Lecture 2: Verification of Concurrent Programs

Part 2: Under Approximate Analysis

Ahmed Bouajjani

LIAFA, University Paris Diderot – Paris 7

VTSA, MPI-Saarbrücken, September 2012

Concurrent Programs with Procedures

- Parallel threads (with/without procedure calls)
- Shared memory
- Interleaving semantics (sequential consistency)
- Model = Concurrent Pushdown Systems (Multistack systems)

Concurrent Programs with Procedures

- Parallel threads (with/without procedure calls)
- Shared memory
- Interleaving semantics (sequential consistency)
- Model = Concurrent Pushdown Systems (Multistack systems)
- Turing powerful: 2 threads
- $\Rightarrow$ Restrictions: Consider only some schedules
- Aim: detect bugs

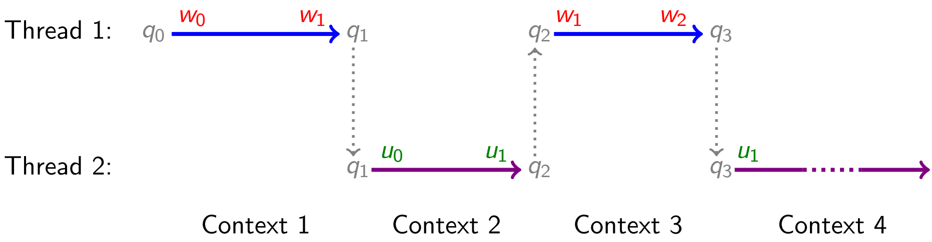
Concurrent Programs with Procedures

- Parallel threads (with/without procedure calls)
- Shared memory
- Interleaving semantics (sequential consistency)
- Model = Concurrent Pushdown Systems (Multistack systems)
- Turing powerful: 2 threads
- $\Rightarrow$ Restrictions: Consider only some schedules
- Aim: detect bugs
- What is a good concept for restricting the set of behaviors ?

Context-Bounded Analysis

[Qadeer, Rehof, 2005]

The number of context switches in a computation is bounded

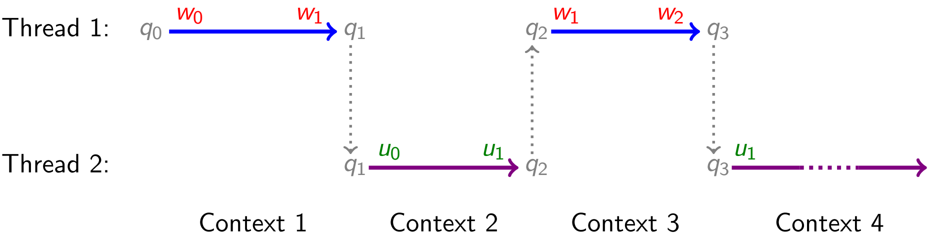


- Suitable for finding bugs in concurrent programs.
- Concurrency bugs show up after a small number of context switches.

Context-Bounded Analysis

[Qadeer, Rehof, 2005]

The number of context switches in a computation is bounded



- Suitable for finding bugs in concurrent programs.
- Concurrency bugs show up after a small number of context switches.
- Infinite-state space: Unbounded sequential computations
- Decidability ?

Basic case: Pushdown system

- Pushdown system $= (Q, \Gamma, \Delta)$
- Configuration: (q, w) where $q \in Q$ is a control state, $w \in \Gamma$ is the stack content.

Basic case: Pushdown system

- Pushdown system $= (Q, \Gamma, \Delta)$
- Configuration: (q, w) where $q \in Q$ is a control state, $w \in \Gamma$ is the stack content.
- Symbolic representation: A finite state automaton.
- Computation of the predecessors/successors:

  For every regular set of configurations C, the $pre^*(C)$ and $post^*(C)$ are regular and effectively constructible.
  [Büchi 62], ..., [B., Esparza, Maler, 97], ...

- Reachability: Polynomial algorithms.
- Can be generalized to model checking.

