Bounded context switching for valence systems

Roland Meyer\textsuperscript{1}, Sebastian Muskalla\textsuperscript{1}, and Georg Zetzsche\textsuperscript{2}

September 4, CONCUR 2018, Beijing

\textsuperscript{1} TU Braunschweig, Germany
\{roland.meyer, s.muskalla\}@tu-bs.de

\textsuperscript{2} IRIF (Université Paris-Diderot, CNRS), France
zetzsche@irif.fr
Result and structure

Theorem
Reachability under *bounded context switching*
for *valence systems over graph monoids*
is *always in NP* (for all graph monoids).
Theorem

Reachability under *bounded context switching*

for *valence systems over graph monoids*

is *always in NP* (for all graph monoids).

1. What is *bounded context switching (BCS)*?
### Theorem

Reachability under **bounded context switching**  
for **valence systems over graph monoids**  
is always in NP (for all graph monoids).

1. What is **bounded context switching (BCS)**?  
2. What are **valence systems over graph monoids**?
Theorem

Reachability under \textit{bounded context switching}

for \textit{valence systems over graph monoids}

is \textit{always in NP} (for all graph monoids).

1. What is \textit{bounded context switching (BCS)}?

2. What are \textit{valence systems over graph monoids}?

3. What is \textit{BCS for valence systems}?
1. BCS
Setting:
Concurrent system, each component modeled as automaton
The problem

Setting:
Concurrent system, each component modeled as automaton

Problem:
If components are beyond finite-state, reachability (safety verification) is difficult
The problem

Setting:
Concurrent system, each component modeled as automaton

Problem:
If components are beyond finite-state, reachability (safety verification) is difficult

Solution:
Consider bounded context switching (BCS)
Context: Infix of the (sequentialized) computation where a single thread is active
**Context**: Infix of the (sequentialized) computation where a single thread is active

**BCS**: Number of contexts switches ($\#\text{contexts} - 1$) bounded by a constant
Bounded context switching

**Context**: Infix of the (sequentialized) computation where a single thread is active

**BCS**: Number of contexts switches (#contexts $- 1$) bounded by a constant

---

**Reachability under bounded context switching** (BCSREACH)

**Given**: Concurrent system $S$, number $k$ (in unary)

**Decide**: Final configuration reachable from initial one in $S$ by a computation with $\leq k$ context switches?
Reachability under bounded context switching (BCSREACH)

Given: Concurrent system \( S \), number \( k \) (in unary)

Decide: Final configuration reachable from initial one in \( S \) by a computation with \( \leq k \) context switches?
Reachability under bounded context switching (BCSREACH)

Given: Concurrent system $S$, number $k$ (in unary)
Decide: Final configuration reachable from initial one in $S$
by a computation with $\leq k$ context switches?

Under-approximation of reachability
### Reachability under bounded context switching (BCSREACH)

<table>
<thead>
<tr>
<th><strong>Given:</strong></th>
<th>Concurrent system $S$, number $k$ (in unary)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Decide:</strong></td>
<td>Final configuration reachable from initial one in $S$ by a computation with $\leq k$ context switches?</td>
</tr>
</tbody>
</table>

Under-approximation of reachability

Complexity is typically much lower
Reachability under bounded context switching (BCSREACH)

Given: Concurrent system $S$, number $k$ (in unary)

Decide: Final configuration reachable from initial one in $S$ by a computation with $\leq k$ context switches?

Under-approximation of reachability

Complexity is typically much lower

Useful as bugs usually occur within few context switches [MQ07,LPSZ08]
Example [QR05]:

Concurrent system where each component is a PDS, communicating via finite control
Example [QR05]:

Concurrent system where each component is a PDS, communicating via finite control

\[\downarrow\] essentially a MPDS
Example [QR05]:

Concurrent system where each component is a PDS, communicating via finite control

↕ essentially a MPDS

Reachability is *undecidable* if \#components \( \geq 2 \)
Example [QR05]:

Concurrent system where each component is a PDS, communicating via finite control

\[ \Downarrow \] essentially a MPDS

Reachability is undecidable if \#components \( \geq 2 \)

Context: Infix in which only one stack is used
Example [QR05]:

Concurrent system where each component is a PDS, communicating via finite control

\[ \Rightarrow \text{essentially a MPDS} \]

Reachability is undecidable if \( \# \text{components} \geq 2 \)

**Context:** Infix in which only one stack is used

Reachability under BCS is NP-complete
Related work

Similar results for

- various types of components,
- various types of communication,
- various BCS-like restrictions.
Related work

Similar results for

- various types of components,
- various types of communication,
- various BCS-like restrictions.

For example:

- Queues as storages [LMP08]
- Pushdowns with dynamic thread creation [ABQ09]
- Pushdowns communicating via queues [HLMS12]

...
Our goal: General BCS result
Related work II

**Our goal:** General BCS result

**Other people’s work:**

- Using graph-theoretic measures (tree width, ...)
  - Can handle queues
  - Cannot handle counters
  - Applies to settings where the complexity is beyond NP
- Reductions to PA-satisfiability
  - Can handle reversal-bounded counters
  - Does not allow nested combination of counters and stack
- Results incomparable to ours
- Our technique provides an algebraic view
Our goal: General BCS result

Other people’s work:

Using graph-theoretic measures (tree width, ...) [MP11, A14]
  • Can handle queues
  • Cannot handle counters
  • Applies to settings where the complexity is beyond NP
Related work II

Our goal: General BCS result

Other people’s work:

Using graph-theoretic measures (tree width, ...) [MP11, A14]
- Can handle queues
- Cannot handle counters
- Applies to settings where the complexity is beyond NP

Reductions to $\exists PA$-satisfiability [HL12, EGT14]
- Can handle reversal-bounded counters
- Does not allow nested combination of counters and stack
Our goal: General BCS result

