Verification of parameterized shared-memory systems

Javier Esparza
Joint work with Pierre Ganty and Rupak Majumdar
Parameterized systems

• Systems consisting of an arbitrary number N of processes generated by instantiating a „program template“

• Verification problem: show that the system satisfies the desired properties whatever the value of N

• Equivalently: prove that each member of an infinite family of systems (one for each value of N) satisfies the properties

• Undecidable in general, even if each member is finite-state
Non-atomic networks

Leader

Store

Contributors

\[ r_c(g) \]
\[ w_c(g) \]

\[ r_c(g) \]
\[ w_c(g) \]

\[ r_c(g) \]
\[ w_c(g) \]

\[ r_c(g) \]
\[ w_c(g) \]
• Why shared memory?
  – Supported by different APIs

• Why a leader?
  – Base in vehicular and mobile phone systems.
  – Server in client-server applications.
  – Adds generality to the model.

• Why anonymous contributors?
  – Replicated programs (Kroening, Wahl, et al.)
  – Cache-coherence protocols
  – Systems where privacy matters
Why reads and writes?

- Previous work uses a transition system semantics
- Leader and contributors defined as transition systems
  - **States**: pairs \((q, x)\) where \(q\) local state and \(x\) valuation of global variables
  - **Transitions**: \((q, x) \rightarrow (q', x')\)
  - **Interleaving rule**: if \((q_i, x) \rightarrow (q'_i, x')\) then \((q_1, \ldots, q_i, \ldots, q_n, x) \rightarrow (q_1, \ldots, q'_i, \ldots, q_n, x')\)
Why reads and writes?

- Previous work uses a transition system semantics.
- Leader and contributors are modeled as transition systems.
  - **States**: \( (q, \mathbf{x}) \) with local state and \( \mathbf{x} \) valuation.
  - **Transitions**: \( (q, \mathbf{x}) \rightarrow (q', \mathbf{x}') \).
  - **Interleaving rule**: If \( q_i \rightarrow (q_i', \mathbf{x}') \) then \( q_1, \ldots, q_i, \ldots, q_n, \mathbf{x}' \rightarrow q_1, \ldots, q_i', \ldots, q_n, \mathbf{x}' \).
Too powerful because ...

- \((q, x) \rightarrow (q', x')\) is an atomic test-and-set
- Can be used to lock shared memory
- But: memory locks **dangerous** and very hard to implement for ad-hoc networks, vehicular systems ...
  - Contributors may enter and leave the system at any time, also inside a lock ...
- Model is too powerful \(\rightarrow\) complexity lower bounds for verification problems may be too pessimistic ...
And which are these bounds?

• Reachability in parametric family of finite state machines (FMS) is EXSPACE-hard
  – Reduction from coverability problem for Petri nets

• Reachability in parametric family of pushdown systems (PDS) is undecidable
  – Reduction from problem for two pushdowns
  – Use test-and-set to give identities to two pushdowns, which from that moment will be the only ones to proceed!
Could it be that by making the model more realistic we also reduce the complexity of verification?
Could it be that by making the model more realistic we also reduce the complexity of verification?

Our answer: yes

EXPSPACE-complete $\rightarrow$ NP-complete for FMSs
Undecidable $\rightarrow$ PSPACE-complete for PDSs
Verification problem (FSM version)

Given:

Decide: Is there a number $N$ such that some computation of \((\text{leader} + N \text{ contributors})\) reaches the final state?
\( N = 2 \)
$N = 2$

$\omega_d(1) \rightarrow r_d(2) \rightarrow r_d(1) \rightarrow r_d(3) \rightarrow 0$

$\omega_c(1) \rightarrow r_c(2) \rightarrow r_c(1) \rightarrow 0$

$\omega_c(3) \rightarrow r_c(2) \rightarrow r_c(1) \rightarrow 0$
\( w_d(1) \)

\( r_d(2) \)

\( r_d(1) \)

\( w_d(1) \)

\( r_d(3) \)
The traces $w_d(1) r_d(2) r_d(1) r_d(3)$ are compatible.

The traces

$w_d(1) r_d(2) r_d(1) r_d(3)$
$r_c(1) w_c(2) r_c(1) w_c(3)$
$r_c(1) w_c(1)$

are compatible.
Language-theoretic formulation of the verification problem

• Control reachability problem.

Given:

– a leader language $L_d$ (traces of leader from initial to final state)
– a contributor language $L_c$ (traces of contributor)

is there $w_d \in L_d$ and a multiset $W \subseteq L_c$ such that $\{w\} \oplus W$ are compatible?
Plan of attack

• Interesting lemmata:
  – Copycat, Simulation (Hague).
  – Monotonicity (new).

