Verification of Concurrent Systems

Ahmed Bouajjani

LIAFA, University Paris Diderot – Paris 7

MOVEP'12, CIRM, December 2012

Concurrent Programs

- Parallel threads (with/without procedure calls)
- Static/Dynamic number of threads
- Communication

Concurrent Programs

- Parallel threads (with/without procedure calls)
- Static/Dynamic number of threads
- Communication
  - Shared memory
    - Notion of action atomicity
    - Actions by a same threads are executed in the same order (Sequential Consistency)
    - Actions by different threads are interleaved non-deterministically

Concurrent Programs

- Parallel threads (with/without procedure calls)
- Static/Dynamic number of threads
- Communication
  - Shared memory
    - Notion of action atomicity
    - Actions by a same threads are executed in the same order (Sequential Consistency)
    - Actions by different threads are interleaved non-deterministically
  - Message passing
    - Channels (queues)
    - Unordered/FIFO ...
    - Perfect/Lossy

Concurrent Programs

- Parallel threads (with/without procedure calls)
- Static/Dynamic number of threads
- Communication
  - Shared memory
    - Notion of action atomicity
    - Actions by a same threads are executed in the same order (Sequential Consistency)
    - Actions by different threads are interleaved non-deterministically
  - Message passing
    - Channels (queues)
    - Unordered/FIFO ...
    - Perfect/Lossy
- We assume finite data domain (e.g., booleans).

Finite number of threads + Shared variables

- Fixed number of threads
- Iterative processes (no recursive procedure calls)
- Finite number of variables

Finite number of threads + Shared variables

- Fixed number of threads
- Iterative processes (no recursive procedure calls)
- Finite number of variables
- A variable has a finite number of possible values
- $\Rightarrow$ Finite product of finite-state systems (threads + variables)
- $\Rightarrow$ Decidable

Finite number of threads + Shared variables

- Fixed number of threads
- Iterative processes (no recursive procedure calls)
- Finite number of variables
- A variable has a finite number of possible values
- $\Rightarrow$ Finite product of finite-state systems (threads + variables)
- $\Rightarrow$ Decidable
- Product grows exponentially in # threads and # variables.
- Reachability is decidable, and PSPACE-complete.
  [Kozen, FOCS'77]

Finite number of threads + bounded queues

- Fixed number of threads
- Iterative processes (no recursive procedure calls)
- Bounded channels

Finite number of threads + bounded queues

- Fixed number of threads
- Iterative processes (no recursive procedure calls)
- Bounded channels
- ⇒ Finite number of possible channel contents
- ⇒ Finite product of finite-state systems (threads + channels)
- ⇒ Decidable

Finite number of threads + bounded queues

- Fixed number of threads
- Iterative processes (no recursive procedure calls)
- Bounded channels
- $\Rightarrow$ Finite number of possible channel contents
- $\Rightarrow$ Finite product of finite-state systems (threads + channels)
- $\Rightarrow$ Decidable
- Product grows exponentially in # threads and size of channels.
- Reachability is decidable, and PSPACE-complete.

Facing the state-space explosion

- Partial order techniques
  - Independent actions $\Rightarrow$ commutable actions $\Rightarrow$ many interleavings
  - Explore representatives up to independent actions commutations
  - Compact representations of sets of behaviors (Unfoldings)

  Godefroid, Wolper, Peled, Holzman, Valmari, McMillan, Esparza, ...

- Symbolic techniques
  - Compact representations of sets of states (e.g., BDD)
  - Encoding bounded-length computation + SAT solvers

  Clarke, McMillan, Somenzi, Biere, Cimatti, ...

Beyond the finite-state case

- Unbounded (parametric/dynamic) number of threads
  - Undecidable in general if threads Ids are allowed
  - $\Rightarrow$ Anonymous threads
- Unbounded channels
  - Undecidable in general in case of FIFO queues
  - $\Rightarrow$ Unordered queues (multisets), lossy queues

Programs with Dynamic Creation of Threads

- Finite number of variables
- Finite data domain
- $\Rightarrow$ Threads are anonymous (no way to refer to identities)

Programs with Dynamic Creation of Threads

- Finite number of variables
- Finite data domain
- $\Rightarrow$ Threads are anonymous (no way to refer to identities)
- Iterative processes (no recursive procedure calls)
- $\Rightarrow$ Counting abstraction
  - Finite number of possible local states $\ell_1, \ldots, \ell_m$
  - Count how many threads are in a given local state

Programs with Dynamic Creation of Threads

- Finite number of variables
- Finite data domain
- $\Rightarrow$ Threads are anonymous (no way to refer to identities)
- Iterative processes (no recursive procedure calls)
- $\Rightarrow$ Counting abstraction
  - Finite number of possible local states $\ell_1, \ldots, \ell_m$
  - Count how many threads are in a given local state
- Safety is reducible to state reachability in VASS / Coverability in PN

Vector Addtion Systems with States

- Finite state machine + finite number of counter $C = \{c_1, \ldots, c_n\}$.
- Operations: (No test to zero)
  - $c_i := c_i + 1$
  - $c_i > 0 \;/\; c_i := c_i - 1$
- Configuration: (q, V) where q is a control state and $V \in \mathbb{N}^n$
- Initial configuration: $(q_0, \mathbf{0})$ where $\mathbf{0} = 0^n$.
- Transition relation:

$$(q_1, V_1) \xrightarrow{op} (q_2, V_2) \textit{ iff}$$

  - $op =$ "$c_i := c_i + 1$", *and* $V_2 = V_1[c_i \leftarrow (V_1(c_i) + 1)]$
  - $op =$ "$c_i > 0 \;/\; c_i := c_i - 1$, *and* $(V_1(c_i) > 0$ *and* $V_2 = V_1[c_i \leftarrow (V_1(c_i) - 1)])$