Other people’s work:

Using graph-theoretic measures (tree width, ...) [MP11, A14]
  • Can handle queues
  • Cannot handle counters
  • Applies to settings where the complexity is beyond NP

Reductions to $\exists P A$-satisfiability [HL12, EGT14]
  • Can handle reversal-bounded counters
  • Does not allow nested combination of counters and stack

Results incomparable to ours
Our goal: General BCS result

Other people’s work:

Using graph-theoretic measures (tree width, ...) \([\text{MP11, A14}]\)

- Can handle queues
- Cannot handle counters
- Applies to settings where the complexity is beyond NP

Reductions to \(\exists \Pi_2 \text{-satisfiability} \) \([\text{HL12, EGT14}]\)

- Can handle reversal-bounded counters
- Does not allow nested combination of counters and stack

Results incomparable to ours

Our technique provides an algebraic view
2. Valence systems
Valence systems

Need a single model that can represent various types of memory
Valence systems

Need a single model that can represent various types of memory

Introducing valence systems over some monoid $\mathbb{M}$
Valence systems

Need a single model that can represent various types of memory

Introducing **valence systems** over some monoid $\mathbb{M}$

Monoid $\mathbb{M}$ represents the **storage** of the system
Valence systems

Need a single model that can represent various types of memory

Introducing valence systems over some monoid $\mathbb{M}$

Monoid $\mathbb{M}$ represents the storage of the system

Syntax:
- Finite control
- Transitions labeled by generators of $\mathbb{M}$
Valence systems

Need a single model that can represent various types of memory

Introducing valence systems over some monoid \( \mathbb{M} \)

Monoid \( \mathbb{M} \) represents the storage of the system

Syntax:
- Finite control
- Transitions labeled by generators of \( \mathbb{M} \)

Semantics:
- Configurations \((q, m)\) with \(q\) control state, \(m \in \mathbb{M}\)
- Transition \(q \xrightarrow{m'} q'\) leads to \((q', m \cdot m')\)
Valence system over $\mathbb{Z} \times \mathbb{Z}$ (with component-wise addition)
Valence system over $\mathbb{Z} \times \mathbb{Z}$ (with component-wise addition)

(essentially an integer 2-VASS)
Want results of the following shape:

**Theorem**

*If monoid $\mathbb{M}$ satisfies condition $c$, then checking property $P$ for all valence systems over $\mathbb{M}$ is in complexity class $\mathcal{C}$.*
Graph monoids

Want results of the following shape:

**Theorem**

*If monoid $\mathbb{M}$ satisfies condition $c$, then checking property $P$ for all valence systems over $\mathbb{M}$ is in complexity class $\mathcal{C}$.***

Best case: *Complete for classification* for property $P$
Graph monoids

Want results of the following shape:

**Theorem**

*If monoid $\mathbb{M}$ satisfies condition $c$, then checking property $P$ for all valence systems over $\mathbb{M}$ is in complexity class $\mathcal{C}$.***

Best case: **Complete for classification** for property $P$

For example, want classification of $P = \text{reachability}$

**Reachability for valence systems**

*Given:* Valence system $\mathcal{A}$ over monoid $\mathbb{M}$

*Decide:* $(q_{\text{init}}, 1_{\mathbb{M}}) \rightarrow^* (q_{\text{final}}, 1_{\mathbb{M}})\,$?
Problem: Monoids are too diverse
Graph monoids

**Problem:** Monoids are too diverse

Focus on an interesting subclass of monoids
Graph monoids

Problem: Monoids are too diverse

Focus on an interesting subclass of monoids

Finitely generated monoids: Too diverse
Graph monoids

**Problem:** Monoids are too diverse

Focus on an interesting subclass of monoids

**Finitely generated monoids:** Too diverse

**Finite monoids:** Not expressive
Graph monoids

**Problem:** Monoids are too diverse

Focus on an interesting subclass of monoids

*Finitely generated monoids:* Too diverse

*Finite monoids:* Not expressive

*Graph monoids*
Example: Graph monoid

Consider the following undirected graph:

\[ \begin{align*}
  &a & b \\
\end{align*} \]
Example: Graph monoid

Consider the following undirected graph:

Nodes $a, b$ are counters / stack symbols
Consider the following undirected graph:

Nodes $a, b$ are counters / stack symbols

Operations: $a^+, b^+$ ("push $a / b$", "increment $a / b$") and $a^-, b^-$ ("pop $a / b$", "decrement $a / b$")
Example: Graph monoid

Consider the following undirected graph:

\[
\begin{array}{c}
\bullet \\
a \\
\bullet \\
b
\end{array}
\]

Nodes \(a, b\) are counters / stack symbols

Operations: \(a^+, b^+\) ("push a / b", "increment a / b")
and \(a^-, b^-\) ("pop a / b", "decrement a / b")

Monoid elements: Sequences of operations
Example: Graph monoid

Consider the following undirected graph:

\[ a \quad b \]

Nodes \( a, b \) are counters / stack symbols

Operations: \( a^+, b^+ \) ("push a / b", "increment a / b")

and \( a^-, b^- \) ("pop a / b", "decrement a / b")

