table>

**Example: Functional layer**

**User-defined (polymorphic) data types**

```haskell
data InfInt = Finite(Int inflntValue) | Infinite ;
```

**User-defined (polymorphic) functions**

```haskell
def Bool lessThan(InfInt d1, InfInt d2) =
case d1 {
    Infinite => False;
    Finite(v1) => case d2 {
        Infinite => True;
        Finite(v2) => v1 < v2;
    };
};
```

- User-defined data types and functions compress the size of a model significantly, compared to a purely OO model (approx. 50%)
Example: Imperative layer

Interfaces define behaviors (types for objects and references)

```java
interface CompareInfInts {
  InfInt findLargest(InfInt d1, InfInt d2);
  Bool compare(InfInt d1, InfInt d2);
}
```

Classes describe implementations of behaviors

```java
class Demo implements CompareInfInts {
  InfInt findLargest(InfInt d1, InfInt d2){
    InfInt reply = Finite(0);
    if (lessThan(d1,d2)) {reply=d2;} else {reply = d1;}
    return reply;
  }

  Bool compare(InfInt d1, InfInt d2){ return (lessThan(d1,d2));}
}
```

Communication and Synchronization in ABS

ABS decouples communication and synchronization

- ABS supports **asynchronous method calls**, using futures.
- Futures are first-class values: f1=x!m(); f2=y!p(f1);
- **Flexible synchronization**: blocking or suspending activities [Futures]
  - Blocking the object: `get-operation` on a future: v=f1.get;
  - Suspending the process: `polling` a future: `await` f1?
  - Polling as part of a guarded command: `await` b && f1? && f2?
- **Cooperative scheduling** of method activations
- Easy to **combine active and reactive behavior**
- **Concurrent object groups**: scope of cooperative scheduling [COGs]


Low-level Concurrency: Interleaving Semantics

Realized in Java, C, . . .

- Task preemption possible at any time
  - myriads of possible interleavings
  - data races
- Invariants tend to be implicit, very hard to establish
  - very difficult to ensure that result is independent of scheduling
- Cannot decompose formal analysis of concurrent programs

Concurrent Object Groups (COGs)

- Unit of distribution, one “processor” resource
- Own heap of shared objects and data
- (Cooperative) multitasking inside COGs
  - One processor, several tasks
  - Intra-group communication by synchronous (sequential)/asynchronous method calls
  - Multiple tasks originating from asynchronous calls within COG
- Inter-group communication only via asynchronous method calls

Concurrency Model

Layered Concurrency Model

Upper tier: asynchronous, no shared state, actor-based
Lower tier: synchronous + asynchronous, shared state, multitasking
Cooperative Multitasking and Invariant Reasoning

- Asynchronous method call creates new task
- Use future as handle for result
- Assume invariant $l_o$ at beginning of $m()$
- Caller continues interrupted unless explicit wait for result
- On ending $m()$ must reestablish its invariant $l_o$
- Only at suspension points the caller must establish its invariant $l_o$
- Futures can be passed around, multiple results synchronized

```java
Fut<T> f = o!m();
... await f? & g? ...
```

Cooperative Scheduling in ABS

Important Consequences

- Task suspension is syntactically explicit decision of modeller
- No preemptive scheduling ⇒ no data races
  - Scheduling has no influence on result of computation
- Non-deterministic scheduling otherwise
  - User-defined configuration of schedulers via annotations
  - Results of analysis of ABS programs valid for any scheduler

Reading Futures

- $f$.get - reads future $f$ and blocks execution until result is available
- Deadlocks possible (use static analyzer for detection)
- Programming idiom: use await $f?$ to prevent blocking (safe access)
  - Fut<T> $v = o!m(e);$...; await $v?$; $r = v$.get;
ABS Concurrency Model: Summary

Method calls with shared heap access encapsulated in COGs

COG
- One activity at a time
- One lock
- Cooperative scheduling
- Callbacks (recursion) ok
- Shared access to data

Violet Ka I Pun  SLA for Cloud with ABS
ABS Concurrency Model: Summary

Distributed computation: async. calls/message passing/separate heap

ABS for Cloud Services
Deployment: A Quick Look at the Landscape

Static deployment scenarios
“Putting software on machines”

Dynamic deployment scenarios
“The software puts itself on machines”

Cloud Computing

Is cloud computing different from distributed computing?

Cloud Computing
- Execution environment with elastic resource provisioning, several stakeholders, and a metered service at multiple granularities for a specified level of quality of service (QoS)
- A host offers services to clients, including infrastructure and platform functionalities and software services to virtualize resource deployment

Virtualization
- Virtualization provides an elastic amount of resources to application-level services, e.g., by allocating a changing processing capacity to a service depending on demand
- We say that application-level services are virtualized if they can adapt to the elasticity of cloud computing
Virtualization & SLA

Can we integrate these pieces for model-based analyses?

Conceptual Parts of a Deployed Cloud Service

Combine techniques based on abstract executable models
- **Formal modeling** using Abstract Behavioral Specifications (ABS)
- **Formal methods**: Verification, Performance Analysis, Cost Analysis, Advanced Type Systems, Code Generation, Test-Case Generation
- **Monitoring**: Framework to generate monitors for SLA-compliance
Modeling the Cloud

Berndnaut Smilde: Nimbus II, 2012

Model-based analysis of performance vs. cost

How will the response time and cost of running my system change if I double the number of servers?

How do fluctuations in client traffic influence the performance of my system on a given deployment architecture?

Can I control the performance of my system better by means of application-specific load balancing?

Use the model to predict behavior

Modeling Deployment Decisions

Abstract Behavioral Specification of Deployment Decisions

Design decisions

- **Abstraction**: Deployment decisions should be expressed at the abstraction level/style of the modeling language (avoid "model drift")
- **Incrementality**: deployment decisions can be added at any stage in the model development
- **Separation of concerns**: Deployment decisions are often fairly orthogonal to the behavioral code
- **Annotations**: Optional annotations locally override defaults

Quality of Service

- Deployment decisions are often closely related to timed behavior
- To compare decisions we use the *timed semantics* of Real-Time ABS
Measuring Quality-of-Service for ABS Models

Let us consider response time

Real-Time ABS
- Real-Time ABS combines OO modeling with linguistic primitives to specify real-time behavior
- Extends ABS with data types for time and duration
- Non-intrusive extension: Untimed models work in Real-Time ABS
- Associate deadlines to method calls

The passage of time
- ABS has a duality between blocking and suspending processes
- Real-Time ABS: block or suspend a process for a period of time
- **Explicit time**: duration(b,w), await duration(b,w)
- **Implicit time**: now() expression reads the current time
- Maximal progress semantics

Deadlines to method calls
- Processes have a local associated deadline
- deadline() expression: time remaining before local deadline
- Default deadline is infinity, can be overridden by annotations
Real-Time ABS Example

Method definition

```c
Bool m() {
    duration(2,5);
    return deadline() > 0;
}
```

Method calls:

**Regular method call:** Default deadline: infinity

```c
Bool success = o.m();
```

**Annotation with fixed deadline:**

```c
[Deadline: 5] Bool success = o.m();
```

**Annotation with state-dependent deadline:**

```c
[Deadline: e] Bool success = o.m();
```

Workloads and Deadline Misses

**Hard real-time scheduling problem**

Is a given workload (jobs with duration & deadline) schedulable?

Can we meet the deadlines?

**Priority-based “conventional” scheduling**

- **Hard real-time:** mission critical deadlines
- **Soft real-time:** missing deadlines = degradation of QoS

**Soft real-time requirements**

- What is a **good priority** for a video player decoding 24 frames per second without processing needs in-between?
- **Which jobs to miss?**

If we miss too many deadlines, the deployed service violates SLA
Why SLA-violation is a problem?

Service provider

Service requester

Service Level Agreements (SLAs)
- Legal contracts between service providers and customers
- Define the quality of service

Violet Ka I Pun SLA for Cloud with ABS
Solution:

- Approximate execution time for each task
- Schedule tasks to resources with “appropriate” speed

Approximate Execution Time

The service: Travel from \( x \) to \( y \)
Distance: \( \sim 460 \) km, Deadline: 6 hrs

The server:
Capacity: 60 km/hour
Approximate Execution Time

The service: Travel from \( x \) to \( y \)
Distance: \( \sim 460 \) km, Deadline: 6 hrs

The server:
Capacity: \( > 200 \) km/hour

Formal Analysis of SLA with ABS tool suite
The ABS tool suite

- Simulation tool
- Deadlock analysis tool
- Systematic testing tool
- Test case generation tool
- Termination and resource consumption tool
- ABS Smart Deployer
- Code generation tools
- Formal verification tool
- Monitoring framework for SLA metrics

http://abs-models.org/

Workflow of service configuration and deployment

**Negotiation Phase**

"the service completes 6 queries per minute from 9:00 to 18:00 and 4 queries per minute otherwise"

\[
ST_s([l.t, 365.t'), 60) =
\begin{cases}
4 & \text{if } t = 0 \text{ and } t' = 32399 \\
6 & \text{if } t = 32400 \text{ and } t' = 64800 \\
4 & \text{if } t = 64801 \text{ and } t' = 86399
\end{cases}
\]
Workflow of service configuration and deployment

The Service code – A case study from Fredhopper*

```java
String searchDB(Database DB, String s) {
    String u;
    u = DB.query(s);
    job(h);
    .......
    return u;
}
```

- resource-aware specification language
- job(h) needs h CPU resources
- the time is needed CPU resources
- allocated CPU resources
Static Analysis

```java
String searchDB(DataBase DB, String s) {
    String u;
    u = DB.query(s);
    job(15);
    ...
    return u;
}
```

```java
class DataBase {
    String query(String s) {
        ...
        job(5);
    }
}
```

Cost Equations

\[
\text{searchDB}(x, y) = \text{query}(y) + \frac{15}{x}
\]
\[
\text{query}(y) = \frac{5}{y}
\]

- \(x\) is the capacity of the object running `searchDB`, and
- \(y\) is the capacity of the one running `query`.

Time estimation

\[
\frac{15}{x} + \frac{5}{y}
\]

\[
\text{ST}_{\text{searchDB}}([1.t, 365.t'], 60) = \begin{cases} 
4 & \text{if } t = 0 \text{ and } t' = 32399 \\
6 & \text{if } t = 32400 \text{ and } t' = 64800 / 9am - 6pm \\
4 & \text{if } t = 64801 \text{ and } t' = 86399
\end{cases}
\]

“the service completes 6 queries per minute from 9:00 to 18:00 and 4 queries per minute otherwise”
### Static Analysis

**Cost Equations**

\[
\text{searchDB}(x, y) = \text{query}(y) + \frac{15}{x} \\
\text{query}(y) = \frac{5}{y}
\]

### Time estimation

\[
\frac{15}{2} + \frac{5}{1} = 12.5
\]

\[
\text{ST}_{\text{searchDB}}([1, t, 365, t'], 60) = \begin{cases} 
4 & \text{if } t = 0 \text{ and } t' = 32399 \\
6 & \text{if } t = 32400 \text{ and } t' = 64800 \quad (/9am - 6pm) \\
4 & \text{if } t = 64801 \text{ and } t' = 86399
\end{cases}
\]

---

### Adjustment

- **Time estimation**
  \[
  \frac{15}{2} + \frac{5}{1} = 12.5
  \]

---

"the service completes 6 queries per minute from 9:00 to 18:00 and 4 queries per minute otherwise"
Adjustment

Time estimation

\[
\frac{15}{2} + \frac{5}{2} = 10
\]

\[ST_{searchDB}([1.t, 365.t'], 60) =
\begin{cases}
4 & \text{if } t = 0 \text{ and } t' = 3299 \\
6 & \text{if } t = 32400 \text{ and } t' = 64800 \\
4 & \text{if } t = 64801 \text{ and } t' = 86399 \\
\end{cases}\]

No static violation detected

Runtime Analysis

- **Observation:**
  measurements on services are taken
- **Reaction:**
  adjustments for resource allocations

"the service completes 6 queries per minute from 9:00 to 18:00 and 4 queries per minute otherwise"
Runtime Analysis

- **Observation**: measurements on services are taken
- **Reaction**: adjustments for resource allocations

**Observation phase – Monitoring Add-on**
- The Service time monitor
  - intercepts all the HTTP invocations/replies to/from a service
  - records the time taken by every request to be completed
- E.g., a **throughput monitor** takes as input a logfile and a time window and returns true if the throughput complies with the definition of ST

**Reaction phase – Monitoring Platform**
- In case of violations, triggers a resource adjustment

---

ST\text{searchDB}([1.t, 365.t'], 60) = \begin{cases} 4 & \text{if } t = 0 \text{ and } t' = 32399 \\ 6 & \text{if } t = 32400 \text{ and } t' = 64800 \text{ /}9am - 6pm \\ 4 & \text{if } t = 64801 \text{ and } t' = 86399 \end{cases}

**Time estimation**
\[
\frac{15}{2} + \frac{5}{1} = 12.5
\]

- [Static Analysis] The initial 2 CPU resources respect the SLA.
- The throughput monitor records only 4 requests per minute.
Runtime Analysis

\[ ST_{searchDB}([1.t, 365.t'], 60) = \begin{cases} 
4 & \text{if } t = 0 \text{ and } t' = 32399 \\
6 & \text{if } t = 32400 \text{ and } t' = 64800 / 9am - 6pm \\
4 & \text{if } t = 64801 \text{ and } t' = 86399 
\end{cases} \]

**Time estimation**

\[ \frac{15}{2} + \frac{5}{2} = 10 \]

- **[Static Analysis]** The initial 2 CPU resources respect the SLA.
- The throughput monitor records only 4 requests per minute.
- Reaction: requests **1 additional** CPU resource.
- Too much resources during the night.
  - Respects the SLA, but
  - Waste of money and resources.

## Auxiliary Metrics

**Time estimation**

\[ \frac{15}{x} + \frac{5}{y} \]

- Each CPU resource \( x \) costs 10
- Each CPU resource \( y \) costs 20
Auxiliary Metrics

**Time estimation**

\[
\frac{15}{2} + \frac{5}{1} = 12.5
\]

- Each CPU resource \( x \) costs 10
- Each CPU resource \( y \) costs 20
- Total cost = 4060

---

**Budget metric function**

\[
\text{Budget}_{\text{searchDB}}([1, t, 365, t']) = \begin{cases} 
40 & \text{if } t = 0 \text{ and } t' = 32399 \\
60 & \text{if } t = 32400 \text{ and } t' = 64800 \text{ /} 9\text{am} - 6\text{pm} \\
40 & \text{if } t = 64801 \text{ and } t' = 86399 
\end{cases}
\]

- The static analysis will approve \( \text{Budget}_{\text{searchDB}} \).
- When the runtime CPU reallocation is triggered by the throughput monitor, the nightly budget is not met anymore.
- The budget monitor reacts by requiring a deallocation of the extra CPU resource during the night.
Summary

Statically verifiable

- Maximum number of virtual machines
- Upper bound of response time

Dynamically verifiable

- Availability
- Budget compliances
Lecture 6

Contract-Based Model of Evolving Architecture for Intelligent CPS

by Zhiming Liu
Points of Discussion

1. Introduction to CPS
2. Smart City, IoT, BigData and AI in CPS
3. Evolving architecture is essential — any existing CPS?
4. A Proposal for a contract-based architecture model
5. Advance rCOS to CPS
CPS

- A CPS combines a **cyber side** with a **physical side**.
- **Cyber side** – computing and networking
- **Physical side** – mechanical, electrical, and chemical processes
- The **cyber components** control the physical side using sensors and actuators, as well as providing services to users
- **Sensors** and **actuators** are interfaces that observe/sense the physical system and actuate the controls
- **Human actors** – involve human-machine interaction
- **CPS include IoT, Data Centres and M2M**

Software in CPS

- **Systems of multi-scales systems [SomS]**
  - Distributed computing
  - Managing internet and mobile communication networks [SDN]
  - Embedded software
  - Service based, OO, AI
- **Multi-scale: space, time and functionality**
- **Ubiquitous interaction of systems of computation, control, and services**
  - Complex dynamics, and require self-X
**BigData**

- Where are bigdata from, what they are for, and why 4Vs?
- Through sensors, “things” are virtualized as data so as to be identified and shared/trnasmitted
- Not only exit in CPS, collected, processed and used as historic data.
- Also dynamically used in and drive the evolution of CPS.
- The value of data as virtualisation of things is to create views from separate data sources.
- Views are what makes BigData useful, by link data sets together to deliver new products and services.
- Views implemented as APIs, and they are used to realise business processes and workflows.
- BigData are an integrate part of CPS and used to develop value added services and drive system’s evolving behavior.

**Evolving Architecture**

CPSs cannot be built from scratch, but they are ever evolving

1. Develop new components and plug into the system.
2. Dynamically find and connect components.
3. Adding more interfaces and/or improving performance of interface, as to allow cyber components to
   1) sense more and better about its environment;
   2) make more intelligent control decisions, and provide smarter services,
   3) control and coordinate more and better physical components.
4. With 1&2 to “connect” what were originally separated components to allows them to interact, collaborate and coordinate.
5. With more and better connectors, coordinators, interfaces to improve trustworthiness.
BMS as CPS

Battery Management System

- Traction inverter
- Vehicle controller
- Battery charger

Interface module

- Controller area network

Battery pack

Capacitor group

- Cell

Controller area network

Sensors

Research content — Framework

The framework of the cloud based BMS

Solution:

- Refer to experience
- Use CPS architecture
- Unified standard management
- Employ data from multidimensional

Architecture:

- Multilevel CPS
- Dynamic evolution architecture
- Adaptive components
- Interface based interconnection
Evolution in Railway Systems

MIT Robot Garden, Automatic Driving Cars, ....

AI in CPS

• Knowledge reasoning, as well as data analytics, is involved in
  – create views,
  – assembling views into new services,
  – decision making in business processes and workflows
  – dynamic service discovery and binding – Evolving SOA.
• Computer vision and Natural language processing in
  – human-machine interaction (HMI), and
  – removing the barrier to software requirements elicitation, analysis, formalisation, and prototyping.
• Robots are obviously in CPS.
Challenges

- Handling complexity *dynamic emergent behaviour*.
- Rigorously model the interactions between the physical sides and the cyber sides.
- Requirements
  - Cope with changes and uncertainties both during development and at runtime
  - The relation between performance and functionalities
  - Multiviews and competing requirements of different stakeholders.
- Handling evolving architecture
  - dynamic discovery and binding services and components, (*evolving SOA*).
  - robust, self-organising, self-adaptive.
- SE+AI

A Proposed Position

- A combined component-based and service-oriented design and evolution
  - System architecture is horizontally component-based and vertically service-oriented
  - Systems architecture Design new cyber components in the context of an existing CPS
  - Upgrade existing cyber components in an existing CPS
  - Develop new layer of services, monitoring and control (SOA)
  - Refine existing layer of coordination, monitoring and control
- Integration testing and verification, such as *emergent properties* [mostly SOA]
- The notion of *Contracts* in Key in this framework
Objectives/Vision

- A model-driven and component-based approach to CPS Design and Evolution

1) Seamlessly and coherently combination of the various dimensions of the multi-scale design space – behavior, QoS, space and time.

2) Definition and management of the entire processes and not to consider only point solutions of methodology, tools, and models that ease part of the design.

- Provide correct, secure and intelligence and healthy evolution by architectural design – a theory of generic refinement
  - in which integration verification and simulation techniques are driven by construction.

Contracts in General

- A contract is a very general notion in many disciplines.
- For CPS design, a contract \( C \) is given by a pair of properties

\[
C = A |\rightarrow G
\]

\( A \): assumptions on the environment, and
\( G \): the promise of the component under these assumptions.

- The model of contracts is general for functions, interactions, and QoS, i.e. multi-concerns.
Contract-Based Model Sports CBD and SOA

- Use as much as possible elements from available components and services
- Component composition $C = A \mid-- G$ of $C_1 = A_1 \mid-- G_1$ and $C_2 = A_2 \mid-- G_2$

$$A = (A_1 \land A_2) \lor \neg (G_1 \land G_2), \quad G = G_1 \land G_2$$

- Contracts (at the level of interfaces) support Layered SOA
- Key challenge in CPS is to develop a model of contracts to
  - model interactions between physical and components
  - mix different physical systems, control logic, and implementation architectures

Support MBD

- The main philosophy of MBD is integration virtualisation, and contract composition is model composition.
- Support correctness preservation model transformations, by refinement

$$C_1 \sqsubseteq C_2 \text{ if } A_1 \leq A_2 \text{ and } G_1 \Rightarrow G_2$$

- Support layered design, correct realisation of contract or a component in a higher layer by assembling components in a lower layer
- Theorem: $(Contracts, \sqsubseteq)$ forms a partial order, the lower and upper bounds are

$$C_1 \land C_2 = A_1 \lor A_2 \mid-- G_1 \land G_2 \quad C_1 \lor C_2 = A_1 \land A_2 \mid-- G_1 \lor G_2$$

- Separation of Multiple viewpoints, e.g. $C_i \sqcap C_i = A_i \lor A_i \mid-- G_i \land G_i$
Well-Known Contract-Based Models

- Imperative Programs
  - Hoare Logic \{\text{Pre}\} \ P \ \{\text{Post}\}
  - UTP \ P \ \{\text{Pre}\} \ |- \ \{\text{Post}\} \ \text{[Hoare&He]}
- OOP: Meyer’s Design by Contracts, rCOS \ [He, Li & Liu]
- Reactive Systems
  - Jones’ theory of Rely-Guarantee
  - Lamport’s TLA: \ E \implies S
- Component-Based Systems: rCOS \ [He, Li & Liu]

---

Model of Architectural Components

- **Service component**
  
  \text{Component} \text{ M1}
  
  \text{Z d;}
  
  \text{provided} \ \text{interface} \text{ M1IF} \ {
  \ W(Z \ v) \ { d:=v } ; \ R(;v) \ { v:=d } ; 
  
  }

- **Reactive component**
  
  \text{Component} \text{ M}
  
  \text{Z d, Bool w = true;}
  
  \text{provided} \ \text{interface} \text{ MIF} \ {
  \ W(Z \ v) \ { w&(d:=v, w:= \text{not w}) } ; 
  \ R(;Z \ v) \ { \text{not w&w}(v:=d ; w:=\text{not w})} ; 
  
  }
Equivalent Models

Component M
Z d;
provided interface MIF {
W(Z v) { d:=v }; R(v) { v:=d };
protocol { (WR)*+(WR)*W } // ** generally traces }

Component M requires M1 //** M is obtained through coordinating M1
Bool w = true;
provided interface MIF {
W(Z v) { w&(M1.W(v); w:=not w) };
R(Z v) { not w&(M1.R(v); w:=not w) };
}

Component M requires M1 //** M is obtained through coordinating M1
provided interface MIF {
W(Z v) { M1.W(v) };
R(Z v) { M1.W(v) };
protocol { (M1.WM1.R)*+(M1.WM1.R)*W }
}

More General Component

component fM {
Z d;
provided interface MIF {
W(Z v) { d:=v };
R(v) { v:=d };
protocol { (WR)*+(WR)*W }
}
actions { //fault modelling corruption of memory
fault {true| d'< > d } }

• Renaming as a built-in connector,
  fMi=fM[fMi.W/W,fMi.R/R], i=1,2,3,

• But can be built by composition

component fMi requires fM
  provided interface MIFi {
fMi.W(Z v) { fM.W(v) };
fMi.R(Z v) { fM.R(v) };
}
Orchestration

component V requires fM1, fM2, fM3 Z v1,v2,v3,
provided interface VIF {
    W(Z v) { fM1.W(v)||fM2.W(v)||fM3.W(v) };
    R(;v) { fM1.R(;v1)||fM2.R(;v2)||fM3.R(;v3);v=vote(v1,v2,v3) };
    protocol { W({W,R}) } /* notice one can specify different protocols */
}

• V ⊑ M. i.e. V refines M, provided at most one memory is corrupted
  Verification
• The proof need to introduce auxiliary variables

Note three different instances of fM are running in parallel in side these fMi’s, this in V

Architecture Decision for Fault-Tolerance
Model-Driven Development with rCOS

- Each phase is based on the construction of **verifiable models**.
- Models are analysed and verified.
- Refined models are constructed by **model transformations**.
- **Code** is generated from design models.
- **Proof obligations** are generated by model transformations.
- **rCOS modeler** integrates UML model notation into rCOS.

---

**Component Interface Sequence Diagram**

- Diagram showing interaction between components.
- Sequence of steps in a process (startSale, enterItem, endEntry, makeCashPayment).
Component Interface Sequence Diagram

Component Class Diagram
Operation Contract makeCashPayment(a)

• **Preconditions**
  1. Store exits
  2. CashDesk exits
  3. Catalog exits
  4. Sale is complete

• **Postconditions**
  5. A new cashpayment is created
  6. Amount of the new cashpayment is set to a
  7. The new cashpayment is linked to the sale

**OO Design: makeCashPayment()**
OO Design to Component-Based Design

Automatically Generate Component Diagram
rCOS Design Process
Summary

1. use case as component
2. refine use case operations by design patterns to generate an oo interaction model
3. generate design class model
4. transform the oo interaction model to a component interaction model
5. generate the component diagram
6. transform oo interfaces to specific middlewares, e.g. RMI, CORBA etc.
7. Integrates implemented use cases

Code generation and testing after 3 or 6

Cyber–Physical Components

- Physical Interface

- Cyber-Physical Component
**Physical Components and Interfaces**

Component A /*an appliance 
rate: [Time \(\rightarrow\) Real];
status: {on, off};
provided interface {
  rate /*signal: given by manufacture;
  switch() /*operation: switch A on and off
}

Component M /*meter
val: [time \(\rightarrow\) Real];
provided interface {
  read();true \(-\) r'=val};
}
required interface rate /*signal
val= energy(rate)
}

Composition: \(H = A||M\)

---

**System Evolution for Home Automation (a)**

- Add provided signal ‘val’ to M
- Add a control pad \(P\) that requires signal ‘val’, provides ‘set()’, and calls A.switch, etc. \(M' = P || M, H' = M' || A\)
- Refine \(P\) with planning with daily budget, and schedule functionality \(M'' = P' || M, H'' = M'' || A\)
System Evolution for Home Automation (b)

\[ Ai = A[\text{switch/switch}, \text{rate/rate}], \quad A = A_1 || ... || A_m \]
\[ Mi = M[\text{read/read, val/val}], \quad M = M_1 || ... || M_m \]
\[ P_i = P[.........], \quad P = P_1 || ... || P_m \]

- Add a global controller for planning and schedule
  \[ H = G || P || M || A \]
- Control C with mobile phone from car or office
Network Evolution

- Consider k households $H_j$, each with its own budget $H = H_1 || H_2 || ... || H_k$
- Consider Coordinator, interacting the households to coordinate their budgets, based on the interaction with of a utility company Utility $N = H || Coordinator || Utility$
- Evolve to a network $U$ of utility companies $N = H || Coordinator || U$
- Implement $H || Coordinator$ by distributing the coordination activities among the households themselves.

Note: individual households evolve in parallel, and in parallel with the network with more households and companies.

Conclusions

1. CPS as unified view of emerging systems
2. BigData, Machines/Deep Learning, AI are all involved
3. A framework based on the notion of contracts
   - there is already high methodological value when using informal contracts.
   - formalisation promises additional benefits in dealing with system complexity, and formalisation can be done incrementally.
4. Contracts are essential for integration of heterogeneous components, and also important the domain linkages in interdisciplinary collaboration.
5. Architecture models are essential for
   - correct and secure by design,
   - Identification of safety vulnerabilities and security threats, and
   - making architecture decisions for different concerns (Aspect-Orientation)
   - But most of the challenges discussed are largely unsolved
Indirections

“All Problems in Computer Science can be Solved by Another Layer of Indirection.

But that usually will create another problem.”

--- David J. Wheeler

Lord Ernest Rutherford (1871-1937),
Father of Nuclear Physics
Nobel Prize Winner in Chemistry (1908)
President of Royal Society (125-1930)

Sir Edward Victor Appleton (1892-1965),
Physicist, FRS,
Nobel Prize Winner in Physics (1947)

Jack. A. Ratcliffe (1902-1987),
Radio Physicist, FRS

Sir Maurice Wilkes (1913-2010),
Computer Scientist,
FRS, FREng, DFCBS,

David J. Wheeler (1927-2004),
Computer Scientist,
FRS

Mathai Joseph, Computer Scientist

Zhiming Liu
Michael Burrows  
(1963--)  
FRS, Google  
Burrows–Wheeler transform  
Search Engine AltaVista

Andrew Hopper  
(1953--)  
FRS,FREng  
Virtual Network Computing

Bjarne Stroustrup  
(1950-)  
C++

Roger Needham  
(1935-2003)  
FRS, FREng  
Needham–Schroeder protocol  
BAN logic

Mathai Joseph  
Real-Time
1. Centre for Research and Innovation in Software Engineering (RISE),
2. Faculty of Computer and Information Science (CIS),
   Southwest University, Chongqing, China
Email: liubocq@swu.edu.cn; bob.liubo@gmail.com;
URL: http://www.swu-rise.net.cn/bo.liu

Chongqing, China, 17/10/2017

Outline

0 Introduction to QoS-aware SC
1 Q-SC under multi-task
2 Q-SC with severe QoS
3 Q-SC with trust
Why need service composition (SC)
• A single service (atomic service) is useful to provide a specific functionality. It may not necessarily fulfill the requirements of complex applications. Hence several services are combined together to create multi-functionality composite services.

What & Why QoS-aware SC (Q-SC)
• It is a service-oriented technology (SOT), which combines atomic services into a loosely coupled application (composite service), where the QoS properties (e.g., time, cost, reliability) of each service are adopted to guide the process of SC.
• As many services provide identical functionalities but varying in non-functional features (QoS), Q-SC gains more attention.

Traditional Framework of Q-SC
Step 1: Task decomposition
Output: A series of subtasks

Step 2: Service discovery
Output: Sets of candidate services

Step 3: Service composition
Output: Optimal composite service

Step 4: Service execution
Output: User acceptable results
Application of Q-SC into Cloud manufacturing
(A Case: Online Motorcycle Production)

Task Decomposition
Service Discovery
Service Discovery
Service Discovery

Drawbacks in traditional framework

Single-task
1-1 mapping
Trust is required

Multi-task
Large-scale SOA System
(Cloud manufacturing)
Motivation scenario: multi-task req.

- Given two OMP tasks $T_1$ and $T_2$, the QoS requirements are: $QoS\text{-time}1<523$, $QoS\text{-time}2<498$;
- Suppose the sets of candidate services for each subtask are given by the following figure.

The reason lies in the strategy of single task oriented composition:
The first composed service for $T_2$ used up all the best component services, without considering the inter-task constraints between $T_1$ and $T_2$. Thus, the second composition for $T_2$ may be lack of good enough component services.
Multi-task oriented composition

Idea of multi-task oriented service composition:
Conduct a holistic service composition of multiple composite services for both $T_1$ and $T_2$ simultaneously.

Although each composite service for $T_1$ or $T_2$ may not be optimum, both the requirements on $T_1$ and $T_2$ can be met.

Imbprovement to the framework

1. Task decomposition (multi-task):
   $$T = \{ ST_1^i, ST_2^i, ..., ST_j^i, ..., ST_l^i \}$$

2. Service discovery:
   $$CSS = \{ CSS_1^i, CSS_2^i, ..., CSS_j^i, ..., CSS_k^i \}$$
   $$CSS' = \{ CS_1^i, CS_2^i, ..., CS_3^i, ..., CS_l^i \}$$

3. Service composition (holistic com.):
   $$CSS = \{ CSS_1^{1i}, CSS_2^{1i}, ..., CSS_j^{1i}, ..., CSS_k^{1i} \}$$
   $$\{ CS_1^{11}, CS_2^{12}, ..., CS_3^{1l}, ..., CS_l^{1k} \}$$

Find out the optimal holistic solution for all Tasks $T = \{ T_i | (i=1...l) \}$
Formulate a more general problem model:
- It responses to any number of requesting tasks and provides the best-effort composite services to meet the requirements of all tasks.
- Specially, when the number of requesting task $I = 1$, the multi-task oriented problem is specialized into a single task oriented problem.

$$\max QoS(MCCS) \quad \rightarrow \quad \max \sum_{i=1}^{I} QoS(MCCS_i)$$
Simulation experiments

Multi-task requesting in Online Motorcycle Production

Outline
0 Introduction to QoS-aware SC
1 Q-SC under multi-task
2 Q-SC with severe QoS
3 Q-SC with trust & user preferences

Sound performance-wise
Motivation scenario: severe QoS

- Severe QoS: the given QoS constraints outnumber any plainly composed composite services.
- Given two OMP tasks $T_i$ and $T_j$, the QoS requirements are: $QoS_{time1} < 523$, $QoS_{time2} < 350$.
- Suppose the sets of candidate services for each subtask are given by the following figure.

![Motivation diagram](image)

The reason lies in one-to-one mapping based service composition (OOM-SC):

Only one appropriate candidate atomic service is allowed to perform each subtask, while other functionally equivalent atomic services available to the same subtask are kept on the shelf.

Synergistic elementary service groups based service composition

The main idea of **SESG-SC**.

Release the assumption of one-to-one mapping between atomic services and subtasks, allowing free combination of multiple functionally equivalent atomic services into a synergistic elementary service group (SESG) to perform each subtask collectively.
Improvement to the framework

Selective structure

Parallel structure

Hybrid structure

Structure of SESGs

General Structure (Hybrid Structure) of a SESG

We use SESGs instead of atomic services as the component services for respective subtasks in service composition
QoS evaluation of SESGs

Formulation of SESG-SC

Formulate a more general problem model than before:

- **At least one** atomic service should be *combined into* the corresponding **SESG** for each subtask.

- If all the **numbers** of atomic services in each SESG are **restricted to 1**, the SESG-SC model will be specialized into the **OOM-SC model**.
Algorithm to solve the problem

Matrix Coded Genetic Algorithm with Collaboratively Evolutonal Populations (MCGA-CEP)

Simulation experiments

Severe QoS constraints in Online Motorcycle Production

Still, it requires further improvement on performance
Overview

Introduction to QoS-aware SC

Q-SC under multi-task

Q-SC with severe QoS

Q-SC with trust

Introduction: Trust-based Q-SC

Why need Trust-based Q-SC

1. User independent
   - price, popularity, availability, etc
     - Offered by service providers (SP) or third-party registries (e.g., UDDI).

2. User dependent
   - satisfaction degree, etc
     - QoS feedback from client-side is required.

Existed problem: Trustable QoS are required

- Current Q-SC is mainly based on the QoS properties advertised by SP.
- Malicious SPs who offer untrustworthy QoS properties may do harm to the ecological environment of service; services from honest SPs are uncompetitive.
Introduction: Trust-based Q-SC

Aim: Predict the absent QoS values
- A large number of users need to evaluate an even larger number of candidate services, before making a trusted service composition.
- Without user 1’s QoS values on S4, the S4 may not be the candidate to any service composition for User 1 forever. That’s unreasonable.

<table>
<thead>
<tr>
<th></th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
</tr>
</thead>
<tbody>
<tr>
<td>User 1</td>
<td>5</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User 2</td>
<td>4.5</td>
<td></td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>User 3</td>
<td></td>
<td>3.5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User 4</td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User 5</td>
<td>3.5</td>
<td></td>
<td></td>
<td>4.8</td>
<td></td>
</tr>
</tbody>
</table>

The blank indicates User 1 has not ever used S4 before, and no QoS values in it. **Need to be predicted**

Introduce RS to the framework of Q-CS

Recommender system
- User-based CF
- Service-based CF
- Matrix factoring

Predict Client-side QoS

Trusted services recommendation
User-based Collaborative Filtering (CF)

**Idea:** If a user has little knowledge about a service, he can inquire or refer to other users (es. his friends) who has similar interests as him.

\[ sim(u, v) = \frac{\sum_{i \in I} (q_{u,i} - \bar{q}_u)(q_{v,i} - \bar{q}_v)}{\sqrt{\sum_{i \in I} (q_{u,i} - \bar{q}_u)^2} \sqrt{\sum_{i \in I} (q_{v,i} - \bar{q}_v)^2}} \]

**Step 1:** find out similar users

For each pair of users u and v, the Pearson Correlation Coefficient (PCC) can be used to evaluate the similarity of them.

**Step 2:** predict absent QoS

\[ q_{u,x} = \sum_{v \in \text{sim}_u} sim(u, v)q_{v,x} \]

<table>
<thead>
<tr>
<th></th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
</tr>
</thead>
<tbody>
<tr>
<td>User 1</td>
<td>5</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User 2</td>
<td></td>
<td></td>
<td>4.5</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>User 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User 4</td>
<td>2</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User 5</td>
<td>3.5</td>
<td></td>
<td></td>
<td></td>
<td>4.8</td>
</tr>
</tbody>
</table>

Service-based Collaborative Filtering

**Idea:** We can study the historical scores of a user on other similar items, to predict what score he will give to the absent QoS of the item.

