Verification of Concurrent Programs
Decidability, Complexity, Reductions

Ahmed Bouajjani
LIAFA, Univ Paris Diderot - Paris 7

ATVA, Sydney, November 2014
Concurrency at Different Levels

- **Application level**
  - Abstraction: Abstract Data Structures, Transactions, etc.
  - => Atomicity, Isolation, ...

- **Concurrent objects**
  - Performances: avoid coarse-grain locking
  - Consistency: ensure illusion of atomicity, ...

- **Infrastructures**
  - Performances: cashes, replication
  - Relaxed memory models, weak consistency
Issues

- **Formal models**
  - Communication patterns: shared memory, message passing,…
  - Control features: recursion, dynamic thread creation,…
  - Memory models, consistency criteria

- **Decidability**
  - Recursion + synchronization, dynamic thread creation,…
  - Queues (communication channels, buffers,…)

- **Complexity**
  - State-space explosion
  - Huge number of interleaving

- **Efficient Algorithmic Techniques**
  - Applicable to source-code
  - Scalable: under/upper approximations
Reductions to Basic Formal Models

- **Pushdown Systems (Recursive State Machines)**
  - Model for sequential procedural programs
  - Control state/configuration reachability is Polynomial
  - Useful when concurrent behaviours can be “sequentialized”

- **Petri Nets (Vector Addition Systems)**
  - Model for concurrent programs with (anonymous) thread creation
  - Control state reachability (coverability) is EXPSPACE-complete

- **Fifo Channel Systems**
  - Model for message passing programs
  - Control state reachability undecidable in general
  - Becomes decidable for Lossy Channel Systems — highly complex
  - Also useful for reasoning about weak memory models
Reductions to Basic Classes of Programs

- **Code to code translations**
  - Sequential programs
  - Concurrent programs under sequential consistency

- **Separation of concerns**
  - Semantical problem, regardless of decidability issue
  - Holds for arbitrary data types
  - Holds for unbounded control parameters (recursion depth, number of threads, size of buffers/channels, etc.)

- **Decidability, complexity**
  - Finite data domain
  - Bounding some of the parameters
Concurrent Programs with Shared Memory

- Recursion + Synchronization is Turing powerful
- Bounded analysis: what is the parameter to bound?
Concurrent Programs with Shared Memory: Context-Bounded Analysis [Qadeer, Rehof, ’05]

- Recursion + Synchronization is **Turing powerful**
- **Bounded analysis**: what is the **parameter to bound**?

  => Number of context switches
- Bugs show up after a small number of context switches
Concurrent Programs with Shared Memory: Context-Bounded Analysis [Qadeer, Rehof, ’05]

• Recursion + Synchronization is Turing powerful
• Bounded analysis: what is the parameter to bound?
  => Number of context switches
• Bugs show up after a small number of context switches
• Still an unbounded number of reachable configurations
• Decidable? Complexity?
Concurrent Programs with Shared Memory: Context-Bounded Analysis [Qadeer, Rehof, ’05]

- Recursion + Synchronization is Turing powerful
- Bounded analysis: what is the parameter to bound?
  - => Number of context switches
  - Bugs show up after a small number of context switches
  - Still an unbounded number of reachable configurations
  - Decidable? Complexity?
  - => Yes, NP-complete

- Construct the sets of forward/backward reachable configurations (using FSA): Iterate the construction post*/pre* for single pushdown systems.
Concurrent Programs with Shared Memory: Context-Bounded Analysis [Qadeer, Rehof, ’05]

• Recursion + Synchronization is Turing powerful
• Bounded analysis: what is the parameter to bound?
  • => Number of context switches
  • Bugs show up after a small number of context switches
• Still an unbounded number of reachable configurations
• Decidable? Complexity?
  • => Yes, NP-complete
• Construct the sets of forward/backward reachable configurations (using FSA): Iterate the construction post*/pre* for single pushdown systems.
• But this is not scalable…
Sequentialization under Context-Bounding
[Lal, Reps, ’08]