Context-Bounded Analysis: Decidability

- Consider a multi-stack systems with n stacks
- Configuration: $(q, w_1, \ldots, w_n)$, where q is a control state, $w_i \in \Gamma_i$ are stack contents.

Context-Bounded Analysis: Decidability

- Consider a multi-stack systems with n stacks
- Configuration: $(q, w_1, \ldots, w_n)$, where q is a control state, $w_i \in \Gamma_i$ are stack contents.
- Symbolic representation: clusters $(q, A_1, \ldots, A_n)$, q a control state, A_i are FSA over Γ_i
- Given a cluster C, compute a set of clusters characterizing $K\text{-}pre^*(C)$ (resp. $K\text{-}post^*(C)$)

Context-Bounded Analysis: Decidability

- Consider a multi-stack systems with n stacks
- Configuration: $(q, w_1, \ldots, w_n)$, where q is a control state, $w_i \in \Gamma_i$ are stack contents.
- Symbolic representation: clusters $(q, A_1, \ldots, A_n)$, q a control state, A_i are FSA over Γ_i
- Given a cluster C, compute a set of clusters characterizing $K\text{-}pre^*(C)$ (resp. $K\text{-}post^*(C)$)
- Generalize the pre^* / $post^*$ constructions for PDS

Context-Bounded Analysis: Decidability

- Consider a multi-stack systems with n stacks
- Configuration: $(q, w_1, \ldots, w_n)$, where q is a control state, $w_i \in \Gamma_i$ are stack contents.
- Symbolic representation: clusters $(q, A_1, \ldots, A_n)$, q a control state, A_i are FSA over Γ_i
- Given a cluster C, compute a set of clusters characterizing K-$pre^*(C)$ (resp. K-$post^*(C)$)
- Generalize the pre^* / $post^*$ constructions for PDS
- Enumerate sequences of the form $q_0 i_0 q_1 i_1 q_2 i_2 \ldots i_K q_K i_{K+1}$, where q_j's are states, and $i_j \in \{1, \ldots, n\}$ are threads identities.
- Let $X_{K+1} = C$. Compute: for $j = K$ back to 0
  - $A'_{j+1} = pre^*_{i_{j+1}}(X_{j+1}[i_{j+1}]) \cap q_j \Gamma_i^*$
  - $X_j = (q_j, A_1^{j+1}, \ldots, A'_{j+1}, \ldots, A_n^{j+1})$

Dynamic Creation of Threads ?

[Atig, B., Qadeer, 09]

Problem

- Bounding the number of context switches $\Rightarrow$ bounding the number of threads.
- $\Rightarrow$ Inadequate bounding concept for the dynamic case.

  Each created thread must have a chance to be executed

Dynamic Creation of Threads ?

[Atig, B., Qadeer, 09]

Problem

- Bounding the number of context switches $\Rightarrow$ bounding the number of threads.
- $\Rightarrow$ Inadequate bounding concept for the dynamic case.

Each created thread must have a chance to be executed

New definition

- Give to each thread a *context switch budget*
- $\Rightarrow$ The number of context switches is bounded for each thread
- $\Rightarrow$ The global number of context switches in a run is unbounded
- NB: Generalization of Asynchronous Programs

Case 1: Dynamic Networks of Finite-State Processes

Decidable ?

Case 1: Dynamic Networks of Finite-State Processes

Decidable ?

Theorem

The K-bounded state reachability problem is EXPSPACE-complete.

Reduction to/from the coverability problem for Petri.

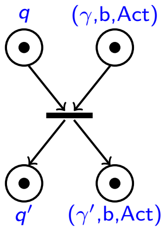
Reduction to coverability in PN

- For every global store $q \in Q$, associate a place q.
- For every stack configuration $\gamma \in \Gamma \cup \{\epsilon\}$ and budget $b \in \{1, \ldots, K\}$ of the active thread, associate a place $(\gamma, \mathrm{b}, \mathrm{Act})$.
- For every stack configuration $\gamma \in \Gamma \cup \{\epsilon\}$ and budget $b \in \{0, \ldots, K\}$ of a pending thread, associate a place $(\gamma, \mathrm{b}, \mathrm{Pen})$.