**Step 1:** find out similar items

For each pair of service i and j, PCC can be used to evaluate the similarity of them.

\[ sim(i, j) = \frac{\sum_{v \in U} (q_{v,i} - \bar{q}_i)(q_{v,j} - \bar{q}_j)}{\sqrt{\sum_{v \in U} (q_{v,i} - \bar{q}_i)^2} \sqrt{\sum_{v \in U} (q_{v,j} - \bar{q}_j)^2}} \]

**Step 2:** predict absent QoS

\[ q_{u,i} = \sum_{j \in \text{sim}_i} sim(i, j)q_{u,j} \]

<table>
<thead>
<tr>
<th></th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
</tr>
</thead>
<tbody>
<tr>
<td>User 1</td>
<td>5</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User 2</td>
<td></td>
<td></td>
<td>4.5</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>User 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User 4</td>
<td>2</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User 5</td>
<td>3.5</td>
<td></td>
<td></td>
<td></td>
<td>4.8</td>
</tr>
</tbody>
</table>
Matrix Factoring

<table>
<thead>
<tr>
<th></th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
</tr>
</thead>
<tbody>
<tr>
<td>User 1</td>
<td>5</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User 2</td>
<td></td>
<td>4.5</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>User 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User 4</td>
<td>2</td>
<td></td>
<td></td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>User 5</td>
<td>3.5</td>
<td></td>
<td></td>
<td>4.8</td>
<td></td>
</tr>
</tbody>
</table>

Denoted by matrix $Q_{n \times m}$ where $n$ denotes the number of users while $m$ denotes that of services.

- A user-service QoS matrix $Q_{n \times m}$ can be approximately divided into two matrix $U_{n \times k}$ and $S_{k \times m}$ with $k$-dimension ($k < \min(m, n)$), such that: $Q_{n \times m} \approx (U_{n \times k})^T S_{k \times m}$.
- $U$ is the latent feature vector of $n$ users, indicating the extent of interest that users have in the items of high on some features, while $S$ is the latent feature vector of $m$ services, indicating the extent to which possesses those features.
- Idea: only a few key features affecting user-service interactions. And a use is impacted by how the feature is applied to him.

We minimize the term to gain the fine division of matrix $U_{n \times k}$ and $S_{k \times m}$:

$$\min L = \sum_{x \in [1,n], i \in [1,m]} \frac{1}{2} (q_{x,i} - U_x^T S_i)^2$$

Regularization terms are usually added to the end of the term to avoid over fitting.

Initial value of $U_{n \times k}$ and $S_{k \times m}$ are randomly generated, followed by updating $U, S$ by employing gradient descent technique.

$$P_u = P_u - \gamma_1 \frac{\partial L}{\partial P_u}$$

$$S_i = S_i - \gamma_2 \frac{\partial L}{\partial S_i}$$

$\gamma_1, \gamma_2$ denotes the learning rates given as parameters.
Challenge 1: **Sparsity**
- A large number of users need to evaluate an even larger number of candidate services, before making a trusted service composition.
- If constructing a user-service QoS matrix, only a few services are evaluated by a few users. It is a sparse matrix with huge number of absent QoS values.

Challenge 2: **Cold start**
- If a newcomer requests service composition, he would be in trouble because he has not any historical QoS data.
- Similarly, a new created service cannot be a candidate of any composite service for the same reason.

Challenge 3: **Trustworthiness of Client-side QoS**
- Lack the capability of differentiating user’s creditability. There may exist spam users who always give fake feedback of QoS properties.

Challenge 4: **Tensor factoring of high dimension**
- When time-, and position-dimension are included into the matrix factoring (Client-side QoS may vary with time and position in real cases).
- Considering multi-QoS properties in the element of user-service QoS matrix.

<table>
<thead>
<tr>
<th></th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
</tr>
</thead>
<tbody>
<tr>
<td>User 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User 5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

It could be a vector of QoS properties
Challenges in Trust-based Q-SC

Challenge 5: **Algorithm of sound performance**
- As the user-service QoS matrix becomes increasingly high-dimensional, it must be a time consuming work to find out a fine result of tensor factoring.

Challenge 6: **Absence of open datasets with sufficient scale and information**
- Many datasets have very limited scale of data
- Lack of combined information with time, position, multiple QoS properties
- Most datasets are not open source

It is a long way to put in real use of Trust-based Q-SC, even the Q-SC. More challenges will raise as the further development and application of them.

**Q&A**
Lecture 8

Coloured Petri Nets: Modelling and Validation of Concurrent Systems

by Rui Wang
Coloured Petri Nets
Modelling and Validation of Concurrent Systems

Overview and introduction

Rui Wang
Email: rwa@hvl.no

Lars M. Kristensen
Email: lmkr@hib.no / WWW: home.hib.no/ansatte/lmkr

Department of Computing, Mathematics, and Physics, Western Norway University of Applied Sciences, NORWAY
The BeChong Summer School, Oct. 2017

Concurrent Systems

• The vast majority of IT systems today can be characterised as concurrent systems:
  • Structured as a collection of concurrently executing software components and applications (parallelism).
  • Operation relies on communication, synchronisation, and resource sharing.

Internet protocols, cloud, and web-based applications
Multi-core platforms and multi-threaded software
Embedded systems and networked control systems
Concurrent Systems

▪ Most software development projects are concerned with concurrent software systems.
▪ The engineering of concurrent systems is challenging due to their complex behaviour:
  ▪ Concurrently executing and independently scheduled software components.
  ▪ Non-deterministic and asynchronous behaviour (e.g., timeouts, message loss, external events, …).
  ▪ Almost impossible for software developers to have a complete understanding of the system behaviour.
  ▪ Reproducing errors is often difficult.
▪ Methods to support the engineering of reliable concurrent systems are important.

Modelling

▪ One way to approach the challenges posed by concurrent systems is to build a model.
▪ A model is an abstract representation which can be manipulated by a computer software tool:
  ▪ Using a model it becomes possible to investigate properties of the system prior to implementation.
Modelling ...

- Used in many other engineering disciplines:
  - When engineers construct a bridge.
  - When architects design a building.

- Modelling is typically done in the early phases of system development.

Coloured Petri Nets (CPNs)

- General-purpose graphical modelling language for the engineering of concurrent systems.
- Combines Petri Nets and a programming language:

Petri Nets: [C.A. Petri'62]
- graphical notation concurrence synchronisation resource sharing

CPN ML (Standard ML):
- data and data manipulation compact modelling parameterisable models
Quick Recap: Petri Net Concepts

State modelling:
- **Places** (ellipses) that may hold tokens
- **Marking (state)**: distribution of tokens on the places.
- **Initial marking**: initial state.

Event (action) modelling:
- **Transitions** (rectangles)
- **Directed arcs**: connecting places and transitions.
- **Arc weights**: specifying tokens to be added/removed.

Execution (token game):
- **Current marking**
- **Transition enabling**
- **Transition occurrence**

Application Areas

- Communication protocols and data networks.
- Distributed algorithms and software systems.
- Embedded systems and control software.
- Business processes and workflow modelling.
- Manufacturing systems.

... [ http://cs.au.dk/cpnets/industrial-use/ ]
CPN Tools [ www.cpntools.org ]

- Practical use of CPNs is supported by CPN Tools:
  - Editing and syntax check.
  - Interactive- and automatic simulation.
  - Application domain visualisation.
  - Verification based on state space exploration.
  - Simulation-based performance analysis.

Examples of CPN Tools users

**North America**
- Boeing
- Hewlett-Packard
- Samsung Information Systems
- National Semiconductor Corp.
- Fujitsu Computer Products
- Honeywell Inc.
- MITRE Corp.
- Scalable Server Division
- E.I. DuPont de Nemours Inc.
- Federal Reserve System
- Bell Canada
- Nortel Technologies, Canada

**Asia**
- Mitsubishi Electric Corp., Japan
- Toshiba Corp., Japan
- SHARP Corp., Japan
- Nippon Steel Corp., Japan
- Hongkong Telecom Interactive Multimedia System

**Europe**
- Alcatel Austria
- Siemens Austria
- Bang & Olufsen, Denmark
- Nokia, Finland
- Alcatel Business Systems, France
- Peugeot-Citroën, France
- Dornier Satellitensysteme, Germany
- SAP AG, Germany
- Volkswagen AG, Germany
- Alcatel Telecom, Netherlands
- Rank Xerox, Netherlands
- Sydkraft Konsult, Sweden
- Central Bank of Russia
- Siemens Switzerland
- Goldman Sachs, UK
Coloured Petri Nets
Modelling and Validation of Concurrent Systems

Basic syntactical constructs and semantical concepts of the CPN modelling language

Kurt Jensen &
Lars Michael Kristensen
{kjensen,lmkristensen}
@cs.au.dk

Simple protocol
Informal description

No loss of packets

Coloured Petri Net

AARHUS UNIVERSITY
Coloured Petri Nets
Department of Computer Science
Kurt Jensen
Lars M. Kristensen
**Places represent the state of the system**

- Each place contains a number of tokens.
- Each token carries a colour (data value).
- The colour set specifies the set of allowed token colours.

**Initial marking (multi-set of tokens)**

Each token in the initial marking must have a colour (value) that belongs to the colour set.

**Definition of colour sets:**

- colset NO = int; (* seq. numbers *)
- colset DATA = string; (* payload *)
- colset NOxDATA = product NO * DATA;

**Current marking during simulation**

The thick green border line indicates that the transition is enabled (ready to occur).

The current marking changes during simulation (execution of the model).

One token with value 1

Circle: 6 tokens

Square: Detailed token values
Transitions and arcs

The type of the arc expression must be equal to the colour set of the attached place (or a multi-set over the colour set)

Declaration of variables:
\[ \text{var } n : \text{NO}; \quad \text{(* integers *)}\]
\[ \text{var } d : \text{DATA}; \quad \text{(* strings *)}\]

Binding of variables:
\[ <n=1,d=\text{"COL"}>\]

Evaluation of expressions:
\[ (n,d) \rightarrow (1,\text{"COL"}) : \text{NOxDATA} \]
\[ n \rightarrow 1 : \text{NO} \]

Enabling of transitions

Transition is enabled if we can find a binding so that each input arc expression evaluates to a multi-set of tokens that is present on the corresponding input place

Two variables:
\[ \text{var } n : \text{NO}; \quad \text{(* integers *)}\]
\[ \text{var } d : \text{DATA}; \quad \text{(* strings *)}\]

Binding:
\[ <n=\?, d=\?>\]
We want to find a binding for the variable \( n \) such that the arc expression \((n,d)\) evaluates to a token which is present on the place \( \text{PacketsToSend} \).

**Enabling of SendPacket**

- **Arc expression**: \((n,d)\)
- **Binding**: \( < n=1, d=? > \)

We want to find a binding for the variable \( d \) such that the arc expression \((n,d)\) evaluates to a token which is present on the place \( \text{PacketsToSend} \).

**Enabling of SendPacket**

- **Six different tokens**
- **Arc expression**: \((n,d)\)
- **Binding**: \( < n=1, d="\text{COL}" > \)
Enabling of SendPacket

We have found a binding so that each input arc expression evaluates to tokens that are present on the corresponding input place.

Binding: \(< n=1, d="COL" >\)

Transition is enabled (ready to occur)

Occurrence of SendPacket in binding \(< n=1, d="COL" >\)

Remove: \(1,"COL"\)

Add a new token: \(1,"COL"\)

Remove: 1
New marking after occurrence of SendPacket in binding <n=1,d="COL">:

- The first packet has been removed.
- A copy of the first packet has been put on A.
- Transition is no longer enabled (thin border line).

New marking M₁:

- Transition enabled.
- Transition enabled.
- Transition enabled.

No token on this place.
**Binding of TransmitPacket**

Current marking

```
1'(1,"COL")
```

Arc expression

```
Binding: < n=1 , d="COL" >
```

---

**Occurrence of TransmitPacket in binding <n=1,d="COL">**

Remove: (1,"COL")

Add a new token: (1,"COL")
New marking $M_2$

Simulation Demo in CPN Tools

Interactive and automatic simulation
### Second version of the protocol

```scala
def constant

\[
\text{val AllPackets} = 1'(1,"COL") ++ 1'(2,"OUR") ++ 1'(3,"ED ") ++
\text{1'}(4,"PET") ++ 1'(5,"RI ") ++ 1'(6,"NET");
\]

\[
\text{val AllPackets} = 1'(1,"COL") ++ 1'(2,"OUR") ++ 1'(3,"ED ") ++
\text{1'}(4,"PET") ++ 1'(5,"RI ") ++ 1'(6,"NET");
\]
```

#### Declaration of constants

- We use the following constant to specify the initial marking of PacketsToSend.

```scala
\[
\text{val AllPackets} = 1'(1,"COL") ++ 1'(2,"OUR") ++
\text{1'}(3,"ED ") ++ 1'(4,"PET") ++
\text{1'}(5,"RI ") ++ 1'(6,"NET");
\]
```

- Saves a little bit of space in the diagram.
- Enhances readability.
- Can be reused (at other places).
Double-headed arcs

A double-headed arc is a shorthand for two oppositely directed arcs with the same arc expression.

Double-headed arc

We no longer remove the tokens from the input places.

Double-headed arc

Retransmission becomes possible.

More complicated arc expression

More complicated arc expression (if-then-else expression)

More complicated arc expression

(if-then-else expression)
**If-then-else expression**

If-then-else expression

If-then-else expression

New variable:

```
var success : BOOL;
```

**Successful transmission over the network**

```
b^+ = <n=1, d="COL", success=true>
b^-= <n=1, d="COL", success=false>
```

**Packet is lost during transmission**

```
b^+ = <n=1, d="COL", success=true>
b^- = <n=1, d="COL", success=false>
```

New variable:

```
var success : BOOL;
```

No packet is added

empty

Packet is lost during transmission

if success then 1'(n,d) else empty

success = true

success = false

No packet is added

empty

Packet is lost during transmission

if success then 1'(n,d) else empty

success = true

success = false
**New name and new type**

Initial marking: empty text string

- AllPackets
- NOxDATA

Aflows:

- Packets To Send
- (n,d)
- NOxDATA
- n
- 1'1
- k
- NO
- x

Bflows:

- Transmit Packet
- (n,d)
- NOxDATA
- if success then 1'(n,d) else empty
- k
- NO
- x

Cflows:

- Transmit Ack
- n
- NO

Dflows:

- Receive Ack
- n
- NO
- if success then 1'n else empty

New name

- Data Received
- DATA
- 1'

New type

- Data Received
- DATA
- 1'
- NO
- x

**New place: NextRec**

Plays a similar role as NextSend

Contains the number of the expected packet

- Packets To Send
- (n,d)
- NOxDATA
- n
- 1'1
- k
- NO

Aflows:

- Send Packet
- (n,d)
- NOxDATA
- n
- 1'1
- k

Bflows:

- Transmit Packet
- (n,d)
- NOxDATA
- if success then 1'(n,d) else empty
- k
- NO

Cflows:

- Transmit Ack
- n
- NO

Dflows:

- Receive Ack
- n
- NO
- if success then 1'n else empty

New place

- Data Received
- DATA
- 1'

AARHUS UNIVERSITY
Coloured Petri Nets
Department of Computer Science
Kurt Jensen
Lars M. Kristensen
Correct packet arrives

Packet no 1 arriving

Packet no 1 expected

Wrong packet arrives

Packet no 1 arriving

Packet no 3 expected
Acknowledgements can be lost

NextSend is updated

Also possible to lose acknowledgements

NextSend is updated with sequence number from acknowledgement
Simulation Demo in CPN Tools

Two enabled transitions

\[ SP = (\text{SendPacket}, \ <n=1, \ d="COL">) \]
\[ TP^+ = (\text{TransmitPacket}, \ <n=1, \ d="COL", \ success=true>) \]
\[ TP^- = (\text{TransmitPacket}, \ <n=1, \ d="COL", \ success=false>) \]

- These binding elements need the same token
- They are in conflict with each other
Two enabled transitions

- These bindings elements use different tokens
- They are concurrently enabled and can occur concurrently

SP = (SendPacket, <n=1, d="COL">)
TP+ = (TransmitPacket, <n=1, d="COL", success=true>)
TP- = (TransmitPacket, <n=1, d="COL", success=false>)

Three enabled transitions

- All other binding elements are concurrently enabled
- These binding elements are in conflict

SP = (SendPacket, <n=1, d="COL">)
TP+ = (TransmitPacket, <n=1, d="COL", success=true>)
TP- = (TransmitPacket, <n=1, d="COL", success=false>)
RP = (ReceivePacket, <n=1, d="COL", k=1, data=""
Editing Demo in CPN Tools
Lecture 9

Model Transformations & QVT

by Dan Li
Model Transformations & QVT

Li Dan, Volker Stolz, Liu Zhiming

BeChong, Chongqing, Oct. 2017

Agenda

- Modeling and Model Transformations
- QVT and Support Tools
- QVTR-XSLT Approach
- Case Study: UML to RDBMS Transformation
- Case Study: Support rCOS development with refinement transformations.
- Conclusions
Agenda

- Modeling and Model Transformations
- QVT and Support Tools
- QVTR-XSLT Approach
- Case Study: UML to RDBMS Transformation
- Case Study: Support rCOS development with refinement transformations.
- Conclusions

Models and Modeling Languages

- A model describes a system at a higher level of abstraction.
- A model needs to be written in a precise, concise and non-ambiguous language---modeling language.
Define Modeling Languages

- **Concrete syntax**: how the language elements appear in a concrete, human-usable form.
  - textual languages defined by, e.g. alphabet, keywords, sentences
  - graphical languages defined by, e.g. symbols, lines, layouts

- **Abstract syntax**: specifies the concepts of the language and their relationships to each other.
  - grammars, e.g. BNF
  - Metamodelling: define a well-defined language using metamodelling.

Define Modeling Languages (2)

- **Semantics**: assign meaning to the modeling language
  - **static semantics**: well-formedness rules over the abstract syntax
  - **dynamic semantics**: either informally or mathematically
    - denotational semantics: mapping syntactic concepts to a semantic domain.
    - operational semantics: interpreting program as a sequences of computational (execution) steps.

- **Persistent formats**: .txt files, json, XML/XMI, database, model repositories
Concrete Syntax Styles

- **Graphical notation**: visually attractive, easier and faster to understand, two or three dimensions.
  - A picture is worth a thousand words
- **Textual notation**: more expressiveness, more compact and has less scalability problems, but only one dimension.
- **Better choice**: take the advantage of both notations, and keep a balance between them:
  - graphically specify structural concepts and relations;
  - textually specify constraints, calculation, value assignment, etc.

Graphical / Textual Notation Examples

class Table{
    attribute name : String
    attribute cols : Column[1..*]
    reference pkey : Column[1..*]
    attribute fkeys : FKey[0..*]
}
class FKey{
    reference references : Table
    reference cols : Column[1..*]
}
class Column{
    attribute name : String
    attribute type : EString
}
class RDBMSModel{
    attribute table : Table[1..*]
}
Model Transformations

- A model is an abstraction of a reality which captures only certain aspects of features, and is written in a well-defined language.
- Modeling is an efficient way to simply (abstract) and conquer the complexities of the real world. There are models everywhere, for researching, designing, manufacturing…
- **Model Transformation** defines as an automated process that takes one or more source models as input and produces one or more target models as output, following a set of transformation rules.
- Model transformations are the **KEY** to precisely analysis and manipulate models.

Model Transformations (2)
Applications of Model Transformations

- Model driven development
  - model refinement, model refactoring
- Model properties checking
- Model execution.
- Model abstraction
- Code generation

Challenges of Transformations

- Models are among the most complex structures in the world.
- Transformations concern different types of models:
  - Semantic models
  - Concrete syntax (graphical) models
  - Textual specifications, and.
  - Their synchronization.
- Handel different aspects of models:
  - Sub-models, e.g. Use Case, Class, Sequence, State Machine, Component Diagrams
  - Different abstract levels, and
  - Their consistency.
Model Transformation Classification

- **Declarative vs. Imperative (Operational)**
  - what to do: Invariant relations (mapping) between source and target
  - how to do it: How to explicitly derive a target from a source.

- **Visitor-based vs. Pattern-based**
  - visitors to traverse the source models and write targets
  - pattern matching and produce targets

- **General purpose languages vs. transformation languages**
  - no overhead to learn, tool supports
  - higher expression, simplify to develop transformations.

MDA: Standardization of Modeling

- Model driven architecture is a framework promoted by Object Management Group (OMG) for MDD/CBD.

- **MDA** includes a set of standards:
  - MOF – metamodel definition 
  - UML – model definition 
  - OCL – model constrain and query 
  - XMI – model interchange 
  - QVT – model transformation
OMG Metamodel Architecture

- **M₀**: Instance of model
- **M₁**: Model
- **M₂**: Metamodel
- **M₃**: Metameta model

Meta-Object Facility (MOF)

- The UML metamodel and other MM's
- UML models and other M's (CoCoMe)
- Various usages of these models

MOF, EMOF, XMI

- **MOF** (Meta Object Facility) - a generic framework for describing and representing and managing metamodels.
- **EMOF** (Essential MOF) - a subset of MOF for defining simple metamodels using simple concepts.
- **XMI** *(XML Metadata Interchange)* - map the graph-based models to/from the tree-based XML documents, enable easy interchange of models between UML-based modeling tools.

---

Examples: Eclipse Modeling Framework (EMF), Ecore
UML: Unified Modeling Language

- A widely accepted standard graphical modeling language for system analysis and design, with many available tools.
- Many types of diagrams for specifying a system from different views,
  - *structure diagrams*: Class Diagrams, Object Diagrams, Component Diagrams
  - *behavior diagrams*: Use Case diagrams, Sequence Diagrams, State Machine Diagrams

OCL: Object Constraint Language

- OCL is a declarative, no side effects, strongly typed language.
- OCL is easy to read and write.
- OCL provides a way to develop more precise models using UML.
  - To specify *invariant* on a class that must always hold, Also applicable to types and stereotypes
  - To specify *pre-* and *post* conditions on Operations and Methods.
  - To specify *guards*
  - To specify *queries* on the UML model.
  - To specify calculation for the *values* of model elements
Agenda

- Modeling and Model Transformations
- QVT and Support Tools
- QVTR-XSLT Approach
- Case Study: UML to RDBMS Transformation
- Case Study: Support rCOS development with refinement transformations.
- Conclusions

MOF 2.0 Queries/Views/Transformations

- Define a language for querying MOF models
- Define a language for transformation definitions
- Allow for the creation of views of a model
- Ensure that the transformation language is declarative and expresses complete transformations
- Ensure that incremental changes to source models can be immediately propagated to the target models
- Express all new languages as MOF models

(Taken from the QVT specification)
QVT Relations (QVT-R)

- A high level, user friendly declarative model transformation language with both textual and graphical notations.
- A transformation is specified as a set of relations between model elements of source and target models.
- A relation specifies how two types of object diagrams, called domain patterns, relate to each other.
- The use of essential OCL greatly enhances it, and used for querying models.
QVT-R in Graphical Notation

- Provides a concise, intuitive and powerful way to specify transformations.

- Graphical specification is a higher-level view that is easier to understand and communicate than the lexical counterpart.

- UML people might expect to continue the graphical tradition of class diagrams and favor a graphical notation.

- A picture is worth a thousand words

A Relation in QVT-R Graphical Syntax

```
when PackageToSchema(p, s);
where prefix = cn; AttributeToColumn(c, t, prefix);
```
The Relation in Textual Notation

top relation ClassToTable {
   cn, prefix: String;
   checkonly domain uml c:Class {namespace=p:Package {},
      kind='Persistent', name=cn};
   enforce domain rdbms t:Table {schema=s:Schema {}, name=cn,
      column=cl:Column {name=cn+'_tid', type='NUMBER'},
      key=k:Key {name=cn+'_pk', column=cl, }};
   when {
      PackageToSchema(p, s);
   }
   where {
      prefix = cn;
      AttributeToColumn(c, t, prefix);
   }
}

Advantages of QVT-R

- It is a standard language of OMG.
- Declarative style is generally more concise since there is no need to explicitly specify the execution steps, and more easy to maintain traceability.
- Graphical notation is more intuition to define transformation.
- The use of OCL greatly enhances its expressiveness.
Problems of QVT Specification

- Some transformation scenarios, such as in-place transformations, model compositions, bidirectional transformations, are not well specified;
- Do not detailed define model query;
- Correctness and verification of transformations are not addressed;
- Many missing, conflicts and errors left in the definition of graphical notation.
- Do not provide any reference implementation.

Transformation Tools

- Graph Transformation
  - **AGG**: an algebraic approach, with graphical editor and an interpreter, support verification and reasoning.
- QVT Operational
  - Borland Together Architect (M2M Eclipse project)
  - SmartQVT: Eclipse OS implementation
- QVT Relations
  - **ModelMorf**: a command line application
  - **MediniQVT**: Eclipse based, widely used QVT-R tool.
- QVT-like
  - **ATL**: also Eclipse based, a mature and widely used transformation tool.
  - **UMLX**: using graphical syntax
Medini QVT -- IKV++ Technologies

- Eclipse integrated QVT-R textual editor, with syntax highlighting and code completion;
- Java based QVT-R interpreter with debugging facilities;
- Support trace models and bidirectional transformations
- Metamodels and input, output models are XML files of Ecore or UML format;
- Some language features are missing, such as checking and enforce semantics, collection templates;
- Stage: seems the more stable and mature implementation of QVT-Relations

```xml
<transformation
    title="Package to a schema"
    source="UML1.4toSimpleEMLS"
    sourceType="SourceFile">
  <mapEachPackage to="SimpleEMLS:SimpleEMLS">
    <checklnly domain="UML1.4toSimpleEMLS:SimpleEMLS">
      <universe "pn"/>
    </checklnly>
    <enforce domain="SimpleEMLS:SimpleEMLS">
      "rcName" = "pn";
    </enforce>
  </mapEachPackage>
</transformation>
```
ModelMorf – Tata Consultancy Service

- Fully compliant with QVT-Relations language
  - bi-directional, create traces, target-incremental, in-place, composition

- Provides a Java-based engine with a command line user interface, and consumes and produces models in XMI format.

- Plan to provide a commercial tool for Relations that implements both textual and graphical syntax

ATL – QVT like transformation language and tool

- An Eclipse based IDE with dedicated textual editors, debuggers, code completion, syntax highlighting, meta model registry
- Hybrid style - imperative bodies wrapped into declarative shells;
- Based on the OCL specification;
- Support unidirectional transformations;
- A component in the M2M Eclipse project;

- Very stable and mature and constantly improved
ModelMorf – Tata Consultancy Service

- Fully compliant with QVT-Relations language
  - bi-directional, create traces, target-incremental, in-place, composition

- Provides a Java-based engine with a command line user interface, and consumes and produces models in XMI format.

- Plan to provide a commercial tool for Relations that implements both textual and graphical syntax
ATL – QVT like transformation language and tool

- An Eclipse based IDE with dedicated textual editors, debuggers, code completion, syntax highlighting, meta model registry
- Hybrid style - imperative bodies wrapped into declarative shells;
- Based on the OCL specification;
- Support unidirectional transformations;
- A component in the M2M Eclipse project;
- Very stable and mature and constantly improved
Comparison of Transformation Tools

<table>
<thead>
<tr>
<th>Capability for complex transformation</th>
<th>AGG</th>
<th>ATL</th>
<th>UMLX</th>
<th>medini QVT</th>
<th>ModelMorf</th>
</tr>
</thead>
<tbody>
<tr>
<td>Support QVT Standard</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>In-place transformation support</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Declarative style</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Graphical notation</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Able to Integrate in other CASE tools</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Able to work with different programming languages and work in different platforms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Work for QVT-R Graphical Syntax

- **QViT (2017)**: An interactive modeling editor for QVT-Relations (First Prototype). Output to medini QVT
- **Declarative QVT (2010)**: Eclipse M2M project. Based on UMLX, it plan to offer a graphical editor. Not available.
- **Visual QVT/R (2009)**: Max Bureck of the Freie University Berlin proposed to build a graphical editor for QVT-R.
- **Enterprise Architect (2007)**: Based on EA, Oliver Alt of BOSCH designed a UML profile, and enabled EA for QVT editing. A case study for proof of concept
- **VMTS (2007)**: provided a QVT plug-in for VMTS, and defined QVT-R editor. A case study of VMTS.
Agenda

- Modeling and Model Transformations
- QVT and Support Tools
- QVTR-XSLT Approach
- Case Study: UML to RDBMS Transformation
- Case Study: Support rCOS development with refinement transformations.
- Conclusions

Motivation for the QVTR-XSLT Tool

- Graphical notation of QVT Relations (QVTR) provides a concise, intuitive way to specify model transformations.
  - But there is NO practical support tool.
- Most of the models are stored in XML/XMI formats.
- XSLT is a powerful and widely used language for manipulating XML with many industrial-strength processors.
  - But programming in XSLT is difficulty during to its low level syntax
- QVTR-XSLT: a practical model transformation framework that combines the power of graphical notation of QVT-R and XSLT
An approach and tool for the graphical notation of QVT-R

- An approach to map QVT-R transformations into executable XSLT programs, a W3C standard language;
- A method to map OCL into XPath, a sub-language of XSLT;
- An extension to QVT-R with three new constructs;
- A graphical model query facility;
- A UML profile for modeling QVT-R in UML;
- A support tool named QVTR-XSLT that consists of a QVT-R graphical editor and an XSLT code generator, and
- A set of predefined transformations for improving the interoperability of the tool, and for transformation verification and validation.

XSLT

- Extensible Stylesheet Language for Transformations (XSLT) is one of the W3C standards.
- A declarative rule-based programming language for transforming XML documents.
- Widely used in data-intensive applications.
- An XSLT stylesheet consists of a set of rule templates, each of them matches elements in source model, and produces output to the target model.
- XSLT has many industrial strength implementations, and can work with most of the programming languages.
Mapping Transformation to Stylesheet

- **QVT-R**
  - Transformation
  - Relation
  - Primitive domain
  - Function
  - Key
  - OCL expression

- **XSLT**
  - Stylesheet
  - Rule template
  - Template parameter
  - Function
  - Key
  - XPath expression

Mapping Relation to Rule Template

- **Relation**
  - Source domain pattern
  - When clause
  - Source domain pattern
  - Where clause
  - Target domain pattern
  - Relation calls

- **Rule Template**
  - Match expression
  - Variable declarations
  - Construction instructions
  - Template calls
Mapping OCL to XPath

OCL:

```
dn.outgoing->
select(name='else')->size()=1
```

XPath:

```
count(my:xmiXMIrefs(current()/@outgoing)[@name='else'])=1
```

Supports of QVT-R

- QVTR-XSLT supports unidirectional non-incremental enforcement transformations of QVT-R, where a set of output models is produced from a set of input models.

- Mechanisms for reuse and extension of generic transformation definitions;
- Transformation traceability;
- Multiple source and target metamodels and models;
- In-place updates where the source and target models are the same.
Pattern Matching

- QVT-R supports definition of complex patterns, which can involve sets of objects, non-existence of objects, ordered links, and navigate and gather information from different places of a model.

- The pattern matching semantics of QVT-R can be fully supported by the path expression of XPath.

Rule scheduling

- How the set of relations is arranged in the execution of a transformation.
- The when and where clauses of relations form a dependency tree, rooted from a specific top-level relation.
- The invocation of a relation not only depends on its domain patterns, but also determined by its dependencies.

- In QVTR-XSLT, a where relation call is implemented as an XSLT template invocation instruction.
- A when relation call is simulated as a backward where relation call.
Correctly Execution of Transformations

- Through analyzing the trace-model for:
  - all top-level relations are executed;
  - all elements that have been referred in other relations are correctly created; and
  - no duplicate created element.

Features of QVT-XSLT Tool

- Definition of metamodels using EMOF;
- Definition of transformations using QVT-R graphical notation;
- Model query with graphical notation;
- Support unidirectional non-incremental enforcement transformations of QVT-R;
- Complex pattern matching of object templates, property templates, collection templates, and not templates;
- Transformation inheritance through rule overriding;
- Traceability of transformation executions;
- Multiple input and output models;
- In-place transformations;
Characteristic of the Transformation Approach and Tool

- The approach is founded on the OMG and W3C standards, and makes use of well-known and commonly adopted languages and CASE tools.
- It is easy and user-friendly to design QVT-R rules, and the syntax-directed editing can reduce the number of errors.
- The generated XSLT programs can directly executed under any XSLT processor on any platform, and can be easily integrated into other applications and systems.
- As there are already many industrial-strength XSLT processors, transformations can run fast and efficiently process large-scale models.
QVTR-XSLT Tool

- A Graphical Editor
  - support design of QVTR transformations in graphical notation;
  - save the QVTR transformation models as XML files.

- A Code Generator
  - reads in the transformation model, generates corresponding XSLT stylesheets.

QVT-R Graphical Editor
XSLT Code Generator

Agenda

- Modeling and Model Transformations
- QVT and Support Tools
- QVTR-XSLT Approach
- Case Study: UML to RDBMS Transformation
- Case Study: Support rCOS development with refinement transformations.
- Conclusions
The is the classical simple UML model to simple RDBMS transformation, which is well-suited to demonstrate ideas and technical details of a transformation language.

We design seven rules for the transformation, and configure it with appropriate metamodels.

For comparison purposes, we take the medini QVT as a reference implementation of QVT-R, and execute the same transformation.

The Metamodels

- UML metamodel
- RDBMS metamodel
Transformation Rules (3)

Generate XSLT for the Transformation
Comparison - UML to RDBMS Example

- XSLT generated by the tool: 154 lines of code
- QVT relations in textual: 120 lines
- QVT operational: 100 lines
- QVT core: 400 lines
- MT: 140 lines
- medini QVT: 240 lines

Execution of the Transformation

- Source model: a small library model with 6 classes and 59 features in total;
- Environment: a laptop of Intel M330 2.13 GHz CPU, 3 GB memory, Windows 7 home version

Comparison of execution time (ms)

<table>
<thead>
<tr>
<th></th>
<th>QVTR-XSLT</th>
<th>medini QVT</th>
</tr>
</thead>
<tbody>
<tr>
<td>With trace</td>
<td>22</td>
<td>90</td>
</tr>
<tr>
<td>Without trace</td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>
Comparison of Execution Traces

Transformation name="umlTOrdbms" source="uml" target="rdbms">
  <relation name="PackageToSchema">
    <obj name="p" href="_pXZ9wKS6EduN8Krm-aC8vQ" refname="myPackage" model="uml"/>
    <obj name="s" href="myPackage" model="rdbms" op="c"/>
  </relation>
  <relation name="ClassToTable">
    <obj name="c" href="_pXZ9waS6EduN8Krm-aC8vQ" refname="Library" model="uml"/>
    <obj name="p" href="_pXZ9wKS6EduN8Krm-aC8vQ" refname="myPackage" model="uml"/>
    <obj name="t" href="Library" model="rdbms"/>
    <obj name="cl" href="Library_tid" model="rdbms"/>
    <obj name="k" href="Library_pk" model="rdbms"/>
    <obj name="s" href="myPackage" model="rdbms"/>
  </relation>
  <relation name="AttributeToColumn">
    <primitiveDomain name="prefix" href="Library"/>
    <obj name="c" href="_pXZ9waS6EduN8Krm-aC8vQ" refname="Library" model="uml"/>
    <obj name="t" href="Library" model="rdbms"/>
  </relation>
</Transformation>

QVTR-XSLT trace model

medini QVT trace model

---

Agenda

- Modeling and Model Transformations
- QVT and Support Tools
- QVTR-XSLT Approach
- Case Study: UML to RDBMS Transformation
- Case Study: Support rCOS development with refinement transformations.
- Conclusions
Case Study: Support rCOS Development

- Requirements Model
  - Use cases
  - Concept class diagrams, functionality specification
  - Dynamic behavior (state machine, sequence diagram)

- OO Design Model
  - Object (sequence) diagrams
  - Design class diagrams
  - OO designs

- Component Model
  - Component diagrams
  - Component sequence diagrams
  - Composition/deployment

rCOS Refinement Rules

- Object-oriented refinement:
  - functionality delegation (*expert pattern*), attribute encapsulation, class decomposition.

- Object-oriented model to component-based model:
  - OO sequence diagram to component sequence diagram transformation.

- Component composition:
  - parallel composition, plugging, disjoint union, coordination, renaming interface methods, hiding interface methods, etc.