Monoid elements: Sequences of operations modulo the congruence \( o^+.o^- \cong \varepsilon \)
Example: PDS

\[ \mathbb{M}_G = \{a^+, b^+, a^-, b^-\}^* / \cong \]

\[ o^+.o^- \cong \varepsilon \ \forall o \in \{a, b\} \]
Example: PDS

\[ \mathbb{M}_G = \{a^+, b^+, a^-, b^-\}^*/ \cong \]

\[ o^+.o^- \cong \varepsilon \ \forall o \in \{a, b\} \]
Example: PDS

\[ \mathbb{M}_G = \{a^+, b^+, a^-, b^-\}^*/ \cong \]

\[ o^+.o^- \cong \varepsilon \quad \forall o \in \{a, b\} \]

\[ a^+b^+b^-a^- \cong a^+a^- \]
Example: PDS

\[ M_G = \{a^+, b^+, a^-, b^-\}^*/\cong \]

\[ o^+.o^- \cong \varepsilon \ \forall o \in \{a, b\} \]

\[ a^+b^+b^-a^- \cong a^+a^- \cong \varepsilon = 1_M \]
Example: PDS

\[ M_G = \{ a^+, b^+, a^-, b^- \}^* / \simeq \]

\[ o^+.o^- \simeq \varepsilon \ \forall o \in \{ a, b \} \]

\[ a^+b^+b^-a^- \simeq a^+a^- \simeq \varepsilon = 1_{M_I} \]

\[ a^+b^+a^-b^- \]

irreducible
Example: PDS

\[ M_G = \{ a^+, b^+, a^-, b^- \}^* / \cong \]

\[ o^+.o^- \cong \varepsilon \ \forall o \in \{ a, b \} \]

\[ a^+ b^+ b^- a^- \cong a^+ a^- \cong \varepsilon = 1_{M_I} \]

\[ a^+ b^+ a^- b^- \] irreducible

\[ a^- a^+ \] irreducible
Example: PDS

\[ M_G = \{ a^+, b^+, a^-, b^- \}^* / \cong \]

\[ o^+.o^- \cong \varepsilon \ \forall o \in \{a, b\} \]

\[ a^+b^+b^-a^- \cong a^+a^- \cong \varepsilon = 1_{M_I} \]

\[ a^+b^+a^-b^- \quad \text{irreducible} \]

\[ a^-a^+ \quad \text{irreducible} \]

Valence systems over \( M_G \) are PDS over stack alphabet \( \{a, b\} \)
Graph monoids

Graph monoid $\mathbb{M}_G$ given by undirected graph $G = (V, I)$
Graph monoid $\mathbb{M}_G$ given by undirected graph $G = (V, I)$

Nodes of $G$ are counters
Graph monoid $\mathbb{M}_G$ given by undirected graph $G = (V, \mathcal{E})$

Nodes of $G$ are counters

Operations $\mathcal{O}$ consisting of $o^+, o^-$ for each node $o \in V$
Graph monoids

Graph monoid $\mathbb{M}_G$ given by undirected graph $G = (V, \mathcal{I})$

Nodes of $G$ are counters

Operations $\mathcal{O}$ consisting of $o^+$, $o^-$ for each node $o \in V$

$\mathbb{M}_G = \mathcal{O}^*/\cong$
Graph monoids

Graph monoid $\mathbb{M}_G$ given by undirected graph $G = (V, \mathcal{I})$

Nodes of $G$ are counters

Operations $\mathcal{O}$ consisting of $o^+$, $o^-$ for each node $o \in V$

$\mathbb{M}_G = \mathcal{O}^*/\simeq$

Monoid elements are represented by sequences of operations
Graph monoid $\mathbb{M}_G$ given by undirected graph $G = (V, \mathcal{E})$

Nodes of $G$ are counters

Operations $\mathcal{O}$ consisting of $o^+, o^-$ for each node $o \in V$

$\mathbb{M}_G = \mathcal{O}^*/\simeq$

Monoid elements are represented by sequences of operations

Monoid operation: Concatenation of representatives
Graph monoids

Graph monoid $\mathbb{M}_G$ given by undirected graph $G = (V, \mathcal{I})$

Nodes of $G$ are counters

Operations $\mathcal{O}$ consisting of $o^+, o^-$ for each node $o \in V$

$\mathbb{M}_G = \mathcal{O}^*/\cong$

Monoid elements are represented by sequences of operations

Monoid operation: Concatenation of representatives

Congruence $\cong$ satisfies $o^+.o^- \cong \varepsilon$ for all $o$
Graph monoids

Graph monoid $\mathbb{M}_G$ given by undirected graph $G = (V, I)$

Congruence $\cong$ satisfies $o^+.o^- \cong \varepsilon$ for all $o$
Graph monoids

Graph monoid $\mathbb{M}_G$ given by undirected graph $G = (V, \mathcal{I})$

Congruence $\cong$ satisfies $o^+.o^- \cong \varepsilon$ for all $o$

Edge relation $\mathcal{I}$ called independence relation
Graph monoids

Graph monoid $\mathbb{M}_G$ given by undirected graph $G = (V, \mathcal{I})$

Congruence $\cong$ satisfies $o^+.o^- \cong \varepsilon$ for all $o$

Edge relation $\mathcal{I}$ called independence relation

Intuition:
Graph monoids

Graph monoid $\mathbb{M}_G$ given by undirected graph $G = (V, I)$

Congruence $\equiv$ satisfies $o^+.o^- \equiv \varepsilon$ for all $o$

Edge relation $I$ called independence relation

Intuition:
If $o \mathrel{I} u$, then $o$ and $u$ belong to independents part of the storage
Graph monoids

Graph monoid $\mathbb{M}_G$ given by undirected graph $G = (V, \mathcal{I})$

Congruence $\cong$ satisfies $o^+.o^- \cong \varepsilon$ for all $o$

Edge relation $\mathcal{I}$ called independence relation

Intuition:
If $o \mathcal{I} u$, then $o$ and $u$ belong to independents part of the storage
Congruence should identify computations that order independent operations differently
Graph monoid $M_G$ given by undirected graph $G = (V, \mathcal{I})$

Congruence $\cong$ satisfies $o^+ . o^- \cong \varepsilon$ for all $o$

Edge relation $\mathcal{I}$ called independence relation

Intuition:
If $o \mathcal{I} u$, then $o$ and $u$ belong to independents part of the storage
Congruence should identify computations that order independent operations differently