Reduction to coverability in PN

- For every global store $q \in Q$, associate a place q.
- For every stack configuration $\gamma \in \Gamma \cup \{\epsilon\}$ and budget $b \in \{1, \ldots, K\}$ of the active thread, associate a place $(\gamma$,b,Act$)$.
- For every stack configuration $\gamma \in \Gamma \cup \{\epsilon\}$ and budget $b \in \{0, \ldots, K\}$ of a pending thread, associate a place $(\gamma$,b,Pen$)$.

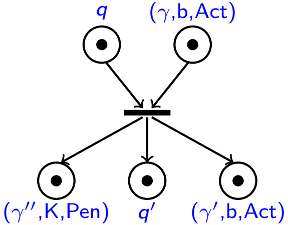
Rule of the form: $q\gamma \longrightarrow q'\gamma'$ $\Longrightarrow$



Reduction to coverability in PN

- For every global store $q \in Q$, associate a place q.
- For every stack configuration $\gamma \in \Gamma \cup \{\epsilon\}$ and budget $b \in \{1, \ldots, K\}$ of the active thread, associate a place (γ,b,Act).
- For every stack configuration $\gamma \in \Gamma \cup \{\epsilon\}$ and budget $b \in \{0, \ldots, K\}$ of a pending thread, associate a place (γ,b,Pen).

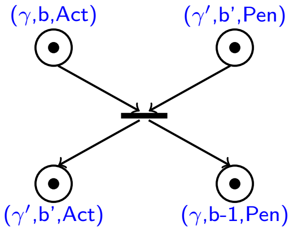
Rule of the form: $q\gamma \longrightarrow q'\gamma' \rhd \gamma''$ $\Longrightarrow$



Reduction to coverability in PN

- For every global store $q \in Q$, associate a place q.
- For every stack configuration $\gamma \in \Gamma \cup \{\epsilon\}$ and budget $b \in \{1, \ldots, K\}$ of the active thread, associate a place $(\gamma,\text{b},\text{Act})$.
- For every stack configuration $\gamma \in \Gamma \cup \{\epsilon\}$ and budget $b \in \{0, \ldots, K\}$ of a pending thread, associate a place $(\gamma,\text{b},\text{Pen})$.

Context switch (with b'$>$ 0) $\Longrightarrow$



Case 2: Dynamic Networks of Pushdown Systems

- Decidable ?

Case 2: Dynamic Networks of Pushdown Systems

- Decidable ?
- Difficulty:
  - Unbounded number of pending local contexts
  - Can not use the same construction as for the case of finite state threads. (This would need an unbounded number of places.)

Case 2: Dynamic Networks of Pushdown Systems

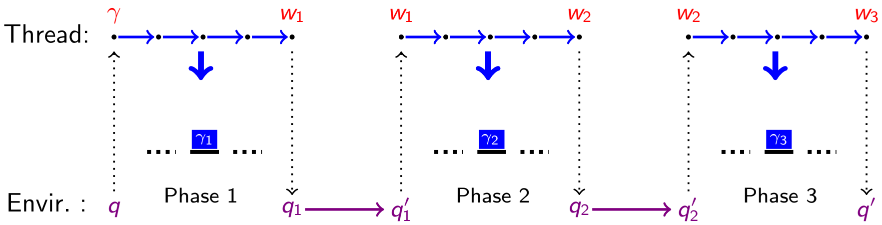
- Decidable ?
- Difficulty:
  - Unbounded number of pending local contexts
  - Can not use the same construction as for the case of finite state threads. (This would need an unbounded number of places.)