- Structural refinement (small step):
  - adding classes, attributes, methods, promoting methods to super classes, etc.
Model Transformations in rCOS

- **Functionality delegation (expert pattern)**: refine a requirements model to an OO design model by assigning responsibilities to information experts;
- **Object sequence diagram to component sequence**: abstract an OO design model into a component model by aggregating objects into components;
- **Component composition**: rename method, hide method, parallel composition, plug components;
- **Generate state machines from sequence diagrams**;
- **Small-step class structure refactoring**: rename, add, forward, decompose, remove, merge.

rCOS Refinement Transformations

- **Requirements model to OO design model**
  ➢ generate controller classes with guarded designs and state machines
- **Functionality delegation (expert pattern)**
  ➢ Refine OO design models by assigning responsibilities
- **OO design model to component-based model**
  ➢ aggregate objects into components
- **Component composition**
  ➢ rename method, hide method, parallel composition, plug components
Complexity of the Transformations

- Concern different types of models:
  - Semantic models
  - Concrete syntax (graphical) models
  - Textual specifications, and.
  - Their synchronization.
- Handle different aspects of models:
  - Use Case diagrams, Class Diagrams, Sequence Diagrams, State Machine Diagrams, Component Diagrams, and
  - Their consistency.

Implementation of the Transformations

- The transformations are designed using the graphical notation of QVT-R in the graphical editor of the QVTR-XSLT tool.
- XSLT programs are generated for the transformations using the XSLT code generator.
- The XSLT programs are integrated into the rCOS modeler and invoked from it.
- For example, the OO design model to component-based model transformation is specified as 105 relations with 45 functions and queries. About 6300 lines of XSLT code are generated.
rCOS Sequence Diagrams

Visual Diagram Modeling

UML 2.0 Diagram Interchange (DI)
An Example Relation

An Example Query

- Get corresponding operation of a message
Requirements Modeling

```java
public enterItem (Barcode code, int qty) {
    qty > 0 /* true */;
    /* Item i: store, catalog, contains (i) & barcode = code */
    /* line = LineItem.new(code, qty) */
    /* j: store, catalog, contains (j) & barcode = code */
    /* line subtotal * qty */
    /* sale.lines.add(line) */
```
Object-oriented Design

System sequence diagram

Functionality delegation

OO interaction diagram

OO Model to Component Model

Abstract

OO interaction model

Component interaction model
Component Architecture Design

Composition Operators

Agenda

- Modeling and Model Transformations
- QVT and Support Tools
- QVTR-XSLT Approach
- Case Study: UML to RDBMS Transformation
- Case Study: Support rCOS development with refinement transformations.
- Conclusions
**Limitations of the Approach**

- Not all features of QVT-R standard are supported
  - Relation domains;
  - Transformation directions
  - Check-before-enforce semantics and in-place transformation
- Correctness of the QVT-R implementation is not formally proven.
- The rCOS profile does not fully support rCOS component model
  - coordinators (coordinating processes)
- Generation of proof obligations is not supported.

**Future Work**

- Stream model transformations: XSLT 3.0 supports XML streams.
- Program transformations: LLVM IR (intermediate representation), refactoring, insert trace instructions to detect data race.
- Generation of the proof obligations for verification of the transformation will be investigated.
- Migration of the QVT-R graphical editor to an open source UML CASE tool will be considered.
Thank You!
Lecture 10

Introduction to Association Analysis

by Peng Cheng
Introduction to Association Analysis

Peng Cheng
Southwest University

Association Rule Mining

- Given a set of transactions, find rules that will predict the occurrence of an item based on the occurrences of other items in the transaction

Market-Basket transactions

<table>
<thead>
<tr>
<th>TID</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Bread, Milk</td>
</tr>
<tr>
<td>2</td>
<td>Bread, Diaper, Beer, Eggs</td>
</tr>
<tr>
<td>3</td>
<td>Milk, Diaper, Beer, Coke</td>
</tr>
<tr>
<td>4</td>
<td>Bread, Milk, Diaper, Beer</td>
</tr>
<tr>
<td>5</td>
<td>Bread, Milk, Diaper, Coke</td>
</tr>
</tbody>
</table>

Example of Association Rules

\{\text{Diaper}\} \rightarrow \{\text{Beer}\},
\{\text{Milk}, \text{Bread}\} \rightarrow \{\text{Eggs, Coke}\},
\{\text{Beer}, \text{Bread}\} \rightarrow \{\text{Milk}\},

Implication means co-occurrence, not causality!
Definition: Frequent Itemset

- **Itemset**
  - A collection of one or more items
  - Example: \{Milk, Bread, Diaper\}
  - \(k\)-itemset
    - An itemset that contains \(k\) items

- **Support count (\(\sigma\))**
  - Frequency of occurrence of an itemset
  - E.g. \(\sigma(\{\text{Milk, Bread, Diaper}\}) = 2\)

- **Support**
  - Fraction of transactions that contain an itemset
  - E.g. \(s(\{\text{Milk, Bread, Diaper}\}) = \frac{2}{5}\)

- **Frequent Itemset**
  - An itemset whose support is greater than or equal to a \(\text{minsup}\) threshold

<table>
<thead>
<tr>
<th>TID</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Bread, Milk</td>
</tr>
<tr>
<td>2</td>
<td>Bread, Diaper, Beer, Eggs</td>
</tr>
<tr>
<td>3</td>
<td>Milk, Diaper, Beer, Coke</td>
</tr>
<tr>
<td>4</td>
<td>Bread, Milk, Diaper, Beer</td>
</tr>
<tr>
<td>5</td>
<td>Bread, Milk, Diaper, Coke</td>
</tr>
</tbody>
</table>

Definition: Association Rule

- **Association Rule**
  - An implication expression of the form \(X \Rightarrow Y\), where \(X\) and \(Y\) are itemsets
  - Example: \(\{\text{Milk, Diaper}\} \Rightarrow \{\text{Beer}\}\)

- **Rule Evaluation Metrics**
  - Support (\(s\))
    - Fraction of transactions that contain both \(X\) and \(Y\)
  - Confidence (\(c\))
    - Measures how often items in \(Y\) appear in transactions that contain \(X\)

  Example:
  \[
  \{\text{Milk, Diaper}\} \Rightarrow \text{Beer} \\
  s = \frac{\sigma(\text{Milk, Diaper, Beer})}{|T|} = \frac{2}{5} = 0.4 \\
  c = \frac{\sigma(\text{Milk, Diaper, Beer})}{\sigma(\text{Milk, Diaper})} = \frac{2}{3} = 0.67
  \]
Association Rule Mining Task

- Given a set of transactions $T$, the goal of association rule mining is to find all rules having
  - support $\geq \text{minsup}$ threshold
  - confidence $\geq \text{minconf}$ threshold

- Brute-force approach:
  - List all possible association rules
  - Compute the support and confidence for each rule
  - Prune rules that fail the $\text{minsup}$ and $\text{minconf}$ thresholds

$\Rightarrow$ Computationally prohibitive!

Mining Association Rules

Example of Rules:

- $\{\text{Milk,Diaper}\} \rightarrow \{\text{Beer}\}$ (s=0.4, c=0.67)
- $\{\text{Milk,Beer}\} \rightarrow \{\text{Diaper}\}$ (s=0.4, c=1.0)
- $\{\text{Diaper,Beer}\} \rightarrow \{\text{Milk}\}$ (s=0.4, c=0.67)
- $\{\text{Beer}\} \rightarrow \{\text{Milk,Diaper}\}$ (s=0.4, c=0.67)
- $\{\text{Diaper}\} \rightarrow \{\text{Milk,Beer}\}$ (s=0.4, c=0.5)
- $\{\text{Milk}\} \rightarrow \{\text{Diaper,Beer}\}$ (s=0.4, c=0.5)

Observations:

- All the above rules are binary partitions of the same itemset: $\{\text{Milk, Diaper, Beer}\}$
- Rules originating from the same itemset have identical support but can have different confidence
- Thus, we may decouple the support and confidence requirements
Mining Association Rules

● Two-step approach:
  1. Frequent Itemset Generation
     - Generate all itemsets whose support $\geq \text{minsup}$
  2. Rule Generation
     - Generate high confidence rules from each frequent itemset, where each rule is a binary partitioning of a frequent itemset

● Frequent itemset generation is still computationally expensive

Frequent Itemset Generation

Given d items, there are $2^d$ possible candidate itemsets
Frequent Itemset Generation

- **Brute-force approach:**
  - Each itemset in the lattice is a *candidate* frequent itemset
  - Count the support of each candidate by scanning the database
  - Match each transaction against every candidate
  - Complexity ~ $O(NMw)$ => Expensive since $M = 2^d$ !!!

<table>
<thead>
<tr>
<th>TID</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Bread, Milk</td>
</tr>
<tr>
<td>2</td>
<td>Bread, Diaper, Beer, Eggs</td>
</tr>
<tr>
<td>3</td>
<td>Milk, Diaper, Beer, Coke</td>
</tr>
<tr>
<td>4</td>
<td>Bread, Milk, Diaper, Beer</td>
</tr>
<tr>
<td>5</td>
<td>Bread, Milk, Diaper, Coke</td>
</tr>
</tbody>
</table>

Computational Complexity

- Given $d$ unique items:
  - Total number of itemsets = $2^d$
  - Total number of possible association rules:
    \[
    R = \sum_{k=1}^{d-1} \binom{d}{k} \cdot \sum_{j=1}^{d-k} \binom{d-k}{j}
    \]
    \[
    = 3^d - 2^d + 1
    \]
    If $d=6$, $R = 602$ rules
Frequent Itemset Generation Strategies

- Reduce the number of candidates \((M)\)
  - Complete search: \(M = 2^d\)
  - Use pruning techniques to reduce \(M\)

- Reduce the number of transactions \((N)\)
  - Reduce size of \(N\) as the size of itemset increases
  - Used by vertical-based mining algorithms

- Reduce the number of comparisons \((NM)\)
  - Use efficient data structures to store the candidates or transactions
  - No need to match every candidate against every transaction

Reducing Number of Candidates

- Apriori principle:
  - If an itemset is frequent, then all of its subsets must also be frequent

- Apriori principle holds due to the following property of the support measure:

\[
\forall X, Y : (X \subseteq Y) \Rightarrow s(X) \geq s(Y)
\]

  - Support of an itemset never exceeds the support of its subsets
  - This is known as the anti-monotone property of support
Illustrating Apriori Principle

<table>
<thead>
<tr>
<th>Item</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bread</td>
<td>4</td>
</tr>
<tr>
<td>Coke</td>
<td>2</td>
</tr>
<tr>
<td>Milk</td>
<td>4</td>
</tr>
<tr>
<td>Beer</td>
<td>3</td>
</tr>
<tr>
<td>Diaper</td>
<td>4</td>
</tr>
<tr>
<td>Eggs</td>
<td>1</td>
</tr>
</tbody>
</table>

Minimum Support = 3

If every subset is considered, $C_1^6 + C_2^6 + C_3^6 = 41$
With support-based pruning, $6 + 6 + 1 = 13$

Pairs (2-itemsets)

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>{Bread, Milk}</td>
<td>3</td>
</tr>
<tr>
<td>{Bread, Beer}</td>
<td>2</td>
</tr>
<tr>
<td>{Bread, Diaper}</td>
<td>3</td>
</tr>
<tr>
<td>{Milk, Beer}</td>
<td>2</td>
</tr>
<tr>
<td>{Milk, Diaper}</td>
<td>3</td>
</tr>
<tr>
<td>{Beer, Diaper}</td>
<td>3</td>
</tr>
</tbody>
</table>

(No need to generate candidates involving Coke or Eggs)

Triplets (3-itemsets)

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>{Bread, Milk, Diaper}</td>
<td>3</td>
</tr>
</tbody>
</table>
Apriori Algorithm

- **Method:**
  - Let $k=1$
  - Generate frequent itemsets of length 1
  - Repeat until no new frequent itemsets are identified
    - Generate length $(k+1)$ candidate itemsets from length $k$ frequent itemsets
    - Prune candidate itemsets containing subsets of length $k$ that are infrequent
    - Count the support of each candidate by scanning the DB
    - Eliminate candidates that are infrequent, leaving only those that are frequent

### The Apriori Algorithm — Example

<table>
<thead>
<tr>
<th>TID</th>
<th>Items</th>
<th>( L_1 )</th>
<th>( C_1 )</th>
<th>Scan D</th>
<th>( L_2 )</th>
<th>( C_2 )</th>
<th>Scan D</th>
<th>( L_3 )</th>
<th>( C_3 )</th>
<th>Scan D</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>1 3 4</td>
<td></td>
<td>{1}</td>
<td></td>
<td>{1 2}</td>
<td></td>
<td></td>
<td>{1 3}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>2 3 5</td>
<td></td>
<td>{2}</td>
<td></td>
<td>{1 3}</td>
<td></td>
<td></td>
<td>{2 3}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>300</td>
<td>1 2 3 5</td>
<td></td>
<td>{3}</td>
<td></td>
<td>{1 5}</td>
<td></td>
<td></td>
<td>{2 5}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>400</td>
<td>2 5</td>
<td></td>
<td>{4}</td>
<td></td>
<td></td>
<td>{3 5}</td>
<td></td>
<td></td>
<td>{3 5}</td>
<td></td>
</tr>
</tbody>
</table>

- Database D

- \( C_1 \): Itemsets of length 1
- \( L_1 \): Frequent itemsets of length 1

- \( C_2 \): Itemsets of length 2
- \( L_2 \): Frequent itemsets of length 2

- \( C_3 \): Itemsets of length 3
- \( L_3 \): Frequent itemsets of length 3
Factors Affecting Complexity

- Choice of minimum support threshold
  - lowering support threshold results in more frequent itemsets
  - this may increase number of candidates and max length of frequent itemsets
- Dimensionality (number of items) of the data set
  - more space is needed to store support count of each item
  - if number of frequent items also increases, both computation and I/O costs may also increase
- Size of database
  - since Apriori makes multiple passes, run time of algorithm may increase with number of transactions
- Average transaction width
  - transaction width increases with denser data sets
  - This may increase max length of frequent itemsets and traversals of hash tree (number of subsets in a transaction increases with its width)

Maximal Frequent Itemset

An itemset is maximal frequent if none of its immediate supersets is frequent
Closed Itemset

- An itemset is closed if none of its immediate supersets has the same support as the itemset

<table>
<thead>
<tr>
<th>TID</th>
<th>Items</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>{A,B}</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>{B,C,D}</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>{A,B,C,D}</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>{A,B,D}</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>{A,B,C,D}</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>{A}</td>
<td>4</td>
</tr>
<tr>
<td>{B}</td>
<td>5</td>
</tr>
<tr>
<td>{C}</td>
<td>3</td>
</tr>
<tr>
<td>{D}</td>
<td>4</td>
</tr>
<tr>
<td>{A,B}</td>
<td>4</td>
</tr>
<tr>
<td>{A,C}</td>
<td>2</td>
</tr>
<tr>
<td>{A,D}</td>
<td>3</td>
</tr>
<tr>
<td>{B,C}</td>
<td>3</td>
</tr>
<tr>
<td>{B,D}</td>
<td>4</td>
</tr>
<tr>
<td>{C,D}</td>
<td>3</td>
</tr>
</tbody>
</table>

Maximal vs Closed Itemsets

<table>
<thead>
<tr>
<th>TID</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ABC</td>
</tr>
<tr>
<td>2</td>
<td>ABCD</td>
</tr>
<tr>
<td>3</td>
<td>BCE</td>
</tr>
<tr>
<td>4</td>
<td>ACDE</td>
</tr>
<tr>
<td>5</td>
<td>DE</td>
</tr>
</tbody>
</table>

Transaction Ids

Not supported by any transactions
Maximal vs Closed Frequent Itemsets

Minimum support = 2

# Closed = 9
# Maximal = 4
Rule Generation

● Given a frequent itemset \( L \), find all non-empty subsets \( f \subset L \) such that \( f \rightarrow L - f \) satisfies the minimum confidence requirement
  
  - If \{A,B,C,D\} is a frequent itemset, candidate rules:
    
    \[
    \begin{align*}
    &ABC \rightarrow D, \quad ABD \rightarrow C, \quad ACD \rightarrow B, \quad BCD \rightarrow A, \\
    &A \rightarrow BCD, \quad B \rightarrow ACD, \quad C \rightarrow ABD, \quad D \rightarrow ABC, \\
    &AB \rightarrow CD, \quad AC \rightarrow BD, \quad AD \rightarrow BC, \quad BC \rightarrow AD, \\
    &BD \rightarrow AC, \quad CD \rightarrow AB,
    \end{align*}
    \]
  
  ● If \( |L| = k \), then there are \( 2^k - 2 \) candidate association rules (ignoring \( L \rightarrow \emptyset \) and \( \emptyset \rightarrow L \))

Rule Generation

● How to efficiently generate rules from frequent itemsets?
  
  - In general, confidence does not have an anti-monotone property
    
    \( c(ABC \rightarrow D) \) can be larger or smaller than \( c(AB \rightarrow D) \)
  
  - But confidence of rules generated from the same itemset has an anti-monotone property
  
    e.g., \( L = \{A,B,C,D\} \):
    
    \[
    c(ABC \rightarrow D) \geq c(AB \rightarrow CD) \geq c(A \rightarrow BCD)
    \]
  
    ● Confidence is anti-monotone w.r.t. number of items on the RHS of the rule
Rule Generation for Apriori Algorithm

Lattice of rules

Low Confidence Rule

Pruned Rules

Pattern Evaluation

- Association rule algorithms tend to produce too many rules
  - many of them are uninteresting or redundant
  - Redundant if \{A,B,C\} → \{D\} and \{A,B\} → \{D\} have same support & confidence

- Interestingness measures can be used to prune/rank the derived patterns

- In the original formulation of association rules, support & confidence are the only measures used
Computing Interestingness Measure

Given a rule \( X \rightarrow Y \), information needed to compute rule interestingness can be obtained from a contingency table.

Contingency table for \( X \rightarrow Y \):

<table>
<thead>
<tr>
<th></th>
<th>( Y )</th>
<th>( \overline{Y} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( X )</td>
<td>( f_{11} )</td>
<td>( f_{10} )</td>
</tr>
<tr>
<td>( \overline{X} )</td>
<td>( f_{01} )</td>
<td>( f_{00} )</td>
</tr>
<tr>
<td>( f_{+1} )</td>
<td>( f_{+0} )</td>
<td>(</td>
</tr>
</tbody>
</table>

- \( f_{11} \): support of \( X \) and \( Y \)
- \( f_{10} \): support of \( X \) and \( \overline{Y} \)
- \( f_{01} \): support of \( \overline{X} \) and \( Y \)
- \( f_{00} \): support of \( \overline{X} \) and \( \overline{Y} \)

Used to define various measures:
- support, confidence, lift, Gini, J-measure, etc.

Drawback of Confidence

<table>
<thead>
<tr>
<th></th>
<th>Coffee</th>
<th>Coffee</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tea</td>
<td>15</td>
<td>5</td>
</tr>
<tr>
<td>Tea</td>
<td>75</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>90</td>
<td>10</td>
</tr>
</tbody>
</table>

Association Rule: \( \text{Tea} \rightarrow \text{Coffee} \)

Confidence = \( P(\text{Coffee}|\text{Tea}) = 0.75 \)

but \( P(\text{Coffee}) = 0.9 \)

\( \Rightarrow \) Although confidence is high, rule is misleading

\( \Rightarrow P(\text{Coffee}|\overline{\text{Tea}}) = 0.9375 \)
Statistical Independence

- Population of 1000 students
  - 600 students know how to swim (S)
  - 700 students know how to bike (B)
  - 420 students know how to swim and bike (S,B)

  - \( P(S \cap B) = \frac{420}{1000} = 0.42 \)
  - \( P(S) \times P(B) = 0.6 \times 0.7 = 0.42 \)

  - \( P(S \cap B) = P(S) \times P(B) \Rightarrow \text{Statistical independence} \)
  - \( P(S \cap B) > P(S) \times P(B) \Rightarrow \text{Positively correlated} \)
  - \( P(S \cap B) < P(S) \times P(B) \Rightarrow \text{Negatively correlated} \)

Example: Lift/Interest

<table>
<thead>
<tr>
<th></th>
<th>Coffee</th>
<th>Coffee</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tea</td>
<td>15</td>
<td>5</td>
</tr>
<tr>
<td>Tea</td>
<td>75</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>90</td>
<td>10</td>
</tr>
</tbody>
</table>

Association Rule: Tea \(\rightarrow\) Coffee

Confidence\(= P(\text{Coffee}|\text{Tea}) = 0.75 \)

but \( P(\text{Coffee}) = 0.9 \)

\( \Rightarrow \text{Lift} = \frac{0.75}{0.9} = 0.8333 \) (< 1, therefore is negatively associated)
There are lots of measures proposed in the literature.

Some measures are good for certain applications, but not for others.

What criteria should we use to determine whether a measure is good or bad?

What about Apriori-style support based pruning? How does it affect these measures?
Application case study —— Identify risk factors in AKI disease

AKI (Acute Kidney Injury) Definition

- KDIGO staging system for AKI

<table>
<thead>
<tr>
<th>AKI Stage</th>
<th>Serum Creatinine (SCr) Criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Increase &gt;26.4 µmol/L (0.3 mg/dL) or 1.5-1.9 times baseline</td>
</tr>
<tr>
<td>2</td>
<td>Increase 2.0-2.9 times baseline</td>
</tr>
<tr>
<td>3</td>
<td>Increase creatinine &gt;354 µmol/L (4.0 mg/dL) or 3 times baseline</td>
</tr>
</tbody>
</table>
Cohort Baseline Characteristics

### Demographics

<table>
<thead>
<tr>
<th>Age, n (%)</th>
<th>AKI-1 (n=6396)</th>
<th>AKI-2 (n=678)</th>
<th>AKI-3 (n=185)</th>
<th>Non-AKI (n=69698)</th>
<th>P value</th>
</tr>
</thead>
<tbody>
<tr>
<td>18-25</td>
<td>303(4.74)</td>
<td>29(4.28)</td>
<td>25(13.51)</td>
<td>4596(6.59)</td>
<td>&lt;.001</td>
</tr>
<tr>
<td>26-35</td>
<td>514(8.04)</td>
<td>44(6.49)</td>
<td>23(12.43)</td>
<td>7339(10.53)</td>
<td>NA</td>
</tr>
<tr>
<td>36-45</td>
<td>711(11.12)</td>
<td>76(11.21)</td>
<td>25(13.51)</td>
<td>8601(12.34)</td>
<td>NA</td>
</tr>
<tr>
<td>46-55</td>
<td>1218(19.04)</td>
<td>157(23.17)</td>
<td>35(18.92)</td>
<td>14374(20.62)</td>
<td>NA</td>
</tr>
<tr>
<td>56-64</td>
<td>1672(26.14)</td>
<td>185(27.29)</td>
<td>49(26.49)</td>
<td>16192(23.24)</td>
<td>NA</td>
</tr>
<tr>
<td>&gt;64</td>
<td>1978(30.92)</td>
<td>187(27.56)</td>
<td>28(15.14)</td>
<td>18596(26.68)</td>
<td>NA</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Race, n (%)</th>
<th>AKI-1 (n=6396)</th>
<th>AKI-2 (n=678)</th>
<th>AKI-3 (n=185)</th>
<th>Non-AKI (n=69698)</th>
<th>P value</th>
</tr>
</thead>
<tbody>
<tr>
<td>White</td>
<td>4791(74.91)</td>
<td>487(71.83)</td>
<td>130(70.27)</td>
<td>53177(76.30)</td>
<td>&lt;0.001</td>
</tr>
<tr>
<td>African American</td>
<td>918(14.35)</td>
<td>111(16.37)</td>
<td>36(19.46)</td>
<td>9336(13.39)</td>
<td>0.0025</td>
</tr>
<tr>
<td>Asian</td>
<td>45(0.70)</td>
<td>7(1.03)</td>
<td>2(1.08)</td>
<td>600(0.86)</td>
<td>0.3016</td>
</tr>
<tr>
<td>Other</td>
<td>642(10.04)</td>
<td>73(10.77)</td>
<td>17(9.19)</td>
<td>6585(9.45)</td>
<td>0.0787</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Gender, n (%)</th>
<th>AKI-1 (n=6396)</th>
<th>AKI-2 (n=678)</th>
<th>AKI-3 (n=185)</th>
<th>Non-AKI (n=69698)</th>
<th>P value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Female</td>
<td>2574(40.24)</td>
<td>300(44.25)</td>
<td>76(41.08)</td>
<td>31848(45.69)</td>
<td>NA</td>
</tr>
<tr>
<td>Male</td>
<td>3822(59.76)</td>
<td>379(55.75)</td>
<td>109(58.92)</td>
<td>37850(54.31)</td>
<td>NA</td>
</tr>
</tbody>
</table>

### Clinical Variables

<table>
<thead>
<tr>
<th>Feature Category</th>
<th># of Variables</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Demographics</td>
<td>3</td>
<td>Age, gender, race</td>
</tr>
<tr>
<td>Vitals</td>
<td>5</td>
<td>BMI, diastolic BP, systolic BP, pulse, temperature</td>
</tr>
<tr>
<td>Lab tests</td>
<td>14</td>
<td>Albumin, ALT, AST, Ammonia, Blood Bilirubin, BUN, Ca, CK-MB, CK, Glucose, Lipase, Platelets, Troponin, WBC</td>
</tr>
<tr>
<td>Comorbidities</td>
<td>29</td>
<td>UHC comorbidity</td>
</tr>
<tr>
<td>Admission diagnosis</td>
<td>315</td>
<td>UHC APR-DRG</td>
</tr>
<tr>
<td>Medications</td>
<td>1682</td>
<td>All medications are mapped to RxNorm ingredient</td>
</tr>
<tr>
<td>Problem List</td>
<td>280</td>
<td>ICD9 codes mapped to CCS major diagnoses</td>
</tr>
</tbody>
</table>
Methodology

- Often, a great amount of association rules are generated.

- We utilized quantitative measures to select top-\(n\) rules from numerous candidates. Three measures were adopted as follows:
  - **Confidence**
  - **Risk lift**
  - **Relative risk**

- High confidence means high reliability to result in AKI.
Risk Lift

- Differentiate strength of various risk factors in one rule
  - Often, there are multiple factors in a rule’s antecedent which infer the occurrence of AKI.
  - We are interested to know which factor(s) in a combination play the major role of increasing AKI risk.
  - For example, in the rule \( \{A, B, C\} \rightarrow \{\text{AKI}\} \), we are interested to know which factor(s) contribute to AKI most. It’s possible to be B, and it’s also possible to be the combination of B and C.

Relative risk

- Relative risk is a measure often used in epidemiological studies.
- For a AKI-related association rule, \( \{A, B, C\} \Rightarrow \{\text{AKI}\} \), the inferred risk patterns are \( \{A, B, C\} \), we may get:

<table>
<thead>
<tr>
<th></th>
<th>AKI</th>
<th>NON-AKI</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A B C)</td>
<td>( \text{Prob}({A, B, C}, \text{AKI}) )</td>
<td>( \text{Prob}({A, B, C}, \text{NON-AKI}) )</td>
<td>( \text{Prob}({A, B, C}) )</td>
</tr>
<tr>
<td>( \neg (A,B,C) )</td>
<td>( \text{Prob}(\neg{A,B,C}, \text{AKI}) )</td>
<td>( \text{Prob}(\neg{A,B,C}, \text{NON-AKI}) )</td>
<td>( \text{Prob}(\neg{A,B,C}) )</td>
</tr>
<tr>
<td>total</td>
<td>( \text{Prob}(\text{AKI}) )</td>
<td>( \text{Prob}(\text{NON-AKI}) )</td>
<td>1</td>
</tr>
</tbody>
</table>

- Relative risk = \[
\frac{\text{Supp}(\{A, B, C\} \cup \text{AKI})}{\text{Supp}(A, B, C)} / \frac{\text{Supp}(\neg\{A, B, C\} \cup \text{AKI})}{\text{Supp}(\neg\{A, B, C\})}
\]
Experimental Results

- The original output of association rule mining only includes the most abbreviated form of rules. For easy review, we added derived AKI-related rules.

- For example,
  \[
  \{A, B\} \implies \{C, D, AKI\} \implies \\
  \{A, B\} \implies \{AKI\} \\
  \{A, B, C\} \implies \{AKI\} \\
  \{A, B, C, D\} \implies \{AKI\}
  \]

  The original output only contains \(\{A, B\} \implies \{C, D, AKI\}\). However, we want to see all derived AKI-related rules, so they are also contained in the outcome.

Experimental Results

- high base ratio vs. low base ratio
  - **High base ratio**: LIVER TRANSPLANT (51.6%), TRACH W/DMV W EXTEN PROC (58.3%), RESPIRATORY SYSTEM DIAG (46.6%)
  - **Low base ratio**: BONE MARROW TRANSPLANT (23.7%), HEART FAILURE (21.5%), ACUTE LEUKEMIA (20.7%)

- Post-operative AKI vs. Non-operative AKI
  - **Post-operative**: LIVER TRANSPLANT, TRACH W/DMV W EXTEN PROC, BONE MARROW TRANSPLANT
  - **Non-operative**: RESPIRATORY SYSTEM DIAG, HEART FAILURE, LEUKEMIA
### LIVER TRANSPLANT

**maximal confidence: 0.90000**

( 326 1217 => 318 478 1615  #SUP: 9  #CONF: 0.69231 )

( Coagulopathy, spironolactone => Liver disease, benzoic acid, AKI )

<table>
<thead>
<tr>
<th>Rule</th>
<th>Support Set</th>
<th>Confidence</th>
<th>Lift</th>
<th>Relative Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>1217 =&gt; 1615</td>
<td>15</td>
<td>0.53571</td>
<td>0.01927</td>
<td>1.04125</td>
</tr>
<tr>
<td>326 1217 =&gt; 1615</td>
<td>10</td>
<td>0.76923</td>
<td>0.23352</td>
<td>1.52276</td>
</tr>
<tr>
<td>326 1217 =&gt; 1615</td>
<td>10</td>
<td>0.76923</td>
<td></td>
<td></td>
</tr>
<tr>
<td>326 478 1217 =&gt; 1615</td>
<td>10</td>
<td>0.76923</td>
<td>0.04895</td>
<td>1.61978</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rule</th>
<th>Support Set</th>
<th>Confidence</th>
<th>Lift</th>
<th>Relative Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>1217 =&gt; 1615</td>
<td>15</td>
<td>0.53571</td>
<td></td>
<td></td>
</tr>
<tr>
<td>326 =&gt; 1615</td>
<td>44</td>
<td>0.51163</td>
<td></td>
<td></td>
</tr>
<tr>
<td>478 =&gt; 1615</td>
<td>37</td>
<td>0.30579</td>
<td></td>
<td></td>
</tr>
<tr>
<td>318 =&gt; 1615</td>
<td>130</td>
<td>0.51587</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### LIVER TRANSPLANT

**maximal lift: 0.35500**

( 482 1353 => 1333 1615  #SUP: 7  #CONF: 0.87500 )

( methylene blue, diphenhydramine => fentanyl, AKI )

<table>
<thead>
<tr>
<th>Rule</th>
<th>Support Set</th>
<th>Confidence</th>
<th>Lift</th>
<th>Relative Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>482 =&gt; 1615</td>
<td>13</td>
<td>0.52000</td>
<td>0.00355</td>
<td>1.00750</td>
</tr>
<tr>
<td>482 1353 =&gt; 1615</td>
<td>7</td>
<td>0.87500</td>
<td>0.35500</td>
<td>1.72667</td>
</tr>
<tr>
<td>482 1333 1353 =&gt; 1615</td>
<td>7</td>
<td>0.87500</td>
<td>0.00000</td>
<td>1.72667</td>
</tr>
<tr>
<td>482 =&gt; 1615</td>
<td>13</td>
<td>0.52000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1353 =&gt; 1615</td>
<td>29</td>
<td>0.43284</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1333 =&gt; 1615</td>
<td>96</td>
<td>0.39669</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# LIVER TRANSPLANT

Maximal relative risk: 1.78784

(326 1217 => 318 478 1615) #SUP: 9  #CONF: 0.69231
(Conagulopthy, spironolactone => Liver disease, benzoic acid, AKI)

<table>
<thead>
<tr>
<th>Path</th>
<th>SUP</th>
<th>CONF</th>
<th>Lift</th>
<th>Relative Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>1217 =&gt; 1615</td>
<td>15</td>
<td>0.53571</td>
<td>0.01927</td>
<td>1.04125</td>
</tr>
<tr>
<td>326 1217 =&gt; 1615</td>
<td>10</td>
<td>0.76923</td>
<td>0.23352</td>
<td>1.52276</td>
</tr>
<tr>
<td>326 478 1217 =&gt; 1615</td>
<td>9</td>
<td>0.81818</td>
<td>0.04895</td>
<td>1.61978</td>
</tr>
<tr>
<td>318 326 478 1217 =&gt; 1615</td>
<td>9</td>
<td>0.90000</td>
<td>0.08182</td>
<td>1.78784</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Path</th>
<th>SUP</th>
<th>CONF</th>
<th>Lift</th>
</tr>
</thead>
<tbody>
<tr>
<td>1217 =&gt; 1615</td>
<td>15</td>
<td>0.53571</td>
<td></td>
</tr>
<tr>
<td>326 =&gt; 1615</td>
<td>44</td>
<td>0.51163</td>
<td></td>
</tr>
<tr>
<td>478 =&gt; 1615</td>
<td>37</td>
<td>0.30579</td>
<td></td>
</tr>
<tr>
<td>318 =&gt; 1615</td>
<td>130</td>
<td>0.51587</td>
<td></td>
</tr>
</tbody>
</table>

Thank you for your attendance!
Lecture 11

Optimal Switching Controller Synthesis
by Symbolic Computation – A Case Study

by Hengjun Zhao
Optimal Switching Controller Synthesis by Symbolic Computation — A Case Study

Hengjun Zhao

joint work with Naijun Zhan, Deepak Kapur and Kim G. Larsen

1School of Computer and Information Science, Southwest University
Chongqing, China
BeChong School

October 19, 2017

Outline

1 The Oil Pump Control Problem
2 Motivation
3 The Overall Approach
4 Solving the Oil Pump Problem
   • Step 1
   • Step 2
   • Step 3
   • Results
5 Conclusions
The Oil Pump Control Problem

Motivation

The Overall Approach

Solving the Oil Pump Problem
  - Step 1
  - Step 2
  - Step 3
  - Results

Conclusions

The Hunchback of Notre Dame: Quasimodo
The Oil Pump Control Problem

Motivation

The Overall Approach

Solving the Oil Pump Problem

Conclusions

A Reported Case Study


- Provided by the HYDAC ELECTRONIC GMBH company within the European project Quasimodo
- An oil pump control problem
  - safety
  - robustness
  - optimality
The Hydraulic System

- The system is composed of a machine, an accumulator, a reservoir and a pump.
- The machine consumes oil out of the accumulator; the pump adds oil from the reservoir into the accumulator.

The Consumption Rate

- The oil consumption is periodic. The length of one consumption cycle is 20s (second).
- The profile of consumption rate in one cycle is depicted by
The Pump

- The power of the pump is $2.2 \text{l/s}$ (liter/second)
- 2-second latency: if the pump is switched on ($t_{2k+1}$) or off ($t_{2k+2}$) at time points
  
  \[ 0 \leq t_1 \leq t_2 \leq \cdots \leq t_i \leq t_{i+1} \leq \cdots, \]

  then
  \[ t_{i+1} - t_i \geq 2 \]

  for any $i \geq 1$
- It is obvious that the pump can be turned on at most 5 times in one cycle

The Hydro-Pneumatic Accumulator

Diaphragm & Nitrogen

Figure 2: **diaphragm** accumulator
The Accumulator: Safety

Volume upper and lower bound:

Control Objectives: Safety

Determine the $t_i$'s in order to

- $R_s (\text{safety})$: maintain
  
  $v(t) \in [V_{\text{min}}, V_{\text{max}}]$, \quad \forall t \in [0, \infty)$

  - $v(t)$ denotes the oil volume in the accumulator at time $t$
  - $V_{\text{min}} = 4.9$ l (liter)
  - $V_{\text{max}} = 25.1$ l

- Under the constraint:
  
  $R_{\text{pl}} (\text{pump latency})$: $t_{i+1} - t_i \geq 2$
The Oil Pump Control Problem

**Motivation**

**The Overall Approach**

**Solving the Oil Pump Problem**

**Conclusions**

**Control Objectives: Robustness**

- $R_r$ (robustness): uncertainties of the system should be taken into account:
  - fluctuation of consumption rate (if it is not 0), up to $f = 0.1l/s$
  - imprecision in the measurement of oil volume, up to $\varepsilon = 0.06l$
  - imprecision in the measurement of time, up to $\delta = 0.015s$.