From Multithreaded Programs to VASS

- Associate a control state with each valuation of the globals
- Associate a counter with each valuation of thread locals
- A statement moving globals from g to g' and locals from ℓ to ℓ':

$$g \xrightarrow{c_\ell > 0 / c_\ell := c_\ell - 1 \,;\, c_{\ell'} := c_{\ell'} + 1} g'$$

- Creation of a new thread at initial state ℓ:

$$g \xrightarrow{c_\ell := c_\ell + 1} g$$

VASS: State Reachability

- State reachability problem:

  Given a state q, determine if a configuration (q, V) is reachable, for some $V \in \mathbb{N}^n$ (any one).

- Coverability problem:

  Given a configuration (q, V), determine if a configuration (q, V') is reachable, for some $V' \geq V$. (We say that (q, V) is coverable*.)*

EXSPACE-complete [Rackoff 78]

NB: *Coverability can be reduced to State Reachability and vice-versa.*

Well Structured Systems

[Abdulla et al. 96], [Finkel, Schnoebelen, 00]

- Let U be a universe.
- Well-quasi ordering $\preceq$ over U: $\forall c_0, c_1, c_2, \ldots, \ \exists i < j, \ c_i \preceq c_j$
- $\Rightarrow$ Each (infinite) set has a finite minor set.

- Let $S \subseteq U$. Upward-closure $\overline{S}$ = minimal subset of U s.t.
  - $S \subseteq \overline{S}$,
  - $\forall x, y.\ (x \in S \text{ and } x \preceq y) \Rightarrow y \in \overline{S}$.
- A set is upward closed if $\overline{S} = S$
- Upward closed sets are definable by their minor sets
  - Assume there is a function *Min* which associates a minor to each set.
  - Assume $pre(Min(S))$ is computable for each set S.

- Monotonicity: $\preceq$ is a simulation relation

$$\forall c_1, c_1', c_2.\ \big((c_1 \longrightarrow c_1' \text{ and } c_1 \preceq c_2) \Rightarrow \exists c_2'.\ c_2 \longrightarrow c_2' \text{ and } c_1' \preceq c_2'\big)$$

Key lemma

Lemma

The pre and pre* images of upward closed set are upward closed

Key lemma

Lemma

The pre and pre^* images of upward closed set are upward closed

1. Let S be an upward closed set.
2. Assume $pre(S)$ is not upward closed.
3. Let $c_1 \in pre(S)$, and let $c_2 \in U$ such that $c_1 \preceq c_2$ and $c_2 \notin pre(S)$

Key lemma

Lemma

The pre and pre* images of upward closed set are upward closed

1. Let S be an upward closed set.
2. Assume $pre(S)$ is not upward closed.
3. Let $c_1 \in pre(S)$, and let $c_2 \in U$ such that $c_1 \preceq c_2$ and $c_2 \notin pre(S)$
4. Let $c_1' \in S$ such that $c_1 \rightarrow c_1'$

Key lemma

Lemma

The pre and pre* images of upward closed set are upward closed

1. Let S be an upward closed set.
2. Assume $pre(S)$ is not upward closed.
3. Let $c_1 \in pre(S)$, and let $c_2 \in U$ such that $c_1 \preceq c_2$ and $c_2 \notin pre(S)$
4. Let $c_1' \in S$ such that $c_1 \rightarrow c_1'$
5. Monotonicity $\Rightarrow$ there is a c_2' such that $c_2 \rightarrow c_2'$ and $c_1' \preceq c_2'$

Key lemma

Lemma

The pre and pre* images of upward closed set are upward closed

1. Let S be an upward closed set.
2. Assume $pre(S)$ is not upward closed.
3. Let $c_1 \in pre(S)$, and let $c_2 \in U$ such that $c_1 \preceq c_2$ and $c_2 \notin pre(S)$
4. Let $c_1' \in S$ such that $c_1 \rightarrow c_1'$
5. Monotonicity $\Rightarrow$ there is a c_2' such that $c_2 \rightarrow c_2'$ and $c_1' \preceq c_2'$
6. S is upward closed $\Rightarrow$ $c_2' \in S$

Key lemma

Lemma

The pre and pre* images of upward closed set are upward closed

1. Let S be an upward closed set.
2. Assume $pre(S)$ is not upward closed.
3. Let $c_1 \in pre(S)$, and let $c_2 \in U$ such that $c_1 \preceq c_2$ and $c_2 \notin pre(S)$
4. Let $c_1' \in S$ such that $c_1 \rightarrow c_1'$
5. Monotonicity $\Rightarrow$ there is a c_2' such that $c_2 \rightarrow c_2'$ and $c_1' \preceq c_2'$
6. S is upward closed $\Rightarrow c_2' \in S$
7. $\Rightarrow c_2 \in pre(S)$, contradiction.

Key lemma

Lemma

The pre and pre* images of upward closed set are upward closed

1. Let S be an upward closed set.
2. Assume $pre(S)$ is not upward closed.
3. Let $c_1 \in pre(S)$, and let $c_2 \in U$ such that $c_1 \preceq c_2$ and $c_2 \notin pre(S)$
4. Let $c_1' \in S$ such that $c_1 \rightarrow c_1'$
5. Monotonicity $\Rightarrow$ there is a c_2' such that $c_2 \rightarrow c_2'$ and $c_1' \preceq c_2'$
6. S is upward closed $\Rightarrow c_2' \in S$
7. $\Rightarrow c_2 \in pre(S)$, contradiction.
8. For pre^*: the union of upward closed sets is upward closed.