- Compute valid Input-Output interfaces
- Check that they are composable
- \[\Rightarrow\] Code to code reduction to reachability in a sequential program
Dynamic (finite-state) thread creation

- Finite number of variables
- Finite data domain
- \( \Rightarrow \) Threads are anonymous (cannot refer to their identities)

- Iterative programs, no recursion
- \( \Rightarrow \) Arbitrary number of finite-state processes
Dynamic (finite-state) thread creation

- Finite number of variables
- Finite data domain
- \( \Rightarrow \) Threads are anonymous (cannot refer to their identities)

- Iterative programs, no recursion
- \( \Rightarrow \) Arbitrary number of finite-state processes

- Counter abstraction
  - Finite number of local control states
  - Count how many threads are at each control state

- State reachability \( \Rightarrow \) coverability in Petri nets (VASS)
Dynamic thread creation + recursion: Asynchronous Programs

- Event-driven programming
- Synchronous (recursive) procedure calls (as usual)
- Asynchronous procedure calls, or task creation
  - Tasks are executed until completion
  - Newly created tasks are stored in a buffer
  - Pending tasks are scheduled in any order
- Formal model: Multiset Pushdown Systems
  [Sen, Viswanathan, ’06]
  - Execution of task: pushdown process (using the stack)
  - Created tasks are stored in the multiset
  - When the stack is empty, a task is moved from the multiset to the top of the stack
From Asynchronous Programs to Petri nets
[Atig, B., Touili, ’08]

• Unbounded multiset —> counters (one per type of task)
• How to get rid of stacks?
From Asynchronous Programs to Petri nets
[Atig, B., Touili, ’08]

• Unbounded multiset —> counters (one per type of task)
• How to get rid of stacks ?

• => Summarization
  • Compute the start-end points reachability relation of tasks
  • + the effect of their execution: the multiset of created tasks
From Asynchronous Programs to Petri nets
[Atig, B., Touili, ’08]

• Unbounded multiset —> counters (one per type of task)
• How to get rid of stacks ?

• =&gt; Summarization
  • Compute the start-end points reachability relation of tasks
  • + the effect of their execution: the multiset of created tasks

• Are summaries representable by a finite-state automata ?
From Asynchronous Programs to Petri nets

- \( m_1 \) and \( m_2 \) are Parikh images of Context-Free Languages.
- They are effectively computable semi-linear sets.
- There are Regular Languages that have the same Parikh images.
From Asynchronous Programs to Petri nets

- $M_1$ and $M_2$ are Parikh images of Context-Free Languages
- They are effectively computable semi-linear sets
- There are Regular Languages that have the same Parikh images

$\Rightarrow$ Replace Pushdown tasks by Finite-State Automata
Complexity

- Polynomial size reduction
  - Pushdown system $\rightarrow$ Context Free Grammar
  - CFG $\rightarrow$ Synchronisation-Free Petri Net (BPP)

- Petri nets are particular cases

$\Rightarrow$ State reachability in AP is EXPSACE-complete
[Ganty, Majumdar, '12]
Asynchronous Programs
with Priorities & Preemption [Atig, B., Touili, ’08]
Asynchronous Programs with Priorities & Preemption [Atig, B., Touili, ’08]

• Guess macro steps due to Level 2
• => Insert labeled transitions in the pushdown model of Level 1
• Parikh image —> automaton A (q0, s0, q’0)
Asynchronous Programs with Priorities & Preemption

- Guess **macro steps** due to Level 2
- \(\Rightarrow\) Insert labeled transitions in the pushdown model of Level 1
- **Parikh image** \(\rightarrow\) automaton \(A\) \((q_0, s_0, q'0)\)
- Simulation using \(A\): **what to do for red transitions?**
Asynchronous Programs with Priorities & Preemption

