Verification of Concurrent Programs

Decidability, Complexity, Reductions.

Ahmed Bouajjani

U Paris Diderot – Paris 7

Locali Workshop, Beijing, November 2013
Concurrency at different levels

- **Application level:**
  - Needs abstraction:
    - Abstract data structures, transactions, ...
  - Assumes:
    - Atomicity, isolation, ... (+ sequential specification...)

- **Performances**
  - Overlaps between parallel actions, sharing, etc.
  - Ensures:
    - (Illusion of) atomicity, isolation ...
  - Assumes:
    - Memory model (sequential consistency, causal delivery, etc.)

- **Infrastructures**
  - Performances
    - Store buffers, cashes, replicas, etc.
  - Relaxed memory models, weak consistency criteria.
  - (action reordering, lossyness, duplication, etc.)
Concurrency at different levels

- **Application level:**
  - Needs abstraction:
    - Abstract data structures, transactions, ...
  - Assumes:
    - Atomicity, isolation, ... (+ sequential specification...)

- **Implementation of concurrent data structures, and system services**
  - Performances ⇒ overlaps between parallel actions, sharing, etc.
  - Ensures:
    - (Illusion of) atomicity, isolation ...
  - Assumes:
    - Memory model (sequential consistency, causal delivery, etc.)
Concurrent at different levels

- **Application level:**
  - Needs abstraction:
    - Abstract data structures, transactions, ...
  - Assumes:
    - Atomicity, isolation, ... (+ sequential specification...)

- **Implementation of concurrent data structures, and system services**
  - Performances ⇒ overlaps between parallel actions, sharing, etc.
  - Ensures:
    - (Illusion of) atomicity, isolation ...
  - Assumes:
    - Memory model (sequential consistency, causal delivery, etc.)

- **Infrastructures**
  - Performances ⇒ Store buffers, cashes, replicas, etc.
  - Relaxed memory models, weak consistency criteria.
    - (action reordering, lossyness, duplication, etc.)
Issues at different levels

- **Applications**
  - **Correctness**: Program (model) satisfies Specification (of some service)

- Libraries of concurrent objects
  - Ensuring atomicity (+ specification):
    - Linearizability (shared concurrent data structures), equivalent to Observational Refinement: \( \forall \text{Client}. \forall n. \text{Client}_n[\text{Impl}] \subseteq \text{Client}_n[\text{Spec}] \)
    - Serializability (transactions),
    - Eventual consistency (distributed data structures), etc.

- Satisfaction of a specification over a relaxed memory model.
- Robustness against a memory model: Given a program \( P \) and two memory models \( M_1 \leq M_2 \), \( [\![P]\!]_{M_1} = [\![P]\!]_{M_2} \)?
Issues at different levels

- Applications
  - Correctness: Program (model) satisfies Specification (of some service)
  - Complexity (state-space explosion),
    Undecidability (recursion + synchronization, dynamic thread creation)
Issues at different levels

- **Applications**
  - Correctness: Program (model) satisfies Specification (of some service)
  - Complexity (state-space explosion),
  - Undecidability (recursion + synchronization, dynamic thread creation)

- **Libraries of concurrent objects**
  - Ensuring atomicity (+ specification):
    - *Linearizability* (shared concurrent data structures),
      equivalent to *Observational Refinement*:
      \[ \forall \text{Client}. \forall n. \text{Client}^n[\text{Impl}] \subseteq \text{Client}^n[\text{Spec}] \]
    - *Serializability* (transactions),
    - *Eventual consistency* (distributed data structures), etc.
Issues at different levels

- **Applications**
  - Correctness: Program (model) satisfies Specification (of some service)
  - Complexity (state-space explosion),
    Undecidability (recursion + synchronization, dynamic thread creation)

- **Libraries of concurrent objects**
  - Ensuring atomicity (+ specification):
    - **Linearizability** (shared concurrent data structures), equivalent to *Observational Refinement*:
      \[ \forall \text{Client} . \forall n. \text{Client}^n[\text{Impl}] \subseteq \text{Client}^n[\text{Spec}] \]
    - **Serializability** (transactions),
    - **Eventual consistency** (distributed data structures), etc.
  - Satisfaction of a specification over a relaxed memory model.
  - Robustness against a memory model:
    
