Parameterized Communicating Automata
Complementation and Model Checking

Benedikt Bollig, Paul Gastin, and Akshay Kumar

Laboratoire Spécification et Vérification
ENS Cachan & CNRS, France

Indian Institute of Technology Kanpur, India

FSTTCS 2014
India International Centre, New Delhi
December 15–17, 2014
Focus of previous work has been on verification:

«Is a system correct independently of the number of processes / the way they are arranged?»

Focus of previous work has been on verification:

«Is a system correct independently of the number of processes / the way they are arranged?»


In this talk, we study language-theoretic questions / expressiveness:

- Graph-based semantics (cf. WYSIWYG-Lecture)
- Complementation
- Equivalent characterization in terms of MSO logic

We are looking for a «robust» model of parameterized systems.
Focus of previous work has been on verification:

«Is a system correct independently of the number of processes / the way they are arranged?»


In this talk, we study language-theoretic questions / expressiveness:

- Graph-based semantics (cf. WYSIWYG-Lecture)
- Complementation
- Equivalent characterization in terms of MSO logic

We are looking for a «robust» model of parameterized systems.

There have been robust models for fixed process architectures:

Finite Automata

finite automaton

\[ a \rightarrow s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow s_4 \]
\[ a \rightarrow s_0 \rightarrow s_2 \rightarrow s_3 \rightarrow s_6 \]
\[ b \rightarrow s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow s_5 \]
Finite Automata

finite automaton

determinization

![Finite Automaton Diagram](image)

![Determinization Diagram](image)
Finite Automata

finite automaton

determinization

complementation
Finite Automata

Theorem [Büchi-Elgot-Trakhtenbrot 1960s]:
Finite Automata = MSO

∀x(a(x) → ∃y(succ(x, y) ∧ b(y)))
Theorem [Büchi-Elgot-Trakhtenbrot 1960s]:
Finite Automata = MSO

\[ \forall x (a(x) \rightarrow \exists y (\text{succ}(x, y) \land b(y))) \]
Parameterized Communicating Automata (PCA)
Parameterized Communicating Automata (PCA)
Parameterized Communicating Automata (PCA)
Parameterized Communicating Automata (PCA)
Parameterized Communicating Automata (PCA)

non-fixed & unbounded
Parameterized Communicating Automata (PCA) over Rings

non-fixed & unbounded

Diagram showing labeled transitions between states: $s_0, s_1, s_2, s_3, s_4, s_5, s_6$ with labels $a$ and $b$. Connections are marked with $l$ and $r$.
A PCA is given by:

- finite automaton over \( \{l, r\} \times \{!, ?\} \times Msg \) (here: \( Msg = \{0, 1\} \))
- acceptance condition
A PCA is given by:

- finite automaton over \( \{l, r\} \times \{!, ?\} \times \text{Msg} \) (here: \( \text{Msg} = \{0, 1\} \))
- acceptance condition

Parameterized Communicating Automata (PCA) over Rings
A PCA is given by:

- finite automaton over \( \{l, r\} \times \{!, ?\} \times \text{Msg} \)  
  (here: \( \text{Msg} = \{0, 1\} \))
- acceptance condition
Parameterized Communicating Automata (PCA) over Rings

A PCA is given by:
- finite automaton over \( \{l, r\} \times \{!, ?\} \times \text{Msg} \)  
  (here: \( \text{Msg} = \{0, 1\} \))
- acceptance condition
A PCA is given by:

- finite automaton over \( \{l, r\} \times \{!, ?\} \times \text{Msg} \)  \hspace{1cm} (here: \( \text{Msg} = \{0, 1\} \))
- acceptance condition
Parameterized Communicating Automata (PCA) over Rings
Remark:
Behavior abstracts away message contents from $Msg = \{0, 1\}$
(like states, or stack symbols in pushdown automata).
Parameterized Communicating Automata (PCA) over Rings
Parameterized Communicating Automata (PCA) over Rings
Parameterized Communicating Automata (PCA) over Rings
Parameterized Communicating Automata (PCA) over Rings
Parameterized Communicating Automata (PCA) over Rings
Parameterized Communicating Automata (PCA) over Rings
Parameterized Communicating Automata (PCA) over Rings
Parameterized Communicating Automata (PCA) over Rings
Parameterized Communicating Automata (PCA) over Rings
Parameterized Communicating Automata (PCA) over Rings
Acceptance condition:
MSO formula over rings whose nodes are labeled with states.
Signature: $s(x) \ x \ r \ l \ y$
Thus, there are no constant processes (e.g., no «first» or «last» process).
Acceptance condition:
MSO formula over rings whose nodes are labeled with states.
Signature: \( s(x) \xRightarrow{r} y \)