If $o \mathcal{I} u$, then $o^\pm$ and $u^\pm$ commute: $o^\pm . u^\pm \cong u^\pm . o^\pm$
Example: VASS

\[ M_G = \{ a^+, b^+, a^-, b^- \}^*/ \cong \]

\[ o^+.o^- \cong \varepsilon \ \forall o \in \{a, b\} \]

\[ o^\pm .u^\pm \cong u^\pm .o^\pm \text{ where } \{u, o\} = \{a, b\} \]
Example: VASS

\[ M_G = \{a^+, b^+, a^-, b^-\}^* / \sim \]

\[ o^+.o^- \sim \varepsilon \ \forall o \in \{a, b\} \]

\[ o^\pm.u^\pm \sim u^\pm.o^\pm \text{ where } \{u, o\} = \{a, b\} \]

\[ a^+b^+b^-a^- \sim a^+a^- \sim \varepsilon = 1_M \text{ still valid} \]
Example: VASS

\[ \mathbb{M}_G = \{a^+, b^+, a^-, b^-\}^*/ \cong \]

\[ o^+.o^- \cong \varepsilon \ \forall o \in \{a, b\} \]

\[ o^\pm.u^\pm \cong u^\pm.o^\pm \text{ where } \{u, o\} = \{a, b\} \]

\[ a^+b^+b^-a^- \cong a^+a^- \cong \varepsilon = 1_{\mathbb{M}} \text{ still valid} \]

\[ a^+b^+a^-b^- \cong a^+b^+b^-a^- \cong \varepsilon \]
Example: VASS

\[ \mathbb{M}_G = \{a^+, b^+, a^-, b^-\}^* / \cong \]

\[ o^+.o^- \cong \varepsilon \ \forall o \in \{a, b\} \]

\[ o^\pm.u^\pm \cong u^\pm.o^\pm \text{ where } \{u, o\} = \{a, b\} \]

\[ a^+b^+b^-a^- \cong a^+a^- \cong \varepsilon = 1_{\mathbb{M}} \text{ still valid} \]

\[ a^+b^+a^-b^- \cong a^+b^+b^-a^- \cong \varepsilon \]

\[ a^-a^+ \text{ still irreducible} \]
Example: VASS

\[ M_G = \{a^+, b^+, a^-, b^-\}^*/ \cong \]

\[ o^+.o^- \cong \varepsilon \quad \forall o \in \{a, b\} \]

\[ o^\pm.u^\pm \cong u^\pm.o^\pm \quad \text{where} \quad \{u, o\} = \{a, b\} \]

\[ a^+ b^+ b^- a^- \cong a^+ a^- \cong \varepsilon \cong 1_{M} \quad \text{still valid} \]

\[ a^+ b^+ a^- b^- \cong a^+ b^+ b^- a^- \cong \varepsilon \]

\[ a^-a^+ \quad \text{still irreducible} \]

Valence systems over \( M_G \) are \( 2\text{-VASS} \)
Example: integer VASS

\[ M_G = \{ a^+, b^+, a^-, b^- \}^* / \cong \]

\[ o^+.o^- \cong \varepsilon \ \forall o \in \{ a, b \} \]

\[ o^\pm .u^\pm \cong u^\pm .o^\pm \ \forall u, o \in \{ a, b \} \]
**Example: integer VASS**

\[ M_G = \{a^+, b^+, a^-, b^-\}^* / \sim \]

\[ o^+.o^- \sim \varepsilon \quad \forall o \in \{a, b\} \]

\[ o^\pm.u^\pm \sim u^\pm.o^\pm \quad \forall u, o \in \{a, b\} \]

\[ a^+b^+b^-a^- \sim a^+a^- \sim \varepsilon = 1_M \quad \text{still valid} \]
Example: integer VASS

\[ M_G = \{ a^+, b^+, a^-, b^- \}^* / \cong \]

\[ o^+.o^- \cong \epsilon \ \forall o \in \{a, b\} \]

\[ o^\pm.u^\pm \cong u^\pm.o^\pm \ \forall u, o \in \{a, b\} \]

\[ a^+ b^+ b^- a^- \cong a^+ a^- \cong \epsilon = 1_M \ \text{still valid} \]

\[ a^+ b^+ a^- b^- \cong a^+ b^+ b^- a^- \cong \epsilon \]
Example: integer VASS

\[ \mathbb{M}_G = \{a^+, b^+, a^-, b^-\}^* / \cong \]

\[ o^+ . o^- \cong \varepsilon \quad \forall o \in \{a, b\} \]

\[ o^\pm . u^\pm \cong u^\pm . o^\pm \quad \forall u, o \in \{a, b\} \]

\[ a^+ b^+ b^- a^- \cong a^+ a^- \cong \varepsilon = 1_{\mathbb{M}} \quad \text{still valid} \]

\[ a^+ b^+ a^- b^- \cong a^+ b^+ b^- a^- \cong \varepsilon \]

\[ a^- a^+ \cong a^+ a^- \cong \varepsilon \]
Example: integer VASS

$$\mathbb{M}_G = \{a^+, b^+, a^-, b^-\}^*/\cong$$

$$o^+.o^- \cong \varepsilon \ \forall o \in \{a, b\}$$

$$o^\pm.u^\pm \cong u^\pm.o^\pm \ \forall u, o \in \{a, b\}$$

$$a^+b^+b^-a^- \cong a^+a^- \cong \varepsilon = 1_\mathbb{M} \quad \text{still valid}$$

$$a^+b^+a^-b^- \cong a^+b^+b^-a^- \cong \varepsilon$$

$$a^-a^+ \cong a^+a^- \cong \varepsilon$$

Valence systems over $\mathbb{M}_G$ are integer 2-VASS
Example: MPDS

\begin{center}
\begin{tikzpicture}

\node[draw, circle, fill=black] (a) at (0,0) {$a_\ell$};
\node[draw, circle, fill=black] (b) at (1,1) {$b_\ell$};
\node[draw, circle, fill=black] (c) at (1,-1) {$b_r$};
\node[draw, circle, fill=black] (d) at (0,0) {$a_r$};