- Guess **macro steps** due to Level 2
- => Insert labeled transitions in the pushdown model of Level 1
- **Parikh image** —> automaton $A$ ($q_0$, $s_0$, $q'0$)
- Simulation using $A$: what to do for red transitions?
- => Construct a Petri net (as for plain AP) + test to zero of $M_2$
- **This is OK** [Reinhardt ’06-08], [Bonnet, ’11]
Dynamic thread creation: Context-Bounded Analysis
[Atig, B., Qadeer, ’09]

- Global bounding => bounding the number of threads
- Each thread must have a chance to be scheduled
Dynamic thread creation: Context-Bounded Analysis
[Atig, B., Qadeer, ’09]

• Global bounding => bounding the number of threads
• Each thread must have a chance to be scheduled
• => Bounded number of context switches per thread
• => The global number of context switches is unbounded
• NB: Generalisation of Asynchronous Programs

Decidability? Complexity?
Dynamic thread creation + recursion: Context-Bounded Analysis cont.

- Difficulties
  - Unbounded number of pending threads at different rounds
  - Unbounded number of local contexts (local state+stack)
  - Naïve use of counter abstraction does not work: needs an unbounded number of counters.
Dynamic thread creation + recursion: Context-Bounded Analysis cont.

• **Difficulties**
  
  • Unbounded number of pending threads at different rounds
  • *Unbounded number of local contexts (local state+stack)*
  • Naïve use of counter abstraction does not work: needs an unbounded number of counters.

• **Approach: Compositional reasoning!**
  
  • Summarize the interaction of a thread with its environment
  • Regular representations of these summaries (using FSA)
  • Simulate threads using their summaries —> Petri net
Regular Summaries of Tasks Behaviours

Task $t$

$q_0$ \[ \rightarrow \] $r_1$ \[ \rightarrow \] ... \[ \rightarrow \] $r_n$ \[ \rightarrow \] $q_1$

$q'_1$ \[ \rightarrow \] $s_1$ \[ \rightarrow \] ... \[ \rightarrow \] $s_m$ \[ \rightarrow \] $q_2$

$q'_2$ \[ \rightarrow \] $t_1$ \[ \rightarrow \] ... \[ \rightarrow \] $t_k$ \[ \rightarrow \] $q_3$
Regular Summaries of Tasks Behaviours

• Pushdown automaton with state-jump transitions
Regular Summaries of Tasks Behaviours

- Pushdown automaton with state-jump transitions

- Regular summary: Downward closure w.r.t. subword relation
  - Order between tasks is important
  - Some of the created tasks may not be useful (never scheduled)
  - Downward closure of a CFL is a regular effectively constructible set
Reduction to Coverability in Petri Nets

- Places for representing the control states
- Place for counting the number of pending tasks at some local state, with a context switch budget (from 0 to K)
- For each created task, guess the sequence of context switches (interface)
- Simulate a full sequential execution using the FSA corresponding to the interface
- Transition labelled with a task \( t \) correspond to a task creation
- For a transition \((q, q')\), the control is left: current task waits for \( q' \), and its context switch budget is decremented
The reduction creates a PN of \textit{exponential size}\n\n\Rightarrow \text{The problem decidable and is in 2EXPSPACE}
Complexity

- The reduction creates a PN of exponential size
- => The problem decidable and is in 2EXPSPACE

- The problem is EXSPACE-hard
- => Is it possible to improve the complexity?
The reduction creates a PN of exponential size

\[ \Rightarrow \text{The problem decidable and is in 2EXPSPACE} \]

The problem is EXSPACE-hard

\[ \Rightarrow \text{Is it possible to improve the complexity?} \]

The complexity is high anyway!

\[ \Rightarrow \text{Sequentialization?} \]
Sequentialization: A General Method
[B., Emmi, Parlato, ’11]

• Interface: Sequence of state jumps \((q_1, q'_1)\)…\((q_n, q'_n)\)

• Transforming Interfaces:
Sequentialization: A General Method
[B., Emmi, Parlato, ’11]