**Control Objectives: Optimality**

Considering the energy cost and wear of the system,

- $R_o$ (optimality): minimize the average accumulated oil volume in the limit, i.e. minimize

$$\lim_{T \to \infty} \frac{1}{T} \int_{t=0}^{T} v(t) \, dt$$
The General Problem: **Switching Control**

Given a hybrid system $\mathcal{H}$ in which transition conditions $h_{ij}$ are not determined but parameterized by $u$, a vector of control parameters.

Our task is to determine $u$ such that $\mathcal{H}$ can make discrete jumps at desired points, thus guaranteeing that

- a safety property $S$ is satisfied, i.e. $x \in S$ at any time
- an optimization goal, e.g. $\min_u g(u)$, is achieved, where $g(u)$ is a certain objective function.

---

Outline

1. The Oil Pump Control Problem
2. Motivation
3. The Overall Approach
4. Solving the Oil Pump Problem
   - Step 1
   - Step 2
   - Step 3
   - Results
5. Conclusions
Existing Solutions

- HYDAC
  - bang-bang controller
  - smart controller
- Matlab/Simulink
  - correctness and robustness not verified

Existing Solutions

- HSCC’09
- F. Cassez etc: Efficient On-the-fly Algorithms for the Analysis of Timed Games, CONCUR’05
- Timed Game Automata, UPPAAL-TIGA
Existing Solutions

- HSCC’09
- optimality improved: 30-40%
- correctness and robustness have to be verified posteriorly using PHAVer:

Numeric vs Symbolic Computation

- Numeric computation (discretization) suffers from numerical imprecision
  - incorrect controllers
  - not optimal

- Symbolic approach (first-order logic about real numbers, quantifier elimination) is precise so it can generate correct controllers and better optimal values
Real Quantifier Elimination

- First-order theory of reals
  - polynomials (with rational coefficients): $3x^2 + 2y$
  - equations and inequalities: $>, \geq, <, \leq, =$
  - logical symbols: $\forall, \exists, \land, \lor, \ldots$
  - $\exists x. x^2 < 0$
- Quantifier Elimination (QE)
  - $Qx. \phi(x) \equiv true (false): \exists x. x^2 < 0 \equiv (false)$
  - $Qx. \phi(x, y) \equiv \varphi(y): \exists x. x^2 < y \equiv y > 0$
- QE is an expensive operation (doubly exponential)
- It has to be used judiciously

Outline

1. The Oil Pump Control Problem
2. Motivation
3. The Overall Approach
4. Solving the Oil Pump Problem
   - Step 1
   - Step 2
   - Step 3
   - Results
5. Conclusions
The General Problem

Given a hybrid system $\mathcal{H}$ in which transition conditions $h_{ij}$ are not determined but parameterized by $u$, a vector of control parameters.

Our task is to determine $u$ such that $\mathcal{H}$ can make discrete jumps at desired points, thus guaranteeing that

- a safety property $S$ is satisfied, i.e. $x \in S$ at any time
- an optimization goal, e.g. $\min_u g(u)$, is achieved

Our Approach – Step 1

Derive constraint $D(u)$ on $u$ from the safety requirements $S$

- Compute reachable set $\text{Reach}_\mathcal{H}(x, u)$ of $\mathcal{H}$ as polynomial formulas
- Suppose $S$ is also modeled by polynomial formulas, then $D(u)$ can be obtained by applying QE to

$$\forall x. \left( \text{Reach}_\mathcal{H}(x, u) \rightarrow S \right)$$
Our Approach – Step 2

Encode the optimization problem (suppose the objective function $g$ is a polynomial) over constraint $D(u)$ into a quantified first-order polynomial formula $Qu.\varphi(u, z)$ by introducing a fresh variable $z$

- Minimize $u^2$ on $[-1, 1]$

- Introduce a fresh variable $z$: $u \geq -1 \land u \leq 1 \land u^2 \leq z$
Our Approach – Step 2

Encode the optimization problem (suppose the objective function \( g \) is a polynomial) over constraint \( D(u) \) into a quantified first-order polynomial formula \( Qu.\varphi(u, z) \) by introducing a fresh variable \( z \):

- Minimize \( u^2 \) on \([-1, 1]\)
- Introduce a fresh variable \( z \):
  \[
  u \geq -1 \land u \leq 1 \land u^2 \leq z
  \]
- Projection to the \( z \)-axis:
  \[
  \exists u. (u \geq -1 \land u \leq 1 \land u^2 \leq z)
  \]
- After QE: \( z \geq 0 \), which means
  \[
  \min_{u \in [-1, 1]} u^2 = 0
  \]

Lemma

Suppose \( g_1(u_1), g_2(u_1, u_2), g_3(u_1, u_2, u_3) \) are polynomials, and \( D_1(u_1), D_2(u_1, u_2), D_3(u_1, u_2, u_3) \) are nonempty compact semi-algebraic sets. Then there exist \( c_1, c_2, c_3 \in \mathbb{R} \) s.t.

\[
\exists u_1. (D_1 \land g_1 \leq z) \iff z \geq c_1, \quad (1)
\]

\[
\forall u_2. (\exists u_1. D_2 \implies \exists u_1. (D_2 \land g_2 \leq z)) \iff z \geq c_2, \quad (2)
\]

\[
\exists u_3. ((\exists u_1, u_2, D_3) \land \forall u_2. (\exists u_1, D_3 \implies \exists u_1. (D_3 \land g_3 \leq z))) \iff z \geq c_3, \quad (3)
\]

where \( \triangleright \in \{>, \geq\} \), and \( c_1, c_2, c_3 \) satisfy

\[
c_1 = \min_{u_1} g_1(u_1) \quad \text{over } D_1(u_1), \quad (4)
\]

\[
c_2 = \sup_{u_2} \min_{u_1} g_2(u_1, u_2) \quad \text{over } D_2(u_1, u_2), \quad (5)
\]

\[
c_3 = \inf_{u_3} \sup_{u_2} \min_{u_1} g_3(u_1, u_2, u_3) \quad \text{over } D_3(u_1, u_2, u_3). \quad (6)
\]
Our Approach – Step 3

Eliminate quantifiers in $Qu.\varphi(u, z)$ and from the result we can retrieve the optimal value and the corresponding optimal controller $u$. 

Outline

1. The Oil Pump Control Problem
2. Motivation
3. The Overall Approach
4. Solving the Oil Pump Problem
   - Step 1
   - Step 2
   - Step 3
   - Results
5. Conclusions
Localize the Controller

- $0 \leq t_1 \leq t_2 \leq \cdots \leq t_i \leq t_{i+1} \leq \cdots$
- Employing the periodicity
- Stable interval $[L, U] \subseteq [V_{\text{min}}, V_{\text{max}}]$

Repeated Cycles
The Oil Pump Control Problem

Motivation

The Overall Approach

Solving the Oil Pump Problem

- Step 1
- Step 2
- Step 3
- Results

Conclusions

Modeling Oil Consumption

<table>
<thead>
<tr>
<th>time</th>
<th>[2,4]</th>
<th>[8,10]</th>
<th>[10,12]</th>
<th>[14,16]</th>
<th>[16,18]</th>
</tr>
</thead>
<tbody>
<tr>
<td>rate</td>
<td>1.2</td>
<td>1.2</td>
<td>2.5</td>
<td>1.7</td>
<td>0.5</td>
</tr>
</tbody>
</table>

- fluctuation of consumption rate: $f = 0.1$
The Oil Pump Control Problem

Motivation

The Overall Approach

Solving the Oil Pump Problem

Conclusions

Step 1

Step 2

Step 3

Results

Modeling Oil Consumption

<table>
<thead>
<tr>
<th>time</th>
<th>[2,4]</th>
<th>[8,10]</th>
<th>[10,12]</th>
<th>[14,16]</th>
<th>[16,18]</th>
</tr>
</thead>
<tbody>
<tr>
<td>rate</td>
<td>1.2</td>
<td>1.2</td>
<td>2.5</td>
<td>1.7</td>
<td>0.5</td>
</tr>
</tbody>
</table>

- fluctuation of consumption rate: \( f = 0.1 \)

\[
\begin{align*}
(0 \leq t \leq 2) & \implies V_{out} = 0 \\
\land (2 \leq t \leq 4) & \implies 1.1(t-2) \leq V_{out} \leq 1.3(t-2) \\
\land (4 \leq t \leq 8) & \implies 2.2 \leq V_{out} \leq 2.6 \\
\land (8 \leq t \leq 10) & \implies 2.2 + 1.1(t-8) \leq V_{out} \leq 2.6 + 1.3(t-8) \\
\land (10 \leq t \leq 12) & \implies 4.4 + 2.4(t-10) \leq V_{out} \leq 5.2 + 2.6(t-10) \\
\land (12 \leq t \leq 14) & \implies 9.2 \leq V_{out} \leq 10.4 \\
\land (14 \leq t \leq 16) & \implies 9.2 + 1.6(t-14) \leq V_{out} \leq 10.4 + 1.8(t-14) \\
\land (16 \leq t \leq 18) & \implies 12.4 + 0.4(t-16) \leq V_{out} \leq 14 + 0.6(t-16) \\
\land (18 \leq t \leq 20) & \implies 13.2 \leq V_{out} \leq 15.2 \\
\end{align*}
\]

Modeling the Pump

- We will first assume that the pump is activated at most twice in one cycle: \( t_1, t_2, t_3, t_4 \)
- \( t_{i+1} - t_i \geq 2 \):

\[
\begin{align*}
C_2 \equiv & \quad (t_1 \geq 2 \land t_2 - t_1 \geq 2 \land t_3 - t_2 \geq 2 \land t_4 - t_3 \geq 2 \land t_4 \leq 20) \\
\lor & \quad (t_1 = 20 \land t_2 = 20 \land t_3 = 20 \land t_4 = 20) \\
\lor & \quad (t_1 = 20 \land t_2 = 20 \land t_3 = 20 \land t_4 = 20)
\end{align*}
\]

- \( 2.2 l/s \)

\[
\begin{align*}
(0 \leq t \leq t_1) & \implies V_{in} = 0 \\
\land (t_1 \leq t \leq t_2) & \implies V_{in} = 2.2(t-t_1) \\
\land (t_2 \leq t \leq t_3) & \implies V_{in} = 2.2(t_2-t_1) \\
\land (t_3 \leq t \leq t_4) & \implies V_{in} = 2.2(t_3-t_1) + 2.2(t-t_3) \\
\land (t_4 \leq t \leq 20) & \implies V_{in} = 2.2(t_2 + t_4 - t_1 - t_3)
\end{align*}
\]
Encoding Safety Requirements

- Oil volume in the accumulator:
  \[ C_4 \triangleq v = v_0 + V_{in} - V_{out} \]

- Safety and inductiveness (considering robustness):
  \[
  C_5 \triangleq t = 20 \rightarrow L + 0.2 \leq v \leq U - 0.2 \\
  C_6 \triangleq 0 \leq t \leq 20 \rightarrow V_{min} + 0.2 \leq v \leq V_{max} - 0.2
  \]

\[ S \triangleq \forall t, v, V_{in}, V_{out}. (C_1 \land C_3 \land C_4 \rightarrow C_5 \land C_6). \]

- \( C_1 \): oil consumed
- \( C_3 \): oil pumped
- \( C_4 \): oil in the accumulator
- \( C_5 \): (local) safety
- \( C_6 \): inductiveness

\[ C_7 \triangleq \forall v_0. \left( C_7 \rightarrow \exists t_1 t_2 t_3 t_4. (C_2 \land S) \right). \]

- \( C_7 \): \( L \leq v_0 \leq U \)
- \( C_2 \): 2-second latency
Deriving Constraints

Applying QE to

\[ C_8 \equiv \forall v_0. \left( C_7 \rightarrow \exists t_1t_2t_3. \left( C_2 \land S \right) \right), \]

we get

\[ C_9 \equiv L \geq 5.1 \land U \leq 24.9 \land U - L \geq 2.4. \]

Deriving Constraints (Contd.)

\[ C_{10} \equiv C_2 \land C_7 \land C_9 \land S. \]

- \( C_2 \): 2-second latency
- \( C_7 \): \( L \leq v_0 \leq U \)
- \( C_9 \): constraint on \( L, U \)
- \( S \): safety and inductiveness

After QE:

\[ D(L, U, v_0, t_1, t_2, t_3, t_4) \equiv \bigvee_{i=1}^{92} D_i \]
The Oil Pump Control Problem

Motivation

The Overall Approach

Solving the Oil Pump Problem

- Step 1
- Step 2
- Step 3
- Results

Conclusions

Optimization Criterion

\( R_{o}(optimality): \) minimize the average accumulated oil volume in the limit, i.e. minimize

\[
\lim_{T \to \infty} \frac{1}{T} \int_{t=0}^{T} v(t) dt
\]
The Oil Pump Control Problem

Motivation

The Overall Approach

Solving the Oil Pump Problem

Conclusions

Step 1

Step 2

Step 3

Results

Optimization Criterion (Contd.)

- $R'_0 : \min_{[L, U]} \max_{v_0 \in [L, U]} \min_{t} \frac{1}{20} \int_{t=0}^{20} \nu(t) dt$.

Encoding Optimization Criteria

Lemma

Suppose $g_1(u_1)$, $g_2(u_1, u_2)$, $g_3(u_1, u_2, u_3)$ are polynomials, and $D_1(u_1)$, $D_2(u_1, u_2)$, $D_3(u_1, u_2, u_3)$ are nonempty compact semi-algebraic sets. Then there exist $c_1$, $c_2$, $c_3 \in \mathbb{R}$ s.t.

\begin{align*}
&\exists u_1. (D_1 \land g_1 \leq z) \iff z \geq c_1, \\
&\forall u_2. (\exists u_1. D_2 \rightarrow \exists u_1. (D_2 \land g_2 \leq z)) \iff z \geq c_2, \\
&\exists u_3. ((\exists u_1, u_2. D_3) \land \forall u_2. (\exists u_1. D_3 \rightarrow \exists u_1. (D_3 \land g_3 \leq z))) \iff z \geq c_3,
\end{align*}

where $\triangleright \in \{>, \geq\}$, and $c_1$, $c_2$, $c_3$ satisfy

\begin{align*}
c_1 &= \min_{u_1} g_1(u_1) \quad \text{over } D_1(u_1), \\
c_2 &= \sup_{u_2} \min_{u_1} g_2(u_1, u_2) \quad \text{over } D_2(u_1, u_2), \\
c_3 &= \inf_{u_3} \sup_{u_2} \min_{u_1} g_3(u_1, u_2, u_3) \quad \text{over } D_3(u_1, u_2, u_3).
\end{align*}
Encoding the Optimization Criterion

Cost function:

\[
g(v_0, t_1, t_2, t_3, t_4) \equiv \frac{1}{20} \int_{t=0}^{20} v(t) \, dt = \frac{20v_0 + 1}{20} (t_2^2 - t_3^2 - t_3^3 - 40t_3 - 40t_4 - 40t_3 - 40t_4 - 132.2)
\]

\( R'_o \) can be encoded into

\[\exists L, U. (C_9 \land \forall v_0. (C_7 \rightarrow \exists t_1 t_2 t_3 t_4. (D \land g \leq z))) ,\]

which is equivalent to \( z \geq z^* \) or \( z > z^* \)

Outline

1. The Oil Pump Control Problem
2. Motivation
3. The Overall Approach
4. Solving the Oil Pump Problem
   - Step 1
   - Step 2
   - Step 3
   - Results
5. Conclusions
Performing QE

\[ \exists L, U, (C_9 \land \forall v_0, (C_7 \rightarrow \exists t_1 t_2 t_3 t_4, (D \land g \leq z))) \]

- the inner \( \exists \): quadratic programming
- the outer \( \exists \): discretization

\[ L \geq 5.1 \land U \leq 24.9 \land U - L \geq 2.4 \]

- the middle \( \forall \): divide and conquer

Outline

1. The Oil Pump Control Problem
2. Motivation
3. The Overall Approach
4. Solving the Oil Pump Problem
   - Step 1
   - Step 2
   - Step 3
   - Results
5. Conclusions
Optimal Controllers with 2 Activations

- In [Cassez et al hscc09], the optimal value 7.95 is obtained at interval [5.1, 8.3]
- Using our approach, the optimal value is 7.53 (a 5% improvement) and the corresponding interval is [5.1, 7.5]
- Comparison of local optimal controllers: (the left one comes from [Cassez et al hscc09])

Local Optimal Controllers — 2 Activations

\[ t_1 = \frac{10v_0 - 25}{13} \land t_2 = \frac{10v_0 + 1}{13} \land t_3 = \frac{10v_0 + 153}{22} \land t_4 = \frac{157}{11} \]
Improvement by Increasing Activations

- The pump is allowed to be switched on at most 3 times in one cycle.
- The optimal average accumulated oil volume 7.35 (a 7.5% improvement) is obtained at interval [5.2, 8.1].
- The local optimal controllers corresponding to \( v_0 \in [5.2, 8.1] \):

\[
\begin{align*}
t_1 &= \frac{10v_0 - 26}{13} & t_2 &= \frac{10v_0}{13} & t_3 &= \frac{5v_0 + 76}{11} & t_4 &= 12 & t_5 &= 14 & t_6 &= 359/22 & v_0 \in [5.2, 6.8] \\
t_1 &= \frac{10v_0 - 26}{13} & t_2 &= \frac{10v_0}{13} & t_3 &= \frac{5v_0 + 76}{11} & t_4 &= \frac{5v_0 + 98}{11} & t_5 &= \frac{5v_0 + 92}{9} & t_6 &= \frac{20v_0 + 3095}{198} & v_0 \in [6.8, 7.5] \\
t_1 &= \frac{10v_0 - 26}{13} & t_2 &= \frac{10v_0}{13} & t_3 &= \frac{5v_0 + 76}{11} & t_4 &= \frac{5v_0 + 98}{9} & t_5 &= \frac{5v_0 + 92}{9} & t_6 &= \frac{20v_0 + 3095}{198} & v_0 \in [7.5, 7.8] \\
t_1 &= \frac{10v_0 - 26}{13} & t_2 &= \frac{10v_0}{13} & t_3 &= \frac{45v_0 + 1300}{143} & t_4 &= 14 & t_5 &= \frac{359}{22} & t_6 &= 20 & v_0 \in [7.8, 8.1]
\end{align*}
\]
Simulation of Our Controllers
The Oil Pump Control Problem

Motivation

The Overall Approach

Solving the Oil Pump Problem

Conclusions

Step 1

Step 2

Step 3

Results

Simulation (Contd.)

<table>
<thead>
<tr>
<th>Controller</th>
<th>Bang-Bang</th>
<th>Smart</th>
<th>Uppaal-Tiga</th>
<th>2-actn</th>
<th>3-actn</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial (l)</td>
<td>8.3</td>
<td>8.3</td>
<td>8.3</td>
<td>6.5</td>
<td>8.0</td>
</tr>
<tr>
<td>Duration (s)</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>200</td>
</tr>
<tr>
<td>Mean Vol (l)</td>
<td>13.45</td>
<td>11.16</td>
<td>7.44</td>
<td>7.11</td>
<td>6.77</td>
</tr>
</tbody>
</table>

- **Initial**: oil volume at the beginning of simulation
- **Duration**: run time of simulation (20 cycles, 200s)
- **Mean Vol**: the average accumulated oil volume over 200s

Tools
The Oil Pump Control Problem

Motivation

The Overall Approach

Solving the Oil Pump Problem

Conclusions

Step 1

Step 2

Step 3

Results

Tools: Mjollnir

http://www-verimag.imag.fr/~monniaux/mjollnir.html

http://redlog.dolzmann.de/

All experiments are done on a desktop running Linux with a 2.66 GHz CPU and 3 GB memory

<table>
<thead>
<tr>
<th>formula</th>
<th>C8</th>
<th>C10</th>
<th>$\theta_{kkt}$ (all 92)</th>
<th>all the rest</th>
</tr>
</thead>
<tbody>
<tr>
<td>tool</td>
<td>Mjollnir</td>
<td>Mjollnir</td>
<td>Mjollnir</td>
<td>Redlog/Reduce</td>
</tr>
<tr>
<td>time</td>
<td>8m8s</td>
<td>4m13s</td>
<td>31s</td>
<td>&lt;1s *</td>
</tr>
</tbody>
</table>

*All 15400 iterations will cost more than 10 hours using a single process
Modeling, verification, synthesis and optimization of controllers for hybrid systems are integrated into one framework based on first-order polynomial formulas and real quantifier elimination.

Judicious interplay of:
- Symbolic computation: QE and formula simplification
- Numerical computation: discretization in an optimization fashion

Avoid bad design of controllers and obtain better optimal values.
Thanks!

Questions?
Part II

BeChong Spring Meeting

Bergen, Norway


Organisers: Assoc. Prof. Volker Stolz (HVL), Prof. Zhiming LIU (RISE, Southwest University (SWU), Chongqing, China)

Location: HVL, Campus Kronstad

Supported by the SIU project “Methods and Tool Support for Refinement, Model Transformation and Verification of Network Systems”.

Guest speakers:
- Prof. Martin Leucker (Univ. Lübeck, Germany)

Further lecturers:
- Dr. Svetlana Jaksic (HVL)
- Prof. Lars Michael Kristensen (HVL)
- Prof. Yngve Lamo (HVL/HUS)
- Dr. Dan Li (HVL/Guizhou Academy of Sciences)
- Dr. Bo Liu (SWU)
- Prof. Zhiming Liu (RISE/SWU)
- Assoc. Prof. Violet Ka I Pun (HVL/UIO)
- Assoc. Prof. Adrian Rutle (HVL)
- Assoc. Prof. Volker Stolz (HVL)
- Dr. Xia Zeng (SWU)
- Dr. Hengjun Zhao (SWU)

Agenda:

<table>
<thead>
<tr>
<th>Time</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>10:00-10:15</td>
<td>Welcome by Zhiming Liu and Volker Stolz</td>
</tr>
<tr>
<td>10:15-11:00</td>
<td>Relation method of programming: a relational semantics that unifies</td>
</tr>
<tr>
<td></td>
<td>Hoare Logic, Predicate Transformer and UTP</td>
</tr>
<tr>
<td></td>
<td>Zhiming Liu (SWU)</td>
</tr>
<tr>
<td>11:00-11:30</td>
<td>Coffee break</td>
</tr>
<tr>
<td>11:30-12:15</td>
<td>Hardware-assisted data race detection with COEMS</td>
</tr>
<tr>
<td></td>
<td>Svetlana Jaksic (HVL)</td>
</tr>
<tr>
<td>12:15-14:00</td>
<td>Lunch</td>
</tr>
<tr>
<td>14:00-14:45</td>
<td>Linking Theories of Modelling for Intelligent Cyber-Physical Systems</td>
</tr>
<tr>
<td></td>
<td>Zhiming Liu, Centre for Research and Innovation in Software Engineering (RISE)</td>
</tr>
<tr>
<td>14:45-15:30</td>
<td>Initial Modelling and validation of the JAliEn Grid middleware</td>
</tr>
<tr>
<td></td>
<td>Maksim Melnik Storetvedt (HVL)</td>
</tr>
<tr>
<td>15:30-16:00</td>
<td>Short presentations by Master students</td>
</tr>
<tr>
<td></td>
<td>Kang Chen, Yukun Zhang (SWU)</td>
</tr>
<tr>
<td>16:00-18:00</td>
<td>Free time</td>
</tr>
<tr>
<td>18:00-</td>
<td>Social dinner</td>
</tr>
</tbody>
</table>
### Tuesday (27 Feb)

<table>
<thead>
<tr>
<th>Time</th>
<th>Session</th>
<th>Speaker</th>
</tr>
</thead>
<tbody>
<tr>
<td>09:30-10:15</td>
<td>Transforming CPN Models into Code for TinyOS: A Case Study of the RPL Protocol</td>
<td><em>Lars Michael Kristensen (HVL)</em></td>
</tr>
<tr>
<td>10:15-10:45</td>
<td>Coffee break</td>
<td></td>
</tr>
<tr>
<td>10:45-11:30</td>
<td>Modelling and validation of the MQTT protocol for machine-to-machine communication</td>
<td><em>Alejandro Rodriguez Tena (HVL)</em></td>
</tr>
<tr>
<td>11:30-12:15</td>
<td>Model-based Test-Case Generation with Coloured Petri Nets</td>
<td><em>Rui Wang (HVL)</em></td>
</tr>
<tr>
<td>12:15-14:00</td>
<td>Lunch</td>
<td></td>
</tr>
<tr>
<td>14:00-14:45</td>
<td>Basic concepts of LLVM</td>
<td><em>Dan Li (HVL)</em></td>
</tr>
<tr>
<td>14:45-15:30</td>
<td>Model-coverage for Coloured Petri Net models</td>
<td><em>Faustin Ahishakiye (HVL)</em></td>
</tr>
<tr>
<td>15:30-16:15</td>
<td>Darboux-type Barrier Certificates for Safety Verification of Nonlinear Hybrid Systems</td>
<td><em>Xia Zheng (SWU)</em></td>
</tr>
</tbody>
</table>

### Wednesday (28 Feb)

<table>
<thead>
<tr>
<th>Time</th>
<th>Session</th>
<th>Speaker</th>
</tr>
</thead>
<tbody>
<tr>
<td>09:30-10:30</td>
<td>Automatic Evolution of Transformation Rules</td>
<td><em>Adrian Rutle (HVL)</em></td>
</tr>
<tr>
<td>10:30-10:45</td>
<td>Coffee break</td>
<td></td>
</tr>
<tr>
<td>10:45-11:30</td>
<td>Multilevel-Modelling and Transformations with MultEcore; Rearchitecting Models</td>
<td><em>Fernando Macías (HVL)</em></td>
</tr>
<tr>
<td>11:30-12:15</td>
<td>Modelling and abstraction of the Machine Learning Domain</td>
<td><em>Angela Barriga Rodriguez (HVL)</em></td>
</tr>
<tr>
<td>12:15-14:00</td>
<td>Lunch</td>
<td></td>
</tr>
<tr>
<td>14:00-14:45</td>
<td>API/Client Co-Evolution</td>
<td><em>Anna M. Eilertsen (UIB)</em></td>
</tr>
<tr>
<td>14:45-15:30</td>
<td>Invariant Generation and Safety Verification for Hybrid Systems</td>
<td><em>Hengjun Zhao (SWU)</em></td>
</tr>
<tr>
<td>15:30-15:45</td>
<td>Coffee break</td>
<td></td>
</tr>
<tr>
<td>15:45-16:30</td>
<td>A Formal Model of Parallel Execution on Multicore Architectures with Multilevel Caches</td>
<td><em>Violet Ka I Pun (HVL)</em></td>
</tr>
<tr>
<td>16:30-17:00</td>
<td>Short presentations by Master students</td>
<td><em>Roy Nesbø (HVL), Xiao Qin (SWU)</em></td>
</tr>
<tr>
<td>Time</td>
<td>Event</td>
<td>Speaker/Presenter</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------------------------------------------------------------</td>
<td>----------------------------------------</td>
</tr>
<tr>
<td>Thursday (1 Mar)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>09:30-10:30</td>
<td>Specification Languages</td>
<td>Martin Leucker (U. Lübeck)</td>
</tr>
<tr>
<td>10:30-10:45</td>
<td>Coffee break</td>
<td></td>
</tr>
<tr>
<td>10:45-11:30</td>
<td>INTROMAT</td>
<td>Yngve Lamo (HVL)</td>
</tr>
<tr>
<td>11:30-12:15</td>
<td>The role of Ontology in Data Mining</td>
<td>Fazle Rabbi (HVL)</td>
</tr>
<tr>
<td>12:15-14:00</td>
<td>Lunch</td>
<td></td>
</tr>
<tr>
<td>Friday (2 Mar)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>09:30-10:30</td>
<td>Temporal Logics</td>
<td>Volker Stolz (HVL)</td>
</tr>
<tr>
<td>10:30-10:45</td>
<td>Coffee break</td>
<td></td>
</tr>
<tr>
<td>10:45-11:30</td>
<td>QoS-aware Service Composition and Recommendation</td>
<td>Bo Liu (SWU)</td>
</tr>
<tr>
<td>11:30-12:15</td>
<td>Multimodel Correspondence through Inter-Model Constraints</td>
<td>Patrick Stünkel (HVL)</td>
</tr>
<tr>
<td>12:15-12:30</td>
<td>Closing</td>
<td>Zhiming Liu, Volker Stolz</td>
</tr>
<tr>
<td>12:30-14:00</td>
<td>Lunch</td>
<td></td>
</tr>
</tbody>
</table>
Lecture 12

Relational Semantics and Unifying Logics of Programming

by Zhiming Liu
Relational Semantics and Unifying Logics of Programming

Zhiming Liu
Centre for Research and Innovation in Software Engineering
Southwest University, Chongqing China
Email: zhimingliu88@swu.edu.cn

@BeChong Spring Meeting, Bergen, Norway

February 26, 2018

Outline

- Aims and Themes
- Formal Methods
- Relational Semantics
- UTP
- Relations to Hoare Logic and WP Transformer
The Goals & Themes

- **Goals**
  1. Establish a good theoretical understanding of how to describe what programs do, how they do it, and why they work
  2. Learn how to apply this understanding to the construction and analysis of programs
  3. Focus on correct by design – linking to software engineering
  4. Links of Theories of Modelling of Software and Systems

- **Themes**
  1. Basic concepts and techniques
  2. Relational model
  3. The unifying views of different theories

Formal Method in General

- How do you ensure that your program is *correct*?
- Why is testing not enough?

  *Testing can only show the presence of a bug in a program, but NOT the absence of bugs in the program.*

  *E.W. Dijkstra*
What is Formal Method About?

- Established in 1970’s (R.W. Floyd, E.W. Dijkstra, C.A.R. Hoare - all Turing Award winners)
- About disciplines of programming to ensure the correctness of a program
  - Specification and abstraction
    - How to describe what program do, or
    - Define the correctness criteria of a program
  - Semantics
    - How do they do it: semantics of programs
  - Verification and Analysis
    - Prove the correctness criteria of a program, i.e. verify why it does it
  - Refinement, Synthesis
    - Develop a program to ensure the correctness

What Should a FM Provide?

- A mathematical model of programs – formal semantics, formal specification
- A systematic way of construction, manipulation and modification of program specifications/models – refinement, formal synthesis
- A systematic way to compare programs and specifications/models – formal verification, formal analysis
- Tools for specification and verification, and this needs formalisation
Expected Features of a Formal Method

- Information hiding
- Compositional analysis, design, reasoning, verification & validation
- Algorithmic techniques for analysis and verification - automatic proving and verification

The Key Issues:
- Abstraction, Refinement and Verification
- Incremental application of formalisation

Programs, Computers and Computations

- Computer carries out a computation by executing a program
- A program consists of a number of variables and a sequence of commands
- At the beginning of the execution, the computer receives an input
- During the execution, the commands are carried out according to the flow of control defined by the program
- At the end of the execution, an output is produced.
- The same program can be executed for different inputs and produce different outputs.
Input and Output

- The way in which input and output take place often differs from one computer to another:
  - Different computers are mostly common in what goes on internally
  - It is during the period from the end of the input to the beginning of the output that the real process of computation takes place
  - At the end of the execution, the final values of the variables are converted for output.

States

- Before the execution (after compilation), each program variable is allocated an address of memory to remember the data of that variable
- At the beginning of the execution of a program, the input sets initial values of (some of) the variables/memories
- At anytime during the execution, a command is carried out and the data of the variables are manipulated according to the meaning of the command
- We use the concept state to describe the mechanism of remember values of the variables.
- The input is to determine the initial state and at the end the final state is what must be the output
Computation as Change of State

- **Abstract** from input and output
- A computation is a change from an initial state to a final state
- We can understand an execution of a program on a computer as (effecting) a computation

Semantics of a Program

- It is the *semantics* of the program that determines for a given initial state what must be the final state ready for output
- The *semantics* of a program is the *meaning* of the program. More precisely, the *semantics* of a program is the *meaning* of the execution of the program.
- The semantics of a command should not be dependant on in which program it occurs. So we can have a systematic and generic way of defining the semantics of all programs
- The difference between the semantics of a programming language and semantics of programs written in that language is not significant
Variables

- For a program $P$, assume a finite set of program variables, denoted as $\alpha P$.
- But to study of all programs, assume an infinite set of program variables, denoted by $\text{Var}$, ranged over by $x$, $y$, $z$, etc.
- Each program variable $x$ is associated with a value space, called the type of the variable.
- So operations, such as $+$, $-$ and $\times$ on integers, can be used to form expressions.
- We do not formalise the notion of data types, but assume a universal value space $\text{Val}$.

