Verification of parameterized shared-memory asynchronous systems

Anca Muscholl
LaBRI, Bordeaux University
IAS, TU Munich

Joint work with: M. Fortin, S. LaTorre, I. Walukiewicz
Model: asynchronous, shared memory

Hague, 2011

Esparza, Ganty, Majumdar, 2013

Durand-Gasselin, Esparza, Ganty, Majumdar, 2015

leader

Reg

contributors
Model: asynchronous, shared memory

Hague, 2011
Esparza, Ganty, Majumdar, 2013
Durand-Gasselin, Esparza, Ganty, Majumdar, 2015

Register: lock-free reads/writes

leader D
(C,D)-systems
contributors C
Leader and contributors are pushdown processes.

Contributors do not have identities.

With locks: a unique contributor can be distinguished.

With only one contributor: leader+contributor can simulate a Turing machine.

Without locks and with an arbitrary number of contributors: the model becomes surprisingly manageable.

Parametrization as powerful abstraction (Kahlon, 2008)
Multiset semantics of (C,D)-systems

\[ C = \langle S, \delta \subseteq S \times \Sigma_C \times S, s_{\text{init}} \rangle \quad D = \langle T, \Delta \subseteq T \times \Sigma_D \times T, t_{\text{init}} \rangle . \]

**contributor**

**leader**

\[ G: \text{a finite set of register values} \]

A configuration is \((M, t, g)\), where \(M \in \mathbb{N}^S\), \(t \in T\), \(g \in G\).

\[
\begin{align*}
(M, t, g) \xrightarrow{w(h)} (M, t', h) & \quad \text{if } t \xrightarrow{w(h)} t' \text{ in } \Delta, \\
(M, t, g) \xrightarrow{r(h)} (M, t', h) & \quad \text{if } t \xrightarrow{r(h)} t' \text{ in } \Delta \text{ and } h = g, \\
(M, t, g) \xrightarrow{\bar{w}(h)} (M', t, h) & \quad \text{if } M \xrightarrow{\bar{w}(h)} M' \text{ in } \delta, \\
(M, t, g) \xrightarrow{\bar{r}(h)} (M', t, h) & \quad \text{if } M \xrightarrow{\bar{r}(h)} M' \text{ in } \delta \text{ and } h = g.
\end{align*}
\]

where \(M \xrightarrow{a} M'\) in \(\delta\) \quad \text{if } s \xrightarrow{a} s' \text{ in } \delta \text{ and } M' = M - [s] + [s']\), for some \(s, s' \in S\).
(M, t, g) \xrightarrow{w(h)} (M, t', h) \quad \text{if } t \xrightarrow{w(h)} t' \text{ in } \Delta,
(M, t, g) \xrightarrow{r(h)} (M, t', h) \quad \text{if } t \xrightarrow{r(h)} t' \text{ in } \Delta \text{ and } h = g,
(M, t, g) \xrightarrow{\bar{w}(h)} (M', t, h) \quad \text{if } M \xrightarrow{\bar{w}(h)} M' \text{ in } \delta,
(M, t, g) \xrightarrow{\bar{r}(h)} (M', t, h) \quad \text{if } M \xrightarrow{\bar{r}(h)} M' \text{ in } \delta \text{ and } h = g.

where

\[
M \xrightarrow{a} M' \text{ in } \delta \quad \text{if } s \xrightarrow{a} s' \text{ in } \delta \text{ and } M' = M - [s] + [s'], \text{ for some } s, s' \in S.
\]
C, D may be infinite-state

\[ C = \langle S, \delta \subseteq S \times \Sigma_C \times S, s_{init} \rangle \quad D = \langle T, \Delta \subseteq T \times \Sigma_D \times T, t_{init} \rangle. \]

Contributor \quad Leader

Transition systems C and D need not to be finite. In our case they are given by pushdown systems:

\[ \mathcal{A}_C = \langle P, \Sigma_C, \Gamma_C, \delta, p_{init}, A_{init}^C \rangle \quad \mathcal{A}_D = \langle Q, \Sigma_D, \Gamma_D, \Delta, q_{init}, A_{init}^D \rangle. \]

So \( S = \{ q\alpha : q \in P, \alpha \in \Gamma_C^* \} \)
Every contributor proposes a value
* Leader chooses one of these values
* The rest of the protocol uses the chosen value

Example of properties:
(for every n, for every run)
* Leader eventually decides on a value
* If the leader decides on a value, contributors use only this value
* On runs where only one value is used i.o. the protocol is correct
Example of a system:
- Contributors proposes values.
- Leader chooses one of these values.
- The rest of the protocol uses the chosen value.