Backward Reachability Analysis

Consider the increasing sequence $X_0 \subseteq X_1 \subseteq X_2 \ldots$ defined by:

- $X_0 = Min(S)$
- $X_{i+1} = X_i \cup Min(pre(\overline{X_i}))$

Termination:

There is a index $i \geq 0$ *such that* $X_{i+1} = X_i$

- The set $pre^*(S)$ is upward closed $\Rightarrow$ has a finite minor
- Wait until a minor is collected

Backward Reachability Analysis

Consider the increasing sequence $X_0 \subseteq X_1 \subseteq X_2 \ldots$ defined by:

- $X_0 = Min(S)$
- $X_{i+1} = X_i \cup Min(pre(\overline{X_i}))$

Termination:

There is a index $i \geq 0$ *such that* $X_{i+1} = X_i$

- The set $pre^*(S)$ is upward closed $\Rightarrow$ has a finite minor
- Wait until a minor is collected
- How long shall we wait?

Backward Reachability Analysis

Consider the increasing sequence $X_0 \subseteq X_1 \subseteq X_2 \ldots$ defined by:

- $X_0 = Min(S)$
- $X_{i+1} = X_i \cup Min(pre(\overline{X_i}))$

Termination:

There is a index $i \geq 0$ *such that* $X_{i+1} = X_i$

- The set $pre^*(S)$ is upward closed $\Rightarrow$ has a finite minor
- Wait until a minor is collected
- How long shall we wait?
- Possibly very very long: Non primitive recursive in general

The case of VASS

- Usual $\leq$ order over $\mathbb{N}$ is a WQO (Dickson lemma)
- Product of WQO's is a WQO.
- $\Rightarrow \leq$ generalized to $\mathbb{N}^n$ is a WQO.

The case of VASS

- Usual $\leq$ order over $\mathbb{N}$ is a WQO (Dickson lemma)
- Product of WQO's is a WQO.
- $\Rightarrow \leq$ generalized to $\mathbb{N}^n$ is a WQO.
- Upward-closed sets = finite disjunctions of $\bigwedge_{i=1}^{n} l_i \leq c_i$, where $l_i \in \mathbb{N}$
- Computation of the Pre:
  - $op =$ "$c_j := c_j + 1$" : $(\bigwedge_{i \neq j} l_i \leq c_i) \wedge (max(l_j - 1, 0) \leq c_j)$
  - $op =$ "$c_j > 0 / c_j - 1$": $(\bigwedge_{i \neq j} l_i \leq c_i) \wedge (l_j + 1 \leq c_j)$

The case of VASS

- Usual $\leq$ order over $\mathbb{N}$ is a WQO (Dickson lemma)
- Product of WQO's is a WQO.
- $\Rightarrow \leq$ generalized to $\mathbb{N}^n$ is a WQO.
- Upward-closed sets = finite disjunctions of $\bigwedge_{i=1}^{n} l_i \leq c_i$, where $l_i \in \mathbb{N}$
- Computation of the Pre:
  - $op =$ "$c_j := c_j + 1$" : $(\bigwedge_{i \neq j} l_i \leq c_i) \wedge (max(l_j - 1, 0) \leq c_j)$
  - $op =$ "$c_j > 0 / c_j - 1$": $(\bigwedge_{i \neq j} l_i \leq c_i) \wedge (l_j + 1 \leq c_j)$
- No test to zero, only guards of the form $c > 0 \Rightarrow$ Monotonicity
- $\Rightarrow$ Coverability is decidable.

The case of Lossy Fifo Channel Systems

- Subword relation over a finite alphabet is a WQO (Higman's lemma)

The case of Lossy Fifo Channel Systems

- Subword relation over a finite alphabet is a WQO (Higman's lemma)
- Upward-closed sets = finite unions of

$$\Sigma^* a_1 \Sigma^* a_2 \cdots a_m \Sigma^*$$

- Computation of the Pre:
  - Send: Left concatenation + Upward closure
  - Receive: Right derivation

The case of Lossy Fifo Channel Systems

- Subword relation over a finite alphabet is a WQO (Higman's lemma)
- Upward-closed sets = finite unions of

$$\Sigma^* a_1 \Sigma^* a_2 \cdots a_m \Sigma^*$$

- Computation of the Pre:
  - Send: Left concatenation + Upward closure
  - Receive: Right derivation
- Lossyness ⇒ Monotonicity
- ⇒ Coverability is decidable.

Concurrent Programs with Procedures

- Procedural program $\rightarrow$ Pushdown System (finite control + stack)
- Concurrent program $\rightarrow$ Concurrent PDS's (Multistack systems)

Concurrent Programs with Procedures

- Procedural program $\rightarrow$ Pushdown System (finite control + stack)
- Concurrent program $\rightarrow$ Concurrent PDS's (Multistack systems)
- Two stacks can simulate a Turing tape.
- Concurrent programs with 2 threads are Turing powerful.