    *Given a program $P$ and two memory models $M_1 \leq M_2$, $\llbracket P \rrbracket_{M_1} = \llbracket P \rrbracket_{M_2}$?*
Issues at different levels

- **Applications**
  - Correctness: Program (model) satisfies Specification (of some service)
  - Complexity (state-space explosion),
  - Undecidability (recursion + synchronization, dynamic thread creation)

- **Libraries of concurrent objects**
  - Ensuring atomicity (+ specification):
    - **Linearizability** (shared concurrent data structures),
      equivalent to **Observational Refinement**:
      \[ \forall \text{Client}. \forall n. \text{Client}^n[\text{Impl}] \subseteq \text{Client}^n[\text{Spec}] \]
    - **Serializability** (transactions),
    - **Eventual consistency** (distributed data structures), etc.

  - Satisfaction of a specification over a relaxed memory model.

  - Robustness against a memory model:
    
    \text{Given a program P and two memory models } M_1 \leq M_2, \ [P]_{M_1} = [P]_{M_2}?

  - Complexity (huge number of action orders),
  - Undecidability (some commutations allow to encode TM! – queues).
Questions

- Limits of decidability?
- Complexity?
- Basic (conceptual/technical) tools?
- General and efficient algorithmic approaches?
Reductions to Basic Models

- **Pushdown systems** (≡ Recursive state machines)

- **Unbounded Petri nets** (≡ Vector Addition Systems)

- **(Lossy) FIFO-channel systems**
Reductions to Basic Models

- **Pushdown systems** (≡ Recursive state machines)
  - Model for sequential programs (with recursive procedures).
  - State reachability is polynomial.

- **Unbounded Petri nets** (≡ Vector Addition Systems)
  - Model for dynamic concurrent programs with (an arbitrary number of) finite-state (anonymous) threads.
  - State reachability is decidable (EXPSPACE-complete). Research on efficient algorithms + tools.
  - Also useful when recursion (stacks) can be “eliminated” using summarization/finite-state abstraction of interfaces.

- **(Lossy) FIFO-channel systems**
  - Model for message-passing programs,
  - State reachability is decidable for the lossy model (using the theory of WQO).
  - Highly complex (non-primitive recursive), but ...
  - Also useful for reasoning about weak memory models: modeling of the effects of various kind of relaxations.
Reductions to Basic Models

- **Pushdown systems** (≡ Recursive state machines)
  - Model for sequential programs (with recursive procedures).
  - State reachability is polynomial.
  - Also useful when concurrent behaviors can be “sequentialized”.

- **Unbounded Petri nets** (≡ Vector Addition Systems)
  - Model for dynamic concurrent programs with (an arbitrary number of) finite-state (anonymous) threads.
  - State reachability is decidable (EXPSPACE-complete). Research on efficient algorithms + tools.
  - Also useful when recursion (stacks) can be “eliminated” using summarization/finite-state abstraction of interfaces.

- **(Lossy) FIFO-channel systems**
Reductions to Basic Models

- **Pushdown systems** ($\equiv$ Recursive state machines)
  - Model for sequential programs (with recursive procedures).
  - State reachability is polynomial.
  - Also useful when concurrent behaviors can be “sequentialized”.

- **Unbounded Petri nets** ($\equiv$ Vector Addition Systems)
  - Model for dynamic concurrent programs with (an arbitrary number of) finite-state (anonymous) threads.
  - State reachability is decidable (EXPSPACE-complete). Research on efficient algorithms + tools.

- **(Lossy) FIFO-channel systems**
Reductions to Basic Models

- **Pushdown systems** (≡ Recursive state machines)
  - Model for sequential programs (with recursive procedures).
  - State reachability is polynomial.
  - Also useful when concurrent behaviors can be “sequentialized”.

- **Unbounded Petri nets** (≡ Vector Addition Systems)
  - Model for dynamic concurrent programs with (an arbitrary number of) finite-state (anonymous) threads.
  - State reachability is decidable (EXPSPACE-complete). Research on efficient algorithms + tools.
  - Also useful when recursion (stacks) can be “eliminated” using summarization/finite-state abstraction of interfaces.