Thus, there are no constant processes (e.g., no "first" or "last" process).
Parameterized Communicating Automata (PCA) over Rings

\[ \exists x (s_4(x) \land \forall y (y \neq x \rightarrow s_5(y) \lor s_6(y))) \]
Parameterized Communicating Automata (PCA) over Rings

\[ \exists x(s_4(x)) \land \forall y(y \neq x \rightarrow s_5(y) \lor s_6(y)) \]
Parameterized Communicating Automata (PCA) over Rings

\[ \exists x(s_4(x) \land \forall y(y \neq x \rightarrow s_5(y) \lor s_6(y))) \]
Parameterized Communicating Automata (PCA) over Rings

\[ \exists x(s_4(x) \land \forall y(y \neq x \rightarrow s_5(y) \lor s_6(y))) \]

Perspectives: Dynamic Message-Passing Systems
Parameterized Communicating Automata (PCA) over Rings

\[ \exists x(s_z(x) \land \forall y(y \neq x \rightarrow s_5(y) \lor s_6(y))) \]
Parameterized Communicating Automata (PCA) over Rings

$L = \exists x(s_4(x) \land \forall y(y \neq x \rightarrow s_5(y) \lor s_6(y)))$
Parameterized Communicating Automata (PCA) over Rings

$L$ = \[
\begin{align*}
\exists x(s_4(x) \land \forall y(y \neq x \rightarrow s_5(y) \lor s_6(y)))
\end{align*}
\]
Parameterized Communicating Automata (PCA) over Rings

\[
L = \exists x(s_4(x) \land \forall y(y \neq x \rightarrow s_5(y) \lor s_6(y)))
\]
Parameterized Communicating Automata (PCA) over Rings

$L = \exists x(s_4(x) \land \forall y(y \neq x \rightarrow s_5(y) \lor s_6(y)))$
Complementation

\[ L = \exists x(s_4(x) \land \forall y \neg x \rightarrow s_5(y) \lor s_6(y)) \]
Complementation

\[
\begin{align*}
L & \quad = \\
\exists x(s_1(x) \land \forall y(y \neq x \rightarrow s_5(y) \lor s_6(y)))
\end{align*}
\]
Complementation

\[
L = \{ \text{graph 1}, \text{graph 2} \}
\]

\[
\exists x (s_4(x) \land \forall y (y \neq x \rightarrow s_5(y) \lor s_6(y)))
\]
Complementation

\[
L = \{ \end{array} 
\[
\text{\(\exists x(s_4(x) \land \forall y(y \neq x \rightarrow s_5(y) \lor s_6(y)))\)}
\]
Complementation

\[
L = \{ s_0 \rightarrow r!1 \leftarrow l?1 \rightarrow l?0 \rightarrow r!0 \rightarrow r!0 \rightarrow l?0 \rightarrow r!1 \rightarrow s_0 \mid \exists x(s_4(x) \land \forall y(y \neq x \rightarrow s_5(y) \lor s_6(y))) \}
\]
Complementation

L

\[ \exists x(s_4(x) \land \forall y(y \neq x \rightarrow s_5(y) \lor s_6(y))) \]

=  

[Diagram]
Negative Results

**Theorem:**
PCAs over rings are not complementable.
Theorem:
PCAs over rings are not complementable.

Proof:
Theorem:
PCAs over rings are not complementable.

Proof:
Behaviors encode grids.
**Theorem:**
PCAs over rings are not complementable.

**Proof:**
- Behaviors encode grids.
- Grid automata are not closed under complementation
  
  [Matz-Schweikardt-Thomas ’02].

---

[Image of a diagram]
Negative Results

Theorem:
PCAs over rings are not complementable.