• Results for the state-machine case
• Results for the pushdown case
• Complexity of the bounded reachability problem
Copycat lemma

\[ w_d(1) r_d(2) r_d(1) r_d(3) \]
\[ r_c(1) w_c(2) r_c(1) w_c(3) \]
\[ r_c(1) w_c(1) \]

\[ \text{compatible} \]

\[ w_d(1) r_d(2) r_d(1) r_d(3) \]
\[ r_c(1) w_c(2) r_c(1) w_c(3) \]
\[ r_c(1) w_c(1) \]

\[ \text{compatible} \]

\[ w_d(1) r_c(1) r_c(1) r_c(1) r_c(1) \]
\[ w_c(2) r_d(2) w_c(1) \]
\[ r_d(1) r_c(1) w_c(3) \]
\[ r_d(3) \]

\[ \text{w}_c(2) w_c(2) r_d(2) w_c(1) \]
\[ r_d(1) r_c(1) r_c(1) w_c(3) w_c(3) \]
\[ r_d(3) \]
Copycat Lemma

• **Copycat Lemma:**

Let $u \in L_d$, let $W$ be a multiset of traces of $L_c$. If $\{u\} \oplus W$ is compatible and $w' \in W$, then $\{w\} \oplus W \oplus \{w'\}$ is compatible too.
Simulation Lemma

- For the leader, reads are "obstacles" on the path to the final state.
- The leader may need help from the contributors to "overcome" them.
- Consequence of the copycat lemma: 
  If the leader can get help to overcome an obstacle $r_d(g)$ for some value $g$, then it can get help to overcome all obstacles $r_d(g)$.
Simulation Lemma

- The arbitrarily many contributors can be replaced by a finite number of simulators, one for each possible value of the store.
- The role of the simulator for value $g$ is to execute a trace ending to $w_c(g)$, and then keep executing $w_c(g)$.
- Delicate technical points!
\( f_c(g) \) : first write by a contributor of the value \( g \)

\( u_c(g) \) : write of \( g \) by a contributor, not a first write, and immediately overwritten
Simulation Lemma

• **Simulation Lemma.** Let \( \{1, \ldots, k\} \) be the set of possible values of the store, and let \( S_i \) be the set of traces of the \( i \)-th simulator containing a first write \( f_c(i) \).

Let \( u \in L_d \).

\( u \) is compatible with some multiset of traces of \( L_c \)

\[
\text{iff}
\]

\( u \) is compatible with some set \( \{s_1, s_2, \ldots, s_k\} \), where \( s_i \in S_i \).

(Compatibility respects first and useless writes)
$w_d(1) r_d(2) r_d(1) r_d(3)$
$w_d(1) r_d(2) r_d(1) r_d(3)$
$w_d(1) r_d(2) r_d(1) r_d(3)$
$w_d(1) r_d(2) r_d(1) r_d(3)$
$w_d(1) r_d(2) r_d(1) r_d(3)$
$w_d(1) r_d(2) r_d(1) r_d(3)$
$w_c(1)$
$w_c(2)$
$w_c(3)$
$u_c(2)$
Monotonicity Lemma for Simulators

• By the Simulation Lemma our task reduces to finding $u \in L_d$ and $S = \{s_1, s_2, \ldots, s_k\}$, where $s_i \in S_i$, such that $\{u\} \cup S$ is compatible.

• **Monotonicity Lemma**: The search for $s_i$ can be safely limited to traces

  \[ \sigma_i f_c(i) w_c(i)^* \]

  where $\sigma_i$ is a **minimal** trace of $S_i$ w.r.t. the subword order.

  – Intuitively: these are the traces without superfluous reads or useless writes.
Monotonicity Lemma for Simulators

Simulator for $g = 1$

Minimal traces
Control reachability for FSMs

• **Theorem**: the control reachability problem when both leader and contributors are finite state machines is **NP-complete**

• Compare with EXSPACE-hard for the non-atomic case!

• **NP-hardness**: standard reduction from 3SAT

• **Membership in NP**: next slide.
Control reachability for FSMs

- **Membership in NP:**
  1) Guess a sequence $f_c(g_1) \ldots f_c(g_m)$ of first writes
  2) Guess minimal traces $\sigma_{g_1} f_c(g_1) \in S_{g_1}, \ldots, \sigma_{g_m} f_c(g_1) \in S_{g_m}$
     (guaranteed to have polynomial length because simulators are FSMs)
  3) Guess interleaving $\tau$ of $\sigma_{g_1} f_c(g_1) w_c(g_1)^*, \ldots, \sigma_{g_m} f_c(g_m) w_c(g_m)^*$
  4) Check in polynomial time that FSM of leader has a trace $u$ compatible with $\tau$. 
Control reachability for PDSs

• Previously studied by Hague (FSTTCS 11)
  – NP-hard lower bound
  – 2EXPTIME upper bound
• We show: the problem is PSPACE-complete
• Lower and upper bound are both non-trivial:
  – PSPACE-hardness : We describe a network that simulates a given polyspace Turing machine.
  – Membership in PSPACE: We give a polyspace algorithm for control reachability.
PSPACE-hardness

- Both leader and contributors can use their stacks to store a configuration of the Turing machine.
PSPACE-hardness

• Both leader and contributors can use their stacks to store a configuration of the Turing machine.

