Algorithmic Program Verification
Algorithmic Program Verification

... with applications to weak memory models ...
Algorithmic Program Verification

... with applications to weak memory models ...

Parosh Aziz Abdulla
Uppsala University
Web page:

Home Page: Parosh Aziz Abdulla
Tutorials: Postgraduate Course, 10 credits.
Projects:
Awards:
Conferences:
Teaching:
Students:

Next lecture: Wednesday, 2 February, 10:00-12:00, Zoom 589 726 9279.

Current capabilities in computer technology allow enormously complicated implementations of such systems, making the task of producing error-free products more and more difficult. In particular, during the last decade, parallel systems have become a critical part of the infrastructure of our society due to the emergence of modern platforms such as multicores and cloud technology.

It is of great practical and economic importance to developing methods that make the design process less error-prone. There is a real need for techniques for testing and verifying software to guarantee a higher degree of reliability.

Reasoning about concurrent systems is often conducted under the fundamental assumption of sequential consistency (SC) where all components are strongly synchronized so that they all have a uniform view of the global state of the system. However, nowadays, most parallel software run on platforms that do not guarantee SC. More precisely, to satisfy demands on efficiency and energy-saving, such platforms implement optimizations that lead to the relaxation of the inter-component synchronization, hence offering only weak consistency guarantees. Weakly consistent platforms are found at all levels of system design, such as multiprocessors, cache protocols, programming languages, and cloud systems.

Goal:
The participants will learn:

- The principles of model checking: a technique that has led to the most notable advances in algorithmic program verification during the last 20 years.
- Frameworks for verifying communication protocols, distributed systems and algorithms, timed systems, hardware circuits, concurrent programs, and multicore architectures.
- Modeling and verification through classical models such as timed automata, push-down automata, Petri nets, and lossy channels systems.
- Modeling and verifying concurrent programs running on weakly consistent platforms, such as x86- TSO, IBM Power, ARM, C11, and the cloud.

Contents:

- Model checking.
- Infinite-state models.
- Reachability analysis.
- Petri nets.
- Timed automata.
- Push-down automata.
- Lossy channel systems.
- x86- TSO.
- C11.
- Well quasi-ordered systems.
- Program abstraction: monotonic abstraction, view abstraction.
- The Power and ARM architectures.
- Parameterized systems.
- Cloud platforms.

Structure:

- 15 Lectures.
- Project work. The project will consist of implementing some of the algorithms discussed in the class.

Examination:

- Weekly assignments.
- Project work.

Prerequisites:

- Course suitable for both PhD and MSc students.
- Primary knowledge corresponding to the years of an undergraduate program in computer science.
- However, I do not assume any prior knowledge of formal methods, program verification, or weak memory models.

Slides (I will later add slides on weak consistency):

- L1. Petri Nets. [pdf, lecture].
- L2. Well-Quasi-Orderings. [pdf, lecture].
- L3. Lossy Channel Systems. [pdf, lecture].
- L5. Timed Petri Nets. [pdf, lecture].
- L7-8. View Abstraction. [pdf, lecture].

Literature:

- Contact Person:

  Parosh Aziz Abdulla
Outline

• Background
Outline

• Background
• Program Verification
Outline

• Background
• Program Verification
• Infinite-State Models
Outline

• Background
• Program Verification
• Infinite-State Models
• Petri Nets
Outline

• Background
• Program Verification
• Infinite-State Models
• Petri Nets
• Timed Automata
Outline

- Background
- Program Verification
- Infinite-State Models
- Petri Nets
- Timed Automata
- Push-Down Automata
Outline

• Background
• Program Verification
• Infinite-State Models
• Petri Nets
• Timed Automata
• Push-Down Automata
• Lossy Channel Systems
Outline

• Background
• Program Verification
• Infinite-State Models
• Petri Nets
• Timed Automata
• Push-Down Automata
• Lossy Channel Systems
• Parameterized Systems
Outline

• Background
• Program Verification
• Infinite-State Models
• Petri Nets
• Timed Automata
• Push-Down Automata
• Lossy Channel Systems
• Parameterized Systems
Outline

• Background
• Program Verification
• Infinite-State Models
• Petri Nets
• Timed Automata
• Push-Down Automata
• Lossy Channel Systems
• Parameterized Systems
• Well-Structured Systems
Outline

• Background
• Program Verification
• Infinite-State Models
• Petri Nets
• Timed Automata
• Push-Down Automata
• Lossy Channel Systems
• Parameterized Systems
• Well-Structured Systems
• Abstraction Techniques
Outline

• Background
• Program Verification
• Infinite-State Models
• Petri Nets
• Timed Automata
• Push-Down Automata
• Lossy Channel Systems
• Parameterized Systems
• Well-Structured Systems
• Abstraction Techniques
• x86
Outline