Theorem

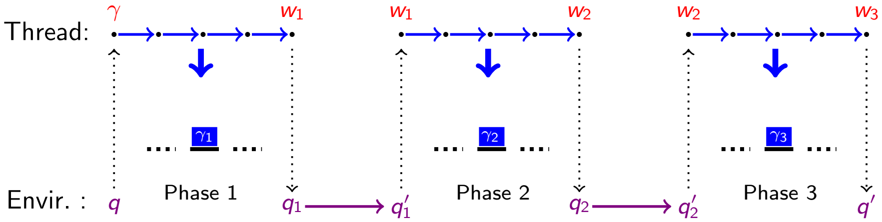
The K-bounded state reachability problem is in 2EXPSPACE.

Exponential reduction to the coverability problem in PN

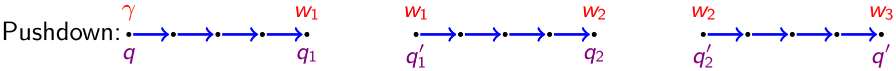
Making visible the interactions



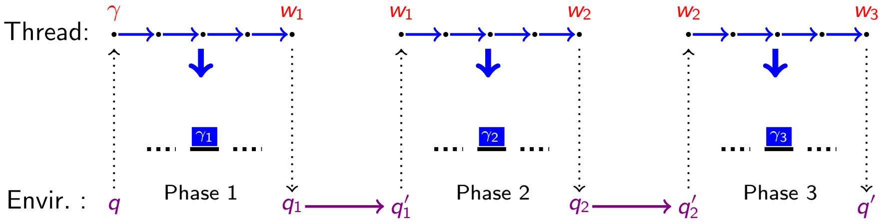
Making visible the interactions



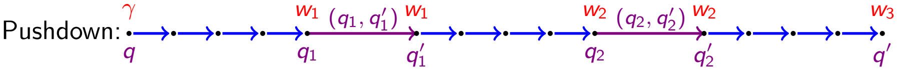
- Construct a labeled pushdown automaton which:
    - Guesses the effect of the environment on the states



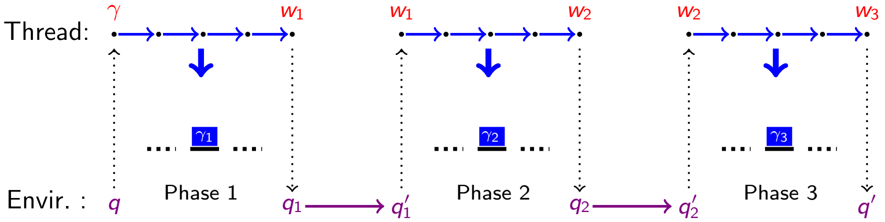
Making visible the interactions



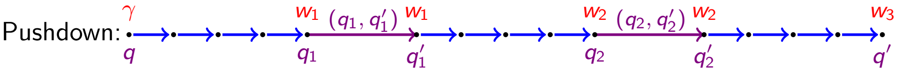
- Construct a labeled pushdown automaton which:
  - Guesses the effect of the environment on the states



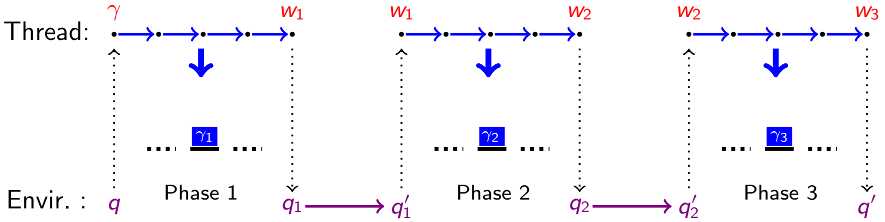
Making visible the interactions



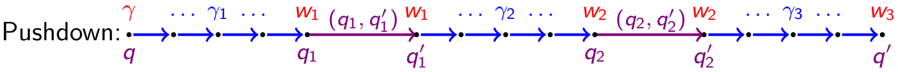
- Construct a labeled pushdown automaton which:
  - Makes visible (as transition labels) the created threads



