Verification of FIFO systems
and more specifically synchronizability

B. Bollig, C. Di Giusto, A. Finkel, L. Germerie, L. Laversa, É. Lozes and A. Suresh

I3S, Univ. Côte d’Azur and LMF, ENS Paris Saclay

june, 17th 2021
FIFO Queues are everywhere

```
func prod(ch chan int) {
for i := 0; i < 5; i++ {
    ch <- i // Send i to ch
}
}
cease(ch) // No further values accepted at ch
}
func cons(ch1, ch2 chan int) {
for {
    select {
    case x := <-ch1: print(x) // Either input from ch1
    case x := <-ch2: print(x) // or input from ch2
    }
}
}
func main() {
    ch1, ch2 := make(chan int), make(chan int)
go prod(ch1)
go prod(ch2)
go prod(ch2)
cons(ch1, ch1)
}
```
The Model-Checking Approach

Program Code over approximation FIFO System

Safety Property Model Checker

yes/no

ex: communication contracts, session types, ...
**FIFO Systems: Definition**

- a finite alphabet of messages
- a finite number of unbounded FIFO queues
- a finite parallel composition $P_1 || \ldots || P_n$
- a process is a goto program with two basic atomic actions: queuing and dequeuing

\[
P ::= q!a.P \quad (\text{enqueue } a \text{ in } q) \\
| q?a.P \quad (\text{dequeue } a \text{ from } q) \\
| P + P' \quad (\text{choice}) \\
| X \quad (\text{goto}) \\
| \text{rec } X.P \quad (\text{label})
\]
Network Topology

- p2p: one queue per pair of processes
- mailbox: one queue per process
- binary system: p2p/mailbox with 2 processes

- general case: a process can queue/dequeue in all queues
Example: a P2P System

Run

τ =
Example: a P2P System

Run

\[ \tau = !a \]
Example: a P2P System

\[ \text{Run} \]

\[ \tau = !a \]
Example: a P2P System

Run
\( \tau = !a?a \)
Example: a P2P System

Run

$\tau = !a?a$
Example: a P2P System

Run
\[ \tau = !a?a!d \]
Example: a P2P System

Run

\[ \tau = !a?a!d \]
Example: a P2P System

Run

\[ \tau = !a?a!d!b \]
Example: a P2P System

Run
\[ \tau = !a?a!d!b \]
Example: a P2P System

Run
\[ \tau = \texttt{!a?a!d!b!b} \]
Example: a P2P System

Run

$$\tau = !a?a!d!b!b$$
Example: a P2P System

Run
\[ \tau = !a ?a !d !b !b \]

?b → 3

?d → 3

P1

P2

P3
Example: a P2P System

Run

\[ \tau = !a?a!d!b!b \]

?b
Example: a P2P System

Run

\[ \tau = !a?a!d!b!b \]

?b?d

\[ \tau = !a?a!d!b!b \]

?b?d

P3
Example: a P2P System

Run

\[ \tau = !a ? a !d !b !b \]

\[ ?b \to b \]

\[ ?d \to d \]
Example: a P2P System

Run

\[ \tau = !a ? a ! d ! b ! b \]
\[ ? b ? d ! c \]
Example: a P2P System

Run

\( \tau = !a?a!d!b!b \)

\( ?b?d!c \)
Example: a P2P System

$$\begin{align*}
\text{Run} & \\
\tau &= !a?a!d!b!b
\end{align*}$$

Note: The diagram illustrates the communication between processes P1, P2, and P3, with labeled transitions and conditions.
Example: a P2P System

Run

\[ \tau = \text{!a?a!d!b!b} \]
\[ \text{?b?d!c?c} \]
Example: a P2P System

Run

\[ \tau = !a ?a !d !b !b \]
\[ ?b ?d !c ?c ?b \]
Verification Problems

- is there a bound on the size of the queues (for all runs)?
- is there a run where a message is sent but never received?
- is there a run where a machine receives an unexpected message?
- is there a reachable configuration where all machines wait for messages but the queues are empty?
- ...