- **(Lossy) FIFO-channel systems**
Reductions to Basic Models

- **Pushdown systems** (≡ Recursive state machines)
  - Model for sequential programs (with recursive procedures).
  - State reachability is polynomial.
  - Also useful when concurrent behaviors can be “sequentialized”.

- **Unbounded Petri nets** (≡ Vector Addition Systems)
  - Model for dynamic concurrent programs with (an arbitrary number of) finite-state (anonymous) threads.
  - State reachability is decidable (EXPSPACE-complete). Research on efficient algorithms + tools.
  - Also useful when recursion (stacks) can be “eliminated” using summarization/finite-state abstraction of interfaces.

- **(Lossy) FIFO-channel systems**
  - Model for message-passing programs,
  - State reachability is decidable for the lossy model (using the theory of WQO). Highly complex (non-primitive recursive), but ...
Reductions to Basic Models

- **Pushdown systems** (≡ Recursive state machines)
  - Model for sequential programs (with recursive procedures).
  - State reachability is polynomial.
  - Also useful when concurrent behaviors can be “sequentialized”.

- **Unbounded Petri nets** (≡ Vector Addition Systems)
  - Model for dynamic concurrent programs with (an arbitrary number of) finite-state (anonymous) threads.
  - State reachability is decidable (EXPSPACE-complete). Research on efficient algorithms + tools.
  - Also useful when recursion (stacks) can be “eliminated” using summarization/finite-state abstraction of interfaces.

- **(Lossy) FIFO-channel systems**
  - Model for message-passing programs,
  - State reachability is decidable for the lossy model (using the theory of WQO). Highly complex (non-primitive recursive), but ...
  - Also useful for reasoning about weak memory models: modeling of the effects of various kinds of relaxations.
Reductions to Basic Classes of Programs

- **Code-to-code translations to:**
  - Sequential programs: getting rid of concurrency
  - Concurrent programs over SC: getting rid of relaxed memory models/weak consistency models

- **Separation of the issues:**
  - As general as possible, regardless from the decidability issue
  - Independent from the used data types
  - Holds for unbounded control parameters: recursion depth, number of processes/created tasks, size of buffers, etc.
  - Precise reduction, under well defined conditions on the control features in programs/classes of computations

- **Decidability and complexity** are derived for particular cases

  *Finite data domains, ...*
Reductions to Basic Classes of Programs

- **Code-to-code translations** to:
  - Sequential programs: getting rid of concurrency
  - Concurrent programs over SC: getting rid of relaxed memory models/weak consistency models

- **Separation of the issues**:
  - As general as possible, regardless from the decidability issue
  - Independent from the used data types
  - Holds for unbounded control parameters: recursion depth, number of processes/created tasks, size of buffers, etc.
  - Precise reduction, under well defined conditions on the control features in programs/classes of computations

- **Decidability and complexity** are derived for particular cases
  
  *Finite data domains*, ... 

- **When is this possible? How?**
Multi-threaded Programs: Sequentialization

- Concurrent programs with shared memory + recursive procedures:
  
  *Reachability is in general *undecidable*: 2-thread boolean programs.*
Multi-threaded Programs: Sequentialization

- **Concurrent programs with shared memory + recursive procedures:**
  
  *Reachability is in general *undecidable*: 2-thread boolean programs.*

- **Context-Bounded Analysis:**
  
  *Finite number of context-switches* [Qadeer, Rehof, 05]
  
  ▶ Few context-switches are needed to catch concurrency bugs,
  
  ▶ Still the program is infinite-state (unbounded call stacks),
  
  ▶ Decidable, NP-complete.
Multi-threaded Programs: Sequentialization

- **Concurrent programs with shared memory + recursive procedures:**
  
  \textit{Reachability is in general \textbf{undecidable}: 2-thread boolean programs.}

- **Context-Bounded Analysis:**
  
  \textit{Finite number of context-switches} \cite{QadeerRehof05}
  
  ▶ Few context-switches are needed to catch concurrency bugs,
  
  ▶ Still the program is infinite-state (unbounded call stacks),
  
  ▶ Decidable, NP-complete.