\draw (a) -- (b);
\draw (a) -- (c);
\draw (b) -- (c);
\draw (b) -- (d);
\draw (c) -- (d);

\end{tikzpicture}
\end{center}
Any \( m \in \{a_\ell^+, a_\ell^-, \ldots\}^* \) can be written as

\[
m \cong m_{\mid_\ell} \cdot m_{\mid_r}\]

such that \( m \cong \varepsilon \) iff \( m_{\mid_\ell} \cong \varepsilon \) and \( m_{\mid_r} \cong \varepsilon \)
Any $m \in \{a_\ell^+, a_\ell^-, \ldots\}^*$ can be written as

$$m \equiv m_{\mid_\ell} \cdot m_{\mid_r}$$

such that $m \equiv \varepsilon$ iff $m_{\mid_\ell} \equiv \varepsilon$ and $m_{\mid_r} \equiv \varepsilon$

Valence systems over $\mathbb{M}_G$ are 2-PDS (with a binary stack alphabet for each stack)
Graph monoids can model:
Graph monoids can model:

- Natural (partially blind) counters
- Integer (blind) counters
Graph monoids can model:

- Natural (partially blind) counters
- Integer (blind) counters
- Combinations of these
Graph monoids can model:

- Natural (partially blind) counters
- Integer (blind) counters
- Combinations of these
- Stacks of these
Graph monoids can model:

- Natural (partially blind) counters
- Integer (blind) counters
- Combinations of these
- Stacks of these

Graph monoids cannot model:
Graph monoids can model:
- Natural (partially blind) counters
- Integer (blind) counters
- Combinations of these
- Stacks of these

Graph monoids cannot model:
- Queues
Graph monoids

Graph monoids can model:
- Natural (partially blind) counters
- Integer (blind) counters
- Combinations of these
- Stacks of these

Graph monoids cannot model:
- Queues
- Higher-order stacks
Characterization results for valence systems/automata:

reachability [Z15]
regularity [Z11]
context-freeness [BZ13]
semilinearity of the Parikh image [BZ13]
...

3. BCS for valence systems
How to define BCS for valence systems over graph monoids?
How to define BCS for valence systems over graph monoids?

Concurrent system as valence system

Assume:
How to define BCS for valence systems over graph monoids?

Concurrent system as valence system

Assume:

The system is modeled as a single valence system
How to define BCS for valence systems over graph monoids?

Concurrent system as valence system

Assume:

- The system is modeled as a single valence system
- The monoid models the total storage of all components
How to define BCS for valence systems over graph monoids?

Concurrent system as valence system

Assume:

- The system is modeled as a single valence system
- The monoid models the total storage of all components
- The components share a control state
  (communication between components)
How to define BCS for valence systems over graph monoids?
How to define BCS for valence systems over graph monoids?

A slight modification
How to define BCS for valence systems over graph monoids?

A slight modification

Consider configurations of the shape \((q, m)\) where \(m\) is a sequence of operations.
How to define BCS for valence systems over graph monoids?

A slight modification

Consider configurations of the shape $(q, m)$ where $m$ is a sequence of operations.

We do not store the monoid element, but its syntactic representation.
How to define BCS for valence systems over graph monoids?

A slight modification

Consider configurations of the shape \((q, m)\) where \(m\) is a sequence of operations

We do not store the monoid element, but its syntactic representation

Crucial as our notion of context is not invariant under congruence
Contexts

\[ a_\ell \quad b_\ell \quad b_r \quad a_r \]
Nodes belonging to independent parts of the storage are connected by an edge.
Nodes belonging to independent parts of the storage are connected by an edge

Intuitively:

$$m = \ldots o^\pm . u^\pm \ldots$$

with $o I u$, then this constitutes a context switch
Nodes belonging to independent parts of the storage are connected by an edge

Intuitively:

\[ m = \ldots o^\pm . u^\pm \ldots \]

with \( o \mathcal{I} u \), then this constitutes a context switch

In general, we need a more restrictive definition
Definition

A sequence of operations $m$ is called dependent if for all $o, u$ in $m$ with $o \neq u$, $o \not\equiv u$ does not hold.
Definition

A sequence of operations $m$ is called **dependent** if for all $o^\pm, u^\pm$ in $m$ with $o \neq u$, $o \not\mathcal{I} u$ does not hold.
Definition
A sequence of operations $m$ is called dependent if for all $o^\pm, u^\pm$ in $m$ with $o \neq u$, $o \mathcal{I} u$ does not hold.

\[ a^+c^+ \text{ dependent} \]
Dependent computations

**Definition**

A sequence of operations $m$ is called **dependent** if for all $o^\pm, u^\pm$ in $m$ with $o \neq u$, $o \not\equiv u$ does not hold.

\[
\begin{align*}
  a^+c^+ & \quad \text{dependent} \\
  b^+c^+ & \quad \text{dependent}
\end{align*}
\]
Dependent computations

**Definition**
A sequence of operations $m$ is called **dependent** if for all $o^\pm, u^\pm$ in $m$ with $o \neq u$, $o \not\equiv u$ does not hold.

\[
\begin{array}{ccc}
\text{c} & \cdot & \text{a} + c^+ & \text{dependent} \\
& & b^+ c^+ & \text{dependent} \\
\text{a} & \text{b} & a^+ b^+ & \text{not dependent}
\end{array}
\]
**Dependent computations**