All these questions (and many others) are undecidable
[Brand Zafiropulo, JACM 1983]
Unary FIFO system and Tiling

Set of tiles

Solution

The process $P$ guesses the solution. First it guesses the first row and queues it, then it guesses the tile above the first row and queues it. The process then starts again with the next row.
FIFO system

The process P guesses the solution
Unary FIFO system and Tiling

Set of tiles

Solution

FIFO system

Guess first row and queue it

The process P guesses the solution
Unary FIFO system and Tiling

Set of tiles

Solution

FIFO system

Guess tile above

The process P guesses the solution
Unary FIFO system and Tiling

Set of tiles

Solution

FIFO system

Guess tile above

The process P guesses the solution

and queue it
Unary FIFO system and Tiling

Set of tiles

Solution

FIFO system

Guess tile above

The process P guesses the solution
Unary FIFO system and Tiling

Set of tiles

Solution

FIFO system

Guess tile above

The process P guesses the solution and queue it
Unary FIFO system and Tiling

Set of tiles

Solution

FIFO system

Guess tile above

The process P guesses the solution
Unary FIFO system and Tiling

**Set of tiles**

Guess first row and queue it

Guess tile above and queue it

Start again with the next row

**FIFO system**

**Guess tile above**

The process P guesses the solution

and queue it
Unary FIFO system and Tiling

**Set of tiles**

- Square tiles in red and blue colors.

**Solution**

- An arrangement of square tiles forming a pattern.

**FIFO system**

- Process P guesses the solution.
  - First row: Guess the first tile and queue it.
  - Next row: Guess the tile above the previously guessed and queue it.
  - Start again with the next row.

- The process P guesses the solution.
Unary FIFO system and Tiling

Set of tiles

Solution

FIFO system

Guess tile above

The process P guesses the solution and queue it
Unary FIFO system and Tiling

The process P guesses the solution first row and queue it. Then queue the tile above and start again with the next row. The process P guesses the solution.
Unary FIFO system and Tiling

Set of tiles

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Solution

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

FIFO system

The process P guesses the solution
Unary FIFO system and Tiling

**FIFO system**

The process P guesses the solution.

**Set of tiles**

Guess first row and queue it

Guess tile above and queue it

Start again with the next row

**Solution**
Unary FIFO system and Tiling

The process $P$ guesses the solution:

1. Guess first row and queue it.
2. Guess tile above and queue it.
3. Start again with the next row.
Unary FIFO system and Tiling

Set of tiles

Solution

FIFO system

The process P guesses the solution
Unary FIFO system and Tiling

Set of tiles

Solution

FIFO system

The process P guesses the solution
Unary FIFO system and Tiling

FIFO system

The process P guesses the solution
Unary FIFO system and Tiling

The process $P$ guesses the solution step by step:

1. **Guess first row** and queue it.
2. **Guess tile above** and queue it.
3. **Start again with the next row**.

---

**FIFO system**

The process $P$ guesses the solution.
Unary FIFO system and Tiling

**Set of tiles**

**Solution**

**FIFO system**

The process P guesses the solution
How does a model-checker work?
Solution 1: Second Over-Approximation

Program Code \(\xrightarrow{\text{over approximation}}\) FIFO System \(\xrightarrow{\text{over approximation}}\) Non-FIFO Semantics

Safety Property \(\xrightarrow{}\) Model Checker

yes/no

ex: Out-of-order (bag) semantics, Lossy semantics.
HIGH COMPLEXITY (non primitive recursive, non Ackermanian)
Solution 2: Under-Approximation

The Completeness Problem: for a given system, is the under-approximation actually exact?
Half-Duplex Semantics

Introduced by Cece and Finkel, CAV’97

**H.D. Semantics**: block sending until no more incoming messages

Safety Model-Checking in PTIME
Completeness Problem in PTIME

Note: Cece and Finkel only considered binary systems
Can be extended to any mailbox system [Germerie, Di Giusto and L., ICE’21].
Bounded Context-Switch Semantics

Introduced by Madhusudan, La Torre and Parlato, TACAS’08

Bounded Context Switch Semantics

$\textit{switch at most } k \textit{ times from one process to another}$

Safety Model-Checking in 2EXPTIME
Completeness Problem not really addressed
Synchronous Semantics

a run is synchronous if it is of the form

\[ !a_1?a_1!a_2?a_2 \cdots !a_n?a_n \]

Synchronous Semantics: consider only synchronous runs.

Completeness could be

\text{every run (that ends with empty buffers) is synchronous.}

Decidable, but very limited!
e.g. for 2 processes, means half-duplex + ”ping-pong”. 
Stating Completeness Differently

\[ P_1 = 1!a \quad P_2 = 1?a.2?b.1?c \quad P_3 = 2!b.1!c \]

\( P_1\|P_2\|P_3 \) admits the non-synchronous run

\[ 1!a \cdot 2!b \cdot 1?a \cdot 2?b \cdot 1!c \cdot 1?c \]

BUT it also admits this \textit{equivalent}, synchronous one

\[ 1!a \cdot 1?a \cdot 2!b \cdot 2?b \cdot 1!c \cdot 1?c \]