• If the leader and a contributor, say C, had the store for them alone, the solution would be easy:
  – Leader pops its configuration and sends it symbol by symbol to C, who stores it in its stack.
  – C pops the configuration, and while doing so computes its successor configuration, and sends it to Leader
PSPACE-hardness

- **Problem**: interference from other contributors. Leader doesn’t know if it is always talking to C!

- **Solution**: counting.
  - W.l.o.g. we assume that the computation of the TM has length $2^{\frac{2}{n}}$, where $n$ is the length of input.
  - Leader and contributors also use their stacks to count the number of times they receive a new configuration.
  - After counting to $2^{\frac{2}{n}}$, Leader waits for a message from some contributor that has also counted up to $2^{\frac{2}{n}}$. If the message arrives, then the contributor has received all configurations from Leader.
PSPACE-hardness

• **Problem**: interference from other contributors. Leader doesn’t know if it is always talking to C!

• **Solution**: counting.
  – W.l.o.g. we assume that the computation of the TM has length $2^n$, where $n$ length of input.
  – **Leader** and contributors also use their stacks to count the number of times they receive a new configuration.
  – After counting to $2^n$, **Leader** waits for a message from some contributor that has also counted up to $2^n$. If the message arrives, then the contributor has received all configurations from **Leader**.
Membership in PSPACE

• Very tricky, here only sketch! (and not completely correct ...)

• Simulation Lemma reduces the problem to reachability for a PDS-leader and a finite number of PDS-simulators.

• General proof strategy: nondeterministically guess a computation of the simulation network step by step, and apply NPSPACE=PSPACE
Membership in PSPACE

- **Problem**: a configuration of the network consists of
  - a configuration of the leader (control+stack content), plus
  - a configuration of each simulator (control+stack content)

In principle, both can be arbitrarily large! No guarantee that a configuration of the network can be stored in polynomial space.
Membership in PSPACE

• **Monotonicity Lemma** helps: it suffices to consider computations of simulators with „small stack“: stack size is at most the size of the simulator.

• So: configurations of simulators can be stored in polynomial space.

• But the configuration of the leader can not.

• Easy to find instances in which every computation of the network leading to the desired state requires leader stack of exponential size ...
Leader Monotonicity Lemma

- **Leader Monotonicity Lemma**: the search for computations of the leader can be restricted to any cover of $L_d$.
- A subset $C \subseteq L_d$ is a cover of $L_d$ if every word of $L_d$ is a subword of (is covered by) some word of $C$.
- But can we find some useful cover of $L_d$?
- $L_d$ is a context-free language (leader is a PDS)
- Let $G_d$ be a context-free grammar for $L_d$
Bounded-index approximations

• Let $G$ be a context-free grammar and $k \geq 1$. The $k$-index approximation of $L(G)$, denoted by $L^{(k)}(G)$ consists of the words of $L(G)$ with a derivation in which every intermediate word contains at most $k$ occurrences of non-terminals.

• Theorem: If $G$ has $m$ non-terminals, then $L^{(3m)}(G)$ is a cover of $L(G)$. 
Example

- \( X \to aXX \mid b \)
- \( b \) is the only word of index 1
- \( aabbb \) has index 2
  \[ X \to aXX \to aXb \to aaXXb \to aabXb \to aabbb \]
- \( aabbabb \) has index 3
  \[ X \to aXX \to aXaXX \to \cdots \to aXabb \to aaXXabb \to \cdots \to aabbabb \]
Guessing words of a cover

- Guessing a word of $L(G_d)$ may require arbitrary space. But guessing a word of $L^{(3m)}(G_d)$ requires only space $O(m)$
Guessing words of a cover

• Guessing a word of $L(G_d)$ may require arbitrary space. But guessing a word of $L^{(3m)}(G_d)$ requires only space $O(m)$

• However: the word is not guessed in the right order, so the guesses cannot be „synchronized“ with the guesses for the simulators!

• Dead end?
No …

- FSMs $A_1, \ldots, A_n$ for the simulators can be constructed in exponential space (stack depth is bounded).
- Their product with $G_d$ yields an exponential CFG grammar $G'_d$, which we should check for emptiness.
- By monotonicity, it suffices to check emptiness of $L^{(3m)}(G'_d)$.
- Theorem: this can be done in $\text{NSPACE}(m \log |G'_d|)$.
- So an expspace algorithm generates $|G'_d|$, and a logspace algorithm checks its emptiness.
- A generic theorem of complexity theory tells us that their combination can be replaced by a polyspace algorithm.
Bounded control reachability

- **Bounded control reachability problem.**
  Given:
  - a leader language $L_d$ (traces of leader from initial to final state)
  - a contributor language $L_c$ (traces of contributor)
  - a number $k$ in unary
  is there $w_d \in L_d$ and a multiset $\mathcal{W} \subseteq L_c$, with all traces of length at most $k$, such that $\{w\} \oplus \mathcal{W}$ are compatible?
Bounded control reachability

- **Theorem:** Bounded control reachability is **NP-complete**, even when the leader and the contributors are Turing machines.
Conclusions

• The complexity of verifying models without atomic test-and-set actions is much lower

• Very much to be taken into account when test-and-set actions are discouraged!
Thank you for your attention...