Making visible the interactions

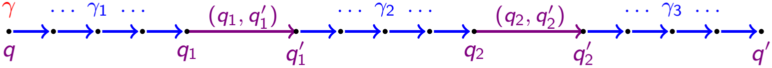


- Construct a labeled pushdown automaton which:
  - Makes visible (as transition labels) the created threads



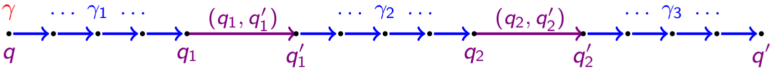
Constructing a regular interface

Pushdown:



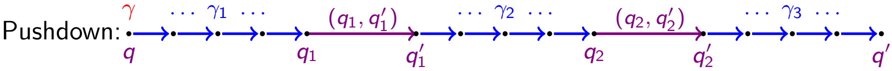
Constructing a regular interface

Pushdown:



- The set of traces L characterizes the interaction between the thread and its environment (L is a CFL)

Constructing a regular interface



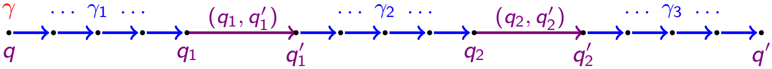
- The set of traces L characterizes the interaction between the thread and its environment (L is a CFL)

Observations: For the state reachability problem

- Order of events is important
- Some created threads may never be scheduled

Constructing a regular interface

Pushdown:



- The set of traces L characterizes the interaction between the thread and its environment (L is a CFL)

Observations: For the state reachability problem

- Order of events is important
- Some created threads may never be scheduled

$\Rightarrow$ Replace L by its downward closure w.r.t. the sub-word relation $L\downarrow$

Constructing a regular interface (cont.)

- The interactions of a thread with its environment can be characterized by the downward closure $L\downarrow$ of the context-free language L
- $L\downarrow$ is regular and effectively constructible ([Courcelle, 1991])
- The size of an automaton for $L\downarrow$ can be exponential in the PDA defining L

Constructing the Petri Net

- Use places for representing the control, one per state
- Count pending tasks having some context switch budget (from 0 to K), and waiting to start at some state
- For each created task, guess a sequence of K states (for context switches)
- At context switches, control is given to a pending task waiting for the current state
- Simulate a full sequential computation (following the FSA automaton of the interface) until next transition (g, g')
- During the simulation, each transition labelled γ corresponds to a task creation
- At a transition (g, g'), leave the control at g (to some other thread) and wait for g' (with a lower switch budget)

Sequentialization under Context Bounding

Question:

Is it possible to reduce CBA of a Concurrent Program to the Reachability Analysis of a Sequential Program ?

Sequentialization under Context Bounding

Question:

Is it possible to reduce CBA of a Concurrent Program to the Reachability Analysis of a Sequential Program ?

Yes: Use compositional reasoning !

[Lal, Reps, 2008]

Sequentialization under Context Bounding: Basic Idea

- Consider a Program with 2 threads T_1 and T_2, and global variables X
- Consider the problem: Can the program reach the state (q_1, q_2)
- Assume that the threads are scheduled in a Round Robin manner
- Let K be the number of rounds
- Guess an *interface* of each thread:
  - $I^i = (I^i_1, \dots I^i_K)$, the global states when T_i starts/is resumed
  - $O^i = (O^i_1, \dots O^i_K)$, the global states when T_i terminates/is interrupted
- Check that T_1 can reach q_1 by a computation that fulfills its interface
- Check that T_2 can reach q_2 by a computation that fulfills its interface
- Check that the interfaces are composable
  - $O^1_j = I^2_j$ for every $j \in \{1, \dots, K\}$
  - $O^2_j = I^1_{j+1}$ for every $j \in \{1, \dots, K-1\}$

Sequentialization: Code-to-code translation

Given a concurrent program P, construct a sequential program P_s such that (q_1, q_2) is reachable under K-CB in P iff q_{win} in reachable in P_s.