- **Sequentialization under Context-bounding** \cite{LalReps08}
  
  ▶ Each thread has a finite number of execution rounds
  
  ▶ Bounded Input/Output interfaces: memory states at the starting/ending points of each round
  
  ▶ Assume-Guarantee approach: Guess the Input states (nondeterministic assignments), produce the Output states, Check composability
  
  ▶ Code-to-code translation to a sequential program
Multi-threaded Programs with Dynamic Thread Creation

- *Each* thread has a bounded number of execution rounds.
- The number of context switches is not bounded globally.
Multi-threaded Programs with Dynamic Thread Creation

- Each thread has a bounded number of execution rounds.
- The number of context switches is not bounded globally.
- Still CBA is decidable [Atig, B., Qadeer, 09]
  - Reduction to state reachability (coverability) in Petri nets
  - Based on a finite-state abstraction of the interface of each thread
    - A thread generates a context-free set $S$ of sequences of thread creation events, but not all created threads must contribute to a computation.
    - It is sound to close the set $S$ by the sub-word relation.
  - Use counters (places) to count the number of threads that are at particular states.
Multi-threaded Programs with Dynamic Thread Creation

- Each thread has a bounded number of execution rounds.
- The number of context switches is **not bounded globally**.
- Still CBA is decidable [Atig, B., Qadeer, 09]
  - Reduction to state reachability (coverability) in Petri nets
  - Based on a finite-state abstraction of the interface of each thread
    - A thread generates a context-free set $S$ of sequences of thread creation events, but not all created threads must contribute to a computation.
    - It is sound to close the set $S$ by the sub-word relation.
  - Use counters (places) to count the number of threads that are at particular states.
- Lower bound: At least as hard as state-reachability in Petri nets
  - Polynomial sequentialization cannot be done precisely for CBA.
Multi-threaded Programs with Dynamic Thread Creation

- Each thread has a bounded number of execution rounds.
- The number of context switches is not bounded globally.
- Still CBA is decidable [Atig, B., Qadeer, 09]
  - Reduction to state reachability (coverability) in Petri nets
  - Based on a finite-state abstraction of the interface of each thread
    - A thread generates a context-free set $S$ of sequences of thread creation events, but not all created threads must contribute to a computation.
    - It is sound to close the set $S$ by the sub-word relation.
  - Use counters (places) to count the number of threads that are at particular states.
- Lower bound: At least as hard as state-reachability in Petri nets
  - Polynomial sequentialization cannot be done precisely for CBA.
- General sequentialization schema: tree traversal $+$ bounded interfaces [B., Emmi, Parlato, 11] (Bounded tree-width behaviors)
Many other works

- Asynchronous programs
  - Synchronous procedure calls + Asynchronous task creation
  - Tasks are run until completion
  
  [Sen, Viswanathan, 06], [Jhala, Majumdar, 07], ...

- Asynchronous programs + priorities & preemption
  
  [Atig, B., Touili, 08], [Emmi, Qadeer, Lal, 12]

- Recursively parallel programs [B., Emmi, 12]
  
  Cilk, X10, ...

- ...
Libraries of Concurrent Objects

Specification given by a regular language
Example of a valid sequence: \( \text{Push}(6)\text{Push}(7)\text{Pop}(7) \)
Linearizability [Herlihy, Wing, 90]

A linearizable execution:

```
call(Push,2) ok
```

```
call(Push,4) ok
```

```
call(Pop) ret(2)
```
A linearizable execution:

```
call(Push,2)  --- ok
--- call(Push,4)   --- ok
---                   --- call(Pop)  ret(2)

Push(4) Push(2) Pop(2) ∈ Specification
```
Checking Linearizability

- Fixed number of finite-state threads [Alur, McMillan, Peled, 96]
  - Reduction to a problem of the form:
    \[ \text{MostGeneralClient[Impl]} \subseteq \text{Closure}(\text{Spec}) \]
  - \( \Rightarrow \) Non-Linearizability \( \leadsto \) a state reachability problem:
    \[ \text{MostGeneralClient[Impl]} \cap \overline{\text{Closure}(\text{Spec})} \neq \emptyset \]
  - Complexity: PSPACE-hard and in EXPSPACE.