States

- A program $P$ is a sequence of commands and the execution of each command changes the values of the variables of $P$.
- During the execution, the computer remembers the state.
- A state over a set $X \subseteq \text{Val}$ of variable, e.g. $\alpha P$, is a mapping from $X$ to $\text{Val}$

$$X = \{x_1, \ldots, x_n, \ldots\}$$

$$\sigma : X \rightarrow \text{Val}$$

$$\sigma = \{x_1 \mapsto v_1, \ldots, x_n \mapsto v_n, \ldots\}, ~ \sigma.x_i = v_i, i = 1, \ldots, n$$

- Let $\Sigma_X$ denote the set of all states over $X$.
- $\Sigma$ be the set of states over $\text{Var}$.
Compare States

- $\sigma_1 = \sigma_2$ iff $\sigma_1.x = \sigma_2.x$ for all $x \in \alpha P$
- $= \text{upto } x$

$$\sigma[x \mapsto a] \overset{\text{def}}{=} \left\{\begin{array}{ll}
\sigma(y) & y \neq x \\
\alpha & y = x
\end{array}\right. \text{ for all } y \in \alpha P$$

$$\sigma[x_1 \mapsto a_1, \ldots, x_k \mapsto a_k] \overset{\text{def}}{=} \sigma[x_1 \mapsto a_1], \ldots, [x_k \mapsto a_k]$$

- Lemma

$\sigma[x \mapsto \sigma.x] = \sigma$, $\sigma[x \mapsto a][x \mapsto b] = \sigma[x \mapsto b]$

Programming Language

$$C ::= \text{skip} | \text{chaos} | x ::= e | C; C | C < B \triangleright C | C[] | B \ast C$$

where $x \in \text{Var}$, $e$ is an expression, and $B$ is a boolean expression
Relational Semantics of Programs

- Undefined state
  - Let \( \bot \) represent the undefined value, e.g. \( x/0 \)
  - Let \( \bot \) represent the state in which a variable \( x \) takes the undefined value \( \bot \)
  - Let \( \Sigma^\bot = \Sigma \cup \{\bot\} \), and \((\Sigma^\bot, \leq)\) be the partial order such that
    \[
    \sigma_1 \leq \sigma_2 \iff \sigma_1 = \sigma_2 \lor \sigma_1 = \bot
    \]

- Semantics of Programs: \([C] \subseteq \Sigma^\bot \times \Sigma^\bot\)
  - \([\text{skip}] \) \(\overset{\text{def}}{=} \{(\sigma, \sigma) \mid \sigma \in \Sigma^\bot\} \cup \{\bot, \sigma) \mid \sigma \in \Sigma^\bot\}\)
  - \([\text{chaos}] \) \(\overset{\text{def}}{=} \Sigma^\bot \times \Sigma^\bot\)
  - \([x := e] \) \(\overset{\text{def}}{=} \{(\sigma, \sigma[x \mapsto \sigma.e]) \mid \sigma \in \Sigma^\bot\} \cup \{\bot, \sigma) \mid \sigma \in \Sigma^\bot\}\)
  - \([C_1; C_2] \) \(\overset{\text{def}}{=} [C_1] \cdot [C_2]\)
  - \([C_1 \langle B \rangle C_2] \) \(\overset{\text{def}}{=} \Sigma^\bot.B \cap [C_1] \cup \Sigma^\bot.\neg B \cap [C_2]\)
  - \([C_1 \parallel C_2] \) \(\overset{\text{def}}{=} [C_1] \cup [C_2]\)
  - \([B; C] \) is the least fixedpoint of
    \[
    X = (C; X) \langle B \rangle \text{skip}
    \]

The Relational CPO

The existence of a fixed point and the unique definition of the semantic of the loop is based on the theory of the CPO below

- \(\mathcal{D} \) \(\overset{\text{def}}{=} \{r \mid r \subseteq \Sigma^\bot \times \Sigma^\bot\}, \subseteq\) is a CPO with regards to program constructors below
  - \( : \mathcal{D} \times \mathcal{D} \rightarrow \mathcal{D}\)
  - \(\langle B \rangle : (\mathcal{D} \times \mathcal{D}) \times (\mathcal{D} \times \mathcal{D}) \rightarrow \mathcal{D}\), for any boolean expression \(B\)
  - \(\parallel : (\mathcal{D} \times \mathcal{D}) \times (\mathcal{D} \times \mathcal{D}) \rightarrow \mathcal{D}\)
  - \(B; C : \mathcal{D} \times \mathcal{D} \rightarrow \mathcal{D}\) for any boolean expression \(B\)

- They are monotonic and continuous
Program Refinement

- Program $C_1 \sqsubseteq C_2$ if $\lbrack C_2 \rbrack \sqsubseteq \lbrack C_1 \rbrack$
- Laws of programming
  - $\text{chaos} \sqsubseteq C$
  - $C \sqsubseteq C$
  - $C_1 || C_2 \sqsubseteq C_1$ and $C_1 || C_2 \sqsubseteq C_2$
  - if $C_1 \sqsubseteq C_2$, then $\text{Context}(C_1) \sqsubseteq \text{Context}(C_2)$

Laws of Programming

Theorem
We can replace a command in a program:

1. $C \equiv C$
2. If $C_1 \equiv C_2$, then $C_2 \equiv C_1$
3. If $C_1 \equiv C_2$ and $C_2 \equiv C_3$, then $C_1 \equiv C_3$
4. skip; $C_1 \equiv C_1 \equiv C_1$; skip
5. $(C_1; C_2); C_3 \equiv C_1; (C_2; C_3)$
6. If $C_1 \equiv C_2$, then for any $C$ and $C_3$

$$C; C_1; C_3 \equiv C; C_2; C_3$$
UTP: A Relational Design Calculus

- A **design of a program** $D$ is tuple $(\alpha, P)$, where
  - $\alpha$ is a finite set of program variables, called the alphabet of $D$
  - $P$ is a predicate in the form $p \models R$ such that $p$ is a predicate over $\alpha$ and $R$ is a predicate over $\alpha$ and the primed versions $\alpha'$
  - The meaning of $D$ is defined by the implication $p \Rightarrow R$ for **partial correctness**
  - For **total correctness** the meaning is $(\text{ok} \wedge p) \Rightarrow R \wedge \text{ok}'$
  - $p$ is called the **precondition** and $R$ is called the **postcondition**.
- Framed design $\beta : p \models R \overset{\text{def}}{=} p \models R \wedge y \notin \beta (y' = y)$

Programs as Designs

<table>
<thead>
<tr>
<th>command: c</th>
<th>design: [c]</th>
</tr>
</thead>
<tbody>
<tr>
<td>skip</td>
<td>${}$ : true $\models true$</td>
</tr>
<tr>
<td>chaos</td>
<td>${}$ : false $\models true$</td>
</tr>
<tr>
<td>$x := e$</td>
<td>${x}$ : true $\models x' = e$</td>
</tr>
<tr>
<td>$C_1 ; C_2$</td>
<td>$\exists m.(C_1[m/x'] \wedge C_2[m/x])$</td>
</tr>
<tr>
<td>$C_1 \ll B \gg C_2$</td>
<td>$B \wedge C_1 \wedge \neg B \wedge C_2$</td>
</tr>
<tr>
<td>$C_1 [;] C_2$</td>
<td>$C_1 \wedge C_2$</td>
</tr>
<tr>
<td>$B * C$</td>
<td>$X = (C ; B* ; X) \ll B \gg \text{skip}$</td>
</tr>
</tbody>
</table>

- **Execution semantics** $[D] : \Sigma^T \times \Sigma^T \longmapsto \{\text{true}, \text{false}\}$
- **Refinement as Implication** $D_1 \sqsubseteq D_2$ if $D_2 \Rightarrow D_1$
- **All laws discussed earlier still hold**
Linking to Hoare Logic and Wp Calculus

- \{pre\}p \vdash R\{post\} \overset{def}{=} (pre \land p) \Rightarrow (R \land post')
- wp(p \vdash R, q) \overset{def}{=} (p \land \neg(R; \neg q))
- All proofs in Hoare Logic and Wp calculus are valid with these embeddings

Axioms and Rules Hoare Logic

- The axioms of integers
- \{R[e/x]\}x := e\{R\}
- Inference rules
  - The rules of the integer logic, and
  - Rules that governing program properties

(Sequential Composition) \quad \frac{(Q)S_1\{Q_1\};\{Q_1\}S_2\{R\}}{(Q)S_1S_2\{R\}}

(Conditional Choice) \quad \frac{(B \land Q)S_1\{R\};\neg(B \land Q)S_2\{R\}}{(Q)S_1\neg(B)\lor S_2\{R\}}

(Loop) \quad \frac{I \land B \vdash S\{I\}}{I \land B \vdash S\{\neg(B \land I)\}}

(Pre- and Postcondition) \quad \frac{Q \rightarrow Q_1;\{Q_1\}S(R_1)_1;\quad R_1 \rightarrow R}{\{Q\}S\{R\}}
Example: SWAP1

\[ \begin{align*} x, y, z &: \text{ int;} \\
x &:= x + y; y := x - y; x := x - y \\
\{(P : y = X \land x = Y \land z = Z)\} \\
\{P : y = X \land (x + y) - y = Y \land z = Z\} \\
x &:= x + y; \\
\{Q_1 : y = X \land x - y = Y \land z = Z\} \\
\{Q_1 : x - (x - y) = X \land x - y = Y \land z = Z\} \\
y &:= x - y; \\
\{Q_0 : x - y = X \land y = Y \land z = Z}\}; \\
x &:= x - y \\
\{x = X \land y = Y \land z = Z\} \end{align*} \]

Example Program SWAP2

\[ \begin{align*} x, y, z &: \text{ int;} \\
\{y = X \land x = Y\} \\
z &:= x; \\
\{y = X \land z = Y \land z = Z\} \\
x &:= y; \\
\{x = X \land z = Y \land z = Z\} \\
y &:= z \\
\{x = X \land y = Y \land z = Z\} \end{align*} \]

Compare SWAP1 and SWAP2: SWAP1 \(\equiv\) SWAP2
Further Development

- A relational model for object-oriented designs
- Contract-based model of component-based design (rCOS)
- A relational model of concurrent programs and the temporal logic of actions
- Real-time systems
- Fault-tolerance
- ..........

Sir Professor Tony Hoare, Professor He Jifeng, and Professor Zhiming Liu

http://en.wikipedia.org/wiki/Tony_Hoare
http://en.wikipedia.org/wiki/He_Jifeng
Lecture 13

Hardware-assisted Data race Detection
with COEMS

by Svetlana Jakšić
Hardware-assisted data race detection with COEMS

BeChong meeting, Bergen, 26.02.2018

Svetlana Jakšić, Dan Li, Volker Stolz

Project overview

Hardware-assisted tracing/monitoring:

• Increase test efficiency
  Automated online observation of embedded processors without time limitation and without intrusiveness simplifying the overall test process

• Increase debug efficiency
  New verification methodology automatically controls a multitude of constraints – a plurality of elusive, non-deterministically occurring failures can be detected.

• Increase test effectiveness
  Efficient combination of functional tests, structural tests (both data and control flow) and runtime verification

• Improve embedded systems performance
  Detailed performance data and execution times, supporting resource optimization (power consumption, runtime, processing performance, deployment)
Hardware Platform

- **Hardware**
  - Virtex-7 series FPGA (available)
  - Zynq Ultrascale+ SOC (under development)
  - RLDRAM3 memory for fast lookup tables
  - Interface to Aurora (Nexus, HSSTP)
  - VPX / FMC form factor

- **Functionality**
  - Online trace data processing (Coresight trace data -> event stream)
  - Supported architectures: ARM Cortex-A9, ARM Cortex-A53*, QorIQ PPC*, Infineon Aurix*
  - Online processing of event stream

*under development

System Overview

- **DuT (ARM Cortex-A9 / Cortex-A53)**
- **Hardware Platform (FPGA)**
- **Event Stream Analysis (via Monitoring)**
- **COEMS Elements**
- **Trace Data Pre-processing and Control Flow Reconstruction**
- **Observation Configuration**
- **Monitor Configuration**
- **TeSSLa Compiler**
- **Monitor Specification (in TeSSLa language)**
- **Front End**
- **C-Compiler**
- **Object code, debug symbols**
- **C-Code**
- **Observation Specification**
- **Report**
- **Trace Buffer and Concentrator**
- **Trace Port**
- **C-Compiler Analyser**
- **TeSSLa Compiler**
- **Monitor Configuration**
Applications

- Finding Data Races
- Finding Timing Bugs
- Finding Functional Bugs

Measuring Coverage
- Execution Coverage
- Branch Coverage
- MC/DC Coverage

Measurement of Worst-Case Execution Time and Worst-Case Response Time

Requirements from the industry partners

- Detection and optimization of shared resources
- Waiting time for a lock
- Performance analysis on functions – hot spots
- Bus and memory usage – bottlenecks
- Inter-thread communication
- Detection and optimization of thread migration and data exchange
- Lower cost of logging/instrumentation than printf
- Logging write accesses to selected variables
- Execution, branch and MC/DC coverage
Example – data race, no locks

Data races in multi-threaded programs:

- two or more threads access the same memory location concurrently,
- at least one of the accesses is a write,
- the threads are not using any exclusive locks to control their accesses to that memory.

```c
#include <stdio.h>
#include <pthread.h>

int x = 0;

void* count(void *arg) {
    for (int i = 0; i < 100; i++) {
        x++;// unprotected!
    }
    return NULL;
}

int main() {
    pthread_t p1, p2;
    pthread_create(&p1, NULL, count, NULL);
    pthread_create(&p2, NULL, count, NULL);
    // fight!
    pthread_join(p1, NULL);
    pthread_join(p2, NULL);
    printf("Counted %d \n", x);
    return 0;
}
```
Example – data race, no locks

dataRace(
    shared variables = \{x\},
    number of threads = 2
)

def two_threads_read := T1\_reads\_x == () \&\& T2\_reads\_x == ()
def a\_thread\_writes := merge(T1\_writes\_x,T2\_writes\_x) == ()
def dataRace := if two\_threads\_read \&\& a\_thread\_writes then ()
out dataRace

def dataRace\_in\_line := if \{time(dataRace)==time(line)\} then line
out dataRace\_in\_line

def time\_read\_x := time(merge(T1\_reads\_x,T2\_reads\_x))
def time\_write\_x := time(merge(T1\_writes\_x,T2\_writes\_x))
def badInterleave := if last(time\_read\_x,time\_read\_x) > default(time\_write\_x,0) then ()
out badInterleave
Example – data race, no locks

```
1674914046544: instruction = "load"
1674914046544: function = "count"
1674914046544: line = 8
1674914046544: column = 10
1674914046544: threadid = 140592004323072
1674914046544: readvar = "x"
1674914046544: readvaraddr = 6299740
1674914048890: instruction = "add"
1674914048890: function = "count"
1674914048890: line = 8
1674914048890: column = 10
1674914048890: threadid = 140592004323072
1674914050773: instruction = "store"
1674914050773: function = "count"
1674914050773: line = 8
1674914050773: column = 10
1674914050773: threadid = 140592004323072
1674914050773: writevar = "x"
1674914050773: writevaraddr = 6299740
```

Example – data race, no locks

```
1749594547010: T1_reads_x = ()
1749594550976: T1_writes_x = ()
1749594571346: T1_reads_x = ()
1749594575683: T1_writes_x = ()
1749595268897: T2_reads_x = ()
1749595268897: dataRace_in_line = 8
1749595273268: T2_writes_x = ()
```
Example – data race, with locks

```c
#include <stdio.h>
#include <pthread.h>

pthread_mutex_t m;
int x=0;
pthread_mutex_t l;
int y=0;

void* f(void *arg) {
    for ( int i = 0; i < 10; i++ ) {
        pthread_mutex_lock(&m);
        x++;
        pthread_mutex_unlock(&m);
    }
    return NULL;
}

void* g(void *arg) {
    for ( int i = 0; i < 10; i++ ) {
        pthread_mutex_lock(&l);
        y++;
        x++;
        pthread_mutex_unlock(&l);
    }
    return NULL;
}

int main() {
    pthread_t p1;
    pthread_t p2;
    pthread_mutex_init(&m, NULL);
    pthread_mutex_init(&l, NULL);
    pthread_create(&p1, NULL, f, NULL);
    pthread_create(&p2, NULL, g, NULL);
    pthread_join(p1, NULL);
    pthread_join(p2, NULL);
    printf("x = %d\n", x);
    printf("y = %d\n", y);
    return 1;
}
```

Lockset algorithm (Eraser) enforces
- that every shared variable is always protected by some set of locks
- this set of locks must be held by any thread accessing the variable
Example – data race, with locks

```python
def correctLocking(
    shared variables = \{x, y\}
    total number of locks = 2
    total number of threads = 2
)

def lock_1_always_protects_1 := T1_protecting_1_with_1 \&\& T2_protecting_1_with_1
def lock_2_always_protects_1 := T1_protecting_1_with_2 \&\& T2_protecting_1_with_2

def error_1 := !(lock_1_always_protects_1 || lock_1_always_protects_1)
```

Example – data race, with locks

```
1980123194759: instruction = "call"
1980123194759: function = "g"
1980123194759: line = 21
1980123194759: column = 9
1980123194759: threadid = 140338129528576
1980123194759: functioncall = "pthread_mutex_lock"
1980123194759: mutexlock = "l"
1980123194759: mutexlockaddr = 6303872
1980123201358: instruction = "load"
1980123201358: function = "g"
1980123201358: line = 22
1980123201358: column = 10
1980123201358: threadid = 140338129528576
1980123201358: readvar = "y"
1980123201358: readvaraddr = 6303864
1980123204027: instruction = "add"
1980123204027: function = "g"
1980123204027: line = 22
1980123204027: column = 10
1980123204027: threadid = 140338129528576
```
Example – data race, with locks

.def lock_1_always_protects_1 := T1_protecting_1_with_1 && T2_protecting_1_with_1
.out lock_1_always_protects_1
.def lock_2_always_protects_1 := T1_protecting_1_with_2 && T2_protecting_1_with_2
.out lock_2_always_protects_1
.def error_1 := !(lock_1_always_protects_1 || lock_1_always_protects_1)

Workflow
### Workflow

- GCC
- .o
- .out
- .o
- Instrumentation
- .bc
- LLVM
- LLVM
- .bc'
- LLVM

### Counting threads and locks

- Counting
  - Number of threads that can be created
  - Number of locks that can be initialized
- Data flow analysis:
- Control flow graph, basic blocks, worklist, lattice, transfer function
- Never underestimate!
- Examples
- LLVM IR, C++ implementation

---

Limitations and optimizations

- TeSSLa is limited with respect to data example: which lock is taken?
- Need ITM-instructions to log additional data
- Avoid instrumentation when not necessary
  - On constant variables
  - On variables that are never written to
  - Collapsing multiple accesses into one
  - Using aliasing/escape analysis
  - Variable accesses that are clear from the assembly code?
Without Instrumentation

• We can use static analysis for lock operations
• Overapproximation of locks in *Pun/Steffen/Stolz: Effect-polymorphic behaviour inference for deadlock checking. 2016*
• Should work for both source- and object code.
• Warnings with low number of false positives

Conclusion

• Related work: static and dynamic race detectors; ThreadSanitizer, ERASER
• Static analysis-component focused on *races*
• Can be reused for other purposes (coverage, ...)
• Beyond COEMS
  • What can be done in no-hardware/no-instrumentation setting?
  • Some expertise on IntelPT now
  • Static analysis results on binary(!) carry over to both IntelPT and CoreSight
• Publications:
  • T. Scheffel *et al.*: “Rapidly Adjustable Non-Intrusive Online Monitoring for Multi-core Systems”, SBFM 2017
  • S. Jakšić *et al.*: “COEMS — open traces from the industry”, RV-CUBES 2017
Lecture 14

Linking Modelling Theories for Intelligent CPS Design

by Zhiming Liu
Linking Modelling Theories for Intelligent CPS Design

Zhiming Liu

Centre for Research & Innovation in Software Engineering (RISE)
Southwest University, Chongqing, China

zhimingliu88@swu.edu.cn

http://www.swu-rise.net.cn/zhiming.liu

Self Introduction

• CV
  > Before 1978: Personal Civilisation not started
  > 1978-1982: PLA Luoyang Institute of Foreign Languaees
  > 1982-1985: PLA Institute of Information Engineering
  > 1985-1987: MSc at IOS/CAS
  > 1988-1994: University of Warwick
  > 1994-2005: University of Leicester
  > 2002-2013: UNU-IIST
  > 2013-2015: Birmingham City University
  > 2016-: Southwest University - 西南大学 （重庆）

• Research
  3) rCOS: formal MDA for object-oriented, Service-based component systems [2000 – – –]
  4) SE+AI in Inter-Networking Systems (??)
RISE at Southwest University

RISE - Centre for Research and Innovation in Software Engineering
1. 11 Staff = 1 Pro + 3 AP + 6 Lect + 1 PostDoc
2. 8 Adjunct Professors 2 CN + 3 UK + 1 USA + 1 LU + 1 IL
3. 16 Students = 9 MSc + 3 PhD + 4 BSc
4. Research:
   - Theory & Methods for Software Engineering
   - Security
   - AI
   - Focus: software in emerging systems - networked systems of heterogeneous components (SOS)
   - Research grants: 千人计划基金 10m, NSFC 面上&2青年, NSFC 重点, 重庆ST

Points of Discussion

1. The need of a unified modelling Framework
2. The ideas/experience of rCOS modelling
3. CPS and Evolving architecture
4. Advance rCOS to CPS?
5. Trustworthy AI – some wild ideas
Modelling
Trinity of Model, Thing & Framework

- **Model:** an abstraction of a thing, but not the thing
- **Faithfulness of model:** $M|\phi$ vs $O|\phi$
- **Modelling framework:** syntax, semantics, theory, techniques and tools for
  - model creation, analysis, manipulation, and verification
- The thing are often modelled from **different viewpoints**
  - Orthogonal or but often interrelated
  - Different viewpoints maybe modelled different with different notations and paradigms
  - Models of different views of the thing need to be integrated to form a “whole” model of the thing
- **Thing can be a model too**

“You will never strike oil by drilling through the map” [Golomb 1971]

---

Model-Driven System Engineering

- **Software and Systems Engineering**
  - The thing is a target system
  - A model to represents the expectation for the system
  - Goal is that the target model satisfies model
- **Model-driven system engineering:** design, analysis, verification, integration and **maintenance** virtualization, i.e. based on models in all stages of the
- **The target system is treated as a model too**
- **Modelling framework:** syntax, semantics, theory, techniques and tools for
  - model creation, analysis, transformations, refinement and verification
Traditional Modelling Frameworks

- **Event-Based – Trace Models:**
  - Automata-based, CCS and CPS-like languages abstract data away
- **Data State-Based with an Operational semantic Model**
  - TLA, Action Systems, B, Alloy – State Transition Systems
- **Declarative and State-Based:** VDM, Z, JML (Hoare-Logic Based)
- **Combination trace-based and data-based:**
  - Value-passing CCS and CSP (limited), Occam
- **Multi-view modelling:** UML-like, Simulink, rCOS
- **Timing model for scheduling**
- **Need a unified modelling framework, but yet explicitly support separation of concerns**
  - Unified semantic theory, model composition and integration, and tool chain
  - Institutions [Bustall & Goguen, 1984], UTP [Hoare & He, 1988]

rCOS

Build **system models** to gain confidence in requirements and designs

- Use **decomposition and separation of concerns**
  - \( C = (V, p, r, \mathcal{P}, \mathcal{R}, \mathcal{P}) \)
  - \( V \) can be of object types
  - provided/required operations, pre | post
  - \( \mathcal{P} \) specifies the interaction protocol
- Use case driven **incremental development to deal with changes**
- use **rigor/formalization**
  - repeatable process
  - analyzable artifacts
  - reduce uncertainty

Basis for Tool Support
Model of Architectural Components

- **Service component**
  
  Component M1
  
  ```
  Z d;
  provided interface M1IF {
    W(Z v) { d:=v };  R(;v) { v:=d };  
  }
  ```

- **Reactive component**
  
  Component M
  
  ```
  Z d, Bool w = true;
  provided interface MIF {
    W(Z v) { w&(d:=v,w:= not w );
    R(;Z v) { not w&(v:=d; w:=not w); 
  }
  ```

- Neither refines the other – requirement change

More General Component

```
component fM{
  Z d; Bool w = true;
  provided interface MIF {
    W(Z v) { w&d:=v, w:= not w }; 
    R(;v) { not w& v:=d; w:=not w};
  }
  actions {
    //fault modelling corruption of memory
    fault {true- d’< > d } //uncertainty, even no known probability
  }
}
```

- **Renaming as a built-in connector**
  
  fMi=fM[fM.W/W,fM.R/R], i=1,2,3,

- **But can be built by composition**
  
  component fMi requires fM
  provided interface MIFi {
    fMi.W(Z v) {fM.W(v)};
    fMi.R(;Z v) {fM.R(;v)}
  }
```
Equivalent Models

- Guarded commands model can be used as programming model
- But they are difficult for third party composition or verification

Component $M$

Z d;
provided interface MIF {
  W(Z v) { d:=v }; R(;v) { v:=d }
protocol {(WR)*+(WR)*W} // ** generally traces }

Component $M$ requires $M_1$ // ** $M$ is obtained through coordinating $M_1$

Bool w = true;
provided interface MIF {
  W(Z v) { w&(M_1.W(v);w:=not w });
  R(;Z v) { not w&(M_1.R(;v);w:=not w };
}

Component $M$ requires $M_1$ // ** $M$ is obtained through coordinating $M_1$

provided interface MIF {
  W(Z v) { M_1.W(v) }; R(;Z v) { M_1.W(;v) ;
  protocol {(M_1.WM_1.R)^*(M_1.WM_1.R)^*W}
}

Orchestration

component $V$ requires $fM_1$, $fM_2$, $fM_3$

Z v1,v2,v3,
provided interface VIF {
  W(Z v) { fM_1.W(v)||fM_2.W(v)||fM_3.W(v) }
  R(;v) { fM_1.R(;v1)||fM_2.R(;v2)||fM_3.R(;v3); v=vote(v1,v2,v3) }
protocol {(WR)^*(WR)^*W} // notice one can specify different
protocols
}

• $V \subseteq M$, i.e. $V$ refines $M$, provided at most one memory is
  corrupted (handling restricted uncertainty)
• The proof need to introduce auxiliary variables
• TOPLAS 1999 [Z. Liu & M. Joseph]
Architecture Decision for Fault-Tolerance

Case Study: CoCoMe
Use Case as Initial Components

- Component Interface Sequence Diagram
- UML sequence diagram does not describe non-determinism

State Diagram/Machine
Component Class Diagram

Operation Contract: makeCashPayment(a)

• Preconditions
  1. Store exits
  2. CashDesk exits
  3. Catalog exits
  4. Sale is complete
• Postconditions
  5. a new \textit{cashpayment} is created
  6. amount of the new \textit{cashpayment} is set to a
  7. the new \textit{cashpayment} is linked to the sale

• Formalised in rCOS [TCS 2006, Liu, He & Li]
**OO Design of each Interface Operation**

E.G. `makeCashPayment()`

![Diagram of OO Design](image)

**OO Design to CB Design**

- Interactive with designer
  1. Select a set of lifelines
  2. Automatically validate 6 conditions
  3. If the checking passes, fine
  4. Otherwise go back one

- FACS [Li, Li & Liu, 2011]
Automatically Generate Component Diagram

- Incremental and iterative design: in the next cycle
  - Extend the current with use case and upgrade the models, or
  - Take on another new use case to repeat the process
  - Either will be based on models created in the previous cycle with reuse

- Long term evolution along with business growing and technology advancements

Consistency, Feature Interactions, and Traceability

CoCoMe

Store

Cash Desk Line

Cash Desk
Cash Desk
Cash Desk
Cash Desk
Cash Desk

Store Server

Store Client
CPS

- A CPS combines a **cyber side** with a **physical side**.
- **Cyber side** – computing and networking
- Physical side – mechanical, electrical, and chemical processes
- The **cyber components** control the physical side using sensors and actuators, as well as **providing services** to users
- **Sensors** and **actuators** are interfaces that observe/sense the physical system and actuate the controls
- **Human actors** – involve human-machine interaction
- **CPS include IoT, Data Centres and M2M**
Next Generation: CPS Based BMS

Solution:
- Refer to experience
- Use CPS architecture
- Unified standard management
- Employ data from multidimensional

Architecture:
- Multilevel CPS
- Dynamic evolution architecture
- Adaptive components
- Interface based interconnection
Software Paradigms in CPS

- **IoT Layer** - Embedded Software, Drivers and OS in sensors and devices
- **Computation & Coordination Layer** - big data processing and analytics, control/monitoring/coordinating software
- **Application Layer** - Apps, web/cloud services, business & workflow managements
- **Network layer** - communication protocols, network infrastructure & resources management and scheduling (SDN)
- **Involve different software architecture styles and technologies:** OOA, SOA, CBA, MDA, AI

Need seamless integration of architecture styles and technologies

AI in CPS

- **Knowledge reasoning, as well as data analytics in**
  - decision making in business processes and workflows
  - intelligent control – **MAPE-K**
  - dynamic service discovery and binding – Evolving SOA.
- **Computer vision and Natural Language Processing** in
  - human–machine interaction (HMI)
- **Robots** are obviously in CPS.
- **AI is both a solution and problem to uncertainty**

End-to-End Modelling & Verification AI Components & Composition of AI and Non-AI Components
Continuous Evolution

1. New components and plug in
2. Dynamically find and connect components
3. Adding more interfaces and/or improving performance of interface, as to allow cyber components to
   1) monitor more and better about its environment
   2) more self*
   3) make more intelligent control decisions, and provide smarter services
   4) control and coordinate more and better physical components

Evolving to improve trustworthiness, optimization, smartness, connectivity, autonomy

Evolution in Railway Systems

Dealing with uncertainty with communication and fault-tolerance
**Challenges**

- **Requirements**
  - Cope with changes and uncertainties: both during development and at runtime
  - The relation between performance and functionalities

- Handling evolving architecture with heterogeneous components of mixed criticality
  - Dynamic component plug in and play (composability)
  - Dynamic discovery and binding services and components, (evolving SOA)
  - Predictability, safety, security, robust, self-organising, self-adaptive.

- Verification of AI Components and Composition of non-learning and learning components

---

**Cyber–Physical Components**  

- Physical Interface

![Physical Interface Diagram](image)

- Cyber-Physical Component

![Cyber-Physical Component Diagram](image)
Physical Components and Interfaces

Component A /*an appliance
rate: [Time -> Real];
status: {on, off};
provided interface {
  rate ("signal: given by manufacture");
  switch() ("operation: switch A on and off")
}

Component M /*meter
val: [time -> Real];
provided interface {
  read(r){true | r = val};
}

required interface rate ("signal
val= energy(rate) // e.g. val(t) = \int rate(t) dt"
}

Composition: H = A|M
- Type equation here.

System Evolution for Home Automation (a)

- Add provided signal 'val' to M
- Add a control pad P that requires signal 'val', provides 'set()', and calls A.switch, etc.
  \( M' = P | M, H' = M' | A \)
- Refine P with planning with daily budget, and schedule functionality
  \( M'' = P' | M, H'' = M'' | A \)
System Evolution for Home Automation (b)

\[ \text{Ai} = A[\text{switch}/\text{swich}, \text{rate}/\text{rate}], \ A = A_1 || \ldots || A_m \]
\[ \text{Mi} = M[\text{read}/\text{read}, \text{val}/\text{val}], \ M = M_1 || \ldots || M_m \]
\[ P_i = P[\ldots \ldots], \ P = P_1 || \ldots || P_m \]

- Add a global controller for planning and schedule
  \[ H = G || P || M || A \]
- Control \( C \) with mobile phone from car or office
Network Evolution

- Consider $k$ households $H_j$, each with its own budget
  \[ H = H_1 || H_2 || \ldots || H_k \]
- Consider Coordinator, interacting the households to coordinate their budgets, based on the interaction with a utility company Utility
  \[ N = H || \text{Coordinator} || \text{Utility} \]
- Evolve to a network $U$ of utility companies
  \[ N = H || \text{Coordinator} || U \]
- Implement $H || \text{Coordinator}$ by distributing the coordination activities among the households themselves
- Households evolve in parallel, and in parallel with the network with more households and companies

Making ML Component Safer – Wild Idea 1

- How to model the bound and expectation?
- How to make ML use feedback?

Further levels of monitoring and control can be added
Making ML Interpretable and controllable – Wild Idea 2

- Use software to read ML’s mind or inspecting what ML is doing
- Communicate findings (suspicious behaviour of ML) out for control
- MIT New: Reading a Neural Network’s Mind [December 10 2017]
- But Modelling – “pyramid layers”

Conclusions

1. Dealing with requirements change and environment change by Incremental development and Evolution
2. BigData and AI and Learning Software deal with uncertainty as well as causing uncertainties
3. Architecture models are essential for
   - correct and secure by design,
   - identification of safety vulnerabilities and security threats, and
   - making architecture decisions for different concerns (aspect-orientation)
4. There is a need of linking of different modelling paradigms for AI systems, Hybrid Systems, Component-Based and Service Based Software Systems.
5. Need to handle and support healthy/safe evolution from multi-dimensions
Indirections

“All Problems in Computer Science can be Solved by Another Layer of Indirection.

But that usually will create another problem.”

--- David J. Wheeler
Lecture 15

Transforming Coloured Petri Net Models into Code for TinyOS – A Case Study of the RPL Protocol

by Lars M. Kristensen
Transforming Coloured Petri Net Models into Code for TinyOS - A Case Study of the RPL Protocol

Lars M. Kristensen and Vegard Veiset
Department of Computing
Bergen University College, NORWAY
Email: lmkr@hib.no / vegard.veiset@stud.hib.no

Motivation

- Coloured Petri Nets (CPNs) have been widely used for modelling of concurrent systems:
  - specification, validation, and verification
  - what about executable software?

configuration RPLProtocolAppC ( )
implementation ()
  components MainC, RPLProtocolC, DAO, DISC;
  RPLProtocolC.Init -> MainC.Init;
  RPLProtocolC.Send -> DAO.WPackets;
  RPLProtocolC.DIS -> DISC.WPackets;
  DAO.HEAD -> RPLProtocolC.HEAD;
)

100010011100011100011
Overview of Approach

- CPN models are platform independent and at a high level of abstraction:

  - Each manual refinement step consists of:
    - Increasing the level of details to the CPN model.
    - Adding pragmatic annotations to the CPN model.

  - The result is a platform-specific CPN model for automated code generation.

Pragmatic <<annotations>>

- Syntactical annotations [12] on model elements:
  - Adds platform dependent and domain-specific elements.
  - Can be bound to code generation templates.

---

Case Study of the RPL Protocol

IEFT RPL Protocol

- IoT routing protocol for distributed sensor networks currently being developed by the IETF:

- Supports a sensor nodes in establishing a DODAG for data collection purposes.
RPL CPN Model

- A platform independent model specifying the operation of the RPL Protocol:

![Diagram of RPL CPN Model]
Platform: TinyOS and nesC

- Operating system and programming language targeting resource constrained devices.

configuration RPLProtocolAppC {
}

implementation {

components MainC, RPLProtocolC, DAOC, DIOC;
RPLProtocolC.Boot -> MainC.Boot;
RPLProtocolC.DAO -> DAOC.RPLPacket;
RPLProtocolC.DIO -> DIOC.RPLPacket;
DAOC.NODE -> RPLProtocolC.NODE;
}

- Applications are structured into components providing and using interfaces.
- Split-phase programming model based on commands, calls, events and signals.
- Component are wired into a configuration constituting an application.

Proposed Refinement Methodology
Refinement Methodology

- A five step methodology for refining models to an abstraction level suited for code generation:
  1. **Component architecture** identifying components and interfaces, and determining an application configuration.
  2. **Interface naming, provision, and use** allowing reference to the same interface provided by multiple components.
  3. **Component and interface signatures** identifying commands and events and associated types.
  4. **Component classification** into boot-, dispatch-, external-, timed-, and regular components.
  5. **Internal component behaviour** providing control-flow oriented modelling of command and event implementations.

---

Step 1: Component Architecture

- Identify <<components>> and <<interfaces>> via substitutions transitions and socket places:
Step 2: Interface Naming and Use

- Resolve naming conflicts and specify use and provision of interfaces:

Step 3: Interface Signatures

- Refine component <<interfaces>> to specify <<commands>> and <<events>>: 
Step 4: Component Classification

- Classifies components as boot-, timed-, dispatch-, external-, and regular components:

Step 5: Internal Behaviour

- Makes explicit control flow and data access in the command and event implementations:
Automated Code Generation

Code Generation

- A template-based code generator implemented based on the Access/CPN Framework [15]:

  1. Mapping CPN ML datatypes into corresponding nesC datatypes.
  2. Interfaces based on places annotated with `<<interface>>`
  3. Components based on substitution transition with `<<component>>`
  4. Configuration and wiring based on `<<component>>` substitution transitions and `<<interface>>` arcs
  5. Command and event behaviour based on `<<var>>` and `<<id>>` places and structural pattern matching.

- Top-down traversal of the CPN model invoking templates according to encountered pragmatics.

Code Validation

- Deployment in a virtualised sensor networks using the TOSSIM emulator:

- Instrumentation and inspection of event-logs:
  
  DEBUG (0): 0:0:0.0000000300 RPL | Application booted.
  DEBUG (0): 0:0:0.0000000300 RPL | State change: 0 -> 2.
Conclusions and Future Work

- A semi-automatic approach to code generation for the TinyOS Platform:
  - A five step methodology refining the model to a level of detail suitable for generating nesC code for the target platform.
  - Pragmatics used to relate CPN model construct and elements to the target platform via code generation templates.

- The approach has been initially validated on the IETF RPL routing protocol for sensor networks.

- Future work:
  - Formalisation of meta-models and transformation steps for the refinement methodology.
  - Model checking techniques for verification of refined models.
  - Model-based testing for validating the generated code.
Lecture 16

Model-based Test-Case Generation with Coloured Petri Nets

by Rui Wang
Model-based Test-Case Generation with Coloured Petri Nets

Rui Wang¹
rwa@hvl.no

L. Kristensen¹  H. Meling²  V. Stolz¹

¹Department of Computing, Mathematics, and Physics, Western Norway University of Applied Sciences, Norway
²Department of Electrical Engineering and Computer Science, University of Stavanger, Norway

BeChong Spring Meeting, Bergen, 27.Feb.2018

Motivation

- Cloud-based services and distributed systems
  - Correctness
  - Availability
  - Fault-tolerance

- Implementation of complex distributed systems is challenging and error-prone¹

- Challenges
  - Concurrency and communication must be handled
  - Fault-tolerance must be implemented correctly

¹Jepsen project - [https://jepsen.io/]
Overview

- Quorum systems are fundamental in building fault-tolerant distributed systems
- **Model-based testing** is a prominent paradigm for error-detection and validation

![Diagram]

**Research Goal**
To investigate the use of Coloured Petri Nets (CPNs) for model-based testing of Quorum-based distributed systems

Outline

- Quorum-based distributed systems and the Gorums framework
- System under test: Gorums and distributed storage
- CPN testing model and test case generation
- Test cases execution and experimental results
- Conclusions and future work
Quorums

- Abstraction for ensuring consistency and fault-tolerance in distributed systems through replication
- **A quorum system**: a collection of subsets of nodes, called quorums, where every pair of subsets intersect
- Example: Read and write quorum system with majority quorum

![Quorum System Diagram](image)

---

Gorums Framework

- A software framework implemented in the Go programming language
- Ease the implementation effort of Quorum-based distributed systems
- Build advanced distributed algorithms and storages, which rely on a quorum system to achieve fault-tolerance

**Long term objective**: validate the implementation of the Gorums framework

- Application: Use the distributed storage implemented using Gorums as the system under test

---


Objective - Overview

![Diagram of test cases and oracles connected to CPN tools, test adapter, and system under test]

Gorums Abstraction

- **Quorum calls**
  - invoke RPCs on set of processes and collect responses
- **Quorum functions**
  - process responses to determine if a quorum has been obtained
SUT: Gorums and Distributed Storage

- A distributed storage system with a single writer and multiple readers, implemented using Gorums
  - Replicated servers for fault-tolerance
  - Read and Write quorum calls
  - Read and Write quorum functions
- Example of a user-defined quorum function:

