Contextual Locking for Dynamic Pushdown Networks

Peter Lammich[1] Markus Müller-Olm[2] Helmut Seidl[1]
Alexander Wenner[2]

[1] TU München

[2] WWU Münster

July 2013

Motivation

- Decidable model for recursive concurrent programs with locks
- Concurrency
  - 2 processes, *n* processes, dynamic process creation
- Locking disciplines
  - Syntactically nested (`synchronized (l) { ... }`)
  - Bounded lock chains ($[_1\ [_2\ ]_1\ [_3\ ]_2\ \dots\ [_c\ ]_{c-1}\ ]_c$)
  - Contextual (`proc p {` $[_1\ [_2\ ]_1\ ]_2$ `... }`)
    - Procedure releases all locks it acquires
    - Procedure releases no „foreign" locks
- Join: Process waits until children have terminated
  - Subsumes parallel procedure calls

Motivation

- Decidable model for recursive concurrent programs with locks
- Concurrency
  - 2 processes, *n* processes, dynamic process creation
- Locking disciplines
  - Syntactically nested (`synchronized (l) { ... }`)
  - Bounded lock chains ($[_1\ [_2\ ]_1\ [_3\ ]_2\ \ldots\ [_c\ ]_{c-1}\ ]_c$)
  - Contextual (`proc p {` $[_1\ [_2\ ]_1\ ]_2$ `... }`)
    - Procedure releases all locks it acquires
    - Procedure releases no „foreign" locks
- Join: Process waits until children have terminated
  - Subsumes parallel procedure calls

Motivation

- Decidable model for recursive concurrent programs with locks
- Concurrency
  - 2 processes, *n* processes, dynamic process creation
- Locking disciplines
  - Syntactically nested (`synchronized (l) { ... }`)
  - Bounded lock chains ($[_1\ [_2\ ]_1\ [_3\ ]_2\ \dots\ [_c\ ]_{c-1}\ ]_c$)
  - Contextual (`proc p {` $[_1\ [_2\ ]_1\ ]_2$ `... }`)
    - Procedure releases all locks it acquires
    - Procedure releases no „foreign“ locks
- Join: Process waits until children have terminated
  - Subsumes parallel procedure calls

Motivation

- Decidable model for recursive concurrent programs with locks
- Concurrency
  - 2 processes, *n* processes, dynamic process creation
- Locking disciplines
  - Syntactically nested (`synchronized (l) { ... }`)
  - Bounded lock chains ($[_1\ [_2\ ]_1\ [_3\ ]_2\ \cdots\ [_c\ ]_{c-1}\ ]_c$)
  - Contextual (`proc p {` $[_1\ [_2\ ]_1\ ]_2$ `... }`)
    - Procedure releases all locks it acquires
    - Procedure releases no „foreign" locks
- Join: Process waits until children have terminated
  - Subsumes parallel procedure calls

Motivation

- Decidable model for recursive concurrent programs with locks
- Concurrency
  - 2 processes, *n* processes, dynamic process creation
- Locking disciplines
  - Syntactically nested (`synchronized (l) { ... }`)
  - Bounded lock chains ($[_1\ [_2\ ]_1\ [_3\ ]_2\ \cdots\ [_c\ ]_{c-1}\ ]_c$)
  - Contextual (`proc p {` $[_1\ [_2\ ]_1\ ]_2$ `... }`)
    - Procedure releases all locks it acquires
    - Procedure releases no „foreign” locks
- Join: Process waits until children have terminated
  - Subsumes parallel procedure calls

- Decidable model for recursive concurrent programs with locks
- Concurrency
  - 2 processes, *n* processes, dynamic process creation
- Locking disciplines
  - Syntactically nested (`synchronized (l) { ... }`)
  - Bounded lock chains ($[_1\ [_2\ ]_1\ [_3\ ]_2\ \dots\ [_c\ ]_{c-1}\ ]_c$)
  - Contextual (`proc p {` $[_1\ [_2\ ]_1\ ]_2$ `... }`)
    - Procedure releases all locks it acquires
    - Procedure releases no „foreign" locks
- Join: Process waits until children have terminated
  - Subsumes parallel procedure calls