- Unbounded number of threads [B., Enea, Emmi, Hamza, 13]
  - Linearizability is undecidable in general.

  - Static Linearizability: Linearization points are fixed in the code, except for read-only methods.
    - Most of implementations of concurrent objects satisfy this condition.
    - Linearization point = commit point
    - Reduction (of non-Static Linearizability) to control state reachability.
    - P/EXPSPACE-complete for fixed/unbounded number of threads.
Checking Linearizability

- **Fixed number of finite-state threads** [Alur, McMillan, Peled, 96]
  - Reduction to a problem of the form:
    \[
    \text{MostGeneralClient}[\text{Impl}] \subseteq \text{Closure}(\text{Spec})
    \]
  - \(\Rightarrow\) Non-Linearizability \(\leadsto\) a state reachability problem:
    \[
    \text{MostGeneralClient}[\text{Impl}] \cap \overline{\text{Closure}(\text{Spec})} \neq \emptyset
    \]
  - Complexity: PSPACE-hard and in EXPSPACE.

- **Unbounded number of threads** [B., Enea, Emmi, Hamza, 13]
  - Linearizability is **undecidable** in general.
Checking Linearizability

- **Fixed number of finite-state threads** [Alur, McMillan, Peled, 96]
  - Reduction to a problem of the form:
    \[ \text{MostGeneralClient}[\text{Impl}] \subseteq \text{Closure}(\text{Spec}) \]
  - \( \Rightarrow \) Non-Linearizability \( \leadsto \) a state reachability problem:
    \[ \text{MostGeneralClient}[\text{Impl}] \cap \overline{\text{Closure}(\text{Spec})} \neq \emptyset \]
  - Complexity: PSPACE-hard and in EXPSPACE.

- **Unbounded number of threads** [B., Enea, Emmi, Hamza, 13]
  - Linearizability is **undecidable** in general.
  - **Static Linearizability:**
    \[ \text{Linearization points are fixed in the code, except for read-only methods.} \]
    - Most of implementations of concurrent objects satisfy this condition.
      \[ \text{Linearization point} = \text{commit point} \]
    - Reduction (of non-Static Linearizability) to control state reachability.
    - P/EXPSPACE-complete for fixed/unbounded number of threads.
Bounded Checking of Linearizability

- Criterion for finding errors in concurrent objects implementations?
Bounded Checking of Linearizability

- Criterion for finding errors in concurrent objects implementations?
- Lin. $\simeq$ Observational Refinement [Filipovic, O’Hearn, Rinetzky, Yang, 10]
  \[ \forall \text{Client}. \forall n. \text{Client}^n[\text{Impl}] \subseteq \text{Client}^n[\text{Spec}] \]
- Writes on external shared variables by the clients induce causality constraints on methods along executions.
Bounded Checking of Linearizability

- Criterion for finding errors in concurrent objects implementations?
- Lin. \(\simeq\) Observational Refinement [Filipovic, O’Hearn, Rinetzky, Yang, 10]
  \[ \forall \text{Client. } \forall n. \text{Client}^n[\text{Impl}] \subseteq \text{Client}^n[\text{Spec}] \]
- Writes on external shared variables by the clients induce causality constraints on methods along executions.
- \(\Rightarrow\) Bounded number of external writes on shared variables by the clients. [B., Enea, Emmi, Hamza, 13].
Bounded Checking of Linearizability

- Criterion for finding errors in concurrent objects implementations?
- Lin. ≃ Observational Refinement [Filipovic, O’Hearn, Rinetzky, Yang, 10]
  \[ \forall Client. \forall n. Client^n[Impl] \subseteq Client^n[Spec] \]
- Writes on external shared variables by the clients induce causality constraints on methods along executions.
- \[ \Rightarrow \] Bounded number of external writes on shared variables by the clients. [B., Enea, Emmi, Hamza, 13].
- Bugs show up within small bounds (0 quite often, or 1)!
Bounded Checking of Linearizability