Concurrent Programs with Procedures

- Procedural program $\rightarrow$ Pushdown System (finite control + stack)
- Concurrent program $\rightarrow$ Concurrent PDS's (Multistack systems)
- Two stacks can simulate a Turing tape.
- Concurrent programs with 2 threads are Turing powerful.
- $\Rightarrow$ Restrictions
  - Classes of programs with particular features
  - Particular kind of behaviors
    (under-approximate analysis for bug detection)

Asynchronous Programs

- Synchronous calls

  Usual procedure calls

- Asynchronous calls
  - *Calls are stored and dispatched later by the scheduler*
  - *They can be executed in any order*

- Event-driven programming (requests, responses)
- Useful model: distributed systems, web servers, embedded systems

Formal Models: Multiset Pushdown Systems

- A task is a sequential (pushdown) process with dynamic task creation
- Created tasks are stored in an unordered buffer (multiset)
- Tasks run until completion
- If the stack is empty, a task in moved from the multiset to the stack

Difficulties

- Unbounded buffer of tasks
- The buffer is a multiset $\Rightarrow$ can be encoded as counters
- Need to combine somehow PDS with VASS
- Stack $\Rightarrow$ not Well Structured
- How to get rid of the stack ?

State Reachability of Multiset PDS

Theorem

The control state reachability problem for MPDS is EXPSPACE-complete.

Reduction to/from the coverability problem for Petri.

First decidability proof by K. Sen and M. Viswanathan, 2006

Semi-linear Sets

- Linear set over $\mathbb{N}^n$ is a set of the form

$$\{\vec{u} + k_1\vec{v_1} + \cdots + k_m\vec{v_m} \ : \ k_1, \ldots, k_m \in \mathbb{N}\}$$

  where $\vec{u}, \vec{v_1}, \ldots, \vec{v_m} \in \mathbb{N}^n$

- Semi-linear set = finite union of linear sets.

- Examples:
  - $\{(0,0) + k(1,1) \ : \ k \geq 0\} \ \equiv \ x_1 = x_2$
  - $\{(0,0) + k(1,2) \ : \ k \geq 0\} \ \equiv \ 2x_1 = x_2$
  - $\{(0,3) + k(1,1) \ : \ k \geq 0\} \ \equiv \ x_1 + 3 = x_2$
  - $\{(0,3) + k_1(0,1) + k_2(1,1) \ : \ k \geq 0\} \ \equiv \ x_1 + 3 \leq x_2$
  - $\{(0,0,0) + k_1(1,0,1) + k_2(0,1,1) \ : \ k_1, k_2 \geq 0\} \ \equiv \ x_1 + x_2 = x_3$
  - $\{(0,0,3) + k_1(1,0,2) + k_2(0,1,1) \ : \ k_1, k_2 \geq 0\} \ \equiv \ 2x_1 + x_2 + 3 = x_3$

Semi-linear Sets

- Linear set over $\mathbb{N}^n$ is a set of the form

$$\{\vec{u} + k_1\vec{v_1} + \cdots + k_m\vec{v_m} \; : \; k_1, \ldots, k_m \in \mathbb{N}\}$$

  where $\vec{u}, \vec{v_1}, \ldots, \vec{v_m} \in \mathbb{N}^n$

- Semi-linear set = finite union of linear sets.

- Examples:
  - $\{(0,0) + k(1,1) \; : \; k \geq 0\} \; \equiv \; x_1 = x_2$
  - $\{(0,0) + k(1,2) \; : \; k \geq 0\} \; \equiv \; 2x_1 = x_2$
  - $\{(0,3) + k(1,1) \; : \; k \geq 0\} \; \equiv \; x_1 + 3 = x_2$
  - $\{(0,3) + k_1(0,1) + k_2(1,1) \; : \; k \geq 0\} \; \equiv \; x_1 + 3 \leq x_2$
  - $\{(0,0,0) + k_1(1,0,1) + k_2(0,1,1) \; : \; k_1, k_2 \geq 0\} \; \equiv \; x_1 + x_2 = x_3$
  - $\{(0,0,3) + k_1(1,0,2) + k_2(0,1,1) \; : \; k_1, k_2 \geq 0\} \; \equiv \; 2x_1 + x_2 + 3 = x_3$

- Theorem [Ginsburg, Spanier, 1966]

  A set is semi-linear iff it is definable in Presburger arithmetics.

Parikh's image

- Let $\Sigma = \{a_1, \ldots, a_n\}$.
- Given a word $w \in \Sigma^*$, the *Parikh image* of w is:

$$\phi(w) = (\#_{a_1}(w), \ldots, \#_{a_n}(w)) \in \mathbb{N}^n$$

- Given a language $L \subseteq \Sigma^*$, $\phi(L) = \{\phi(w) \ : \ w \in L\}$
- Examples:
  - $L_1 = \{a^n b^n \ : \ n \geq 0\}$, $\phi(L_1) = \{(x_1, x_2) \ : \ x_1 = x_2\}$
  - $L_2 = \{a^n b^n c^n \ : \ n \geq 0\}$, $\phi(L_2) = \{(x_1, x_2, x_3) \ : \ x_1 = x_2 \land x_2 = x_3\}$
  - $L_3 = (ab)^* = \{(ab)^n \ : \ n \geq 0\}$, $\phi(L_3) = \{(x_1, x_2) \ : \ x_1 = x_2\}$

Semi-linear sets, CFL's, and RL's

- Parikh's Theorem (1966)

  For every Context-Free Language L, $\phi(L)$ *is a semi-linear set.*

Semi-linear sets, CFL's, and RL's

- Parikh's Theorem (1966)

  For every Context-Free Language L, $\phi(L)$ is a semi-linear set.