Example properties:
(for every n, for every run)
- Leader eventually decides on a value
- If the leader decides on a value, contributors use only this value.
- On runs where only one value is used i.o. the protocol is correct.

There is a run where the leader has decided on some value and afterwards a contributor uses a different value.
Example of a system:

- Contributors proposes values.
- Leader chooses one of these values.
- The rest of the protocol uses the chosen value.

Example properties:
(for every n, for every run)

- Leader eventually decides on a value.
- If the leader decides on a value, contributors use only this value.
- On runs where only one value is used i.o. the protocol is correct.

There is a maximal run where the leader does not decide on any value.
Example of a system:

- Contributors propose values.
- Leader chooses one of these values.
- The rest of the protocol uses the chosen value.

Example properties:
(for every n, for every run)

- Leader eventually decides on a value
- If the leader decides on a value, contributors use only this value.
- On runs where only one value is used i.o., the protocol is correct
(C,D)-systems

- Reachability
- Repeated reachability
- Safety
- Verification of properties

- PSPACE-complete for pushdowns
- Almost always decidable
- Hierarchical (C,D)-systems
- NEXPTIME-complete for pushdowns
- PSPACE-complete if only about infinite runs
- C-stutter-expanding properties
- Nexpctime-Comple
We are interested in the complexity of deciding these properties when C, D are pushdown systems.
(C,D)-systems

- Reachability
  - PSPACE-complete for pushdowns
  - Almost always decidable
  - Hierarchical (C,D)-systems
- Repeated reachability
- Safety
- Verification of properties
Reachability in (C,D)-systems

Given a leader D from some class of systems \( \mathcal{D} \) and a contributor C from some class \( \mathcal{C} \), is there some integer \( n \) such that D, together with \( n \) copies of C, writes a particular value into the register along some run?

**Fact**

When \( C \) and \( D \) are pushdown systems and \( n \) is known, the problem is undecidable.
Reachability in (C,D)-systems

Given a leader D from some class of systems \( \mathcal{D} \) and a contributor C from some class \( \mathcal{C} \), is there some integer n such that D, together with n copies of C, writes a particular value into the register along some run?

**Thm** [Hague, Esparza et al. 2013]

When \( \mathcal{C} \) and \( \mathcal{D} \) are pushdown systems then the reachability problem is decidable (PSPACE-complete). If \( \mathcal{C} \) is finite-state systems, the problem is NP-complete.
Accumulator semantics

\[
(B, t, g) \xrightarrow{w(h)} (B, t', h) \quad \text{if } t \xrightarrow{w(h)} t' \text{ in } \Delta, \quad B \subseteq S
\]

\[
(B, t, g) \xrightarrow{r(h)} (B, t', h) \quad \text{if } t \xrightarrow{r(h)} t' \text{ in } \Delta \text{ and } h = g,
\]

\[
(B, t, g) \xrightarrow{\bar{w}(h)} (B', t, h) \quad \text{if } B \xrightarrow{\bar{w}(h)} B' \text{ in } \delta,
\]

\[
(B, t, g) \xrightarrow{\bar{r}(h)} (B', t, h) \quad \text{if } B \xrightarrow{\bar{r}(h)} B' \text{ in } \delta \text{ and } h = g.
\]

\[
B \xrightarrow{a} B' \text{ in } \delta \quad \text{if } s \xrightarrow{a} s' \text{ in } \delta \text{ and } B' = B \cup \{s'\}, \text{ for some } s, s' \in S.
\]

\[
(M, t, g) \xrightarrow{w(h)} (M, t', h) \quad \text{if } t \xrightarrow{w(h)} t' \text{ in } \Delta,
\]

\[
(M, t, g) \xrightarrow{r(h)} (M, t', h) \quad \text{if } t \xrightarrow{r(h)} t' \text{ in } \Delta \text{ and } h = g,
\]

\[
(M, t, g) \xrightarrow{\bar{w}(h)} (M', t, h) \quad \text{if } M \xrightarrow{\bar{w}(h)} M' \text{ in } \delta,
\]

\[
(M, t, g) \xrightarrow{\bar{r}(h)} (M', t, h) \quad \text{if } M \xrightarrow{\bar{r}(h)} M' \text{ in } \delta \text{ and } h = g.
\]