- DFS + bounded-size multisets of bounded-size interfaces
- Start from a singleton \(\{(q_{\text{init}}, q_{\text{final}})\}\)
- Success if an empty set of interfaces is reached at the end
Weak Memory Models: TSO

- **Writes**: they are sent to store buffers
- **Memory updates**: execution of a write taken from some SB
- **Reads**: read(x)
  - takes the last write to x in the store buffer, if there is one,
  - otherwise the value of x in the main memory
- **AtomicRW**: requires that the SB is empty (~fence)
TSO State Reachability Problem  
[Atig, B., Burckhardt, Musuvathi, ’10]

- **Decidable**  
  - Reducible to reachability in Lossy Channel Systems  
  - => We can lose observations of memory states

- **Non-primitive recursive**  
  ... and vice-versa

- => Huge gap w.r.t. SC reachability (PSPACE-complete)

- **Approximate Analysis**  
  Polynomial Reduction to SC reachability?
TSO State Reachability under Context Bounding

Any execution can be decomposed according to:

\[(P_1 + M_1)^* \cdot (P_2 + M_2)^* \cdot \ldots \cdot (P_n + M_n)^* \cdot (P_1 + M_1)^* \cdot (P_2 + M_2)^* \cdot \ldots \cdot (P_n + M_n)^* \cdot \ldots\]

A K-round bounded execution is of the form:

\[\left( (P_1 + M_1)^* \cdot (P_2 + M_2)^* \cdot \ldots \cdot (P_n + M_n)^* \right)^K\]
Getting rid of store buffers: View of a processor

- Between two rounds, the SB of a processor is not modified
- => Simulate unbounded store buffers using "views"

For each variable: last value in the SB if it exists, or the value in the main memory
Simulating store buffers using views

\[
V_1 = M_1
\]

\[
V_1 = M_1
\]

\[
M_2
\]

\[
M'_2 = V_{12} \times M_2
\]

\[
V_2 = V_{13} \times M'_2
\]

\[
V_{23} \times V_{13}
\]

\[
V_24
\]

\[
N_1 = V_{11} \times M_1
\]

\[
N_2 = M'_2
\]
Other Bounding Concepts for TSO
[Atig, B., Parlato, ’14]

• Memory-centric:
  
  Round: \((M_i + P_1 + \ldots + P_n)^*\)

• Processor-centric:
  
  Round: \((P_i + M_1 + \ldots + M_n)^*\)

• Decidable in 2EXPTIME

  \(\rightarrow\) Bounded-(reversal-)phase multi-pushdown systems

  [La Torre, Madhusudan, Parlato, ’07] [Atig, ’10]
Conclusion

- Complex, undecidable in general
- Reductions to basic models: pushdown systems, Petri nets
- Crucial approach: compositional reasoning
  - Interface computation, Finite/Regular representation
  - Both for decidability and scalability
- Many relevant extensions of Petri nets:
  - PN+ordered counters, PN+stack, Recursive PN, BVASS, etc.
  - Decidability and complexity questions are still open
- PN are expensive: symbolic techniques? approximations?
- Approximate sequentialization (based on bounding concepts)
Some Related Work

• Many decidability results use “sequentialization”
  Atig, La Torre, Madhusudan, Parlato,…

• Many work on bounding + sequentialization
  e.g., Delay bounding [Emmi, Qadeer, Rakamaric, ’11]

• Message passing programs
  • Decidability: [La Torre et. al., ’10],[Heussner et al., ’12]
  • Sequentialization: [B., Emmi,’12] (Phase bounding)

• Infinite runs, liveness properties
  e.g., Scope bounding: [La Torre et. al., ’12] …

• General frameworks for decidability
  • Bounded Tree Width: [La Torre, Madhusudan, Parlato, ’11-12]
  • Bounded Split Width: [Cyriac, Gastin, Kumar, ’12]
Other important issues

• Verification of libraries of concurrent data structures
  => Talk on Wednesday on Refinement Checking

• Robustness against weak memory models

• Distributed data structures