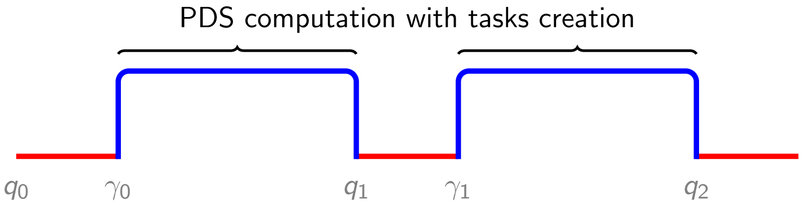
- Proposition

  For every semi-linear set S, there exists a Regular Language L such that $\phi(L) = S$.

- Corollary

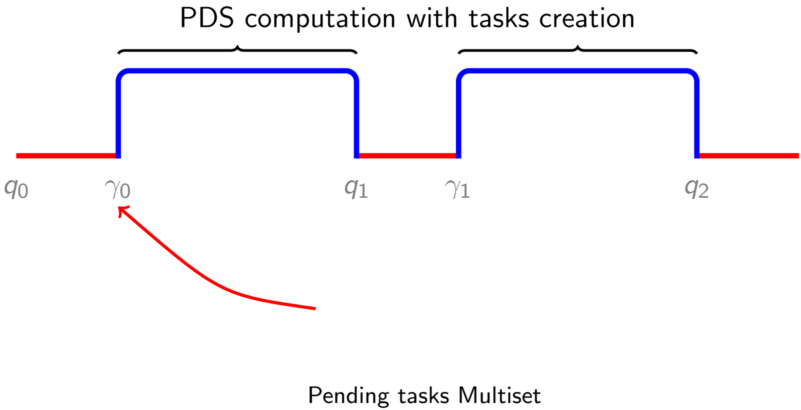
  For every Context-Free Language L, there exists a Regular language L' such that $\phi(L) = \phi(L')$.