Proof:
Behaviors encode grids.
Grid automata are not closed under complementation
[Matz-Schweikardt-Thomas ’02].

Theorem [Emerson-Namjoshi 2003]:
Emptiness is undecidable for PCAs over rings
(even token-passing systems, unless $|Msg| = 1$).
Negative Results

**Theorem:**
PCAs over rings are not complementable.

**Proof:**

- Behaviors encode grids.
- Grid automata are not closed under complementation [Matz-Schweikardt-Thomas ’02].

Context-Bounded Model Checking of Concurrent Software

Shaz Qadeer and Jakob Rehof
Context-bounded PCAs
Context-bounded PCAs

**Idea:** Every process is constrained to a bounded number of contexts.
Context-bounded PCAs

Idea: Every process is constrained to a bounded number of contexts. There are several possible definitions of a context.
Idea: Every process is constrained to a bounded number of contexts. There are several possible definitions of a context.

Here: Process only sends XOR only receives from one fixed neighbor.
Context-bounded PCAs

**Idea:** Every process is constrained to a bounded number of contexts. There are several possible definitions of a context.

**Here:** Process only sends XOR only receives from one fixed neighbor.
Context-bounded PCAs

Idea: Every process is constrained to a bounded number of contexts. There are several possible definitions of a context. 

Here: Process only sends XOR only receives from one fixed neighbor.
Context-bounded PCAs

**Idea:** Every process is constrained to a bounded number of contexts. There are several possible definitions of a context.

**Here:** Process only sends XOR only receives from one fixed neighbor.
Context-bounded PCAs

**Idea:** Every process is constrained to a bounded number of contexts. There are several possible definitions of a context.

**Here:** Process only sends XOR only receives from one fixed neighbor.
Context-bounded PCAs

**Idea:** Every process is constrained to a bounded number of contexts. There are several possible definitions of a context.

**Here:** Process only sends XOR only receives from one fixed neighbor.
Context-bounded PCAs

Idea: Every process is constrained to a bounded number of contexts. There are several possible definitions of a context.

Here: Process only sends XOR only receives from one fixed neighbor.
Context-bounded PCAs

Idea: Every process is constrained to a bounded number of contexts. There are several possible definitions of a context.

Here: Process only sends XOR only receives from one fixed neighbor.
Context-bounded PCAs

Idea: Every process is constrained to a bounded number of contexts. There are several possible definitions of a context.

Here: Process only sends XOR only receives from one fixed neighbor.
Context-bounded PCAs

**Idea:** Every process is constrained to a bounded number of contexts. There are several possible definitions of a context.

**Here:** Process only sends XOR only receives from one fixed neighbor.
Definition: A PCA is $k$-bounded if every accepted behavior is $k$-bounded (can be syntactically enforced).
Definition: A PCA is $k$-bounded if every accepted behavior is $k$-bounded (can be syntactically enforced).
**Definition:** A PCA is *k*-bounded if every accepted behavior is *k*-bounded (can be syntactically enforced).

**Theorem** (Context-bounded PCAs are complementable):
For every bounded PCA $\mathcal{A}$, there is a PCA $\mathcal{B}$ such that $L(\mathcal{B}) = \overline{L(\mathcal{A})}$. 
Proof Outline

nondeterminism

![Diagram of nondeterminism](image)

- Every behavior has a unique run

disambiguation

- Every behavior has a unique run

complementation
Proof Outline

nondeterminism

Every behavior has a unique run

complementation

2-bounded
Proof Outline

nondeterminism

\[ \exists x(s_4(x) \land \forall y(y \neq x \rightarrow s_5(y) \lor s_6(y))) \]

2-bounded

disambiguation
every behavior has a unique run

\[ A \]

complementation

\[ \neg A \]
Proof Outline

nondeterminism

\[ s_0 \]

\( r!1 \quad l?1 \quad l?0 \)

\[ s_1 \quad s_2 \quad s_3 \]