\[
M \xrightarrow{a} M' \text{ in } \delta \quad \text{if } s \xrightarrow{a} s' \text{ in } \delta \text{ and } M' = M - [s] + [s'], \text{ for some } s, s' \in S.
\]
**Multiset**

\[
t \xrightarrow{a} t' \quad t' \quad t'
\]

\[
g \quad g' \quad g'' \quad g'''
\]

\[
s_1 \quad s_1 \xrightarrow{b} s_1' \quad s_1'
\]

\[
\vdots \quad \vdots \quad \vdots \quad \vdots
\]

\[
s_i \quad s_i \quad s_i \xrightarrow{c} s_i' \quad s_i'
\]

\[
\vdots \quad \vdots \quad \vdots \quad \vdots
\]

\[
s_n \quad s_n \quad s_n \quad s_n
\]

**Accumulator**

\[
t \xrightarrow{a} t' \quad t' \quad t'
\]

\[
g \quad g' \quad g'' \quad g'''
\]

\[
s_1 \quad s_1 \quad s_1 \quad s_1 \quad s_1
\]

\[
\vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots
\]

\[
s_i \quad s_i \xrightarrow{b} s_i \quad s_i \quad s_i
\]

\[
\vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots
\]

\[
s_n \quad s_n \quad s_n \xrightarrow{c} s_n \quad s_n \quad s_n
\]

\[
\vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots
\]

\[
s_1' \quad s_1' \quad s_i' \quad s_i' \quad s_i'
\]

\[
\vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots
\]

\[
s_1' \quad s_1' \quad s_i' \quad s_i' \quad s_i'
\]
Let $C$ and $D$ be both effectively closed under synchronized product with finite-state systems. If $C$ has a decidable reachability problem and $D$ has effective downward closure, the reachability for $(C,D)$-systems is decidable.

**Thm** [La Torre, M., Walukiewicz 2015]

Let $C$ and $D$ be both effectively closed under synchronized product with finite-state systems. If $C$ has a decidable reachability problem and $D$ has effective downward closure, the reachability for $(C,D)$-systems is decidable.
Thm

Let $\mathcal{C}$ and $\mathcal{D}$ be both effectively closed under synchronized product with finite-state systems.

If $\mathcal{C}$ has a decidable reachability problem and $\mathcal{D}$ has effective downward closure, then reachability for $(\mathcal{C}, \mathcal{D})$-systems is decidable.

\textit{$\mathcal{C}$ is effectively closed under synchronized product with finite-state systems:} 
given M from $\mathcal{C}$ and a finite automaton A, the synchronized product of M and A belongs to $\mathcal{C}$ and can be effectively constructed.

\textit{$\mathcal{D}$ has effective downward closure:} 
given M from $\mathcal{D}$, the finite automaton accepting all (scattered) subwords of traces of M can be constructed effectively.
Effective downward closure:

- pushdown automata [Courcelle 1991]
- Petri nets [Habermehl et al. 2010]
- stacked counter automata [Zetzsche 2015]
- higher-order pushdown wo / with collapse
  [Hague, Kochems, Ong 2016] [Clemente, Parys, Salvati, Walukiewicz 2016]

Theorem applies to

leader: pushdown automata, Petri nets, decidable subclasses of multi-stack, stacked counter automata.

contributors: any of the above, lossy channel systems, hierarchical composition of (C,D)-systems.
Hierarchical composition of (C,D)-systems

leader $P_0$ and each subtree (C,D)-system is contributor
(C,D)-systems

- Reachability
  - PSPACE-complete for pushdowns
  - Almost always decidable
  - Hierarchical (C,D)-systems

- Repeated reachability
  - PSPACE-complete for pushdowns

- Safety

- Verification of properties
Repeated reachability in (C,D)-systems

Given a leader D from some class of systems \( \mathcal{D} \) and a contributor C from some class \( \mathcal{C} \), is there some integer \( n \) such that D, together with \( n \) copies of C, writes a particular value into the register infinitely often along some run?

Thm [Durand-Gasselin, Esparza, Ganty, Majumdar 2015]

When \( \mathcal{C} \) and \( \mathcal{D} \) are pushdown systems: the repeated reachability problem is PSPACE-hard and in NEXPTIME.
Repeated reachability in (C,D)-systems

Given a leader D from some class of systems $D$ and a contributor C from some class $C$, is there some integer n such that D, together with n copies of C, writes a particular value into the register infinitely often along some run?