**Definition**
A sequence of operations \( m \) is called **dependent** if for all \( o^\pm, u^\pm \) in \( m \) with \( o \neq u \), \( o \not\mathcal{I} u \) does not hold.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Dependence</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a^+c^+ )</td>
<td>dependent</td>
</tr>
<tr>
<td>( b^+c^+ )</td>
<td>dependent</td>
</tr>
<tr>
<td>( a^+b^+ )</td>
<td>not dependent</td>
</tr>
<tr>
<td>( a^+c^+b^+ )</td>
<td>not dependent</td>
</tr>
</tbody>
</table>
Dependent computations

**Definition**

A sequence of operations $m$ is called **dependent** if for all $o^\pm, u^\pm$ in $m$ with $o \neq u$, $o \not I u$ does not hold.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a^+c^+$</td>
<td>dependent</td>
</tr>
<tr>
<td>$b^+c^+$</td>
<td>dependent</td>
</tr>
<tr>
<td>$a^+b^+$</td>
<td>not dependent</td>
</tr>
<tr>
<td>$a^+c^+b^+$</td>
<td>not dependent</td>
</tr>
<tr>
<td>$a^+a^-$</td>
<td>dependent</td>
</tr>
</tbody>
</table>
Definition
A sequence of operations $m$ is called dependent if for all $o^\pm, u^\pm$ in $m$ with $o \neq u$, $o \mathcal{I} u$ does not hold.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a^+ c^+$</td>
<td>dependent</td>
</tr>
<tr>
<td>$b^+ c^+$</td>
<td>dependent</td>
</tr>
<tr>
<td>$a^+ b^+$</td>
<td>not dependent</td>
</tr>
<tr>
<td>$a^+ c^+ b^+$</td>
<td>not dependent</td>
</tr>
<tr>
<td>$a^+ a^-$</td>
<td>dependent</td>
</tr>
<tr>
<td>$a^+ b^+ b^- a^-$</td>
<td>not dependent</td>
</tr>
</tbody>
</table>
**Definition**
A sequence of operations $m$ is called dependent if for all $o^\pm, u^\pm$ in $m$ with $o \neq u$, $o \not I u$ does not hold.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Dependent/Not Dependent</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a^+c^+$</td>
<td>dependent</td>
</tr>
<tr>
<td>$b^+c^+$</td>
<td>dependent</td>
</tr>
<tr>
<td>$a^+b^+$</td>
<td>not dependent</td>
</tr>
<tr>
<td>$a^+c^+b^+$</td>
<td>not dependent</td>
</tr>
<tr>
<td>$a^+a^-$</td>
<td>dependent</td>
</tr>
<tr>
<td>$a^+b^+b^-a^-$</td>
<td>not dependent</td>
</tr>
<tr>
<td>but $a^+a^- \not\equiv a^+b^+b^-a^-$!</td>
<td></td>
</tr>
</tbody>
</table>
Let $m$ be a sequence of operations
Let $m$ be a sequence of operations

Its first context is its maximal dependent prefix
Let $m$ be a sequence of operations

Its first context is its maximal dependent prefix

Inductively:
The $i^{th}$ context of $m$ is the maximal dependent prefix of $m$ with the first $i - 1$ contexts removed
Let $m$ be a sequence of operations

Its first context is its maximal dependent prefix

Inductively:
The $i^{th}$ context of $m$ is the maximal dependent prefix of $m$ with the first $i - 1$ contexts removed

The number of context switches $cs(m)$ is the number of contexts minus 1
In the examples

Assume the number of context switches is bounded by \( k \)

1. \( a \) \( b \)  
   PDS  
   no restriction

2. \( a \) \( b \)  
   VASS  
   changing the counter \( \leq k \) times

3. \( a \) \( b \)  
   integer VASS  
   changing the counter \( \leq k \) times

4. \( b_\ell \) \( b_r \)  
   MPDS  
   changing the stack \( \leq k \) times
BCSREACH for valence systems over graph monoids

Given: Valence system $\mathcal{A}$ over $\mathbb{M}_G$, number $k$ (in unary)

Decide: Is there $(q_{\text{init}}, \varepsilon) \rightarrow (q_{\text{final}}, m)$
with $m \cong \varepsilon$ and $cs(m) \leq k$?
The result

**Theorem**

BCSREACH for valence systems over graph monoids is in NP (for all graph monoids).

---

BCSREACH for valence systems over graph monoids

**Given:** Valence system $\mathcal{A}$ over $\mathbb{M}_G$, number $k$ (in unary)

**Decide:** Is there $\left(q_{\text{init}}, \varepsilon\right) \rightarrow \left(q_{\text{final}}, m\right)$ with $m \cong \varepsilon$ and $\text{cs}(m) \leq k$?
The proof / The algorithm
Proof outline

Need to find a computation \((q_{\text{init}}, \varepsilon) \rightarrow^* (q_{\text{final}}, m)\) with \(m \preceq \varepsilon\)
Proof outline

Need to find a computation \((q_{\text{init}}, \varepsilon) \rightarrow^* (q_{\text{final}}, m)\) with \(m \cong \varepsilon\)

**Good:** Bound \(cs(m) \leq k\)
Proof outline

Need to find a computation \((q_{\text{init}}, \varepsilon) \to^* (q_{\text{final}}, m)\) with \(m \equiv \varepsilon\)

**Good:** Bound \(cs(m) \leq k\)

**Bad:** No bound on length of length of \(m\)
Proof outline

Need to find a computation \((q_{\text{init}}, \varepsilon) \rightarrow^* (q_{\text{final}}, m)\) with \(m \equiv \varepsilon\)

**Good:** Bound \(cs(m) \leq k\)

**Bad:** No bound on length of length of \(m\)

Consider blockwise-reduction
Need to find a computation \((q_{\text{init}}, \varepsilon) \rightarrow^* (q_{\text{final}}, m)\) with \(m \equiv \varepsilon\)