• Background
• Program Verification
• Infinite-State Models
• Petri Nets
• Timed Automata
• Push-Down Automata
• Lossy Channel Systems
• Parameterized Systems
• Well-Structured Systems
• Abstraction Techniques
• x86
• ARM
Outline

• Background
• Program Verification
• Infinite-State Models
• Petri Nets
• Timed Automata
• Push-Down Automata
• Lossy Channel Systems
• Parameterized Systems
• Well-Structured Systems
• Abstraction Techniques
• x86
• ARM
• IBM Power

models

techniques
Outline

• Background
• Program Verification
• Infinite-State Models
• Petri Nets
• Timed Automata
• Push-Down Automata
• Lossy Channel Systems
• Parameterized Systems
• Well-Structured Systems
• Abstraction Techniques
• x86
• ARM
• IBM Power
• C11
Outline

• Background
• Program Verification
• Infinite-State Models
• Petri Nets
• Timed Automata
• Push-Down Automata
• Lossy Channel Systems
• Parameterized Systems
• Well-Structured Systems
• Abstraction Techniques
• x86
• ARM
• IBM Power
• C11
• Cloud Platforms
Outline

• Background
• Program Verification
• Infinite-State Models
• Petri Nets
• Timed Automata
• Push-Down Automata
• Lossy Channel Systems
• Parameterized Systems
• Well-Structured Systems
• Abstraction Techniques
• x86
• ARM
• IBM Power
• C11
• Cloud Platforms
Outline

• Background
• Program Verification
• Infinite-State Models
  • Petri Nets
  • Timed Automata
  • Push-Down Automata
  • Lossy Channel Systems
  • Parameterized Systems
• Well-Structured Systems
• Abstraction Techniques
• x86
• ARM
• IBM Power
• C11
• Cloud Platforms
Outline

- Background
- Program Verification
- Infinite-State Models
- Petri Nets
- Timed Automata
- Push-Down Automata
- Lossy Channel Systems
- Parameterized Systems
- Well-Structured Systems
- Abstraction Techniques
- x86
- ARM
- IBM Power
- C11
- Cloud Platforms
concurrency
concurrency  motivation
Concurrent systems are everywhere
Concurrent systems are everywhere.
Concurrent systems are everywhere.

Motivation:
- Multicore architectures
- Intel
- ARM
- IBM Power

Concurrency
Concurrent systems are everywhere.
Concurrent systems are everywhere.
Concurrent systems are everywhere.
Outline

• Background
• Program Verification
• Infinite-State Models
• Petri Nets
• Timed Automata
• Push-Down Automata
• Lossy Channel Systems
• Parameterized Systems
• Well-Structured Systems
• Abstraction Techniques
• x86
• ARM
• IBM Power
• C11
• Cloud Platforms
Sequential Consistency (SC)
+ simple & intuitive
- expensive

Weak Semantics (WS)
+ efficient, realistic
- complicated
Sequential Consistency (SC)
+ simple & intuitive
- expensive

Weak Semantics (WS)
+ efficient, realistic
- complicated

Weak Consistence
“order in which data becomes visible”
Sequential Consistency (SC)
+ simple & intuitive
- expensive

Weak Semantics (WS)
+ efficient, realistic
- complicated

Weak Consistence
“order in which data becomes visible”
Sequential Consistency (SC)
- simple & intuitive
- expensive

Weak Semantics (WS)
- efficient, realistic
- complicated

Order in which data becomes visible

Weak Consistency
- C11: Release-Acquire, Relaxed
- Java

Architectures
- Intel x86: Total Store Ordering (TSO)
- ARM
- POWER

Programming Languages
- C11: Release-Acquire, Relaxed
- Java

Distributed Systems
- Eventual Consistency
- Causal Consistency

Programming Languages
- C11: Release-Acquire, Relaxed
- Java
**Programming Languages**

- C11: Release-Acquire, Relaxed
- Java

**Architectures**

- Intel x86: Total Store Ordering (TSO)
- ARM
- POWER

**Distributed Systems**

- Eventual Consistency
- Causal Consistency

**Sequential Consistency (SC)**

+ simple & intuitive
- expensive

**Weak Persistence**

- efficient, realistic
- complicated

**Weak Consistency**

“order in which data becomes visible”

**Weak Semantics (WS)**

“order in which data becomes persistent”

**Total Store Ordering (TSO)**
Sequential Consistency (SC)
  + simple & intuitive
  - expensive

Weak Semantics (WS)
  + efficient, realistic
  - complicated

Weak Consistency
  “order in which data becomes visible”

Weak Persistence
  “order in which data becomes persistent”

Architectures
  • Intel x86: Total Store Ordering (TSO)
  • ARM
  • POWER

Programming Languages
  • C11: Release-Acquire, Relaxed
  • Java

Distributed Systems
  • Eventual Consistency
  • Causal Consistency

NVRAMs
  • Intermittent Computing
  • File Systems

Total Store Ordering (TSO)