Thm [Fortin, M., Walukiewicz 2016]

When $C$ and $D$ are pushdown systems: the repeated reachability problem is PSPACE-complete.
NEXPTIME (Esparza et al.): reduce the problem to \( C = \) NFA of exponential size by bounding the stack

The \((C, D)\)-system has a live run iff there is some Büchi trace \( uv^\omega \) s.t.

- leader pushdown has effective stack height 1 after each \( uv^k \) and same (state, top of stack)
- multiset of contributor states is the same after each \( uv^k \)
Separate leader D and contributor C according to the first writes of C:

Trace of D

\[ v_1 \text{ new}(h_1)v_2 \text{ new}(h_2) \cdots v_k \text{ new}(h_k)v_{k+1} \]

is omega-supported if for every \( 1 \leq j \leq k \) there exists a loop of C

\[ u_1 \text{ new}(h_1)u_2 \cdots u_j \text{ new}(h_j)\overline{w}(h_j) \cdots v_k \text{ new}(h_k)u_{k+1} \]

such that \( v_i = \) projection of \( u_i \) on alphabet of D for every \( i \)
For each $j$: contributor run $\rho(j)$ supporting $\text{new}(h_j)$.

Use $(2n+1)$ copies of $\rho(j)$ to support all $n$ reads of $h_j$ in the loop.
(C,D)-systems

Reachability
- PSPACE-complete for pushdowns
- Almost always decidable
- Hierarchical (C,D)-systems

Repeated reachability
- PSPACE-complete for pushdowns

Safety
- NEXPTIME-complete for pushdowns
- PSPACE-complete if only about infinite runs

Verification of properties
Safety in (C,D)-systems

Given a leader D from some class of systems $D$ and a contributor C from some class $C$, is there some integer $n$ such that $D$, together with $n$ copies of $C$, does not write a particular value into the register along some maximal run?

Thm

When $C$ and $D$ are pushdown systems then the safety problem is NEXPTIME-complete.
**Thm**

If $C$ and $D$ are pushdown systems the safety problem is NEXPTIME-complete.

**Thm**

Let $C$ and $D$ be pushdown systems.
Knowing if there is some infinite safe run is PSPACE-complete.
Knowing if there is some maximal finite safe run is NEXPTIME-complete.

**Thm**

If $C$ is finite-state systems and $D$ is pushdown systems the problems are NP-complete.
The intuition is that we consider the case of contributors given by a pushdown automaton. As for liveness we take the transition. The transitions in the set semantics are essentially the same as in the multiset semantics of Section 2 when only finite traces are considered. The idea is that since the number of contributors is arbitrary, we can always add contributors equivalent to the multiset semantics of Section 2.

**Theorem 12.** Let $M$ be a one-counter system with the notations as in Eq. (1) on page 3:

$\Delta = \{ (B, t, g) \xrightarrow{w(h)} (B, t', h) \mid B \subseteq S \}$

- If $t \xrightarrow{w(h)} t'$ in $\Delta$ and $h = g$.
- If $B \xrightarrow{\bar{w}(h)} B'$ in $\delta$.
- If $B \xrightarrow{\bar{r}(h)} B'$ in $\delta$ and $h = g$.

$B \xrightarrow{a} B'$ in $\delta$ if $s \xrightarrow{a} s'$ in $\delta$, and $B'$ is either $B \cup \{s'\}$ or $(B \cup \{s'\}) \setminus \{s\}$ for some $s \in B$.

The (C,D)-system has a finite, maximal run ending in configuration $(M,t,g)$ in the multiset semantics iff it has a finite, maximal run ending in $(B,t,g)$ in the set semantics, with $B = \text{support}(M)$. 

- If $t \xrightarrow{w(h)} t'$ in $\Delta$ and $h = g$.
- If $B \xrightarrow{\bar{w}(h)} B'$ in $\delta$.
- If $B \xrightarrow{\bar{r}(h)} B'$ in $\delta$ and $h = g$.
Prop
When $C$ and $D$ are pushdown systems, the existence of a maximal finite safe run is NEXPTIME-hard.

Reduction from tiling problem:
Find a tiling with symbols from $\Sigma$ of a $2^n \times 2^n$ square.
The tiling should respect neighborhood relations $H, V \subseteq \Sigma \times \Sigma$.

Leader writes: $A_{1,1}, \overline{A}_{1,1}, A_{1,2}, \overline{A}_{1,2}, \ldots, A_{1,2^n}, \overline{A}_{1,2^n}, \ldots, A_{2^n,2^n}, \overline{A}_{2^n,2^n} (\$\$)^{2^n}$. 
and checks the horizontal dependencies.
Prop
When $C$ and $D$ are the class of pushdown systems then the existence of a maximal finite safe run is NEXPTIME-hard.