- Criterion for finding errors in concurrent objects implementations?
- Lin. $\simeq$ Observational Refinement [Filipovic, O’Hearn, Rinetzky, Yang, 10]
  \[ \forall \text{Client} \cdot \forall n. \text{Client}^n[\text{Impl}] \subseteq \text{Client}^n[\text{Spec}] \]
- Writes on external shared variables by the clients induce causality constraints on methods along executions.
- $\Rightarrow$ Bounded number of external writes on shared variables by the clients. [B., Enea, Emmi, Hamza, 13].
- Bugs show up within small bounds (0 quite often, or 1)!
- Precise reduction to State Reachability:
  - OR is equivalent to
    \[ \text{MostGeneralClient}[\text{Impl}] \subseteq \text{MostGeneralClient}[\text{Spec}] \]
  - Use additional counters to reason about number of methods starting and ending between pairs of external writes.
Bounded Checking of Linearizability

- Criterion for finding errors in concurrent objects implementations?
- Lin. \(\cong\) Observational Refinement [Filipovic, O’Hearn, Rinetzky, Yang, 10]
  \[ \forall \text{Client}. \forall n. \text{Client}^n[\text{Impl}] \subseteq \text{Client}^n[\text{Spec}] \]
- Writes on external shared variables by the clients induce causality constraints on methods along executions.
- \(\Rightarrow\) Bounded number of external writes on shared variables by the clients. [B., Enea, Emmi, Hamza, 13].
- Bugs show up within small bounds (0 quite often, or 1)!
- Precise reduction to State Reachability:
  - OR is equivalent to
    \[ \text{MostGeneralClient}[\text{Impl}] \subseteq \text{MostGeneralClient}[\text{Spec}] \]
  - Use additional counters to reason about number of methods starting and ending between pairs of external writes.

- Decidability (finite data domain): Reduction to reachability in Petri nets (using Parikh image computations for Specification closure)
Weak Memory Models: State Space Reachability

- TSO = Writes are sent to store buffers (one per processor).
- SR decidable for TSO (and ...) [Atig, B., Burckhard, Musuvathi, 10-12].
- Holds for unbounded store buffers (and arbitrary number of threads).
Weak Memory Models: State Space Reachability

- TSO = Writes are sent to store buffers (one per processor).
- SR decidable for TSO (and ...) [Atig, B., Burckhardt, Musuvathi, 10-12].
- Holds for unbounded store buffers (and arbitrary number of threads).
- But as hard as State Reachability in Lossy Fifo-Channel Systems (non-primitive recursive)
- \( \Rightarrow \) Precise reduction to State Reachability in SC is not possible.
Weak Memory Models: State Space Reachability

- TSO = Writes are sent to store buffers (one per processor).
- SR decidable for TSO (and ...) [Atig, B., Burckhardt, Musuvathi, 10-12].
- Holds for unbounded store buffers (and arbitrary number of threads).
- But as hard as State Reachability in Lossy Fifo-Channel Systems (non-primitive recursive)
- \( \Rightarrow \) Precise reduction to State Reachability in SC is not possible.
- (Code-to-code) translation to State Reachability is possible under “Age-bounding” [Atig, B., Parlato, 12]
  
  *Each write action in a buffer must be executed after at most\( K\) context switches of that thread.*
Weak Memory Models: State Space Reachability

- TSO = Writes are sent to store buffers (one per processor).
- SR decidable for TSO (and ...) [Atig, B., Burckhardt, Musuvathi, 10-12].
- Holds for unbounded store buffers (and arbitrary number of threads).
- But as hard as State Reachability in Lossy Fifo-Channel Systems (non-primitive recursive)
  ⇒ Precise reduction to State Reachability in SC is not possible.
- (Code-to-code) translation to State Reachability is possible under “Age-bounding” [Atig, B., Parlato, 12]

  Each write action in a buffer must be executed after at most $K$ context switches of that thread.

- Other Works: abstraction/symbolic analysis/bounded model checking:
  - [Kuperstein, Vechev, Yahav, 11]
  - [Linden, Wolper, 10-11]
  - [Abdulla, Atig, Chen, Leonardson, Rezine, 12]
  - [Alglave, Kroening, Nimal, Tautchnig, 13]
Weak Memory Models: Robustness against TSO

- State-robustness as hard as State Reachability in TSO.