Intel x86:
  • Architectures
  • ARM
  • POWER

Java

C11:
  • Programming Languages
  • ARM
  • POWER

Distributed Systems
  • Eventual Consistency
  • Causal Consistency

Eventual Consistency

Causal Consistency

Total Store Ordering (TSO)

Intel x86:
  • Architectures
  • ARM
  • POWER

Java

C11:
  • Programming Languages
  • ARM
  • POWER

Distributed Systems
  • Eventual Consistency
  • Causal Consistency

Eventual Consistency

Causal Consistency

Total Store Ordering (TSO)
Sequential Consistency (SC)
+ simple & intuitive
- expensive

Weak Semantics (WS)
+ efficient, realistic
- complicated

“order in which data becomes visible”

“order in which data becomes persistent”

Weak Consistency

Weak Persistence

Architectures
- Intel x86: Total Store Ordering (TSO)
- ARM
- POWER

Programming Languages
- C11: Release-Acquire, Relaxed
- Java

Distributed Systems
- Eventual Consistency
- Causal Consistency

• NVRAMs
• Intermittent Computing
• File Systems

Total Store Ordering (TSO)

Intel x86: Architectures
ARM
POWER

Java

Eventual Consistency
Causal Consistency

C11: Release-Acquire, Relaxed

Distributed Systems

Sequential Consistency

Intel x86
ARM
POWER

NVRAMs
Intermittent Computing
File Systems
Sequential Consistency (SC)
  + simple & intuitive
  - expensive

Weak Semantics (WS)
  + efficient, realistic
  - complicated

Architectures
  • Intel x86: Total Store Ordering (TSO)
  • ARM
  • POWER

Programming Languages
  • C11: Release-Acquire, Relaxed
  • Java

Distributed Systems
  • Eventual Consistency
  • Causal Consistency

Recent work
  • NVRAMs
  • Intermittent Computing
  • File Systems

“order in which data becomes visible”

Weak Consistency

“order in which data becomes persistent”

Weak Persistence

Weak Semantics (WS)

Sequential Consistency (SC)
+ simple & intuitive
- expensive

Weak Semantics (WS)
+ efficient, realistic
- complicated
Program Verification (SC)

Sequential Consistency (SC)
+ simple & intuitive
- expensive

Weak Semantics (WS)
+ efficient, realistic
- complicated
Program Verification
(SC)

Sequential Consistency (SC)
+ simple & intuitive
- expensive

Program Verification
(WS)

Weak Semantics (WS)
+ efficient, realistic
- complicated
Sequential Consistency (SC)
+ simple & intuitive
- expensive

Weak Semantics (WS)
+ efficient, realistic
- complicated
Outline

• Background
• Program Verification
• Infinite-State Models
• Petri Nets
• Timed Automata
• Push-Down Automata
• Lossy Channel Systems
• Parameterized Systems
• Well-Structured Systems
• Abstraction Techniques
• x86
• ARM
• IBM Power
• C11
• Cloud Platforms
Outline

• Background
• Program Verification
• Infinite-State Models
• Petri Nets
• Timed Automata
• Push-Down Automata
• Lossy Channel Systems
• Parameterized Systems
• Well-Structured Systems
• Abstraction Techniques
• x86
• ARM
• IBM Power
• C11
• Cloud Platforms
Model Checking
Model $\models$ (safety) property
Classical Approach
Finite-State Systems

Model Checking
Model $\models$ (safety) property
Classical Approach
Finite-State Systems

Model Checking
Model $\models$ (safety) property

Challenge:
Infinite-State Systems
Classical Approach
Finite-State Systems

Model Checking
Model $\models$ (safety) property

Challenge:
Infinite-State Systems

Sources of "Infiniteness":

Classical Approach
Finite-State Systems

Model Checking
Model $|= \text{(safety) property}$

Challenge:
Infinite-State Systems

Sources of “Infiniteness”:
Unbounded Data Structures
• stacks (recursion)
• queues (protocols)
• counters (programs)
• clocks (time)
• lists, trees, graphs (heaps)
Classical Approach
Finite-State Systems

Model Checking
Model $|= (\text{safety})$ property

Challenge:
Infinite-State Systems

Sources of "Infiniteness":

Unbounded Data Structures
- stacks (recursion)
- queues (protocols)
- counters (programs)
- clocks (time)
- lists, trees, graphs (heaps)

Unbounded Control Structures
- parameterized systems
- multithreaded programs
- concurrent libraries
- Petri nets
Classical Approach
Finite-State Systems

Model Checking
Model $\models$ (safety) property

Challenge:
Infinite-State Systems

Sources of "Infiniteness":

Unbounded Data Structures
- stacks (recursion)
- queues (protocols)
- counters (programs)
- clocks (time)
- lists, trees, graphs (heaps)

Unbounded Control Structures
- parameterized systems
- multithreaded programs
- concurrent libraries
- Petri nets

Multiple Sources:
- timed Petri nets
- recursive programs with unbounded data
- channels with time stamps
- etc