Reduction from tiling problem:
Find a tiling with symbols from $\Sigma$ of an $2^n \times 2^n$ square.
The tiling should respect neighborhood relations $H, V \subseteq \Sigma \times \Sigma$.

Leader writes: $A_{1,1}, \overline{A_{1,1}}, A_{1,2}, \overline{A_{1,2}}, \ldots, A_{1,2^n}, \overline{A_{1,2^n}}, \ldots, A_{2^n,2^n}, \overline{A_{2^n,2^n}} (\$\$)^{2^n}$

and checks the horizontal dependencies.

Contributors check vertical dependencies, using counting.

Leader ensures that

- every vertical dependency is checked by some contributor, and
- inconsistencies lead to write an error value in the register
(C,D)-systems

- **Reachability**
  - PSPACE-complete for pushdowns
  - Almost always decidable
  - Hierarchical (C,D)-systems

- **Repeated reachability**
  - PSPACE-complete for pushdowns

- **Safety**
  - NEXPTIME-complete for pushdowns
  - PSPACE-complete if only about infinite runs

- **Verification of properties**
  - C-stutter-expanding properties
  - NEXPTIME-complete
A **trace** is a sequence of register operations during a run. 
**Maximal trace** comes from a maximal run (finite or infinite).

A property of traces is $P \subseteq (\Sigma_D \cup \Sigma_C)^\infty$

A property is **C-stutter-expanding** if it is closed under **replicating** contributor actions.

If $x \bar{w}(g) y \in P$ then $x \bar{w}(g) \bar{w}(g) y \in P$

**Verification of properties of (C,D)-systems**

Given a C-stuttering-expanding property $P$. Given a leader $D$ from some class of systems $\mathcal{D}$ and a contributor $C$ from some class $\mathcal{C}$, is there some integer $n$ such that $D$, together with $n$ copies of $C$, has a **maximal** trace in $P$?
Verification of properties of \((C, D)\)-systems

Given a C-stuttering-expanding property \(P\). Given a leader \(D\) from some class of systems \(\mathcal{D}\) and a contributor \(C\) from some class \(\mathcal{C}\), is there some integer \(n\) such that \(D\), together with \(n\) copies of \(C\), has a \textbf{maximal} trace in \(P\)?

All previously considered properties are special instances:

- **reachability**: \(P\) is the set of traces containing the special action.
- **repeated reachability**: \(P\) is the set of traces containing the special action infinitely often.
- **safety**: \(P\) is the set of traces without the special action.
Verification of properties of (C,D)-systems

Given a C-stuttering-expanding property \( P \). Given a leader \( D \) from some class of systems \( \mathcal{D} \) and a contributor \( C \) from some class \( \mathcal{C} \), is there some integer \( n \) such that \( D \), together with \( n \) copies of \( C \), has a maximal trace in \( P \)?

A property of traces is \( P \subseteq (\Sigma_D \cup \Sigma_C)^\infty \).
A property is C-stutter-expanding if it is closed under replicating actions of contributors.

\[
\text{If } x \bar{w}(g) y \in P \text{ then } x \bar{w}(g)\bar{w}(g) y \in P
\]

Rem: Verification of arbitrary regular properties is undecidable, as with a property we can require that there is only one copy of a contributor.

Thm

When \( C \) and \( D \) are pushdown systems then the verification of regular properties of (C,D)-systems is NEXPTIME-complete.
A property of traces is $P \subseteq (\Sigma_D \cup \Sigma_C)^\infty$.

A property is **C-stutter-expanding** if it is closed under duplicating actions of contributors.

If $x \bar{w}(g) y \in P$ then $x \bar{w}(g)\bar{w}(g) y \in P$

The verification of C-stutter-expanding properties of (C,D)-systems reduces to the verification of properties over leader actions.

Leader can keep a local copy of the register and simulate contributor actions: action-requests from contributors are acknowledged by leader, information flows through the register.
Conclusion

Changing from one to arbitrary many contributors turns the problem from undecidable to manageable.

(C,D)-systems of pushdown process have very good algorithmic properties

- Verification of C-stutter-expanding properties is decidable in NEXPTIME
- For some relevant subclasses it is PSPACE.

NEXPTIME-hardness shows that they can exhibit a quite nontrivial behavior.