- Traces \cite{Shasha, Snir, 88}: Capture the control and data flow in SC computations.
  
  Trace-robustness is reducible to State Reachability in SC! \cite{B., Derevenetc, Meyer, 13}
  
  Code-to-code translation, precise (no approximations), holds for an arbitrary number of threads, unbounded buffers, arbitrary data domain.
  
  Finite data domain: PSPACE/EXPSPACE-complete for a fixed/arbitrary number of threads.
  
  Optimal fence insertion.

Other Work

- Testing: \cite{Burckhardt, Musuvathi, CAV'08}, \cite{Burnim, Stergiou, Sen, 11}
  
- Upper-approximate analysis: \cite{Alglave, Maranget, 11}
  
- Stronger criterion: Triangular data races \cite{Owens, 10}
Weak Memory Models: Robustness against TSO

- State-robustness as hard as State Reachability in TSO.
- Traces [Shasha, Snir, 88]: Capture the control and data flow in SC computations.
- Trace-robustness is reducible to State Reachability in SC! [B., Derevenetc, Meyer, 13]
Weak Memory Models: Robustness against TSO

- State-robustness as hard as State Reachability in TSO.
- Traces [Shasha, Snir, 88]: Capture the control and data flow in SC computations.
- Trace-robustness is reducible to State Reachability in SC!
  [B., Derevenetc, Meyer, 13]
- Code-to-code translation, precise (no approximations), holds for an arbitrary number of threads, unbounded buffers, arbitrary data domain.
Weak Memory Models: Robustness against TSO

- State-robustness as hard as State Reachability in TSO.
- Traces [Shasha, Snir, 88]: Capture the control and data flow in SC computations.
- Trace-robustness is reducible to State Reachability in SC!
  [B., Derevenetc, Meyer, 13]
- Code-to-code translation, precise (no approximations), holds for an arbitrary number of threads, unbounded buffers, arbitrary data domain.
- Finite data domain: PSPACE/EXPSPACE-complete for a fixed/arbitrary number of threads.
Weak Memory Models: Robustness against TSO

- State-robustness as hard as State Reachability in TSO.
- Traces [Shasha, Snir, 88]: Capture the control and data flow in SC computations.
- Trace-robustness is reducible to State Reachability in SC!
  [B., Derevenetc, Meyer, 13]
- Code-to-code translation, precise (no approximations), holds for an arbitrary number of threads, unbounded buffers, arbitrary data domain.
- Finite data domain: PSPACE/EXPSPACE-complete for a fixed/arbitrary number of threads.
- Optimal fence insertion.

Other Work

- Testing: [Burckhardt, Musuvathi, CAV'08], [Burnim, Stergiou, Sen, 11]
- Upper-approximate analysis: [Alglave, Maranget, 11]
- Stronger criterion: Triangular data races [Owens, 10]
Weak Memory Models: Robustness against TSO

- State-robustness as hard as State Reachability in TSO.
- Traces [Shasha, Snir, 88]: Capture the control and data flow in SC computations.
- Trace-robustness is reducible to State Reachability in SC!
  [B., Derevenetc, Meyer, 13]
- Code-to-code translation, precise (no approximations), holds for an arbitrary number of threads, unbounded buffers, arbitrary data domain.
- Finite data domain: PSPACE/EXPSPACE-complete for a fixed/arbitrary number of threads.
- Optimal fence insertion.

Other Work

- Testing: [Burckhardt, Musuvathi, CAV’08], [Burnim, Stergiou, Sen, 11]
- Upper-approximate analysis: [Alglave, Maranget, 11]
- Stronger criterion: Triangular data races [Owens, 10]
Conclusion / questions

- A lot remains to be understood concerning decidability frontiers, complexity, and reducibility to problems such as state reachability in basic models.

- In particular: correctness over weak memory models, correctness criteria in the distributed case (papers in POPL’14), etc.

- Generic reductions for general classes of programs and general families of correctness criteria?

- Sequentialization (What is pushdown representable?) is related to the notion of “bounded tree-width” [La Torre, Parlato, Madhusudan, 11].

- We need a general framework for reasoning about order constraints and their violations: What is Petri net representable (Petrifiable)?