```
Algorithm 1 Read quorum function
1: func (qs QUORUMSPEC) ReadQF(replies []READREPLY)
2:   if len(replies) < qs.ReadQSize then ▷ read quorum size
3:     return nil, false ▷ no quorum yet, await more replies
4:   highest := ⊥ ▷ reply with highest timestamp seen
5:   for r := range replies do
6:     if r.Timestamp ≥ highest.Timestamp then
7:       highest := r
8:   return highest, true ▷ found quorum
```

CPN Testing Model for the Distributed Storage
Test Case Generation - Quorum functions

- Unit tests for quorum functions
  - Test cases can be obtained by considering occurrences of the `ApplyReadQF` transition
  - Test oracle can be obtained by considering the value of the token put back on place `Waiting Reply`

![Diagram of Test Case Generation - Quorum functions]

Test Case Generation - Quorum calls

- System tests involving quorum calls
  - Test drivers for the system are used to generate different test scenarios of read and write quorum calls
  - Example: Parallel read/write quorum calls followed by server failures and a new read and a write call can be invoked

![Diagram of Test Case Generation - Quorum calls]
Test Oracle for Quorum Calls

- Concurrent and sequential executions of quorum calls:

  ![Diagram of quorum call executions]

  - There are several legal outcomes of a test case execution due to interleaving.

Test Case and Test Oracle Generation

- State-space exploration
  - Explore the full state space of the CPN testing model
  - Search for arcs corresponding to transitions representing the application of the quorum function
  - Obtain test cases with test oracles

- Simulation
  - Run a simulation of the CPN testing model
  - Use the monitoring facilities of the CPN tools simulator
  - Detect occurrences of the transitions representing the application of the quorum function
  - Emit the corresponding test cases

- Runtime monitor
  - Address non-determinism of concurrent executions to observe legal results as test oracles
The QuoMBT Framework

- Test cases are represented in an XML format
- Test adapter for reading and executing test cases
- Handles both unit tests and system level tests

Experimental Results

- Approach evaluated using test drivers with sequential and parallel execution of read and write quorum calls

**Table**: Experimental results for successful scenarios.

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Nodes</th>
<th>Arcs</th>
<th>Time (seconds)</th>
<th>QC</th>
<th>QF</th>
<th>Gorums Library</th>
<th>QCs RD WR</th>
<th>QFs RD WR</th>
<th>System</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>RD</td>
<td>39</td>
<td>72</td>
<td>&lt;1</td>
<td>1</td>
<td>3</td>
<td>24.6</td>
<td>84.4</td>
<td>0</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>S2</td>
<td>WR</td>
<td>39</td>
<td>72</td>
<td>&lt;1</td>
<td>1</td>
<td>3</td>
<td>24.6</td>
<td>0</td>
<td>84.4</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>S3</td>
<td>RD;WR</td>
<td>254</td>
<td>543</td>
<td>&lt;1</td>
<td>1</td>
<td>7</td>
<td>39.1</td>
<td>84.4</td>
<td>84.4</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>S4</td>
<td>WR;RD</td>
<td>254</td>
<td>543</td>
<td>&lt;1</td>
<td>1</td>
<td>12</td>
<td>40.8</td>
<td>84.4</td>
<td>84.4</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>S5</td>
<td>WR;RD</td>
<td>1,349</td>
<td>4,379</td>
<td>1</td>
<td>6</td>
<td>17</td>
<td>40.8</td>
<td>84.4</td>
<td>84.4</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>S6</td>
<td>(WR;RD);RD</td>
<td>3,035</td>
<td>7,867</td>
<td>2</td>
<td>6</td>
<td>17</td>
<td>40.8</td>
<td>84.4</td>
<td>84.4</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>

RD = Read Quorum Call, WR = Write Quorum Call

**Main results**
- for read & write quorum functions: 100%, both system and unit tests
- for read & write quorum calls: up to 84.4%
- 40.8% for the Gorums library as a whole
Experimental Results

- Experimental results for scenarios involving server failures and programming errors
  - Programming errors: injected in user defined quorum functions
  - Server and communication failures:
    - terminate one or more servers
    - the client makes quorum calls before the servers have started

- Results
  - for read & write quorum functions: still 100%, both system and unit tests
  - for read & write quorum calls: increase to 96.7%
  - 52.3% for the Gorums library as a whole

Conclusions

- Initial experiments on a distributed storage system promising, giving good code coverage
  - successful scenarios
  - scenarios involving server failures

- Testing approach is able to detect programming errors by executing the unit and system tests

- A general infrastructure to validate quorum-based systems implemented by Gorums
  - CPN modeling approach
  - Test case generation algorithms
  - Test case execution framework with runtime monitor

- The CPN testing model can serve as a basis for model-based testing of other quorum-based systems

- The state space and simulation-based test case generation are independent of the Quorum systems under test
Future Work

- Test the quorum calls under additional failures scenarios and adverse conditions, such as network errors, in order to increase the coverage of Gorums library
- Extend the current CPN model and test adapter in order to support such failure scenarios and conditions in system tests, such as generating timeouts
- Experiments on additional complex quorum-based systems implemented by Gorums
  - Multi-writer storages with multiple clients
  - Advanced distributed protocols, such as Paxos
Program analysis with LLVM

BeChong Spring School, Feb. 2018, Bergen

Dan Li, Volker Stolz, Svetlana Jakšić
HVL

Content

1. COMES and Program analysis
2. What is LLVM
3. LLVM Intermediate Representation (IR).
4. Programming LLVM IR.
5. Our work with LLVM
Content

1. COMES and Program analysis
2. What is LLVM
3. LLVM Intermediate Representation (IR).
4. Programming LLVM IR.
5. Our work with LLVM

COEMS System Overview
**Requirements from the industry partners**

- Detection and optimization of shared resources (TAT-TSR:022, 023)
- Waiting time for a lock (TAT-TSR:036)
- Performance analysis on functions – hot spots (AGI:039)
- Bus and memory usage – bottlenecks (AGI:040)
- Inter-thread communication (AGI:041)
- Detection and optimization of thread migration and data exchange (TAT-TSR:014, 015, 016, 017)
- Lower cost of logging/instrumentation than printf (AGI:044)
- Logging write accesses to selected variables (AGI:045)
- Execution, branch and MC/DC coverage (TAT-TSR:064, 065)

**Works in COEMS**

- Finding Data Races
- Finding Timing Bugs
- Finding Functional Bugs

**Measuring Coverage**
- Execution Coverage
- Branch Coverage
- MC/DC Coverage

**Measurement of Worst-Case Execution Time and Worst-Case Response Time**
Data Race Problem

• **Problem:** Data race detection in multithreaded programs. (Implies shared memory)

• A data race occurs when two concurrent threads access a shared variable and
  – at least one access is a write and
  – the threads use no explicit mechanism to prevent the accesses from being simultaneous

• In other words, a data race can lead to a potential violation of mutual exclusion.

• **Solution:** a tool that automates the problem of detecting potential data races

• **Approaches:** Static & Dynamic & Hybrid program analysis

Program Analysis

• Program analysis is about developing algorithms and tools that can analyze other programs

• A wide variety of applications
  – Checking whether a program is correct w.r.t. a given specification.
  – Finding bugs (e.g., model checking, testing, etc.)
  – Optimizing performance (e.g., compiler optimizations, bloat detection, etc.)
  – Detecting security vulnerabilities (e.g., detecting violations of security policies, etc.)
  – Improving software maintainability and understandability
Static v.s. Dynamic Analysis

• Static analysis: performed at compile time or earlier
  – Attempt to understand certain program properties
  – Without running a program
  – Techniques: Type-based analysis, Language features, Data flow analysis

• Dynamic analysis: runtime analysis, monitor programs during execution
  – The program may need to be “instrumented” with additional instructions
  – The additions don’t change program functionality but are used to monitor conditions of interest, for example, access to shared variables.
  – Post mortem or on-the-fly analysis of traces.
  – Add overhead to running time and memory.

Dataflow Analysis

• A class of static analyses that aim to understand how data flows in the program

• Typical examples
  – Available expression analysis
  – Reaching definition analysis
  – Live variable analysis
  – Constant propagation
Constant Propagation

• Determine, for each program point, whether or not a variable has a constant value whenever execution reaches the point.

• For example:
  
  \[
  \begin{align*}
  & [x := 6]^{1}; [y := 3]^{2}; \text{while } [x > y]^{3} \text{ do } ([x := x - 1]^{4}, [z := y + y]^{6}) \\
  & \text{Transform into:} \\
  & [x := 6]^{1}; [y := 3]^{2}; \text{while } [x > 3]^{3} \text{ do } ([x := x - 1]^{4}, [z := 9]^{6}) \\
  & \text{Finally into} \\
  & [x := 3]^{1}; [y := 3]^{2}; [z := 9]^{6}
  \end{align*}
  \]

• Solution: least fixed-point: a worklist-based algorithm

Least Fixed Point Computation

• Represent program as a control-flow graph
• Want to compute abstract values at every program point
• Initialize all abstract states to ⊥
• Repeat until no abstract state changes at any program point:
  • Compute abstract state on entry to a basic block B by taking the join of B’s predecessors
  • Symbolically execute each basic block using abstract semantics
**Instrument in Dynamic Analysis**

```c
void func(int i) {
    Instrument
    if (i>0) {
        Manual
        Automatic
        func(i-1);
    }
    exit(func);
}
```

**Levels of Program Analysis**

- **Program Source Code in Textual Format**
- **AST**
- **Assembly Code**
- **Machine Code**
Levels of Program Analysis

Content

1. COMES and Program analysis
2. What is LLVM
3. LLVM Intermediate Representation (IR).
4. Programming LLVM IR.
5. Our work with LLVM
What is LLVM

- The LLVM is an umbrella project that hosts and develops a "collection of modular and reusable compiler and toolchain technologies" used to develop compiler front ends and back ends.
- An open source project grows up from a Master's thesis research 15 years ago.
- Industrial-strength tools and library. Its C/C++ compiler competitive with GCC.
- Supports all major architectures (x86, ARM, MIPS, PowerPC, ...)
- Wildly used in the industry (Adobe, Apple, Cray, Intel, NVIDIA, Siemens, Sun, ...)  
- Supports dozens of languages (C, C++, FORTRAN, COBOL, Rust, Julia, Swift, Objective-C, ...)
- Compiler framework designed to support transparent lifelong program analysis and transformation.
- The LLVM compiler generates Intermediate Representation (IR) for all of the languages as an intermediate step

LLVM Architecture

- **Frontend**: Translate program code into the LLVM IR. This includes a lexical analyzer, a syntax parser, a semantic analyzer, and the LLVM IR code generator. Plugin interface and a separate static analyzer tool are provided to allow deep analyses.
- **Optimizer**: Target-independent optimizations, including tools and libraries provide interfaces to IR construction, assembling, and disassembling. In where most part of optimizations is applied by invoking LLVM passes.
- **Backend**: Responsible for code generation. It converts LLVM IR to target-specific assembly code or object code binaries. Register allocation, loop transformations, peephole optimizers, and target-specific optimizations/transformations belong to the backend.
Three Primary LLVM Components

• The LLVM Instruction Set
  – Intermediate Representation (IR).
  – The common language- and target-independent IR
  – Internal (IR) and external (persistent) representation

• A collection of well-integrated libraries
  – Analyses, optimizations, code generators, JIT compiler, garbage collection support, profiling, ...

• A collection of tools built from the libraries
  – Assemblers, automatic debugger, linker, code generator, compiler driver, modular optimizer, …
LLVM Standalone Tools

- **clang**: LLVM frontend for C/C++. It compiles C/C++ source code into IR.
- **opt**: LLVM analyzer and optimizer which runs certain optimizations. The input must be an LLVM IR file and generates an output file of same type.
- **llc**: Converts the LLVM bitcode to a target-machine assembly code or object file of a specific backend.
- **lli**: Directly executes program’s bit-code using JIT.
- **llvm-link**: Links two or more llvm bitcode files into one file.
- **llvm-as**: Transforms human-readable LLVM IR files, called LLVM assemblies, into LLVM bitcodes.
- **llvm-dis**: Decodes LLVM bitcodes into LLVM assemblies.
- **More**: http://llvm.org/docs/GettingStarted.html#llvm-tools

LLVM Clang vs GCC

- Clang is very competitive when compared with gcc in performance.
- One of the two compilers is faster in some benchmarks, and slower in others. Usually clang/clang++ have faster compilation times, but the results are very close.

More LLVM/Clang-based Tools

• Sanitizer projects:
  – AddressSanitizer: Fast memory error detector
  – ThreadSanitizer: Detects data races
  – LeakSanitizer: Memory leak detector
  – MemorySanitizer: Detects reads of uninitialized variables
  – UBSanitizer: Detects undefined behavior

LLVM Basic Libraries

• libLLVMCore: This contains all the logic related to the LLVM IR: IR construction (data layout, instructions, basic blocks, and functions) and the IR verifier. It also provides the pass manager.

• libLLVMAnalysis: This groups several IR analysis passes, such as alias analysis, dependence analysis, constant folding, loop info, memory dependence analysis, and instruction simplify.

• libLLVMSupport: This comprises a collection of general utilities. Error, integer and floating point handling, command-line parsing, debugging, file support, and string manipulation are examples of algorithms that are implemented in this library, which is universally used across LLVM components.

• libLLVMCodeGen and libLLVMTarget: This implements target-independent code generation and machine level—the lower level version of the LLVM IR—analyses and transformations. This provides access to the target machine information by generic target abstractions.
The *opt* tool performs machine independent optimizations. That’s the point to do our program analysis.

There are already many optimizations available through opt. And we can use them in our analysis.

LLVM optimizations manipulate LLVM IR, so does our analysis program.

Once we have optimized the intermediate program, we can translate it to machine code -- JIT or Offline.

---

**Getting LLVM to Work**

- The official version of LLVM is 5.01, and the newest version in svn is 7.
- We can download the LLVM binary files, or check out the source code from svn and compile them. But it takes hours.

- If you use Ubuntu 17 or later version, you can install all required LLVM tools and libraries using the following command:

```bash
$> apt-get update
$> apt-get install build-essential clang llvm libboost-program-options-dev
```
Content

1. COMES and Program analysis
2. What is LLVM
3. LLVM Intermediate Representation (IR).
4. Programming LLVM IR.
5. Our work with LLVM

LLVM IR: Intermediate Representation

- LLVM IR is a typed, assembly-like language based on static single assignment (SSA) form.
- The representation of the LLVM middle-end
- Simple, low-level control flow constructs, but permits to represent high-level languages cleanly.
  - No high-level constructs, e.g. classes, inheritance.
  - No runtime system or object model.
  - Does not guarantee safety
- The majority of optimizations is done at LLVM IR level
About LLVM IR

- Low-level and target-independent type system
- RISC-like three address instruction set, with usual opcodes
  add, mul, or, shift, branch, load, store, etc
- Infinite virtual register (local value) set.
- Load/store instructions with typed-pointers.
- Three different formats: bitcode (.bc, compact on-disk format), in-memory representation and textual representation (.ll, LLVM assembly language)
- Control flow graph is represented explicitly in the code.
- Support use-def and def-use chains.
- One IR for analysis and optimization.
- A local value starts with %, and a global variable starts with @

IR Instruction Set

- Arithmetic: add, sub, mul, udiv, sdiv, ...
  \%tmp = add i32 \%indvar, 5
- Logical operations: shl, lshr, ashr, and, or, xor
  \%shr21 = ashr i32 \%mul20, 8
- Memory access: load, store, alloca, getelementptr
  \%tmp3 = load i64* \%tmp2
  store i32 \%0, i32* \%2
- Comparison: icmp, select
  \%cmp12 = icmp ne i32 \%add, 0
- Control flow: call, ret, br, ...
  call void @foo(i32 \%x)
LLVM Type System

- LLVM type consists of:
  - Primitives: integer (arbitrary bitwidth (i32, i64, i1), floating point, label, void
  - Derived: pointer, array, structure, function, vector, ...
  - No high-level types: type-system is language neutral

- Allows arbitrary casts.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>i1</td>
<td>1-bit integer (may be used for bool)</td>
</tr>
<tr>
<td>i8</td>
<td>8-bit integer (used for char)</td>
</tr>
<tr>
<td>i32</td>
<td>32-bit integer</td>
</tr>
<tr>
<td>i64</td>
<td>64-bit integer</td>
</tr>
<tr>
<td>i12015</td>
<td>12015-bit integer</td>
</tr>
<tr>
<td>float</td>
<td>32-bit floating-point number</td>
</tr>
<tr>
<td>double</td>
<td>64-bit floating-point number</td>
</tr>
<tr>
<td>void</td>
<td>empty type</td>
</tr>
<tr>
<td>label</td>
<td>named addresses (see later)</td>
</tr>
<tr>
<td>i32 *</td>
<td>pointer to one (or more) i32</td>
</tr>
<tr>
<td>[40 x i8]</td>
<td>array of 40 8-bit numbers (static size)</td>
</tr>
</tbody>
</table>

LLVM IR Program Structure

- **Module** contains functions, global variables, constants, type sizes, and other attributes.
  - Module is the unit of compilation, analysis and optimization
  - A module lies in its own file (.bc, .ll) or in memory.

- **Function** represents a function definition and contains basic blocks/arguments

- **BasicBlocks** contain list of instructions

- **Instructions** is **opcode** + operands, each of which is a Value.
  - All operands have types
  - Instruction result is typed
Compiling c to LLVM IR: a Example

foo.c

```c
int foo(int a) {
    int b = 12;
    if (a) {
        b++;
    }
    return b;
}
```
Compiling c to LLVM IR: an Example

```
int foo(int a) {
    int b = 12;
    if (a){
        b++;
    }
    return b;
}

int main() {
    int a = 1;
    return foo(a);
}
```

$> clang -c -S -emit-llvm foo.c -o foo.ll

```
$> clang -c -S -emit-llvm foo.c -o foo.ll
```

(part 2)
LECTURE 17.
Output as CFG

```
$> opt -dot-cfg foo.ll -o foo.dot
Writing 'cfg.foo.dot'... Writing 'cfg.main.dot'...
```

Optimizing IR Code

```
foo.ll
1 # ModuleID = 'foo.c'
2 source_filename = "foo.c"
3 target datalayout = "a-mie-i686-f80:128-n8:16:i32:64-S128"
4 target triple = "x86_64-pc-linux-gnu"
5 # Function Attrs: noline nounwind untable
6 define i32 @foo(i32) #0 {
7 %2 = alloca i32, align 4
8 %3 = alloca i32, align 4
9 store i32 %0, i32* %2, align 4
10 store i32 %0, i32* %3, align 4
11 store i32 %0, i32* %3, align 4
12 %4 = load i32, i32* %3, align 4
13 %5 = icmp ne i32 %2, 0
14 br label %9
15 %6:
16 %7 = load i32, i32* %3, align 4
17 %8 = add nsw i32 %7, 1
18 store i32 %8, i32* %8, align 4
19 br label %9
20 %9:
21 %10 = load i32, i32* %3, align 4
22 ret i32 %10
23 }
```
Optimizing IR Code

foo.ll

```c
$> opt -O3 foo.ll -S -o foo2.ll

foo2.ll (part)

```
BeChong Spring School, Feb. 2018, Bergen

To Target-Machine Assemblies

foo2.ll (part)

8  %2 = icmp eq i32 %0, 0
9  %_. = select i1 %2, i32 12, i32 13
10 ret i32 %.

$> llc -march=arm -filetype=asm foo2.ll -o foo_arm.asm
To Target-Machine Assemblies

ARM:

\[
\begin{align*}
\text{mov r1, r0} \\
\text{mov r0, #12} \\
\text{cmp r1, #0} \\
\text{movne r0, #13} \\
\text{mov pc, lr}
\end{align*}
\]

\[
8 \quad \text{%2 = icmp eq i32 %0, 0} \\
9 \quad \text{%. = select i1 %2, i32 12, i32 13} \\
10 \quad \text{ret i32 %}.
\]

\[
\begin{align*}
\text{testl %edi, %edi} \\
\text{setne %al} \\
\text{movzbl %al, %eax} \\
\text{orl $12, %eax} \\
\text{ret}
\end{align*}
\]

\[
\begin{align*}
\text{cmplwi 0, 3, 0} \\
\text{beq 0, .LBB0_2} \\
\text{li 3, 13} \\
\text{blr} \\
\text{.LBB0_2:} \\
\text{li 3, 12} \\
\text{blr}
\end{align*}
\]
## To Target-Machine Assemblies

```
foo2.ll (part)
```

```
8 %2 = icmp eq %32 %0, 0
9 %. = select %1 %2, %32 12, %32 13
10 ret %32 %.
```

### x86:
- `testl %edi, %edi`
- `setne %al`
- `movzbl %al, %eax`
- `orl $12, %eax`
- `ret`

### ARM:
- `mov r1, r0`
- `mov r0, #12`
- `cmp r1, #0`
- `movne r0, #13`
- `mov pc, lr`

### PowerPC:
- `cmplwi 0, 3, 0`
- `beq 0, .LBB0_2`
- `li 3, 13`
- `blr`
- `.LBB0_2:
  - `li 3, 12`
  - `blr`

## Content

1. COMES and Program analysis
2. What is LLVM
3. LLVM Intermediate Representation (IR).
4. Programming LLVM IR.
5. Our work with LLVM
Coding LLVM IR

- A set of libraries provided to support C++, uses the STL.
- Some supports for other languages also available, such Python.
- If possible, chose C++ as the programming language.

LLVM Passes

- LLVM applies a chain of analyses and transformations on the target program.
- Each of these analyses or transformations is called a pass.
- Machine independent passes are invoked by `opt`, and Machine dependent passes are invoked by `llc`.
- A pass may require information provided by other passes. Such dependencies must be explicitly stated and managed by the PassManager. It runs until every pass stops.
- Sharing analysis results between passes is possible.
- Each Pass returns a boolean
  - True indicates the pass has made a changed to the IR code.
  - False indicates that no change was made.
Types of Passes

- A pass is an instance of the LLVM class `Pass`.
- There are many kinds of passes.
- Constraints imposed.
  - e.g. `FunctionPass` can only look at “current function”, and cannot maintain state across functions.

LLVM Analysis Basis

- LLVM IR is in SSA form:
  - `use-def` and `def-use` chains are always available
  - All objects have user/use info, even functions

- Control Flow Graph is always available:
  - Exposed as BasicBlock predecessor/successor lists
  - Many generic graph algorithms usable with the CFG

- Higher-level info implemented as passes, and can be used:
  - Dominators, CallGraph, induction vars, aliasing, GVN, ...
Let's write a pass that counts the number of times that each opcode appears in a given function. This pass must print, for each function, a list with all the instructions that showed up in its code, followed by the number of times each of these opcodes has been used.

Example:

Counting Number of Opcodes

Function foo

```c
int foo(int a) {
    int b = 12;
    if (a){
        b++;
    }
    return b;
}
```

Example:

Counting Number of Opcodes

Function main

```c
int main() {
    int a = 1;
    return foo(a);
}
```

Program: Counting Number of Opcodes

```c
#define DEBUG_TYPE "opCounter"
#include "llvm/Pass.h"
#include "llvm/IR/Function.h"
#include "llvm/Support/raw_ostream.h"
#include <map>
using namespace llvm;
namespace {
    struct CountOp : public FunctionPass {
        static char ID;
        CountOp() : FunctionPass(ID) {
        }
        virtual bool runOnFunction(Function &F) {
            errs() << "Function " << F.getName() << "
            std::map<std::string, int> opCounter;
            for (Function::iterator bb = F.begin(), e = F.end(); bb != e; ++bb) {
                for (BasicBlock::iterator i = bb->begin(), e = bb->end(); i != e; ++i) {
                    if (opCounter.find(i->getOpcodeName()) == opCounter.end()) {
                        opCounter[i->getOpcodeName()] = 1;
                    } else {
                        opCounter[i->getOpcodeName()] += 1;
                    }
                }
            }
            std::map<std::string, int>::iterator i = opCounter.begin();
            std::map<std::string, int>::iterator e = opCounter.end();
            while (i != e) {
                errs() << i->first << " : " << i->second << "\n";
                i++;
            }
            errs() << "\n";
        }
        char CountOp::ID = 0;
    };
```

The pass runs once for each function in the program; therefore, it is a `FunctionPass`.

Here defines and registers the name of the pass, `opCounter`, as well as the help string about the pass.
# Deeper into the Pass

```cpp
struct CountOp : public FunctionPass {
  std::map<std::string, int> opCounter;
  static char ID;
  CountOp() : FunctionPass(ID) {}

  virtual bool runOnFunction(Function &F) {
    errs() << "Function " << F.getName() << 'n';
    for (Function::iterator bb = F.begin(), e = F.end(); bb != e; ++bb) {
      for (BasicBlock::iterator i = bb->begin(), e = bb->end(); i != e; ++i) {
        if (opCounter.find(i->getOpcodeName()) == opCounter.end()) {
          opCounter[i->getOpcodeName()] = 1;
        } else {
          opCounter[i->getOpcodeName()] += 1;
        }
      }
    }
    std::map<std::string, int>::iterator i = opCounter.begin();
    std::map<std::string, int>::iterator e = opCounter.end();
    while (i != e) {
      errs() << i->first << ': ' << i->second << 'n';
      i++;
    }
    errs() << 'n';
    opCounter.clear();
    return false;
  }
};
```

- Here is the start point of the pass.
- Return true if the IR code is changed, false if not.

## Iterating Through LLVM IR Code

```cpp
for(Function::iterator bb = F.begin(), e = F.end(); bb != e; ++bb) {
  for(BasicBlock::iterator i = bb->begin(), e = bb->end(); i != e; ++i) {
    if (opCounter.find(i->getOpcodeName()) == opCounter.end()) {
      opCounter[i->getOpcodeName()] = 1;
    } else {
      opCounter[i->getOpcodeName()] += 1;
    }
  }
}
```

- Iterators are used to go over LLVM data.
- An iterator over a Function gets a list of basic blocks.
- An iterator over a Block gets a list of instructions.

• Here is the start point of the pass.
• Return true if the IR code is changed, false if not.
Compiling and Running the Pass

• To compile the pass, we type:

```bash
$> clang -shared -fPIC -Wno-unknown-warning-option `llvm-config --cxxflags --ldflags` opCounter.cpp -o opCounter.so
```

• Our pass is now a shared library `opCounter.so` in current folder

• We can invoke it using the `opt` tool to run on the `foo.ll`

```bash
$> opt -load ./opCounter.so -opCounter < foo.ll > /dev/null
```
Compiling and Running the Pass

• To compile the pass, we type:

```bash
$> clang -shared -fPIC -Wno-unknown-warning-option `llvm-config --cxxflags --ldflags` opCounter.cpp -o opCounter.so
```

• Our pass is now a shared library `opCounter.so` in current folder

• We can invoke it using the `opt` tool to run on the `foo.ll`

```bash
$> opt -load ./opCounter.so -opCounter foo.ll > /dev/null
```

Final Remarks about LLVM Pass

• Major method to program LLVM IR.

• LLVM provides users with a string of analyses and optimizations which are called passes.

• Users can chain new passes into this pipeline.

• The pass manager orders the passes in such a way to satisfies the dependencies.

• Passes are organized according to their granularity, e.g., module, function, loop, basic block, etc.
Content

1. COMES and Program analysis
2. What is LLVM
3. LLVM Intermediate Representation (IR).
4. Programming LLVM IR.
5. Our work with LLVM

Workflow of COEMS Analysis
Two Analysis Tasks in COEMS

• Counting threads and locks
  – Counting number of threads that can be created.
  – Counting number of locks that can be initialized.
  – Counting number of global variables that data race may happen.
  – Never underestimate.
  – Applied the constant propagation.

• Instrument mutex-lock initializing, pthread creating, locking/unlocking and
global variable accessing.
  – Avoid instrumentation when not necessary
    • On constant variables
    • On variables that are never written to
    • On variables that are never accessed from pthread invoked functions.
    • Collapsing multiple accesses into one
    • Using aliasing/escape analysis
  – Decide real accessed variables at running time.

Instrumented Trace Example

```
void* f(void* arg) {
  for (int i = 0; i < 10; i++) {
    pthread_mutex_lock(&m);
    x++;
    pthread_mutex_unlock(&m);
  }
  return NULL;
}
```

```
15158657320600: instruction = "call"
15158657320600: function = "f"
15158657320600: line = 12
15158657320600: column = 8
15158657320600: threadid = 139665906870016
15158657320600: functioncall = "pthread_mutex_lock"
15158657320600: mutexlock = "m"
15158657320600: mutexlockid = 6299816
15158657320600: instruction = "load"
15158657320600: function = "f"
15158657320600: line = 13
15158657320600: column = 6
15158657320600: threadid = 139665906870016
15158657320600: varread = "x"
15158657320600: varaddr = 6299764
15158657320600: varsize = 4
15158657320600: varoffset = 0
```
Thank You!
Lecture 18

Model Transformation Evolution

by Adrian Rutle
Model Transformation Evolution

Adrian Rutle
BeChong Spring School
Bergen, 28. February 2018

Model-Driven Software Engineering

1. Models as first-class entities
2. Domain specific modeling languages
3. Communication with stakeholders
4. Model transformation as means to relate and manipulate models
   - Code generation
   - Behaviour specification
   - Language translation
   - Refactoring
5. Reasoning on model level
General Pattern for transformations

Transformation Approaches

Rule-based approaches

ATL: A rule-based MT language

TLatI
module Person2Contact; create OUT; MMB from IN; MMA;

rule Start { from p: MMA Person ( p.function = 'boss' ) to c: MMB Contact ( name = p.first_name + p.last_name ) }
Company to CRM example

Company to CRM in ATL code

```java
1 2 Module CompanyCRM;
3 Imports ODM :: IN : Company;
4 "rule CompanyCRM:"
5 6 from a :: Company
7 to b :: CRM
8 9 # "a.name.toLower() is 'e.com',
10 accounts <- a.persons,
11 projects <- a.projects
12 13 14 rule PersonWorker(
15 16 from a :: Company/persons.position = 'employee'
17 to b :: CRM/Account
18 19 ;
20 name <- a.firstname.toLowerCase()
21 account <-
22 23 24 rule Person2Client(
25 26 from a :: Company/persons.position = 'client'
27 to b :: CRM/Account
28 29 ;
30 account <-
31 name <- a.firstname.toLowerCase()
32 33 34 rule Project2Project(
35 36 from a :: Company/project
37 to b :: CRM/Account
38 39 name <- a.name
40 41 )
```
The reason a model undergoes modification is disparate.

A model is subject to different iterations before it converges to a stable version.

A model is a living entity, it may be amended or extended in order to accommodate new requirements and/or insights emerging from the domain, e.g.

- Technological shifts over the reference platform
- Lows and regulations

Evolution in our example
Evolved Company to CRM in ATL code

General evolution pattern

- M1 is evolved to M'1
- Deduce the evolved transformation T'
- We only consider changes in the source!
Formalisation

Encode $T$ as traceability mapping
Encode $ev$ as application of graph transformation
Deduce $T'$ by applying the derived $ev$

From ECore to Attributed Graphs

462 LECTURE 18.
Encode T as traceability mapping

Encode ev as application of graph transformation

$\begin{align*}
\text{(1)} &= \{(e : \text{Employee}, p : \text{Person}) \mid p.\text{position.emp} = e\} \\
\text{(2)} &= \{(c : \text{Client}, p : \text{Person}) \mid p.\text{position.cli} = c\}
\end{align*}$
Encode ev as application of graph transformation

Deduce T’ by applying the derived ev
A metamodell for traceability mapping

Company to CRM example
Company evolution

Derived transformation
Lecture 19

Invariant Generation and Safety Verification for Hybrid Systems

by Hengjun Zhao
Invariant Generation and Safety Verification for Hybrid Systems

Hengjun Zhao
zhaohj2016@swu.edu.cn
Southwest University, Chongqing, China
BeChong Spring School, HVLBergen, Norway
Feb 28, 2018
Outline

- Background on Hybrid System
- Invariant-Based Safety Verification
- Summary
Outline

• Background on Hybrid System

• Invariant-Based Safety Verification

• Summary

Classification of Dynamical Systems

• Discrete

ON

\[
\frac{dx}{dt} = f(x)
\]

OFF

• Continuous: temperature, velocity, ...

\[x(t) \quad x_0\]
Hybrid System

- Continuous + Discrete

Universal Law of Gravitation

Hybrid Systems in Real Life

Auto Gear

Cardiac Cell

HSs in Real Life

Electrical Circuits

Chemical Process

http://people.ee.ethz.ch/~mpt/2/docs/demos/twotanks.php

Embedded Control Systems

Control command output

Logic/Computation

Physical laws

Continuous state input
Why Hybrid System?

• Stability of switched linear system

D. Liberzon et al.: Basic problems in stability and design of switched linear systems

Lorenz System: Chaos

\[
\begin{align*}
\frac{dx}{dt} &= \sigma(y - x), \\
\frac{dy}{dt} &= x(\rho - z) - y, \\
\frac{dz}{dt} &= xy - \beta z.
\end{align*}
\]
Chua’s Circuit: Chaos

\[
\begin{align*}
\dot{x} &= \alpha(y - \phi(x)) \\
\dot{y} &= x - y + z \\
\dot{z} &= -\beta y \\
\phi(x) &\triangleq x + g(x) = m_1 x + \frac{1}{2} (m_0 - m_1) (|x + 1| - |x - 1|)
\end{align*}
\]

Programs in safety critical systems

- Ariane 5 flight 501 failure
- On 4 June 1996
- Self-destruction at around 3700 m
- 30 seconds after lift-off
Ariane 5 flight 501 failure

- Inertial Reference System (SRI) executed conversion from 64-bit floating point to 16-bit signed integer value
- Operand Error (too large)
- SRI computer stopped
- No correct guidance information could be obtained
- Failed

The same SRI software as in Ariane 4 was used for Ariane 5, but the horizontal velocity of Ariane 5 was five times more rapid than Ariane 4.

Flight control for Apollo 11 (1969)

- Could have failed!
- When commanded, the engine reacted after a time lag of 0.3 seconds
- During the mission, the thrust level surged up and down
- It was found out that the lag time of the engine had been improved to only 0.075 seconds
- Just inside the stability envelop!
Flight control for Apollo 11 (1969)

- Could have failed!
- Descent trajectory

- Quartic polynomial

Flight control for Apollo 11 (1969)

- Could have failed!
- Flying over a deep crater

- High order polynomial curve fitting
Software correctness in CPS

- Safety critical cyber physical system (CPS)

- Consider the correctness of programs embedded in the physical world: plant, environment

- Combine discrete behaviours with continuous dynamics: hybrid system
Research Topics

Formal Modelling

Finite Automata
Timed Automata

Hybrid Automata: Bouncing Ball

\[ h = 0 \]

\[ v' := -0.8 \cdot v \]

\[ \frac{dh}{dt} = v \]

\[ \frac{dv}{dt} = -g \]

\[ h \geq 0 \]

\[ h > 0 \]

Guard: \( h = 0 \)

Jump & Reset: \( v' := -0.8 \cdot v \)

Continuous Dynamics: \( \frac{dh}{dt} = v \), \( \frac{dv}{dt} = -g \)

Mode: \( h \geq 0 \)

Domain: \( h > 0 \)

Initial State: \( h > 0 \)
Hybrid Automata: Bouncing Ball

\[ h = 0 \quad v' := -0.8 \cdot v \]

\[ \frac{dh}{dt} = v \quad \frac{dv}{dt} = -g \]

Research Topics

**Formal Modelling**

**Control Engineering**
- controller design,
- stability analysis

**Computer Science**
- formal verification:
  - safety
  - liveness

......
......
Problem: safety verification

- Initial set
- Reachable set
- Unsafe region

Why Formal Verification?

- Simulation
  - Numerical errors
  - Coverage

http://spaceex.imag.fr/documentation/publications/spaceex-scalable-verification-hybrid-systems-34
Outline

• Background on Hybrid System

• Invariant-Based Safety Verification

• Summary

Deductive Verification

• Program