From Multiset PDS to VASS

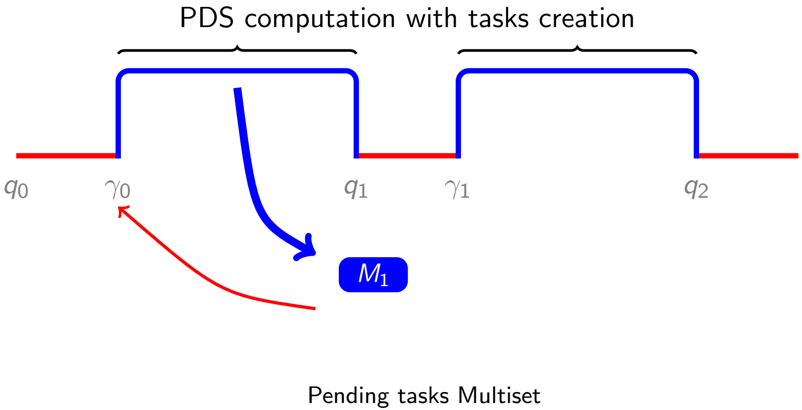


Pending tasks Multiset

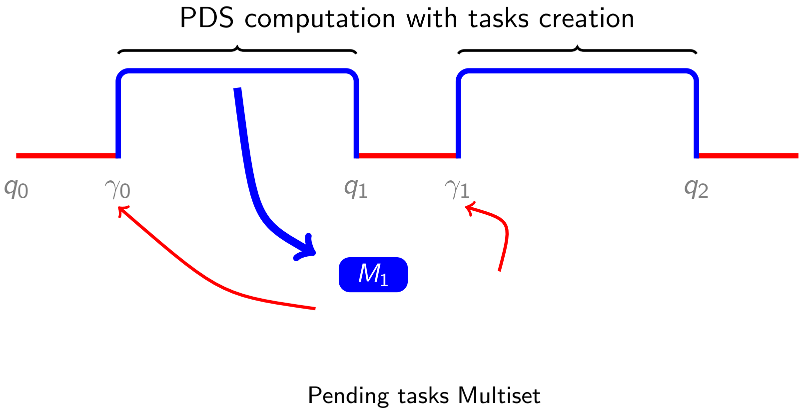
From Multiset PDS to VASS



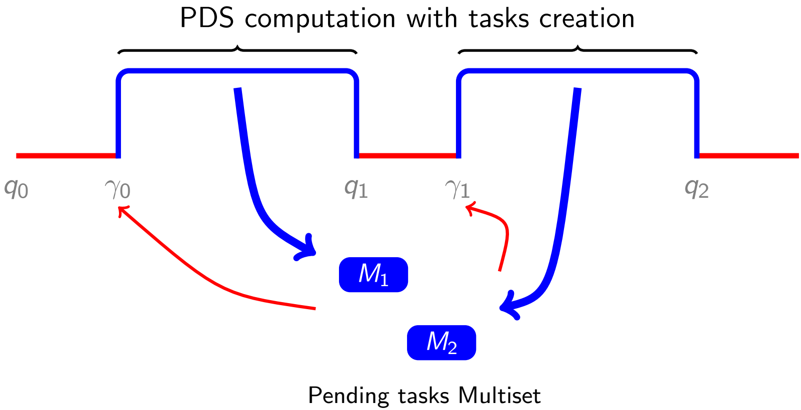
From Multiset PDS to VASS



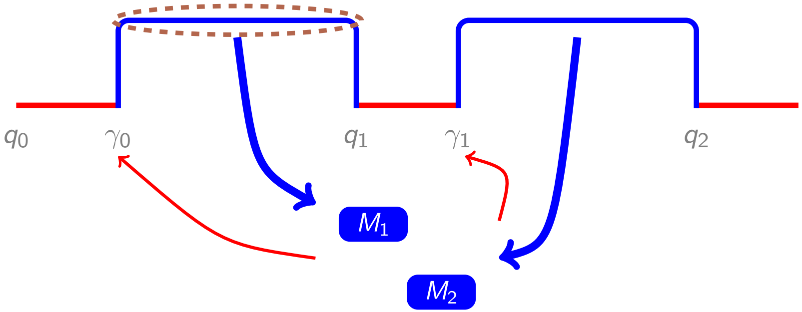
From Multiset PDS to VASS



From Multiset PDS to VASS



From Multiset PDS to VASS



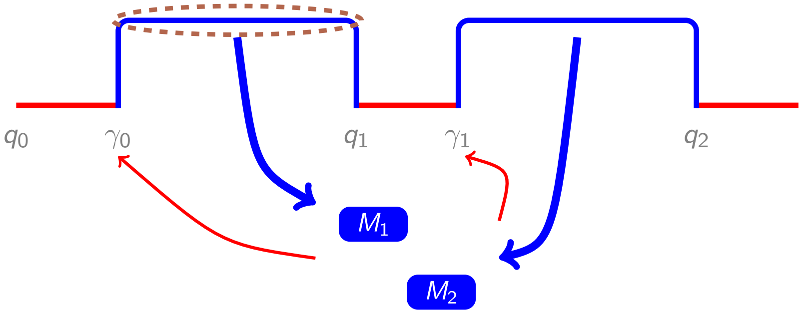
$$q_0, \gamma_0 \overset{L_1}{\Longrightarrow}^* q_1, \epsilon$$

L_1 = Set of sequences of created tasks

L_1 is a Context-Free Language

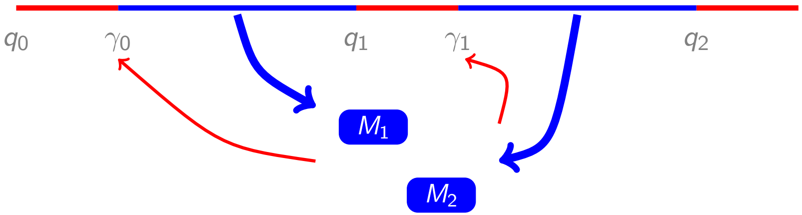
M_1 is the Parikh image of L_1

From Multiset PDS to VASS



Parikh's Theorem: M_i is definable by a finite state automaton S_i

From Multiset PDS to VASS



Parikh's Theorem: M_i is definable by a finite state automaton S_i

Construction of a VASS: Simulation of S_i + task consumption rules

Message-Passing Programs with Procedures

- Undecidable even for unbounded FIFO channels
- Restrictions on
  - Interaction between recursion and communication (e.g., communication with empty stack)
  - Kind of channels (e.g., lossy, unordered)
  - Topology of the network
- Decidable classes

  [La Torre et al. TACAS'08], [Atig et al., CONCUR'08], ...

Concurrent Programs: Under-approximate analysis

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

Concurrent Programs: Under-approximate analysis

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

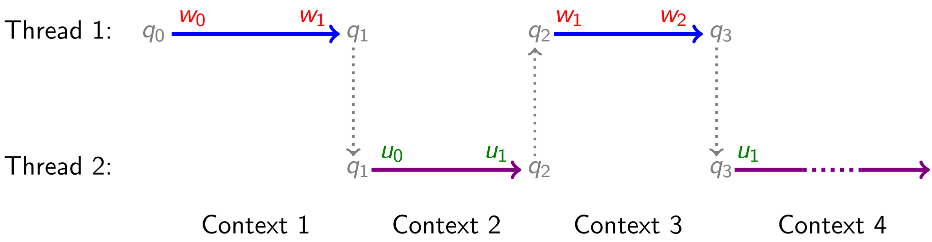
Concurrent Programs: Under-approximate analysis

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

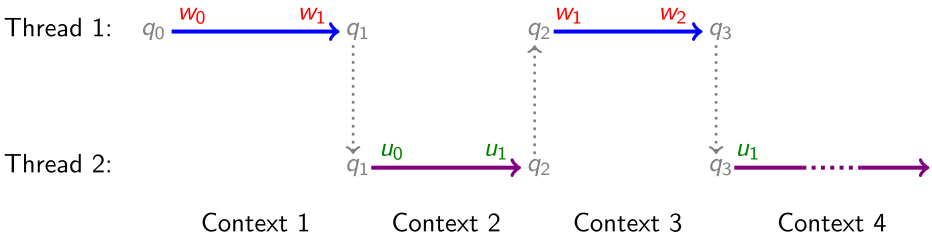


- Suitable for finding bugs in concurrent programs.
- Concurrency bugs show up after a small number of context switches.

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
  - $A'_{j+1} = pre^*_{i_{j+1}}(X_{j+1}[i_{j+1}]) \cap q_j \Gamma_i^*$
  - $X_j = (q_j, A_1^{j+1}, \ldots, A'_{j+1}, \ldots, A_n^{j+1})$

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)

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)
- Round Robin thread scheduling. K = number of rounds

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)
- Round Robin thread scheduling. $K =$ number of rounds
- Guess an *interface* of each thread:
  - $I^i = (I^i_1, \ldots I^i_K)$, the global states when T_i starts/is resumed
  - $O^i = (O^i_1, \ldots O^i_K)$, the global states when T_i terminates/is interrupted

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)
- Round Robin thread scheduling. K = 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

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)
- Round Robin thread scheduling. K = 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

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)
- Round Robin thread scheduling. K = number of rounds
- Guess an *interface* of each thread:
  - $I^i = (I^i_1, \ldots I^i_K)$, the global states when T_i starts/is resumed
  - $O^i = (O^i_1, \ldots O^i_K)$, 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 $j \in \{1, \ldots, K\}$
  - $O^2_j = I^1_{j+1}$ for every $j \in \{1, \ldots, K-1\}$

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.