The synchronous semantics could be considered complete for \( P_1\|P_2\|P_3 \).
The *Happens Before* Partial Order

consider a run of the form \( \cdots \alpha \cdots \beta \cdots \)
\( \alpha \) happens before \( \beta \) if

either \( \text{process}(\alpha) = \text{process}(\beta) \)

or \( \beta \) is the reception matching the send \( \alpha \)

or \( \text{buffer}(\alpha) = \text{buffer}(\beta) \)
and \( \text{kind}(\alpha) = \text{kind}(\beta) \) (\( \in \{!, ?, \} \))

or a chain of such elementary steps
Traces (and Message Sequence Charts, MSC)

\( 1!a \cdot 2!b \cdot 1?a \cdot 2?b \cdot 1!c \cdot 1?c \)
Back to the Completeness of the Synchronous Semantics

We say that

▶ a trace is $\exists$-synchronous if it admits a linearisation (i.e. a run) that is synchronous

▶ the synchronous semantics is complete for a system $P_1 \parallel \cdots \parallel P_n$ if all traces are $\exists$-synchronous

∼ what we called greedy systems in ICE’21 (but we have a different treatment of orphan messages)

The completeness problem is decidable in PTIME (for a fixed number of processes).

Regular safety checking is decidable in PTIME

Remarks:

▶ the session type discipline usually enforces greediness

▶ generalises half-duplex systems
Limitations

Many protocols are not $∃$-synchronous, for instance

Actually, $∃$-synchronous systems are a very light generalisation of half-duplex systems (see more in ICE’21)

How can we generalize this idea more significantly?
A run $r$ is $k$-bounded ($k \geq 1$) if for all prefix $r'$ and for all buffer $i$, $|r'|_{i!} - |r'|_{i?} \leq k$.

A trace is $\exists$-k-bounded if it admits a $k$-bounded linearisation.

(actually more general, we should only count messages that are eventually received)

Regular Safety Checking is in PSPACE. The Completeness Problem (whether a system is $\exists$-k-bounded) is decidable in PSPACE.

Remark: Genest et al only considered p2p systems.
Communication-Closed Protocols

Cezara Drăgoi et al. POPL’16 - CAV’19

messages are timestamped with a round number
messages from older rounds are ignored
A $k$-exchange is a MSC that admits a linearisation of the form

$$!a_1 \cdots !a_l \cdots ?b_1 \cdots ?b_m$$

with $l, m \leq k$.

\[\text{a 3-exchange}\]

\[\text{not a } k\text{-exchange (for any } k)\]
**$k$-synchronous MSCs**

Bouajjani, Enea, Ji and Qadeer, CAV’18

A MSC is *$k$-synchronous* if it is a concatenation of $k$-exchanges.

A 2-synchronous MSC

\[
\begin{array}{ccc}
p & q & r \\
\end{array}
\]

\[
\begin{array}{ccc}
a_0 & a_1 \\
ap & a_2 \\
a_3 & a_4 \\
\end{array}
\]

(a) not $k$-synchronous (for any $k$)
A system is \( k \)-synchronous if all its MCSs (traces?) are \( k \)-synchronous.

- Reachability is in PSPACE under the \( k \)-synchronous semantics.
- The completeness problem (whether a system is \( k \)-synchronous) is in PSPACE.

Proofs later cleaned and extended to p2p systems. See [Laversa, Di Giusto, L. FOSSACS’20].
The Need for a General Framework

∃-k-bounded, k-synchronous, are similar, but

- ∃-k-bounded only considered for p2p systems
  *what about mailbox systems?*

- the proofs are technical and hide similarities
  *what is the key idea?*

- both need to fix a k
  *can we guess k? what about ∞-synchronous?*

Known results:

- guessing k for ∃-k-bounded/p2p is undecidable
  [Genest, Kuske and Muscholl, Fundam. Inform. 2007]

- guessing k for k-synchronous/mailbox is in PSPACE
  [Di Giusto, Laversa, L., CIAA’21]
General Framework: our Proposal

joint work with Bollig, Finkel and Suresh. Submitted.

- a restricted FIFO semantics is a set of MSCs in other words, a set of graphs
- let’s define it with a formula $\phi$ of a logic for graphs: MSO
- let’s add another hypothesis: all MSCs that satisfy $\phi$ have tree-width at most $k$ (for some fixed $k$)

- main result: safety checking and the completeness problems are in EXPTIME. The proof is half a page long.

- side contribution: we identified the $\infty$-synchronous semantics. Safety checking and completeness are
  - EXPTIME (PSPACE?) for mailbox systems
  - undecidable for p2p systems
Thank you!