```plaintext
x:=1;
while (x<=1000000000)
{ x:=x+1; }
x\leq 0
```

• Inductive Invariant

- x=1 \implies x\geq 1
- x\geq 1 \implies x+1\geq 1
- x\geq 1 \implies \neg(x\leq 0)

Continuous system

\[
\frac{dx}{dt} = f(x)
\]
Inductiveness

- Discrete

\[ x_k \in I \rightarrow x_{k+1} \in I \]

- Inductiveness

\[ x_{k+1} = \varphi(x_k) \]

- Transition relation

\[ \Delta t \]

Continuous

- Inductiveness

\[ x(t) \in I \rightarrow x(t + \Delta t) \in I \]

- Transition relation

\[ x(t + \Delta t) = x(t) + x'(t) \cdot \Delta t \]

Invariant & Deductive Verification

\[ \frac{dx}{dt} = f(x) \]
Characterize Inductiveness

- **Inductiveness**
  \[ x(t) \in I \rightarrow x(t + \Delta t) \in I \]

- **Transition relation**
  \[ x(t + \Delta t) = x(t) + x'(t) \cdot \Delta t \]

Derivatives and Invariant

\[
\frac{dx}{dt} = f(x) \\
\frac{dp(x(t))}{dt} < 0 \\
\frac{dp(x(t))}{dt} > 0 \\
\frac{dp(x(t))}{dt} = 0
\]

\[ p(x) = 0 \]

\[ p(x) > 0 \]
Higher-Order Derivatives

\[ \frac{d^1 p}{dt^1} > 0 \]
\[ \forall \frac{d^1 p}{dt^1} = 0 \land \frac{d^2 p}{dt^2} > 0 \]
\[ \forall \frac{d^1 p}{dt^1} = 0 \land \frac{d^2 p}{dt^2} = 0 \land \frac{d^3 p}{dt^3} > 0 \]
\[ \forall \frac{d^1 p}{dt^1} = 0 \land \frac{d^2 p}{dt^2} = 0 \land \frac{d^3 p}{dt^3} = 0 \land \ldots \]

Criterion for Invariant

• \( f(x) \) and \( p(x) \) are polynomials
• Compute an upper bound \( N \) s.t.
• \( p(x) \geq 0 \) is an inductive invariant of iff
  \[ p=0 \implies \left( \frac{d^1 p}{dt^1} > 0 \lor \frac{d^1 p}{dt^1} = 0 \land \frac{d^2 p}{dt^2} > 0 \lor \ldots \right) \]
  \[ \left( \frac{d^1 p}{dt^1} = 0 \land \frac{d^2 p}{dt^2} = 0 \land \ldots \land \frac{d^N p}{dt^N} \geq 0 \right) \]
Main Result

- Semi-algebraic set
  \[
  \bigvee_{i=1}^{l} \bigwedge_{j=1}^{j_i} p_{ij}(x) \triangleright 0 \quad \triangleright \in \{\geq, >\}
  \]
- First-order theory of real numbers is decidable
  - Real quantifier elimination

Checking whether a semi-algebraic set is an inductive invariant of a polynomial continuous dynamical systems is decidable.

Parametric Case

- Parametric polynomials \( p(u,x) \)

Use templates and quantifier elimination to automatically discover inductive invariants.

Improve the efficiency by sum-of-squares relaxation (SOS) and semi-definite programming (SDP).
Non-polynomial Systems

- Elementary functions

\[ f, g \ := \ c | x | f + g | f - g | f \times g | \frac{f}{g} | f^a | e^f | \ln(f) | \sin(f) | \cos(f) , \]

Polynomializing Elementary ODEs

- \( f(x) = \frac{1}{x} \): let \( \dot{v} = \frac{1}{x} \), and thus \( \ddot{v} = -\frac{x}{x^2} \), so (1) is transformed to

\[
\begin{align*}
\dot{x} &= v \\
\ddot{v} &= -v^3
\end{align*}
\]

- \( f(x) = \sqrt{x} \): let \( v = \sqrt{x} \), and thus \( \dot{v} = \frac{x}{2\sqrt{x}} \), so (1) is transformed to

\[
\begin{align*}
\dot{x} &= v \\
\dot{v} &= \frac{1}{2}
\end{align*}
\]

- \( f(x) = e^x \): let \( v = e^x \), and thus \( \dot{v} = e^x \cdot \dot{x} \), so (1) is transformed to

\[
\begin{align*}
\dot{x} &= v \\
\dot{v} &= v^2
\end{align*}
\]
Polynomializing Elementary ODEs

- **$f(x) = \ln x$:** Let $v = \ln x$, and thus $\dot{v} = \frac{\dot{x}}{x}$; further let $u = \frac{1}{x}$, and thus $\dot{u} = -\frac{\dot{x}}{x^2}$. Therefore (1) is transformed to

\[
\begin{align*}
\dot{x} &= v \\
\dot{v} &= uv \\
\dot{u} &= -u^2 v
\end{align*}
\]

- **$f(x) = \sin x$:** Let $v = \sin x$, and thus $\dot{v} = \dot{x} \cdot \cos x$; further let $u = \cos x$, and thus $\dot{u} = -\sin x \cdot \dot{x}$. Therefore (1) is transformed to

\[
\begin{align*}
\dot{x} &= v \\
\dot{v} &= uv \\
\dot{u} &= -u^2
\end{align*}
\]

- **$f(x) = \cos x$:** The transformation is analogous to the case of $f(x) = \sin x$.

Non-polynomial Systems

Transform into polynomial systems by variable replacement
Soft Landing of a Lunar Lander

- 15km: Braking
- 3km: Adjustment
- 2.4km: Approach
- 100m: Obstacle avoidance
- 30m: Slow descent
- 0m: Lunar surface

Slow descent
Slow Descent

- Control objective:
  \[ v = -2 \text{m/s} \]

HA Model

Safety requirement:
\[ |v - (-2)| \leq 0.05 \]
Verification

• Safety requirement: \(|v - (-2)| \leq 0.05\)

• Generated invariant:
  • A polynomial of degree 6 in 3 variables


Summary

• Hybrid systems attracts more and more interests with the development of safety critical embedded systems
• Invariant plays an important role in the study (formal verification) of hybrid systems
• Semi-algebraic inductive invariant checking for polynomial continuous/hybrid systems is decidable
• Non-polynomial systems can be dealt with by variable replacement
• Invariant-based safety verification has good applications in real systems
Thanks!

Questions?
Lecture 20

A Formal Model of Parallel Execution on Multicore Architectures with Multilevel Caches

by Violet Ka I Pun
A Formal Model of Parallel Execution on Multicore Architectures with Multilevel Caches

Violet Ka I Pun
(Shiji Bijo Einar Broch Johnsen S. Lizeth Tapia Tarifa)

BeChong Spring Meeting
26 February – 2 March 2018

Parallel Execution of Programs – Problem

Understand how data moves between multilevel caches and shared memory during the execution of parallel programs

formally describe program execution in a setting with multiple and consistent copies of the same data in memory and caches.
Parallel Execution of Programs – Approach

Abstract Model
- Multicore architecture
- Exclusive multilevel coherent cache
- Shared memory

Operational Semantics
Captures
- Parallel executions of programs
- Data movements between multilevel caches and shared memory

Proof of concept tool for simulations and analysis
- Configurable parameters: e.g., number of cores, number of caches, cache associativity, data layout, etc.
- Measurements: e.g., further into the memory hierarchy, higher the penalties

1 some other policies: inclusive and NINE
A simple language with read and write statements

P ::= Task main{dap}
Task ::= task T{dap}
dap ::= ε | dap; dap | read(r) | write(r) | skip | dap ∩ dap | dap* | spawn(T) | commit(r) | commit

Task read_r{read(r0); spawn write_r; write(r1);}
Task write_r{write(r1)}
main{spawn read_r,}
r0, r1: references inside memory blocks.
Architecture in our settings

- Multilevel private and exclusive caches, single shared memory
- Size of a cache line and memory block are the same
- In-order execution in the cores
- Signalling is instantaneous, data movement is not immediate
- Blocks cannot move directly between caches belonging to different cores, always via shared main memory
- Data in cache lines and memory blocks are ignored

Abstract Model of Multicore Architectures – Overview

- Tasks waiting to be scheduled
- Node 1
  - \{Rd(n), RdX(m)\}
- Node 2
  - \{RdX(m), Rd(n)\}
- Node N
  - \{Rd(n), RdX(m)\}
- Abstract communication medium
- Main memory
  - \{Rd(n), RdX(m)\}
Abstract Model of Multicore Architectures – Nodes

Main memory
L1
Task
Core
Abstract communication medium
flush(n)
!RdX(n)
?Rd(n) | ?RdX(n)
� n
L2
!Rd(n)
?RdX(n)
swap
flush(n)
!RdX(n)
?Rd(n) | ?RdX(n)
flush(n)
Lm
fetch(n)
!RdX(n)
flush(n)
Mm
!RdX(n)
?Rd(n)
flush(n)

Abstract communication medium
Main memory

Formal Model

Syntactic categories.  Definitions.

<table>
<thead>
<tr>
<th>Lid ∈ Int</th>
<th>Cid ∈ Coreld</th>
<th>n ∈ Address</th>
<th>r ∈ Reference</th>
<th>T ∈ TaskId</th>
<th>M ∈ Memory</th>
<th>Tb ∈ TaskTable</th>
<th>rst ∈ RuntimeLang</th>
<th>dst ∈ DataLang</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lid ∈ Int</td>
<td>Cid ∈ Coreld</td>
<td>n ∈ Address</td>
<td>r ∈ Reference</td>
<td>T ∈ TaskId</td>
<td>M ∈ Memory</td>
<td>Tb ∈ TaskTable</td>
<td>rst ∈ RuntimeLang</td>
<td>dst ∈ DataLang</td>
</tr>
<tr>
<td>Lid ∈ Int</td>
<td>Cid ∈ Coreld</td>
<td>n ∈ Address</td>
<td>r ∈ Reference</td>
<td>T ∈ TaskId</td>
<td>M ∈ Memory</td>
<td>Tb ∈ TaskTable</td>
<td>rst ∈ RuntimeLang</td>
<td>dst ∈ DataLang</td>
</tr>
</tbody>
</table>

Config ::= M o T o Ca o CR

CR ∈ Core ::= Cid • rst

Ca ∈ Cache ::= Lev • M • dst

Lev ∈ Cacheld ::= lev(Cid, Lid, flag)

M ∈ MemBlocks ::= M[n→ st]

Status ::= {mo, sh, inv}

AccessPatterns ::= ε | dap | read(r) | write(r)

restart(r) | restart | restart dap

readBl(r) | writeBl(r)

fetchBl(n) | flush

Overbar denotes sets (e.g., CR)
Local Steps using SOS

Include
- Communications between each core and its first level cache
- Movement of data between caches in the same node
- Steps derived from the global communication
- Data update in main memory

Global steps using SOS

- Data movement between caches and the main memory
- Coordinate the signalling and the parallel transitions
Local Semantics – Abstract MSI Protocol

Cache Coherency Protocol: MSI (modified, shared, invalid)

Here $n = \text{addr}(r)$

$Ca \circ CR \rightarrow Ca' \circ CR'$

Violet Ka I Pun  
A Model of Multicore Execution

Local Semantics – Abstract MSI Protocol

Cache Coherency Protocol: MSI (modified, shared, invalid)

Here $n = \text{addr}(r)$

$(\text{Lev} \bullet M \bullet \text{dst}) \circ (\text{Cid} \bullet \text{rst}) \rightarrow 
(\text{Lev}' \bullet M' \bullet \text{dst}') \circ (\text{Cid}' \bullet \text{rst}')$

Violet Ka I Pun  
A Model of Multicore Execution
Local Semantics – \texttt{read}(r)

\begin{enumerate}
\item \texttt{PrRd_1}
\begin{align*}
\text{first}(Lev) &= \text{true} \quad \text{Cid}(Lev) = c \quad n = \text{addr}(r) \\
\text{status}(M, n) &= \text{sh} \lor \text{status}(M, n) = \text{mo}
\end{align*}
\begin{align*}
\text{(Lev \bullet M \bullet dst) \circ (c \bullet \text{read}(r); \text{rst})} & \rightarrow \\
\text{(Lev \bullet M \bullet dst) \circ (c \bullet \text{rst})}
\end{align*}
\item \texttt{PrRd_2}
\begin{align*}
\text{first}(Lev) &= \text{true} \quad \text{Cid}(Lev) = c \quad n = \text{addr}(r) \\
\text{status}(M, n) &= \text{inv} \lor n \notin \text{dom}(M)
\end{align*}
\begin{align*}
\text{(Lev \bullet M \bullet dst) \circ (c \bullet \text{read}(r); \text{rst})} & \rightarrow \\
\text{(Lev \bullet M \bullet dst) \circ (c \bullet \text{rst})}
\end{align*}
\item \texttt{PrRd_3}
\begin{align*}
\text{first}(Lev) &= \text{true} \quad \text{Cid}(Lev) = c \quad n = \text{addr}(r) \\
\text{status}(M, n) &= \text{sh}
\end{align*}
\begin{align*}
\text{(Lev \bullet M \bullet dst) \circ (c \bullet \text{write}(r); \text{rst})} & \rightarrow \\
\text{(Lev \bullet M \bullet dst) \circ (c \bullet \text{rst})}
\end{align*}
\item \texttt{PrRd_4}
\begin{align*}
\text{first}(Lev) &= \text{true} \quad \text{Cid}(Lev) = c \\
\text{n = \text{addr}(r)} \quad \text{status}(M, n) &= \text{inv}
\end{align*}
\begin{align*}
\text{(Lev \bullet M \bullet dst) \circ (c \bullet \text{readBl}(r); \text{rst})} & \rightarrow \\
\text{(Lev \bullet M \bullet dst) \circ (c \bullet \text{readBl}(r); \text{rst})}
\end{align*}
\end{enumerate}

Violet Ka I Pun

Local Semantics – \texttt{write}(r)

\begin{enumerate}
\item \texttt{PrWt_1}
\begin{align*}
\text{first}(Lev) &= \text{true} \quad \text{Cid}(Lev) = c \quad n = \text{addr}(r) \\
\text{status}(M, n) &= \text{mo}
\end{align*}
\begin{align*}
\text{(Lev \bullet M \bullet dst) \circ (c \bullet \text{write}(r); \text{rst})} & \rightarrow \\
\text{(Lev \bullet M \bullet dst) \circ (c \bullet \text{rst})}
\end{align*}
\item \texttt{PrWt_2}
\begin{align*}
\text{first}(Lev) &= \text{true} \quad \text{Cid}(Lev) = c \\
\text{n = \text{addr}(r)} \quad \text{status}(M, n) &= \text{sh}
\end{align*}
\begin{align*}
\text{(Lev \bullet M \bullet dst) \circ (c \bullet \text{write}(r); \text{rst})} & \rightarrow \\
\text{(Lev \bullet M [n \rightarrow \text{mo}] \bullet dst) \circ (c \bullet \text{rst})}
\end{align*}
\item \texttt{PrWt_3}
\begin{align*}
\text{first}(Lev) &= \text{true} \quad \text{Cid}(Lev) = c \\
\text{n = \text{addr}(r)} \quad \text{status}(M, n) &= \text{inv} \lor n \notin \text{dom}(M)
\end{align*}
\begin{align*}
\text{(Lev \bullet M \bullet dst) \circ (c \bullet \text{writeBl}(r); \text{rst})} & \rightarrow \\
\text{(Lev \bullet M [n \rightarrow \text{mo}] \bullet dst \text{: fetch}(n)) \circ (c \bullet \text{writeBl}(r); \text{rst})}
\end{align*}
\item \texttt{PrWt_4}
\begin{align*}
\text{first}(Lev) &= \text{true} \quad \text{Cid}(Lev) = c \\
\text{n = \text{addr}(r)} \quad \text{status}(M, n) &= \text{mo}
\end{align*}
\begin{align*}
\text{(Lev \bullet M \bullet dst) \circ (c \bullet \text{writeBl}(r); \text{rst})} & \rightarrow \\
\text{(Lev \bullet M \bullet dst) \circ (c \bullet \text{rst})}
\end{align*}
\end{enumerate}

Violet Ka I Pun
Local Semantics – Movement of Data Between Caches

\[ C_{id}(Lev_i) = C_{id}(Lev_j) \]
\[ \text{status}(M_j, n) = s_j, s_j \in \{sh, mo\} \]
\[ \text{Lid}(Lev_j) = \text{Lid}(Lev_j) + 1 \]
\[ \text{select}(M_i, n) = n_i \]
\[ \text{status}(M_i, n_i) = s_i \]

\[ (Lev_i \cdot M_j \cdot \text{fetch}(n); dst_i) \rightarrow (Lev_i \cdot M_j \cdot \text{dst}_j) \]

\[ \text{status}(M_j, n) = \text{inv} \cup n \notin \text{dom}(M_j) \]
\[ \text{Lid}(Lev_j) = \text{Lid}(Lev_j) + 1 \]
\[ \text{Cid}(Lev_j) = \text{Cid}(Lev_j) \]

\[ (Lev_i \cdot M_j \cdot \text{fetch}(n); dst_j) \rightarrow (Lev_i \cdot M_j \cdot \text{dst}_j; \text{fetch}(n)) \]

\[ \text{last}(Lev) = \text{true} \]
\[ \text{status}(M, n) = \text{inv} \cup n \notin \text{dom}(M) \]

\[ (Lev \cdot M \cdot \text{fetch}(n); dst) \rightarrow (Lev \cdot M[n \rightarrow \perp]; \text{fetchBl}(n); dst) \]

\[ (\text{flush-one-line}) \]
\[ \text{status}(M, n) = \text{mo} \]

\[ (Lev \cdot M \cdot \text{flush}(n); dst) \rightarrow (Lev \cdot M \cdot \text{flush}(n); dst) \]

\[ (\text{inv-one-line}) \]
\[ \text{status}(M, n) = \text{sh} \]

\[ (Lev \cdot M \cdot \text{flush}(n); dst) \rightarrow (Lev \cdot M[n \rightarrow \text{inv}]; dst) \]

\[ (\text{fetch}) \]
\[ \text{last}(Lev) = \text{true} \]
\[ \text{select}(M_j, n) = n \]
\[ \text{status}(M, n) = \text{sh} \]

\[ M \circ (Lev \cdot M_j \cdot \text{fetchBl}(n); dst) \rightarrow M \circ (Lev \cdot M_j[n \rightarrow \text{sh}]; dst) \]

Local Semantics of the Main Memory

\[ (\text{inv-main-memory}) \]
\[ M \xrightarrow{\text{flush}(n)} M[n \rightarrow \text{inv}] \]

\[ (\text{flush}) \]
\[ \text{status}(M, n) = \text{mo} \]

\[ M \circ (Lev \cdot M_j \cdot \text{flush}(n); dst) \rightarrow M \circ (Lev \cdot M_j[n \rightarrow \text{sh}]; \text{dst}) \]

\[ (\text{fetch}) \]
\[ \text{last}(Lev) = \text{true} \]
\[ \text{select}(M_j, n) = n \]
\[ \text{status}(M, n) = \text{sh} \]

\[ M \circ (Lev \cdot M_j \cdot \text{fetchBl}(n); dst) \rightarrow M \circ (Lev \cdot M_j[n \rightarrow \text{sh}]; \text{dst}) \]
Global steps: Coordinate signalling and Parallel Transitions

Tasks waiting to be scheduled

Node 1

\{!Rd(n), !RdX(m)\}

Node 2

\{!Rd(n), !RdX(m)\}

Node N

\{!RdX(m), !Rd(n)\}

\{?Rd(n), ?RdX(m)\}

\{?Rd(n), ?RdX(m)\}

Abstract communication medium

Main memory

\{?Rd(n), ?RdX(m)\}

\{?RdX(m), !Rd(n)\}

\{!RdX(m), !Rd(n)\}

\{!RdX(m), !RdX(m)\}

\{?Rd(n), ?RdX(m)\}

\{?RdX(m), !Rd(n)\}

Global Semantics – Synchronous Step

(SYNCH)

\( S \neq \emptyset \cap \text{allAddrIn}(S) = \emptyset \)

\( R = \text{dual}(S) \)

\( M \xrightarrow{R} M' \quad \overrightarrow{\mathcal{C} \circ \mathcal{C} R} \quad \overrightarrow{\mathcal{C}' \circ \mathcal{C}' R} \)

\( M \circ \overrightarrow{T} \circ \mathcal{C} \circ \mathcal{C} R \xrightarrow{S} \quad M' \circ \overrightarrow{T} \circ \mathcal{C}' \circ \mathcal{C}' R \)

(SYNCH)

\( \text{belongs}(\mathcal{C} \alpha_1, \mathcal{C} R_1) \quad \text{belongs}(\mathcal{C} \alpha_2, \mathcal{C} R_2) \)

\( S = S_1 \cup S_2 \quad R_1 = \text{dual}(S_1) \quad R_2 = \text{dual}(S_2) \)

\( \overrightarrow{\mathcal{C} \alpha_1 \circ \mathcal{C} R_1} \xrightarrow{S_{1 \cup R_1 \cup R_2}} \overrightarrow{\mathcal{C} \alpha_1 \circ \mathcal{C} R_1'} \quad \overrightarrow{\mathcal{C} \alpha_2 \circ \mathcal{C} R_2} \xrightarrow{S_{2 \cup R_2 \cup R_2}} \overrightarrow{\mathcal{C} \alpha_2' \circ \mathcal{C} R_2'} \)

\( \overrightarrow{\mathcal{C} \alpha_1 \circ \mathcal{C} R_1 \circ \mathcal{C} \alpha_2 \circ \mathcal{C} R_2} \xrightarrow{S_{1 \cup R_1 \cup R_2}} \overrightarrow{\mathcal{C} \alpha_1' \circ \mathcal{C} R_1' \circ \mathcal{C} \alpha_2' \circ \mathcal{C} R_2'} \)

W ::= !Rd(n) | !RdX(n)

S ::= S \cup S

Q ::= ?Rd(n) | ?RdX(n)

R ::= S \cup S

Violet Ka I Pun
A Model of Multicore Execution
Global Semantics –
Asynchronous Step, Scheduling and Spawning

\[ CR = CR_1 \cup CR_2 \cup CR_3 \]
\[ Ca = Ca_1 \cup Ca_2 \cup Ca_3 \cup Ca_4 \]
\[ CR' = CR_1 \cup CR_2 \cup CR_3 \]
\[ Ca' = Ca_1 \cup Ca_2 \cup Ca_3 \cup Ca_4 \]
\[ M \circ Ca \rightarrow M' \circ Ca' \]
\[ T \circ CR \rightarrow T' \circ CR' \]

(Task-Scheduler)

\[ T' = T \setminus \{ T \} \]
\[ dap = Tb(T) \]
\[ T' \circ CR \rightarrow T'' \circ CR' \]
\[ \forall Cid \in \circ \rightarrow (Cid \cdot dap; commit) \]

(Task-Spawn)

\[ T' = T \cup \{ T \} \]
\[ T' \circ CR \rightarrow T'' \circ CR' \]
\[ \forall Cid \in \circ \rightarrow (Cid \cdot spawn(T); dap) \rightarrow (Cid \cdot dap) \]

Correctness

Correctness properties expressed as invariants over any given number of cores and caches

Local and global histories and version number for book keeping to prove properties

- **Program Order:**
  Each core executes a task in program order

- **Data Race Freedom:**
  Single write access and parallel read access to the memory blocks

- **Consistent Shared Copies:**
  Shared copies of a memory block \( n \) in different cores are always consistent

- **Most Recent Value:**
  If a core succeeds to access a memory block, it will get the most recent value
Complementary to the semantics: it makes the semantics executable.

Given a program:

- **Compare the execution of the same program with different setups:** number of cores, number of caches, cache associativity, data layout, etc.
- **Measurements:** e.g., penalties related to data movement. Fetch instructions get more expensive the further we go into the memory hierarchy.
- **Search for bad states:** e.g., invariants (single writer, multiple readers to the same memory block).

**Maude**
- a specification and analysis system based on rewriting logic (RL),
- has been proposed as a framework for language semantics.

Different components in our models are terms and their behaviour is described with rewriting rules and equations.
Example

- **Architectures:**
  - three cores with one, two, and three level of private caches.
- Consider 3 tasks in form of data access patterns.
- Consider three different data layouts.
- Penalties of 1, 10, 100, and 1000 to access L1, L2, L3 and main memory.

![Diagram of memory hierarchy and data access patterns]

Example – Results

- Execution of the model in the Maude proof of concept implementation, for the nine considered scenarios.
- With dispersed data layout, the scenarios need to perform many evictions and fetch operations to access data from main memory.
- With three levels of cache, the penalty is substantially lower.

![Graph showing performance results]

\(^2\)The full Maude specification with additional examples can be downloaded from [http://folk.uio.no/shijib/multilevel.zip](http://folk.uio.no/shijib/multilevel.zip)
Conclusions

- **Our model explains:**
  - Task execution expressed as data-access patterns (e.g., read/write) on cache coherent multicore architectures with multilevel caches.
  - Data movement during task execution in a configurable way.
  - Influence of design choices and data layout.

- **Possible applications:**
  - Study the effects of scheduling.
  - Assist dynamic data allocation and design locking disciplines etc.
  - Study different architectures with shared caches, distributed memory, etc.

Questions?
Lecture 21

Temporal Logics

by Volker Stolz
Introduction

Temporal Logic?

- Temporal logic is the logic of “time”\(^a\)
- It is a modal logic.
- There are different ways of modeling time.
  - linear time vs. branching time
  - time instances vs. time intervals
  - discrete time vs. continuous time
  - past and future vs. future only

\(^a\)pay attention, it will be something kind of abstract, it’s mostly not what’s known as real-time, but there are variants of temporal logics which can handle real-time. They won’t occur in this lecture.
First Order Logic

- We have used FOL to express properties of states.
  - \( \langle x : 21, y : 49 \rangle \models x < y \)
  - \( \langle x : 21, y : 7 \rangle \notmodels x < y \)
- A computation is a sequence of states.
- To express properties of computations, we need to extend FOL.
- This we can do using temporal logic.

LTL: speaking about "time"

In Linear Temporal Logic (LTL) (also called linear-time temporal logic) we can describe such properties as follows: assume time is a sequence\(^1\) of discrete points \( i \) in time, then: if \( i \) is now,

- \( p \) holds in \( i \) and every following point (the future)
- \( p \) holds in \( i \) and every preceding point (the past)

We will only be concerned with the future.

\[ \cdots \rightarrow p_{i-2} \rightarrow p_{i-1} \rightarrow p_i \rightarrow p_{i+1} \rightarrow p_{i+2} \rightarrow \cdots \]

\(^1\) a sequence is linear
LTL operators

We extend our first-order language \( \mathcal{L} \) to a temporal language \( \mathcal{L}_T \) by adding the temporal operators \( \Box, \Diamond, \bigcirc, U, R \) and \( W \).

### Interpretation of the operators

- \( \Box \varphi \): \( \varphi \) will always (in every state) hold
- \( \Diamond \varphi \): \( \varphi \) will eventually (in some state) hold
- \( \bigcirc \varphi \): \( \varphi \) will hold at the next point in time
- \( \varphi U \psi \): \( \psi \) will eventually hold, and until that point \( \varphi \) will hold
- \( \varphi R \psi \): \( \psi \) holds until (incl.) the point (if any) where \( \varphi \) holds (release)
- \( \varphi W \psi \): \( \varphi \) will hold until \( \psi \) holds (weak until or waiting for)

\(^2\)Note: it’s equally ok to extend a propositional language the same way. The difference is between a first-order LTL or propositional LTL.

Syntax

We define LTL formulae as follows.

**Definition**

- \( \mathcal{L} \subseteq \mathcal{L}_T \): first-order formulae are also LTL formulae.
- If \( \varphi \) is an LTL formula, so are the following.
  
  \[ \Box \varphi, \Diamond \varphi, \bigcirc \varphi, \neg \varphi \]

- If \( \varphi \) and \( \psi \) are LTL formulae, so are
  \[ \varphi U \psi, \varphi R \psi, (\varphi W \psi) \]
  \[ (\varphi \lor \psi), (\varphi \land \psi), (\varphi \rightarrow \psi), (\varphi \leftrightarrow \psi) \]

- nothing else
Paths and computations

Definition

- A path is an infinite sequence
  \[ \sigma = s_0, s_1, s_2, \ldots \]
  of states.
- \( \sigma^k \) denotes the path \( s_k, s_{k+1}, s_{k+2}, \ldots \)
- \( \sigma_k \) denotes the state \( s_k \).
- All computations are paths, but not vice versa.

Satisfaction (semantics)

Definition

We define the notion that an LTL formula \( \varphi \) is true (false) relative to a path \( \sigma \), written \( \sigma \models \varphi \) (\( \sigma \not\models \varphi \)) as follows.

- \( \sigma \models \varphi \) iff \( \sigma_0 \models \varphi \) when \( \varphi \in \mathcal{L} \)
- \( \sigma \models \neg \varphi \) iff \( \sigma \not\models \varphi \)
- \( \sigma \models \varphi \lor \psi \) iff \( \sigma \models \varphi \) or \( \sigma \models \psi \)
- \( \sigma \models \Box \varphi \) iff \( \sigma^k \models \varphi \) for all \( k \geq 0 \)
- \( \sigma \models \Diamond \varphi \) iff \( \sigma^k \models \varphi \) for some \( k \geq 0 \)
- \( \sigma \models \Diamond \varphi \) iff \( \sigma^1 \models \varphi \)

(cont.)
Definition (cont.)

\[ \sigma \models \varphi U \psi \iff \sigma^k \models \psi \text{ for some } k \geq 0, \text{ and } \sigma^i \models \varphi \text{ for every } i \text{ such that } 0 \leq i < k \]

\[ \sigma \models \varphi R \psi \iff \text{for every } j \geq 0, \]

\[ \text{if } \sigma^j \not\models \varphi \text{ for every } i < j \text{ then } \sigma^j \models \psi \]

\[ \sigma \models \varphi W \psi \iff \sigma \models \varphi U \psi \text{ or } \sigma \models \Box \varphi \]

Validity and semantic equivalence

Definition

- We say that \( \varphi \) is (temporally) valid, written \( \models \varphi \), if \( \sigma \models \varphi \) for all paths \( \sigma \).
- We say that \( \varphi \) and \( \psi \) are equivalent, written \( \varphi \sim \psi \), if \( \models \varphi \leftrightarrow \psi \) (i.e. \( \sigma \models \varphi \iff \sigma \models \psi \), for all \( \sigma \)).

Example

\( \Box \) distributes over \( \land \), while \( \Diamond \) distributes over \( \lor \).

\[ \Box(\varphi \land \psi) \sim (\Box \varphi \land \Box \psi) \]

\[ \Diamond(\varphi \lor \psi) \sim (\Diamond \varphi \lor \Diamond \psi) \]
### Semantics

\[ \sigma \models \square p \]

\[ \begin{array}{c}
  \bullet^p_0 \\
  \rightarrow \\
  \bullet^p_1 \\
  \rightarrow \\
  \bullet^p_2 \\
  \rightarrow \\
  \bullet^p_3 \\
  \rightarrow \\
  \bullet^p_4 \\
  \rightarrow \ldots
\end{array} \]

\[ \sigma \models \Diamond p \]

\[ \begin{array}{c}
  \bullet_0 \\
  \rightarrow \\
  \bullet_1 \\
  \rightarrow \\
  \bullet_2 \\
  \rightarrow \\
  \bullet^p_3 \\
  \rightarrow \\
  \bullet_4 \\
  \rightarrow \ldots
\end{array} \]

\[ \sigma \models \Box p \]

\[ \begin{array}{c}
  \bullet_0 \\
  \rightarrow \\
  \bullet^p_1 \\
  \rightarrow \\
  \bullet_2 \\
  \rightarrow \\
  \bullet_3 \\
  \rightarrow \\
  \bullet_4 \\
  \rightarrow \ldots
\end{array} \]

\[ \sigma \models p U q \text{ (sequence of } p \text{'s is finite)} \]

\[ \begin{array}{c}
  \bullet^p_0 \\
  \rightarrow \\
  \bullet^p_1 \\
  \rightarrow \\
  \bullet^p_2 \\
  \rightarrow \\
  \bullet^q_3 \\
  \rightarrow \\
  \bullet_4 \\
  \rightarrow \ldots
\end{array} \]

\[ \sigma \models p R q \text{ (The sequence of } q \text{'s may be infinite)} \]

\[ \begin{array}{c}
  \bullet^q_0 \\
  \rightarrow \\
  \bullet^q_1 \\
  \rightarrow \\
  \bullet^q_2 \\
  \rightarrow \\
  \bullet^p^q_3 \\
  \rightarrow \\
  \bullet_4 \\
  \rightarrow \ldots
\end{array} \]

\[ \sigma \models p W q \text{. The sequence of } p \text{'s may be infinite.} \]

\[ (p W q \sim p U q \lor \Box p). \]

\[ \begin{array}{c}
  \bullet^p_0 \\
  \rightarrow \\
  \bullet^p_1 \\
  \rightarrow \\
  \bullet^p_2 \\
  \rightarrow \\
  \bullet^p_3 \\
  \rightarrow \\
  \bullet^p_4 \\
  \rightarrow \ldots
\end{array} \]
The past

Observation

- [Manna and Pnueli, 1992] uses pairs \((\sigma, j)\) of paths and positions instead of just the path \(\sigma\) because they have **past-formulae**: formulae without future operators (the ones we use) but possibly with **past operators**, like \(\Box^{-1}\) and \(\Diamond^{-1}\).

\[
(\sigma, j) \models \Box^{-1} \varphi \quad \text{iff} \quad (\sigma, k) \models \varphi \text{ for all } k, \ 0 \leq k \leq j \\
(\sigma, j) \models \Diamond^{-1} \varphi \quad \text{iff} \quad (\sigma, k) \models \varphi \text{ for some } k, \ 0 \leq k \leq j
\]

- However, it can be shown that for any formula \(\varphi\), there is a **future-formula** (formulae without past operators) \(\psi\) such that

\[
(\sigma, 0) \models \varphi \quad \text{iff} \quad (\sigma, 0) \models \psi
\]

The past: examples

Example

What is a future version of \(\Box (p \rightarrow \Diamond^{-1} q)\)?

\((\sigma, 0) \models \Box (p \rightarrow \Diamond^{-1} q)\)

\[
\bullet \rightarrow \Diamond^{-1} q \rightarrow \bullet \rightarrow \Diamond^{-1} q \rightarrow \bullet \rightarrow \Diamond^{-1} q \rightarrow \bullet \rightarrow \Diamond^{-1} q \rightarrow \bullet
\]

\((\sigma, 0) \models q R (p \rightarrow q)\)

\[
\bullet \rightarrow q \rightarrow \bullet \rightarrow q \rightarrow \bullet \rightarrow q, q \rightarrow \bullet \rightarrow \bullet \rightarrow \bullet \rightarrow \ldots
\]
Examples

Example
\( \varphi \rightarrow \diamond \psi \): If \( \varphi \) holds initially, then \( \psi \) holds eventually.

\[
\begin{array}{cccccccc}
\cdot \varphi & \rightarrow & \cdot & \rightarrow & \cdot & \rightarrow & \cdot \psi & \rightarrow & \cdot & \rightarrow & \ldots
\end{array}
\]

This formula will also hold in every path where \( \varphi \) does not hold initially.

\[
\begin{array}{cccccccc}
\cdot & \neg \varphi & \rightarrow & \cdot & \rightarrow & \cdot & \rightarrow & \cdot & \rightarrow & \cdot & \rightarrow & \ldots
\end{array}
\]

Example: Response

Example (Response)
\( \Box(\varphi \rightarrow \diamond \psi) \)
Every \( \varphi \)-position coincides with or is followed by a \( \psi \)-position.

\[
\begin{array}{cccccccc}
\cdot & \rightarrow & \cdot \varphi & \rightarrow & \cdot & \rightarrow & \cdot \psi & \rightarrow & \cdot & \rightarrow & \cdot \varphi \psi & \rightarrow & \ldots
\end{array}
\]

This formula will also hold in every path where \( \varphi \) never holds.

\[
\begin{array}{cccccccc}
\cdot & \neg \varphi & \rightarrow & \cdot & \neg \varphi & \rightarrow & \cdot & \neg \varphi & \rightarrow & \cdot & \neg \varphi & \rightarrow & \cdot & \neg \varphi & \rightarrow & \ldots
\end{array}
\]
Example
\(\square \Diamond \psi\)
There are infinitely many \(\psi\)-positions.
\(\bullet^\psi \rightarrow \bullet \rightarrow \bullet \rightarrow \bullet^\psi \rightarrow \bullet \rightarrow \bullet^\psi \rightarrow \bullet \rightarrow \ldots\)

This formula can be obtained from the previous one, \(\square (\varphi \rightarrow \Diamond \psi)\), by letting \(\varphi = T: \square (T \rightarrow \Diamond \psi)\).

Example: permanence
Example
\(\Diamond \Box \varphi\)
Eventually \(\varphi\) will hold permanently.
\(\bullet \rightarrow \bullet^\varphi \rightarrow \bullet \rightarrow \bullet \rightarrow \bullet^\varphi \rightarrow \bullet^\varphi \rightarrow \ldots\)

Equivalently: there are finitely many \(\neg \varphi\)-positions.
Example

\((\neg \varphi) W \psi\)

[WRONG SENTENCE] The first \(\varphi\)-position must coincide or be preceded by a \(\psi\)-position.

\(\quad \bullet \neg \varphi \rightarrow \bullet \neg \varphi \rightarrow \bullet \neg \varphi \rightarrow \bullet \psi \rightarrow \bullet \varphi \rightarrow \bullet \rightarrow \bullet \rightarrow \ldots\)

\(\varphi\) may never hold

\(\quad \bullet \neg \varphi \rightarrow \bullet \neg \varphi \rightarrow \bullet \neg \varphi \rightarrow \bullet \neg \varphi \rightarrow \bullet \neg \varphi \rightarrow \bullet \neg \varphi \rightarrow \bullet \neg \varphi \rightarrow \bullet \neg \varphi \rightarrow \ldots\)

LTL Example

Example

\(\Box(\varphi \rightarrow \psi W \chi)\)

Every \(\varphi\)-position initiates a sequence of \(\psi\)-positions, and if terminated, by a \(\chi\)-position.

\(\quad \bullet \rightarrow \bullet \varphi, \psi \rightarrow \bullet \psi \rightarrow \bullet \psi \rightarrow \bullet \chi \rightarrow \bullet \rightarrow \bullet \varphi, \psi \rightarrow \ldots\)

The sequence of \(\psi\)-positions need not terminate.

\(\quad \bullet \rightarrow \bullet \varphi, \psi \rightarrow \bullet \psi \rightarrow \bullet \psi \rightarrow \bullet \psi \rightarrow \bullet \psi \rightarrow \bullet \psi \rightarrow \ldots\)
Nested waiting-for

A nested waiting-for formula is of the form

\[ \Box(\varphi \rightarrow (\psi_m W (\psi_{m-1} W \cdots (\psi_1 W \psi_0) \cdots ))), \]

where \( \varphi, \psi_0, \ldots, \psi_m \in \mathcal{L} \). For the sake of convenience, we write

\[ \Box(\varphi \rightarrow \psi_m W \psi_{m-1} W \cdots W \psi_1 W \psi_0). \]

Every \( \varphi \)-position initiates a succession of intervals, beginning with a \( \psi_m \)-interval, ending with a \( \psi_1 \)-interval and possibly terminated by a \( \psi_0 \)-position. Each interval may be empty or extend to infinity.

\[ \ldots \rightarrow \bullet \varphi, \psi_m \rightarrow \bullet \psi_m \rightarrow \bullet \psi_m \rightarrow \bullet \psi_m\rightarrow \bullet \psi_{m-1} \rightarrow \bullet \psi_{m-1} \rightarrow \ldots \]

\[ \ldots \rightarrow \bullet \psi_2 \rightarrow \bullet \psi_2 \rightarrow \bullet \psi_1 \rightarrow \bullet \psi_1 \rightarrow \bullet \psi_0 \rightarrow \ldots \]

Capturing informally understood temporal specifications formally

It can be difficult to correctly formalize informally stated requirements in temporal logic.