\( l?0 \quad r!1 \quad r!0 \quad r!0 \)

\[ s_4 \quad s_5 \quad s_6 \]

\[ \exists x(s_4(x) \land \forall y(y \neq x \rightarrow s_5(y) \lor s_6(y))) \]

disambiguation

every behavior has a unique run

\[ \neg \varphi \]

complementation

Powerset construction not applicable due to message contents.
Disambiguation of context-bounded PCAs
Disambiguation of context-bounded PCAs
Disambiguation of context-bounded PCAs
Disambiguation of context-bounded PCAs
Disambiguation of context-bounded PCAs

Every process traverses a bounded number of zones.
Every process traverses a bounded number of zones.
Zone numbers can be computed unambiguously by a PCA.
Disambiguation of context-bounded PCAs

- Every process traverses a bounded number of zones.
- Zone numbers can be computed unambiguously by a PCA.
Disambiguation of context-bounded PCAs

- Every process traverses a bounded number of zones.
- Zone numbers can be computed unambiguously by a PCA.
Disambiguation of context-bounded PCAs

- Every process traverses a bounded number of zones.
- Zone numbers can be computed unambiguously by a PCA.
Disambiguation of context-bounded PCAs

- Every process traverses a bounded number of zones.
- Zone numbers can be computed unambiguously by a PCA.
Disambiguation of context-bounded PCAs

- Every process traverses a bounded number of zones.
- Zone numbers can be computed unambiguously by a PCA.
Disambiguation of context-bounded PCAs

\[ R_i \subseteq S^3 \times S^3 \]
Disambiguation of context-bounded PCAs

- Every process traverses a bounded number of zones.
- Zone numbers can be computed deterministically.
- Sending processes compute summaries for zones.
Disambiguation of context-bounded PCAs

- Every process traverses a bounded number of **zones**.
- Zone numbers can be computed deterministically.
- Sending processes compute summaries for zones.
Disambiguation of context-bounded PCAs

Every process traverses a bounded number of zones.

Zone numbers can be computed deterministically.

Sending processes compute summaries for zones.

Acceptance condition checks if summaries correspond to accepting run.
Disambiguation of context-bounded PCAs

- Every process traverses a bounded number of zones.
- Zone numbers can be computed deterministically.
- Sending processes compute summaries for zones.
- Acceptance condition checks if summaries correspond to accepting run.
Every process traverses a bounded number of zones. Zone numbers can be computed deterministically. Sending processes compute summaries for zones. Acceptance condition checks if summaries correspond to accepting run.
Every process traverses a bounded number of zones. Zone numbers can be computed deterministically. Sending processes compute summaries for zones. Acceptance condition checks if summaries correspond to accepting run.
Disambiguation of context-bounded PCAs

Every process traverses a bounded number of zones.
Zone numbers can be computed deterministically.
Sending processes compute summaries for zones.
Acceptance condition checks if summaries correspond to accepting run.
The Logic:
MSO logic over graphs, including process nodes and event nodes.
Logical Characterization of PCAs

The Logic:
MSO logic over graphs, including process nodes and event nodes.
The Logic:
MSO logic over graphs, including process nodes and event nodes.

Corollary: For every bounded set $L$ of behaviors, the following are equivalent:
- $L$ is recognized by some PCA.
- $L$ is definable in MSO logic.
Concluding Remarks

**Complementability and MSO characterization** hold for all topology classes of bounded degree (over fixed set of directions).
Concluding Remarks

**Complementability and MSO characterization** hold for all topology classes of bounded degree (over fixed set of directions).

**Theorem** [B.-Gastin-Schubert 2014]:
Context-bounded emptiness checking is decidable over rings, pipelines, and trees.
Concluding Remarks

**Complementability and MSO characterization** hold for all topology classes of bounded degree (over fixed set of directions).

**Theorem** [B.-Gastin-Schubert 2014]:
Context-bounded emptiness checking is decidable over rings, pipelines, and trees.

**Corollary:**
Context-bounded MSO model checking is decidable over rings, pipelines, and trees.
Concluding Remarks

**Complementability and MSO characterization** hold for all topology classes of bounded degree (over fixed set of directions).

**Theorem [B.-Gastin-Schubert 2014]:**
Context-bounded emptiness checking is decidable over rings, pipelines, and trees.

**Corollary:**
Context-bounded MSO model checking is decidable over rings, pipelines, and trees.

Future Work

- Topologies of unbounded degree (unranked trees, stars, …)
- Include data in messages (e.g., pids)