- Create $2K$ copies of the global variables X_j and X'_j, for $j \in \{1, \ldots, K\}$
- Start the simulation of T_1. At each round $j \in \{1, \ldots, K\}$, thread T_1:
  1. Starts by putting some values in X_j (guesses the input I_j^1)
  2. Copies X_j in X'_j, and runs by using X'_j as global variables
  3. Choses nondeterministically the next context-switch point
  4. Moves to round $j+1$ (locals are not modified) and go to 1 (using new copies of globals X_{j+1} and X'_{j+1}).
- When $T1$ reaches q_1, start simulating T_2. At each round j, thread T_2:
  1. Starts from the content of X'_j that was produced by T_1 in its j-th round
  2. Runs by using X'_j as global variables
  3. Choses nondeterministically the next context-switch point
  4. Checks that $X'_j = X_{j+1}$ (composability check), and move to round $j+1$
- If q_2 is reachable at round K, then go to state q_{win}

Context-bounded analysis: Complexity

- Finite Number of Threads:

	Unbounded	K-Bounded
Finite-state systems	PSPACE-complete	NP-complete
Pushdown systems	Undecidable	NP-complete

- Dynamic Creation of Threads:

	Unbounded	K-Bounded
Finite-state systems	EXPSPACE-complete	EXPSPACE-complete
Pushdown systems	Undecidable	In 2EXPSPACE RR: EXPSPACE-complete [ABQ + Lal]

Sequentialization for Dynamic Programs

- VASS are sequential machines, so there is a precise sequentialization

Sequentialization for Dynamic Programs

- VASS are sequential machines, so there is a precise sequentialization
- What do we mean by "sequentialization" ?

Sequentialization for Dynamic Programs

- VASS are sequential machines, so there is a precise sequentialization
- What do we mean by "sequentialization" ?
- We want to use pushdown systems
- We do not want to expose locals: compositional reasoning
- We want to obtain a program of the same type: we should not add other data structures, variables, etc.

Sequentialization for Dynamic Programs

- VASS are sequential machines, so there is a precise sequentialization
- What do we mean by "sequentialization" ?
- We want to use pushdown systems
- We do not want to expose locals: compositional reasoning
- We want to obtain a program of the same type: we should not add other data structures, variables, etc.
- In this context, a precise sequentialization of dynamic programs cannot exist (we cannot encode VASS with PDS)

Sequentialization for Dynamic Programs

- VASS are sequential machines, so there is a precise sequentialization
- What do we mean by "sequentialization" ?
- We want to use pushdown systems
- We do not want to expose locals: compositional reasoning
- We want to obtain a program of the same type: we should not add other data structures, variables, etc.
- In this context, a precise sequentialization of dynamic programs cannot exist (we cannot encode VASS with PDS)
- Under-approximate sequentialization [B., Emmi, Parlato, 2011]
- Idea:
  - Transform thread creation into procedure calls
  - Allow some reordering using the idea of bounded interfaces

End of Lecture 2:

- Finding adequate bounding notions for concurrent programs is an important issue.
- Adequate bounding should allow to lower the complexity of the analysis, and compositional reductions to sequential analysis.
- Source-to-source reduction are important: allow the use of existing tools.

End of Lecture 2:

- Finding adequate bounding notions for concurrent programs is an important issue.
- Adequate bounding should allow to lower the complexity of the analysis, and compositional reductions to sequential analysis.
- Source-to-source reduction are important: allow the use of existing tools.
- Context-bounding is a interesting concept, but there are others, e.g., delay bounding [Emmi, Qadeer, Rakamaric, 2011]

End of Lecture 2:

- Finding adequate bounding notions for concurrent programs is an important issue.
- Adequate bounding should allow to lower the complexity of the analysis, and compositional reductions to sequential analysis.
- Source-to-source reduction are important: allow the use of existing tools.
- Context-bounding is a interesting concept, but there are others, e.g., delay bounding [Emmi, Qadeer, Rakamaric, 2011]
- Bounding notion for message-passing programs ?
- Phase-bounding has been proposed recently [B., Emmi, 2012]