**Example**

How does one formalize the informal requirement “\( \varphi \) implies \( \psi \)”?

- \( \varphi \rightarrow \psi \)? \( \varphi \rightarrow \psi \) holds in the initial state.
- \( \Box(\varphi \rightarrow \psi) \)? \( \varphi \rightarrow \psi \) holds in every state.
- \( \varphi \rightarrow \Diamond \psi \)? \( \varphi \) holds in the initial state, \( \psi \) will hold in some state.
- \( \Box(\varphi \rightarrow \Diamond \psi) \)? We saw this earlier.
- None of these is necessarily what we intended.
Definition (Duals)

For binary boolean connectives $\circ$ and $\bullet$, we say that $\bullet$ is the dual of $\circ$ if

$$\neg(\varphi \circ \psi) \sim (\neg \varphi \bullet \neg \psi).$$

Similarly for unary connectives: $\bullet$ is the dual of $\circ$ if $\neg \circ \varphi \sim \bullet \neg \varphi$.

*Those are not concrete connectives or operators, they are meant as “placeholders”*

Duality is symmetric:

- If $\bullet$ is the dual of $\circ$ then
- $\circ$ is the dual of $\bullet$, thus
- we may refer to two connectives as dual (of each other).

## Dual connectives

Which connectives are duals?

- $\wedge$ and $\vee$ are duals:
  $$\neg(\varphi \wedge \psi) \sim (\neg \varphi \vee \neg \psi).$$

- $\neg$ is its own dual:
  $$\neg \neg \varphi \sim \neg \neg \neg \varphi.$$

- What is the dual of $\rightarrow$? It’s $\not\rightarrow$:
  $$\neg(\varphi \not\rightarrow \psi) \sim \varphi \not\leftarrow \psi$$
  $$\sim \psi \rightarrow \varphi$$
  $$\sim \neg \varphi \rightarrow \neg \psi$$
Complete sets of connectives

- A set of connectives is complete (for boolean formulae) if every other connective can be defined in terms of them.
- Our set of connectives is complete (e.g., \( \neq \) can be defined), but also subsets of it, so we don’t actually need all the connectives.

**Example**

\( \{ \lor, \neg \} \) is complete.

- \( \land \) is the dual of \( \lor \).
- \( \varphi \rightarrow \psi \) is equivalent to \( \neg \varphi \lor \psi \).
- \( \varphi \leftrightarrow \psi \) is equivalent to \( (\varphi \rightarrow \psi) \land (\psi \rightarrow \varphi) \).
- \( T \) is equivalent to \( p \lor \neg p \)
- \( \bot \) is equivalent to \( p \land \neg p \)

Duals in LTL

We can extend the notions of duality and completeness to temporal formulae.

**Duals of temporal operators**

- What is the dual of \( \Box \)? And of \( \Diamond \)?
- \( \Box \) and \( \Diamond \) are duals.

\[
\neg \Box \varphi \sim \Diamond \neg \varphi \\
\neg \Diamond \varphi \sim \Box \neg \varphi
\]

- Any other?
- \( U \) and \( R \) are duals.

\[
\neg (\varphi U \psi) \sim (\neg \varphi)R(\neg \psi) \\
\neg (\varphi R \psi) \sim (\neg \varphi)U(\neg \psi)
\]
Complete set of LTL operators

We don’t need all our temporal operators either.

Proposition

\{V, \neg, U, \bigcirc\} \text{ is complete for LTL.}

Proof:

\begin{align*}
\Diamond \varphi & \sim T U \varphi \\
\Box \varphi & \sim \bot R \varphi \\
\varphi R \psi & \sim \neg (\neg \varphi U \neg \psi) \\
\varphi W \psi & \sim \Box \varphi \lor (\varphi U \psi)
\end{align*}

Classification of properties

We can classify properties expressible in LTL.

Classification

<table>
<thead>
<tr>
<th>Safety</th>
<th>\Box \varphi</th>
</tr>
</thead>
<tbody>
<tr>
<td>Liveness</td>
<td>\Diamond \varphi</td>
</tr>
<tr>
<td>Obligation</td>
<td>\Box \varphi \lor \Diamond \psi</td>
</tr>
<tr>
<td>Recurrence</td>
<td>\Diamond \Box \varphi</td>
</tr>
<tr>
<td>Persistence</td>
<td>\Diamond \Box \varphi</td>
</tr>
<tr>
<td>Reactivity</td>
<td>\Box \Diamond \varphi \lor \Diamond \Box \psi</td>
</tr>
</tbody>
</table>
Safety

- important basic class of properties
- relation to testing and run-time verification
- “nothing bad ever happens”

Definition (Safety)

- A safety formula is of the form
  \[ \Box \varphi \]
  for some first-order formula \( \varphi \).
- A conditional safety formula is of the form
  \[ \varphi \rightarrow \Box \psi \]
  for (first-order) formulae \( \varphi \) and \( \psi \).
- Safety formulae express invariance of some state property \( \varphi \): that \( \varphi \) holds in every state of the computation.

Safety property example

Example

- Mutual exclusion is a safety property. Let \( C_i \) denote that process \( P_i \) is executing in the critical section. Then
  \[ \Box \neg (C_1 \land C_2) \]
  expresses that it should always be the case that not both \( P_1 \) and \( P_2 \) are executing in the critical section.
- Observe that the negation of a safety formula is a liveness formula; the negation of the formula above is the liveness formula
  \[ \Diamond (C_1 \land C_2) \]
  which expresses that eventually it is the case that both \( P_1 \) and \( P_2 \) are executing in the critical section.
Liveness properties

Definition (Liveness)
- A liveness formula is of the form
  \[ \Diamond \varphi \]
  for some first-order formula \( \varphi \).
- A conditional liveness formula is of the form
  \[ \varphi \rightarrow \Diamond \psi \]
  for first-order formulae \( \varphi \) and \( \psi \).
- Liveness formulae guarantee that some event \( \varphi \) eventually happens: that \( \varphi \) holds in at least one state of the computation.

Connection to Hoare logic

Observation
- Partial correctness is a safety property. Let \( P \) be a program and \( \psi \) the post condition.
  \[ \Box (\text{terminated}(P) \rightarrow \psi) \]
- In the case of full partial correctness, where there is a precondition \( \varphi \), we get a conditional safety formula,
  \[ \varphi \rightarrow \Box (\text{terminated}(P) \rightarrow \psi), \]
  which we can express as \{ \( \varphi \) \} \( P \) \{ \( \psi \) \} in Hoare Logic.
Total correctness and liveness

Observation
- Total correctness is a liveness property. Let $P$ be a program and $\psi$ the post condition.

$$\Diamond (\text{terminated}(P) \land \psi)$$

- In the case of full total correctness, where there is a precondition $\varphi$, we get a conditional liveness formula,

$$\varphi \rightarrow \Diamond (\text{terminated}(P) \land \psi).$$

Duality of partial and total correctness

Observation
Partial and total correctness are dual. Let

$$PC(\psi) \triangleq \Box(\text{terminated} \rightarrow \psi)$$
$$TC(\psi) \triangleq \Diamond (\text{terminated} \land \psi)$$

Then

$$\neg PC(\psi) \sim PC(\neg \psi)$$
$$\neg TC(\psi) \sim TC(\neg \psi)$$
Definition (Obligation)

- A simple obligation formula is of the form

\[ \Box \varphi \lor \Diamond \psi \]

for first-order formula \( \varphi \) and \( \psi \).

- An equivalent form is

\[ \Diamond \chi \rightarrow \Diamond \psi \]

which states that some state satisfies \( \chi \) only if some state satisfies \( \psi \).

Obligation (2)

Proposition

Every safety and liveness formula is also an obligation formula.

Proof: This is because of the following equivalences.

\[ \Box \varphi \sim \Box \varphi \lor \Diamond \bot \]
\[ \Diamond \varphi \sim \Box \bot \lor \Diamond \varphi \]

and the facts that \( \models \neg \Box \bot \) and \( \models \neg \Diamond \bot \).
Recurrence

**Definition (Recurrence)**
- A recurrence formula is of the form
  \[ \Box \Diamond \varphi \]
  for some first-order formula \( \varphi \).
- It states that infinitely many positions in the computation satisfies \( \varphi \).

**Observation**
A response formula, of the form \( \Box(\varphi \rightarrow \Diamond \psi) \), is equivalent to a recurrence formula, of the form \( \Box \Diamond \chi \), if we allow \( \chi \) to be a past-formula.

\[ \Box(\varphi \rightarrow \Diamond \psi) \sim \Box \Diamond (\neg \varphi) W^{-1} \psi \]

Recurrence

**Proposition**
*Weak fairness*\(^a\) can be specified as the following recurrence formula.

\[ \Box \Diamond (\text{enabled}(\tau) \rightarrow \text{taken}(\tau)) \]

\(^a\)weak and strong fairness will be “recurrent” (sorry for the pun) themes. For instance they will show up again in the TLA presentation.

**Observation**
An equivalent form is

\[ \Box (\Box \text{enabled}(\tau) \rightarrow \Diamond \text{taken}(\tau)) \]

which looks more like the first-order formula we saw last time.
**Persistence**

**Definition (Persistence)**
- A *persistence* formula is of the form
  \[ \Diamond \Box \varphi \]
  for some first-order formula \( \varphi \).
- It states that all but finitely many positions satisfy \( \varphi \).
- Persistence formulae are used to describe the eventual *stabilization* of some state property.

---

**Recurrence and Persistence**

**Observation**

Recurrence and persistence are duals.

\[
\neg(\Box \Diamond \varphi) \sim (\Diamond \Box \neg \varphi) \\
\neg(\Diamond \Box \varphi) \sim (\Box \Diamond \neg \varphi)
\]
Reactivity

Definition (Reactivity)

- A simple reactivity formula is of the form
  \[ \Box \Diamond \varphi \lor \Diamond \Box \psi \]
  for first-order formula \( \varphi \) and \( \psi \).
- A very general class of formulae are conjunctions of reactivity formulae.
- An equivalent form is
  \[ \Box \Diamond \chi \rightarrow \Box \Diamond \psi, \]
  which states that if the computation contains infinitely many \( \chi \)-positions, it must also contain infinitely many \( \psi \)-positions.

Proposition

Strong fairness can be specified as the following reactivity formula.

\[ \Box \Diamond enabled(\tau) \rightarrow \Box \Diamond taken(\tau) \]
GCD Example

Below is a computation $\sigma$ of our recurring GCD program.

- $a$ and $b$ are fixed: $\sigma \models \Box(a \equiv 21 \land b \equiv 49)$.
- $at(l)$ denotes the formulae ($\pi \equiv \{l\}$).
- terminated denotes the formula $at(l_b)$.

$P$-computation

States are of the form $\langle \pi, x, y, g \rangle$.

\[
\sigma : \langle l_1, 21, 49, 0 \rangle \rightarrow \langle l_b^b, 21, 49, 0 \rangle \rightarrow \langle l_6, 21, 49, 0 \rangle \rightarrow \\
\langle l_1, 21, 28, 0 \rangle \rightarrow \langle l_b^b, 21, 28, 0 \rangle \rightarrow \langle l_6, 21, 28, 0 \rangle \rightarrow \\
\langle l_1, 21, 7, 0 \rangle \rightarrow \langle l_b^b, 21, 7, 0 \rangle \rightarrow \langle l_4, 21, 7, 0 \rangle \rightarrow \\
\langle l_1, 14, 7, 0 \rangle \rightarrow \langle l_b^b, 14, 7, 0 \rangle \rightarrow \langle l_4, 14, 7, 0 \rangle \rightarrow \\
\langle l_1, 7, 7, 0 \rangle \rightarrow \langle h_7, 7, 7, 0 \rangle \rightarrow \langle l_b^b, 7, 7, 7 \rangle \rightarrow \cdots
\]

GCD Example

Does the following properties hold for $\sigma$? And why?

1. $\Box$terminated (safety)
2. $at(l_1)$ → terminated
3. $at(l_b)$ → terminated
4. $at(l_7)$ → $\Diamond$terminated (conditional liveness)
5. $\Diamond at(l_7)$ → $\Diamond$terminated (obligation)
6. $\Box(gcd(x, y) \equiv gcd(a, b))$ (safety)
7. $\Diamond$terminated (liveness)
8. $\Diamond \Box(y \equiv gcd(a, b))$ (persistence)
9. $\Box \Diamond$terminated (recurrence)
Exercises

1. Show that the following formulae are (not) LTL-valid.
   1.1 $\Box \varphi \leftrightarrow \Box \Box \varphi$
   1.2 $\Diamond \varphi \leftrightarrow \Diamond \Diamond \varphi$
   1.3 $\neg \Box \varphi \rightarrow \Box \neg \varphi$
   1.4 $\Box (\Box \varphi \rightarrow \psi) \rightarrow \Box (\Box \psi \rightarrow \varphi)$
   1.5 $\Box (\Box \varphi \rightarrow \psi) \lor \Box (\Box \psi \rightarrow \varphi)$
   1.6 $\Box \Diamond \Box \varphi \rightarrow \Diamond \Box \varphi$
   1.7 $\Box \Diamond \varphi \leftrightarrow \Box \Diamond \Diamond \varphi$

2. A modality is a sequence of $\neg$, $\Box$ and $\Diamond$, including the empty sequence $\epsilon$. Two modalities $\sigma$ and $\tau$ are equivalent if $\sigma \varphi \leftrightarrow \tau \varphi$ is valid.
   2.1 Which are the non-equivalent modalities in LTL, and
   2.2 what are their relationship (ie. implication-wise)?

References I

The temporal logic of reactive and concurrent systems—Specification.
Springer Verlag, New York.
Lecture 22

QoS-aware Service Composition and Recommandation

by Bo Liu
QoS-aware Service Composition and Recommendation

Dr. Bo Liu

1. Centre for Research and Innovation in Software Engineering (RISE),
2. Faculty of Computer and Information Science (CIS),
   Southwest University, Chongqing, China
   Email: liubocq@swu.edu.cn; bob.liubo@gmail.com;
   URL: http://www.swu-rise.net.cn/bo.liu

Bergen, Norway, 2/3/2018

Outline

1. Introduction to QoS-aware SC
2. Q-SC model and algorithm
3. Improved Q-SC model and algorithm
4. QoS-aware service recommendation
Q1: What kind of service we study?

• Narrow sense: A service actually refers to a Web service, which is kind of program that is encapsulated by using Web services technologies (e.g., WSDL, UDDI, SOAP, REST, etc.) in SOA systems.
• Extended idea: Any networked resources of hardware or software can be encapsulated, virtualized, and accessed as services,
  - Infrastructure as a service (IaaS), Platform as a service (PaaS), Software as a service (SaaS)
• General thought: Everything as a service
  - Manufacturing resource as a service
  - Manufacturing capability as a service
  - Data as a service
  - Knowledge as a service
  - ......

Q2: Why service composition (SC)

• A single service (atomic service) is usually designed to provide a specific functionality. It may not be capable enough to fulfill the requirements of complex applications.
• Several services are required to be combined together to create multi-functionality composite services to undertake complex tasks.

An example (composite service) 
Travelling to China
Q3: Why QoS-aware SC (Q-SC)

- As many services provide identical functionalities but varying in non-functional features it is much necessary to introduce QoS properties (e.g., time, cost, reliability) into the process of Service Composition.
- A typical approach of QoS-aware Service Composition is to combines atomic services into a loosely coupled application (composite service), where the QoS properties of each service are adopted to guide the process of service composition to work out the optimal solution(s).
- When you planning your next travelling to China, you may need to find out the best composite solution with shortest time on journey, lowest money to spend, and least possible troubles.

<table>
<thead>
<tr>
<th>Taxi appointment service</th>
<th>Uber (cheap—with coupon, time consuming - less cars)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Didi (expensive, time saving - more cars &amp; skilled drivers)</td>
</tr>
</tbody>
</table>

Outline

1. Introduction to QoS-aware SC
2. Q-SC model and algorithm
3. Improved Q-SC model and algorithm
4. QoS-aware service recommendation
Traditional Framework of Q-SC

Step 1: Task decomposition
Output: A series of subtasks

Step 2: Service discovery
Output: Sets of candidate services

Step 3: Service composition
Output: Optimal composite service

Step 4: Service execution
Output: User acceptable results

Application of Q-SC into Cloud manufacturing
(A Case: Online Motorcycle Production)

The Q-CS Problem:

- How to make a decision on \textit{which candidate services} from respective candidate sets \textit{should be selected} to form an \textit{solution} with \textit{optimal} QoS evaluation.

- \textit{Given information of} 1) decomposition of subtasks, 2) candidate service set for each subtask, & 3) QoS properties of each candidate service.

- \textit{Based on the facts that} 1) The number of involved \textit{candidate service sets} (for \textit{subtasks}) may be large, 2) The number of \textit{candidate services} in each candidate set is likely to be larger.
The Q-SC model

- \( I \): Number of subtasks;
- \( i \in \{1 \ldots I\} \) denotes sequential number of a subtask (or Candidate Set);
- \( J_i \): Number of candidates for \( i \)-th subtask (or in \( i \)-th Candidate Set).
- \( K \): Number of possible composite service (CS), \( K = \prod_{i=1}^{I} J_i \);
- \( k \in \{1 \ldots K\} \) denotes the \( k \)-th possible CS.
- We need to find out the optimal solution from \( K \) possible CS:

\[
\text{Max QoS}(CS_k)
\]
**Basic GA**

The most responsive to change

- Survival of the fittest
- Good genes to next generation
- Casual mutation to changes
- Better individuals 

Selection operator  Crossover operator  Mutation operator

**Basic GA – an example**

\[ \text{max} f(x) = x^3 - 60x^2 + 900x + 100, \ s.t. \ x \in [0,30] \]
**Basic GA – an example**

\[ \max f(x) = x^3 - 60x^2 + 900x + 100, \quad \text{s.t. } x \in [0,30] \]

\[ F(x) = f(x) \]

Population initialization & Fitness computing

(19) (5) (26) (21) (14)

2399  3225  516  1801  3684

**GA operators - selection**

Survivals selection

\[ P_i = \frac{F_i}{\sum_{i=1}^{NP} F_i} \]

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10011</td>
<td>19</td>
<td>2399</td>
<td>0.206</td>
<td>0.206</td>
</tr>
<tr>
<td>2</td>
<td>00101</td>
<td>5</td>
<td>3225</td>
<td>0.277</td>
<td>0.483</td>
</tr>
<tr>
<td>3</td>
<td>11010</td>
<td>26</td>
<td>516</td>
<td>0.044</td>
<td>0.527</td>
</tr>
<tr>
<td>4</td>
<td>10101</td>
<td>21</td>
<td>1801</td>
<td>0.155</td>
<td>0.682</td>
</tr>
<tr>
<td>5</td>
<td>01110</td>
<td>14</td>
<td>3684</td>
<td>0.317</td>
<td>0.999</td>
</tr>
</tbody>
</table>

\[ \sum f(x) = 11625 \]
GA operators - selection

Survivals selection

<table>
<thead>
<tr>
<th>个体编号</th>
<th>染色体编码</th>
<th>问题的解</th>
<th>适  应度</th>
<th>选择概率</th>
<th>累计概率</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10011</td>
<td>19</td>
<td>2399</td>
<td>0.206</td>
<td>0.206</td>
</tr>
<tr>
<td>2</td>
<td>00101</td>
<td>5</td>
<td>3225</td>
<td>0.277</td>
<td>0.483</td>
</tr>
<tr>
<td>3</td>
<td>11010</td>
<td>26</td>
<td>516</td>
<td>0.044</td>
<td>0.527</td>
</tr>
<tr>
<td>4</td>
<td>10101</td>
<td>21</td>
<td>1801</td>
<td>0.155</td>
<td>0.682</td>
</tr>
<tr>
<td>5</td>
<td>01110</td>
<td>14</td>
<td>3684</td>
<td>0.317</td>
<td>0.999</td>
</tr>
</tbody>
</table>

Random number R between 0 and 1 is
0.070221  0.445929  0.784567  0.566930  0.854641

GA operators - crossover

Example: how a crossover operator work?

(19) (14) (14)  arrow  1001 1 01 110 01110
      (5) (21)

(19) (10) (14)  arrow  10011 01010 01110
      (5) (30)
Example: how a mutation operator work?

<table>
<thead>
<tr>
<th>10011</th>
<th>01010</th>
<th>0110</th>
</tr>
</thead>
<tbody>
<tr>
<td>00101</td>
<td>11110</td>
<td></td>
</tr>
</tbody>
</table>

After several iterations:

01010
(10)
4100

Code solutions as the Chromosome: $x_{11} \cdot x_{12} \cdot x_{13} \cdot x_{21} \cdot x_{22} \cdot x_{23} \cdot \ldots \cdot x_{61} \cdot x_{62} \cdot x_{63}$

The gene $x_{ij} = 1$ indicates the j-th service ($S_j$) in the i-th candidate service set ($CSS_i$) is selected.

Use the basic GA introduced previously, and work out the optimal (or near optimal) solutions in an acceptable duration.
Outline

1. Introduction to QoS-aware SC
2. Q-SC model and algorithm
3. Improved Q-SC model and algorithm
4. QoS-aware service recommendation

Drawbacks of traditional approach

- Single-task
- 1-1 mapping
- Trust is required

Multi-task

Large-scale SOA System
(Cloud manufacturing)

Motivation scenario 1: multi-task req.

- Given two OMP tasks $T_1$ and $T_2$, the QoS requirements are: $QoS\text{-}time_1<523$, $QoS\text{-}time_2<498$.
- Suppose the sets of candidate services for each subtask are given by the following figure.

The reason lies in the strategy of **single task oriented composition**:
The first composed service for $T_2$ used up all the best component services, without considering the inter-task constraints between $T_1$ and $T_2$. Thus, the second composition for $T_2$ may be lack of good enough component services.

Multi-task oriented composition

Idea of **multi-task oriented service composition**:
Conduct a **holistic service composition** of multiple composite services for both $T_1$ and $T_2$ simultaneously.

Although each composite service for $T_1$ or $T_2$ may not be optimum, both the requirements on $T_1$ and $T_2$ can be met.
Improvement to the framework

1. Task decomposition (multi-task):
\[ T = \{ ST_1, ST_2, ..., ST_i, ..., ST_f \} \]

2. Service discovery:
\[ CSS = \{ CSS^1, CSS^2, ..., CSS^j, ..., CSS^m \} \]
\[ CSS_i = \{ CS_{i1}, CS_{i2}, ..., CS_{ij}, ..., CS_{ik} \} \]

3. Service composition (holistic com.):
\[ CSS = \{ CSS_1^1, CSS_2^2, ..., CSS_j^j, ..., CSS_m^m \} \]
\[ \{ CS_{i1}^1, CS_{i2}^2, ..., CS_{ij}^j, ..., CS_{ik}^k \} \]

Find out the optimal holistic solution for all Tasks \( T = \{ T_i | i=1...f \} \).

Formulate a more general problem model:
- It responds to any number of requesting tasks and provides the best-effort composite services to meet the requirements of all tasks.
- Specially, when the number of requesting task \( I = 1 \), the multi-task oriented problem is specialized into a single task oriented problem.

\[ \max QoS(MCCS) \quad \rightarrow \quad \max \sum_{i=1}^{\lfloor I \rfloor} QoS(MCCS_i) \]
Algorithm to solve the problem

Hybrid-Operator based Matrix Coded Genetic Algorithm (HO-MCGA)

\[ X = \begin{bmatrix}
  a_{11} & a_{12} & \cdots & a_{1n} \\
  a_{21} & a_{22} & \cdots & a_{2n} \\
  \vdots & \vdots & \ddots & \vdots \\
  a_{m1} & a_{m2} & \cdots & a_{mn}
\end{bmatrix} \]

Mutation operators

Crossover operators

matrix coding manner

Col.-based operators

row-based operators

point-based operators

Simulation experiments

Multi-task requesting in Online Motorcycle Production

<table>
<thead>
<tr>
<th>Test cases</th>
<th>HO-MCGA</th>
<th>MCGA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Average fitness</td>
<td>Convergence generation time (sec.)</td>
</tr>
<tr>
<td>10:6 MCIs, 5 MFMTs</td>
<td>4.9756</td>
<td>118</td>
</tr>
<tr>
<td>20:6 MCIs, 15 MFMTs</td>
<td>11.4409</td>
<td>357</td>
</tr>
<tr>
<td>30:6 MCIs, 25 MFMTs</td>
<td>18.3657</td>
<td>428</td>
</tr>
<tr>
<td>40:6 MCIs, 35 MFMTs</td>
<td>25.8634</td>
<td>568</td>
</tr>
<tr>
<td>50:6 MCIs, 50 MFMTs</td>
<td>35.6764</td>
<td>622</td>
</tr>
</tbody>
</table>

Sound performance-wise
Motivation scenario 2: severe QoS

- Severe QoS: the given QoS constraints outnumber any plainly composed composite services.
- Given two OMP tasks $T_1$ and $T_2$, the QoS requirements are: $QoS-time_1 < 523$, $QoS-time_2 < 350$.
- Suppose the sets of candidate services for each subtask are given by the following figure.

The reason lies in one-to-one mapping based service composition (OOM-SC): Only one appropriate candidate atomic service is allowed to perform each subtask, while other functionally equivalent atomic services available to the same subtask are kept on the shelf.

Synergistic elementary service groups based service composition

The main idea of SESG-SC:
Release the assumption of one-to-one mapping between atomic services and subtasks, allowing free combination of multiple functionally equivalent atomic services into a synergistic elementary service group (SESG) to perform each subtask collectively.
**Improvement to the framework**

1. Task decomposition
2. Services discovery
3. SESG construction
4. Services composition

**Structure of SESGs**

- atomic service
- parallel structure
- selective structure

General Structure (Hybrid Structure) of a SESG

We use SESGs instead of atomic services as the component services for respective subtasks in service composition
QoS evaluation of SESGs

Let $T(\cdot)$, $C(\cdot)$, $R(\cdot)$ denotes the time, cost, and reliability property of QoS, respectively. Then, the formulas to calculate the QoS of $SESG_i$ are as follows:

$$C(SESG_i) = \sum_{j=1}^{J} C(s_{ij}) \times x_{ij} + \sum_{j=1}^{J} C(s_{ij}) \times p_{ij}$$

(3)

$$T(SESG_i) = \frac{1}{\sum_{j=1}^{J} \frac{1}{T(s_{ij})} \times x_{ij} + \sum_{j=1}^{J} \frac{1}{T(s_{ij})} \times p_{ij}}$$

(4)

$$R(SESG_i) = \left(\prod_{j=1}^{J} R(s_{ij})^{x_{ij} \times \sum_{j=1}^{J} x_{ij}} \times \frac{R(sel(SESG_i))}{R(sel(SESG_i))}\right)$$

(5)

Formulation of SESG-SC

Formulate a more general problem model than before:

- **At least one** atomic service should be **combined into** the corresponding **SESG** for each subtask.

- If all the **numbers** of atomic services in each SESG are **restricted to 1**, the SESG-SC model will be specialized into the **OOM-SC model**.
Algorithm to solve the problem

Matrix Coded Genetic Algorithm with Collaboratively Evolutilional Populations (MCGA-CEP)

START

Subpopulation #1 Initialization  Subpopulation #M Initialization

Fitness evaluation  update elitist #1  update elitist #M

Selection operator execution

Crossover operator execution

Mutation operator execution

Generate new sub-population #1

END

Simulation experiments

Severe QoS constraints in Online Motorcycle Production

Still, it requires further improvement on performance

Outline

1. Introduction to QoS-aware SC
2. Q-SC model and algorithm
3. Improved Q-SC model and algorithm
4. QoS-aware service recommendation

Why SR based on trustable QoS

- Price, popularity, availability, etc
  - Offered by service providers (SP) or third-party registries (e.g., UDDI).
- Satisfaction degree, etc
  - QoS feedback from client-side is required.

Existed problem: Trustable QoS are required

- Current Q-SC is mainly based on the QoS properties advertised by SP.
- Malicious SPs who offer untrustworthy QoS properties may do harm to the ecological environment of service; services from honest SPs are uncompetitive
Introduction: Trust-based Q-SC

When trusted QoS from client-side are considered in Q-SC,

**Aim:** Predict the absent QoS values

- A large number of users need to evaluate an even larger number of candidate services, before making a trusted service composition.
- Without user 1’s QoS values on S4, the S4 may not be the candidate to any service composition for User 1 forever. That’s unreasonable.

<table>
<thead>
<tr>
<th></th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
</tr>
</thead>
<tbody>
<tr>
<td>User 1</td>
<td>5</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User 2</td>
<td></td>
<td></td>
<td>4.5</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>User 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User 4</td>
<td>2</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User 5</td>
<td>3.5</td>
<td></td>
<td></td>
<td></td>
<td>4.8</td>
</tr>
</tbody>
</table>

The blank indicates User 1 has not ever used S4 before, and no QoS values in it. **Need to be predicted**

Introduce RS to the framework of Q-CS

**Recommender system**
- User-based CF
- Service-based CF
- Matrix factoring

**Predict Client-side QoS**

**Trusted services recommendation**

**OMG task T**

1. Task decomposition
2. Services discovery
3. SESG construction
4. Services composition

Service Response (MCCS)

Component service(SESG) of MCCS
User-based Collaborative Filtering (CF)

**Idea:** If a user has little knowledge about a service, he can inquire or refer to other users (e.g., his friends) who has similar interests as him.

\[
sim(u, v) = \frac{\sum_{i \in I} (q_{u,i} - \bar{q}_u)(q_{v,i} - \bar{q}_v)}{\sqrt{\sum_{i \in I} (q_{u,i} - \bar{q}_u)^2 \sum_{i \in I} (q_{v,i} - \bar{q}_v)^2}}
\]

**Step 1:** find out similar users
For each pair of users u and v, the **Pearson Correlation Coefficient (PCC)** can be used to evaluate the similarity of them.

**Step 2:** predict absent QoS
\[
q_{u,x} = \sum_{v \in \text{sim}_u} \sim(u, v) q_{v,x}
\]

<table>
<thead>
<tr>
<th></th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
</tr>
</thead>
<tbody>
<tr>
<td>User 1</td>
<td>5</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User 2</td>
<td></td>
<td>4.5</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>User 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User 4</td>
<td>2</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User 5</td>
<td>3.5</td>
<td></td>
<td></td>
<td></td>
<td>4.8</td>
</tr>
</tbody>
</table>

Service-based Collaborative Filtering

**Idea:** We can study the historical scores of a user on other similar items, to predict what score he will give to the absent QoS of the item.

\[
sim(i, j) = \frac{\sum_{v \in U} (q_{v,i} - \bar{q}_i)(q_{v,j} - \bar{q}_j)}{\sqrt{\sum_{v \in U} (q_{v,i} - \bar{q}_i)^2 \sum_{v \in U} (q_{v,j} - \bar{q}_j)^2}}
\]

**Step 1:** find out similar items
For each pair of service i and j, **PCC** can be used to evaluate the similarity of them.

**Step 2:** predict absent QoS
\[
q_{u,i} = \sum_{j \in \text{sim}_i} \sim(i, j) q_{u,j}
\]

<table>
<thead>
<tr>
<th></th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
</tr>
</thead>
<tbody>
<tr>
<td>User 1</td>
<td>5</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User 2</td>
<td></td>
<td>4.5</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>User 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User 4</td>
<td>2</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User 5</td>
<td>3.5</td>
<td></td>
<td></td>
<td></td>
<td>4.8</td>
</tr>
</tbody>
</table>
Matrix Factoring

Denoted by matrix $Q_{n \times m}$ where $n$ denotes the number of users while $m$ denotes that of services.

- A user-service QoS matrix $Q_{n \times m}$ can be approximately divided into two matrix $U_{n \times k}$ and $S_{k \times m}$ with $k$-dimension ($k < \min(m,n)$), such that $Q_{n \times m} \approx (U_{n \times k})^T S_{k \times m}$.

- $U$ is the latent feature vector of $n$ users, indicating the extent of interest that users have in the services of high on some features, while $S$ is the latent feature vector of $m$ services, indicating the extent to which possesses those features.

- Idea: only a few key features affecting user-service interactions. And a use is impacted by how the feature is applied to him.

We minimize the term to gain the fine division of matrix $U_{n \times k}$ and $S_{k \times m}$:

$$
\min L = \sum_{x \in [1,n], i \in [1,m]} \frac{1}{2} (q_{x,i} - U_x^T S_i)^2
$$

Regularization terms are usually added to the end of the term to avoid over fitting.

Initial value of $U_{n \times k}$ and $S_{k \times m}$ are randomly generated, followed by updating $U, S$ by employing gradient descent technique.

$$
P_u = P_u - \gamma_1 \frac{\partial L}{\partial P_u}
$$

$$
S_i = S_i - \gamma_2 \frac{\partial L}{\partial S_i}
$$

$\gamma_1, \gamma_2$ denotes the learning rates given as parameters.
Challenges in Trust-based Q-SC

**Challenge 1: Sparsity**
- A *large number* of users need to evaluate an *even larger number* of candidate services, before making a trusted service composition.
- If constructing a user-service QoS matrix, only a few services are evaluated by a few users. It is a **sparse matrix** with huge number of absent QoS values.

**Challenge 2: Cold start**
- If a **newcomer** requests service composition, he would be in trouble because he has not any historical QoS data.
- Similarly, a **new created service** cannot be a candidate of any composite service for the same reason.

**Challenge 3: Trustworthiness of Client-side QoS**
- Lack the capability of differentiating user’s creditability. There may exist spam users who always give **fake feedback** of QoS properties.

**Challenge 4: Tensor factoring of high dimension**
- When **time-**, and **position**-dimension are included into the matrix factoring (Client-side QoS may vary with time and position in real cases).
- Considering multi-QoS properties in the element of user-service QoS matrix.

<table>
<thead>
<tr>
<th></th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
</tr>
</thead>
<tbody>
<tr>
<td>User 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User 5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*It could be a vector of QoS properties*
Challenges in Trust-based Q-SC

Challenge 5: **Algorithm of sound performance**
- As the user-service QoS matrix becomes increasingly high-dimensional, it must be a time consuming work to find out a fine result of tensor factoring.

Challenge 6: **Absence of open datasets with sufficient scale and information**
- Many datasets have very limited scale of data
- Lack of combined information with time, position, multiple QoS properties
- Most datasets are not open source

It is a long way to put in real use of Trust-based Q-SC, even the Q-SC. More challenges will raise as the further development and application of them.

Thanks!