**Good:** Bound \(cs(m) \leq k\)

**Bad:** No bound on length of length of \(m\)

Consider blockwise-reduction

If contexts irreducible, get existence of a reducible block decomposition of length \(\leq k^2\)
Proof outline

Need to find a computation \((q_{init}, \varepsilon) \rightarrow^* (q_{final}, m)\) with \(m \cong \varepsilon\)

**Good:** Bound \(cs(m) \leq k\)

**Bad:** No bound on length of length of \(m\)

Consider **blockwise-reduction**

If contexts **irreducible**, get existence of a reducible **block decomposition** of length \(\leq k^2\)

Ensure irreducibility by saturating system
Proof outline

Need to find a computation \((q_{\text{init}}, \varepsilon) \rightarrow^* (q_{\text{final}}, m)\) with \(m \cong \varepsilon\)

**Good:** Bound \(cs(m) \leq k\)

**Bad:** No bound on length of length of \(m\)

Consider **blockwise-reduction**

If contexts **irreducible**, get existence of a reducible **block decomposition of length \(\leq k^2\)**

Ensure irreducibility by saturating system

Then check existence of reducible block decomposition using guessing and representing blocks as **finite automata**
If $m \cong \varepsilon$, then there is a reduction of $m$ that swaps letters and cancels letters.
If $m \equiv \epsilon$, then there is a reduction of $m$ that
swaps letters
cancels letters.

Can define similarly a notation of reduction that
swaps blocks (infixes)
cancels blocks
in one step.
If \( m \cong \varepsilon \), then there is a \textit{reduction} of \( m \) that
swaps letters
cancels letters.

Can define similarly a notation of reduction that
swaps blocks (infixes)
cancels blocks
in one step.

E.g. \( m_1.m_2 \rightarrow m_2.m_1 \) if \textit{every symbol} in \( m_1 \) commutes with \textit{every symbol} in \( m_2 \)
Block decomposition

Let $m = m_1, m_2, \ldots, m_n$ be a decomposition of $m$ into blocks.
Let \( m = m_1, m_2, \ldots, m_n \) be a decomposition of \( m \) into blocks.

If \( m \) can be reduced to \( \varepsilon \) by blockwise operations, call it \textit{freely reducible}. 
Let $m = m_1, m_2, \ldots, m_n$ be a decomposition of $m$ into blocks.

If $m$ can be reduced to $\varepsilon$ by blockwise operations, call it freely reducible.

If $m \cong \varepsilon$, then its decomposition into letters is always freely reducible.
Let $m = m_1, m_2, \ldots, m_n$ be a decomposition of $m$ into blocks.

If $m$ can be reduced to $\varepsilon$ by blockwise operations, call it \textit{freely reducible}.

If $m \cong \varepsilon$, then its decomposition into letters is always freely reducible.

Coarser decompositions might not be freely reducible:

\[ o^+ u^+ , u^- , o^- \]
Sequence is **irreducible** if it is not congruent to a shorter one.
Sequence is \textbf{irreducible} if it is not congruent to a shorter one

<table>
<thead>
<tr>
<th>Theorem</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{Let} ( m ) \textit{be a sequence of operations with}</td>
</tr>
<tr>
<td>( k ) contexts</td>
</tr>
<tr>
<td>\textit{each of them irreducible, and}</td>
</tr>
<tr>
<td>( m \cong \varepsilon. )</td>
</tr>
<tr>
<td>\textit{Then there is a decomposition of} ( m ) \textit{into} ( \leq k^2 ) \textit{blocks that is freely reducible}.</td>
</tr>
</tbody>
</table>
Sequence is **irreducible** if it is not congruent to a shorter one.

**Theorem**

Let $m$ be a sequence of operations with

$k$ contexts

each of them irreducible, and

$m \cong \varepsilon$.

Then there is a decomposition of $m$ into $\leq k^2$ blocks that is freely reducible.

Size of the decomposition is independent of the length of $m$. 
Sequence is **irreducible** if it is not congruent to a shorter one.

**Theorem**

*Let* $m$ *be a sequence of operations with* $k$ *contexts, each of them irreducible, and* $m \cong \varepsilon$.

*Then there is a decomposition of* $m$ *into* $\leq k^2$ *blocks that is freely reducible.*

**Size of the decomposition is independent** of the length of $m$. **Existence can be checked algorithmically.**
The algorithm

**Given:** valence system $\mathcal{A}$, bound $k$
The algorithm, Step I

The algorithm

**Given:** valence system $\mathcal{A}$, bound $k$

**Part I: Enforcing irreducibility**
The algorithm

Given: valence system $\mathcal{A}$, bound $k$

Part I: Enforcing irreducibility

1. Guess $\leq k$ dependent parts of $\mathcal{A}$
The algorithm, Step I

The algorithm

**Given:** valence system $\mathcal{A}$, bound $k$

**Part I: Enforcing irreducibility**

1. Guess $\leq k$ dependent parts of $\mathcal{A}$
2. Saturate each part:
Let $A_{\text{sat}}$ be the resulting valence system.
Let $\mathcal{A}_{\text{sat}}$ be the resulting valence system.