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 $j \in \{1, \ldots, K\}$
- Simulation of T_1. At each round $j \in \{1, \ldots, K\}$ do:
  1. Assign $*$ to all variables of X_j (guesses the input I^1_j)
  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}).
  5. Whenever T_1 reaches q_1, start simulating T_2.

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 $j \in \{1, \ldots, K\}$
- Simulation of T_1. At each round $j \in \{1, \ldots, K\}$ do:
  1. Assign $*$ to all variables of X_j (guesses the input I^1_j)
  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}).
  5. Whenever T_1 reaches q_1, start simulating T_2.
- Simulation of T_2. At each round j do:
  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 $j+1$
  5. If q_2 is reachable at round K, then go to state q_{win}

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.

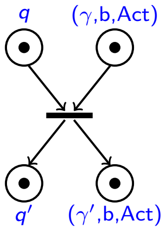
Reduction to coverability in PN

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

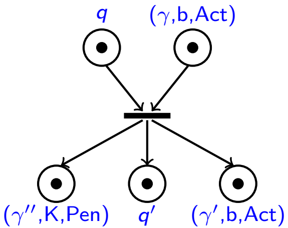
Rule of the form: $q\gamma \longrightarrow q'\gamma'$ $\Longrightarrow$



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).

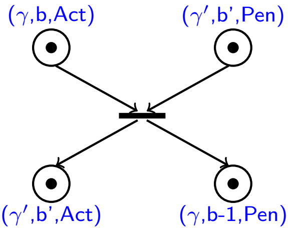
Rule of the form: $q\gamma \longrightarrow q'\gamma' \rhd \gamma''$ $\Longrightarrow$



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

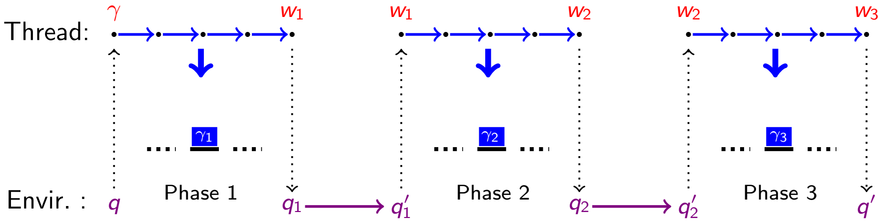
- Decidable ?
- Difficulty:
  - Unbounded number of pending local contexts
  - Can not use the same construction as for the case of finite state threads. (This would need an unbounded number of places.)

Theorem

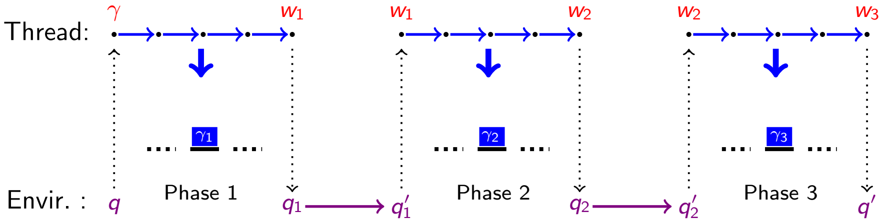
The K-bounded state reachability problem is in 2EXPSPACE.

Exponential reduction to the coverability problem in PN

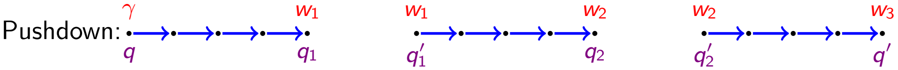
Making visible the interactions



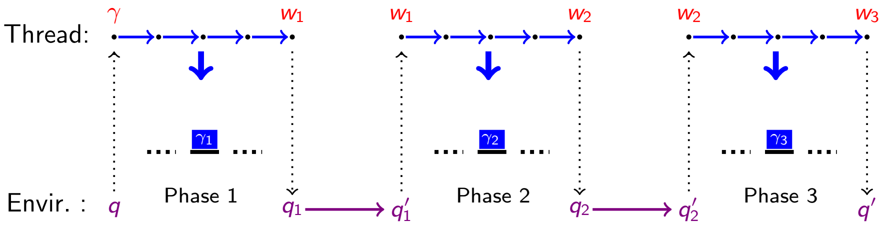
Making visible the interactions



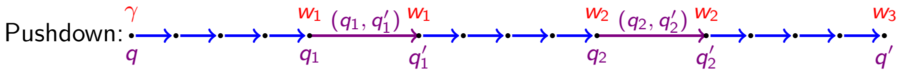
- Construct a labeled pushdown automaton which:
  - Guesses the effect of the environment on the states



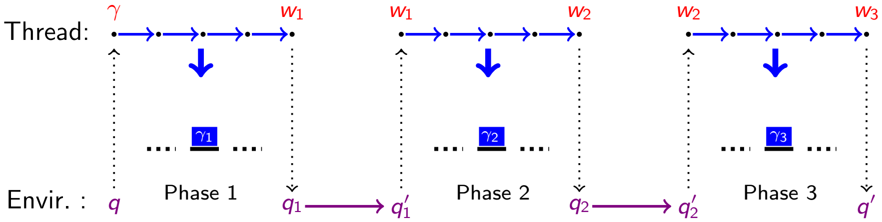
Making visible the interactions



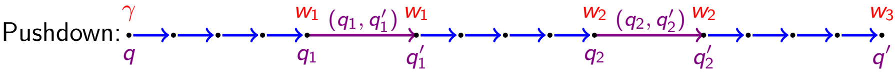
- Construct a labeled pushdown automaton which:
  - Guesses the effect of the environment on the states



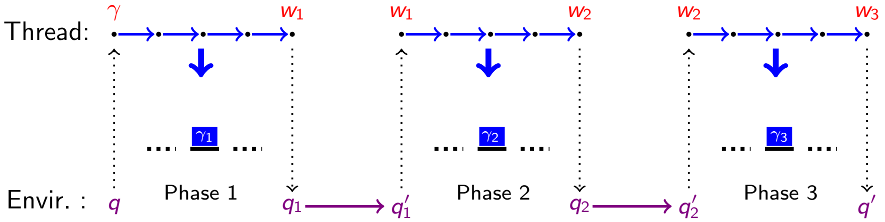
Making visible the interactions



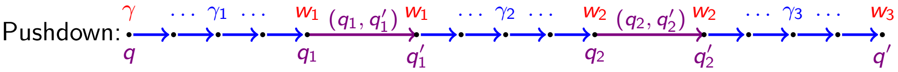
- Construct a labeled pushdown automaton which:
  - Makes visible (as transition labels) the created threads



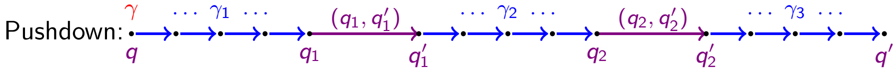
Making visible the interactions



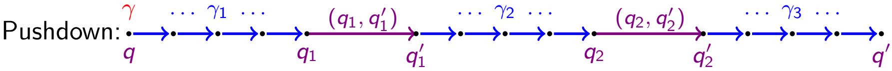
- Construct a labeled pushdown automaton which:
  - Makes visible (as transition labels) the created threads



Constructing a regular interface

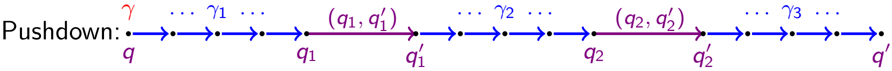


Constructing a regular interface



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

Constructing a regular interface

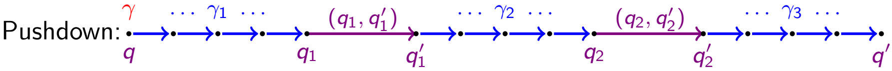


- The set of traces L characterizes the interaction between the thread and its environment (L is a CFL)

Observations: For the state reachability problem

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

Constructing a regular interface



- 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 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

Summary

- Complex / Undecidable in general (communication + recursion)
- Decidable class of concurrent programs: Asynchronous Programs
- Reduction to coverability in VASS (Petri Nets)

Summary

- Complex / Undecidable in general (communication + recursion)
- Decidable class of concurrent programs: Asynchronous Programs
- Reduction to coverability in VASS (Petri Nets)
- Too complex to be scalable

Summary

- Complex / Undecidable in general (communication + recursion)
- Decidable class of concurrent programs: Asynchronous Programs
- Reduction to coverability in VASS (Petri Nets)
- Too complex to be scalable
- Under-approximate analysis: Context-/Delay- Bounded Analysis
- Sequentialization: Code-to-code translation to Sequential Programs

Summary

- Complex / Undecidable in general (communication + recursion)
- Decidable class of concurrent programs: Asynchronous Programs
- Reduction to coverability in VASS (Petri Nets)
- Too complex to be scalable
- Under-approximate analysis: Context-/Delay- Bounded Analysis
- Sequentialization: Code-to-code translation to Sequential Programs
- Other decidability results are based on "sequentialization"

  e.g., Ordered Multi-pushdown systems [Atig, CONCUR'10].

Summary

- Complex / Undecidable in general (communication + recursion)
- Decidable class of concurrent programs: Asynchronous Programs
- Reduction to coverability in VASS (Petri Nets)
- Too complex to be scalable
- Under-approximate analysis: Context-/Delay- Bounded Analysis
- Sequentialization: Code-to-code translation to Sequential Programs
- Other decidability results are based on "sequentialization"
  e.g., Ordered Multi-pushdown systems [Atig, CONCUR'10].
- Message-passing programs: Phase bounding [B., Emmi, TACAS'12]

Summary

- Complex / Undecidable in general (communication + recursion)
- Decidable class of concurrent programs: Asynchronous Programs
- Reduction to coverability in VASS (Petri Nets)
- Too complex to be scalable
- Under-approximate analysis: Context-/Delay- Bounded Analysis
- Sequentialization: Code-to-code translation to Sequential Programs
- Other decidability results are based on "sequentialization"

  e.g., Ordered Multi-pushdown systems [Atig, CONCUR'10].
- Message-passing programs: Phase bounding [B., Emmi, TACAS'12]
- Infinite behaviors (liveness bugs):
  - K-context-bounded ultimately periodic behaviors

    [Atig, B., Emmi, Lal, CAV'12]
  - Scope-bounded analysis

    [LaTorre, Napoli, CONCUR'11], [Atig, B., N. Kumar, Saivasan, ATVA'12]