**Theorem**

$$(q_{\text{init}}, \varepsilon) \rightarrow^* (q_{\text{final}}, m) \text{ in } \mathcal{A} \text{ with } m \equiv \varepsilon \text{ and } cs(m) \leq k,$$
The algorithm, Step 1

Let $A_{\text{sat}}$ be the resulting valence system

**Theorem**

$$(q_{\text{init}}, \varepsilon) \rightarrow^* (q_{\text{final}}, m) \text{ in } A \text{ with } m \cong \varepsilon \text{ and } cs(m) \leq k,$$

iff

$$(q_{\text{init}}, \varepsilon) \rightarrow^* (q_{\text{final}}, m') \text{ in } A_{\text{sat}} \text{ with } m' \cong \varepsilon, cs(m') \leq k,$$

and contexts of $m'$ irreducible.
The algorithm, Step II

Part II:
Checking the existence of a freely reducible block decomposition
The algorithm, Step II

Part II:
Checking the existence of a freely reducible block decomposition

3. For each context $i$, guess part of $A_{sat}$ that is used in block $m_{i,j}$ as NFA $A_{i,j}$
Part II:
Checking the existence of a freely reducible block decomposition

3. For each context $i$, guess part of $A_{sat}$ that is used in block $m_{i,j}$ as NFA $A_{i,j}$

4. Guess reduction on blocks (NFAs) of polynomial length
Part II:
Checking the existence of a freely reducible block decomposition

3. For each context $i$, guess part of $A_{\text{sat}}$ that is used in block $m_{i,j}$ as NFA $A_{i,j}$

4. Guess reduction on blocks (NFAs) of polynomial length

5. Verify reduction step-by-step
The algorithm, Step II

Part II:
Checking the existence of a freely reducible block decomposition

3. For each context \( i \), guess part of \( \mathcal{A}_{\text{sat}} \) that is used in block \( m_{i,j} \) as NFA \( \mathcal{A}_{i,j} \)

4. Guess reduction on blocks (NFAs) of polynomial length

5. Verify reduction step-by-step

   Swap rule applicable to \( \mathcal{A}_{i,j}, \mathcal{A}'_{i',j'} \)
   if \( \forall o^\pm \in \text{Alphabet}(\mathcal{A}_{i,j}) \forall u^\pm \in \text{Alphabet}(\mathcal{A}'_{i',j'}) : o \not\epsilon u \)
The algorithm, Step II

Part II: Checking the existence of a freely reducible block decomposition

3. For each context \( i \), guess part of \( \mathcal{A}_{\text{sat}} \) that is used in block \( m_{i,j} \) as NFA \( \mathcal{A}_{i,j} \)

4. Guess reduction on blocks (NFAs) of polynomial length

5. Verify reduction step-by-step

   - **Swap rule** applicable to \( \mathcal{A}_{i,j}, \mathcal{A}_{i',j'} \)
     
     \[ \text{if } \forall o^\pm \in \text{Alphabet}(\mathcal{A}_{i,j}) \forall u^\pm \in \text{Alphabet}(\mathcal{A}_{i',j'}): o \mathcal{I} u \]

   - **Cancel rule** applicable to \( \mathcal{A}_{i,j}, \mathcal{A}_{i',j'} \)
     
     \[ \text{if } \mathcal{L}(\mathcal{A}_{i,j}) \cap \mathcal{L}(\mathcal{A}_{i',j'})^{\text{inverse}} \text{ is non-empty} \]
Complexity for fixed graphs
Now, assume that the graph \( G \) is fixed, consider \( \text{BCSREACH}(G) \).
Now, assume that the graph $G$ is fixed, consider $\text{BCSREACH}(G)$.

Let $G^-$ denote $G$ with self-loops removed.
Now, assume that the graph $G$ is fixed, consider $\text{BCSREACH}(G)$

Let $G^{-}$ denote $G$ with self-loops removed.

**Theorem**

*If $G^{-}$ is a clique, then $\text{BCSREACH}(G)$ is NL-complete.*
Now, assume that the graph $G$ is **fixed**, consider $\text{BCSREACH}(G)$

Let $G^-$ denote $G$ with self-loops removed.
Now, assume that the graph $G$ is fixed, consider $\text{BCSREACH}(G)$.

Let $G^-$ denote $G$ with self-loops removed.

**Theorem**

*If $G^-$ contains $C_4$ as induced subgraph, then $\text{BCSREACH}(G)$ is NP-complete.*
Now, assume that the graph $G$ is fixed, consider $\text{BCSREACH}(G)$.

Let $G^-$ denote $G$ with self-loops removed.

**Theorem**

*If $G^-$ contains $C_4$ as induced subgraph, then $\text{BCSREACH}(G)$ is NP-complete.*

**Theorem**

*If $G^-$ contains neither $C_4$ nor $P_4$ as induced subgraphs, then $\text{BCSREACH}(G)$ is in $P$.***
Conclusion

Theorem

Reachability under *bounded context switching*

for *valence systems over graph monoids*

is *always in NP*.

+ almost complete classification of complexity for fixed graphs.
Conclusion

Theorem

Reachability under *bounded context switching*

for *valence systems over graph monoids*

*is always in NP.*

+ almost complete classification of complexity for fixed graphs.

Open problems / future work:
Theorem

Reachability under *bounded context switching* for *valence systems over graph monoids* is *always in NP*.

+ almost complete classification of complexity for fixed graphs.

Open problems / future work:

 Complexity for valence systems over P4?
Theorem

Reachability under **bounded context switching**
for **valence systems over graph monoids**
is **always in NP**.

+ almost complete classification of complexity for fixed graphs.

**Open problems / future work:**

Complexity for valence systems over P4?
Bounded **phase** switching?
Conclusions

**Theorem**

Reachability under *bounded context switching* for *valence systems over graph monoids* is *always in NP*.

+ almost complete classification of complexity for fixed graphs.

**Open problems / future work:**

- Complexity for valence systems over P4?
- Bounded *phase* switching?
- BCS for *reachability games*?
Conclusion

Theorem

*Reachability under bounded context switching for valence systems over graph monoids is always in NP.*

+ almost complete classification of complexity for fixed graphs.

Open problems / future work:

- Complexity for valence systems over P4?
- Bounded phase switching?
- BCS for reachability games?
- Richer model supporting queues, higher order?
Thank you!
Questions?