Motivation

- Decidable model for recursive concurrent programs with locks
- Concurrency
  - 2 processes, *n* processes, dynamic process creation
- Locking disciplines
  - Syntactically nested (`synchronized (l) { ... }`)
  - Bounded lock chains ($[_1\ [_2\ ]_1\ [_3\ ]_2\ \cdots\ [_c\ ]_{c-1}\ ]_c$)
  - Contextual (`proc p {` $[_1\ [_2\ ]_1\ ]_2$ `... }`)
    - Procedure releases all locks it acquires
    - Procedure releases no „foreign” locks
- Join: Process waits until children have terminated
  - Subsumes parallel procedure calls

Motivation

- Decidable model for recursive concurrent programs with locks
- Concurrency
  - 2 processes, *n* processes, dynamic process creation
- Locking disciplines
  - Syntactically nested (`synchronized (l) { ... }`)
  - Bounded lock chains ($[_1\ [_2\ ]_1\ [_3\ ]_2\ \ldots\ [_c\ ]_{c-1}\ ]_c$)
  - Contextual (`proc p {` $[_1\ [_2\ ]_1\ ]_2\ \ldots$ `}`)
    - Procedure releases all locks it acquires
    - Procedure releases no „foreign” locks
- Join: Process waits until children have terminated
  - Subsumes parallel procedure calls

Motivation

- Decidable model for recursive concurrent programs with locks
- Concurrency
  - 2 processes, n processes, dynamic process creation
- Locking disciplines
  - Syntactically nested (`synchronized (l) { ... }`)
  - Bounded lock chains ($[_1\ [_2\ ]_1\ [_3\ ]_2\ \ldots\ [_c\ ]_{c-1}\ ]_c$)
  - Contextual (`proc p {` $[_1\ [_2\ ]_1\ ]_2$ `... }`)
    - Procedure releases all locks it acquires
    - Procedure releases no „foreign" locks
- Join: Process waits until children have terminated
  - Subsumes parallel procedure calls

Properties

- Single processes: Control state reachability
  - Can process reach control state p?
  - Can express regular properties of stack
- Relation between processes:
  - n-state reachability:
    - Configuration containing distinct processes at $p_1, \ldots, p_n$ reachable?
  - all-state reachability
    - Configuration where all processes are at p reachable?
    - Can express regular reachability:
      Is configuration with a yield from some regular language reachable?

Properties

- Single processes: Control state reachability
    - Can process reach control state p?
    - Can express regular properties of stack
- Relation between processes:
    - n-state reachability:
        - Configuration containing distinct processes at $p_1, \ldots, p_n$ reachable?
    - all-state reachability
        - Configuration where all processes are at p reachable?
        - Can express regular reachability:
          Is configuration with a yield from some regular language reachable?

Properties

- Single processes: Control state reachability
  - Can process reach control state p?
  - Can express regular properties of stack
- Relation between processes:
  - n-state reachability:
    - Configuration containing distinct processes at $p_1, \dots, p_n$ reachable?
  - all-state reachability
    - Configuration where all processes are at p reachable?
    - Can express regular reachability:
      Is configuration with a yield from some regular language reachable

Properties

- Single processes: Control state reachability
  - Can process reach control state p?
  - Can express regular properties of stack
- Relation between processes:
  - n-state reachability:
    - Configuration containing distinct processes at $p_1, \ldots, p_n$ reachable?
  - all-state reachability
    - Configuration where all processes are at p reachable?
    - Can express regular reachability:
      Is configuration with a yield from some regular language reachable

Properties

- Single processes: Control state reachability
  - Can process reach control state p?
  - Can express regular properties of stack
- Relation between processes:
  - n-state reachability:
    - Configuration containing distinct processes at $p_1, \dots, p_n$ reachable?
  - all-state reachability
    - Configuration where all processes are at p reachable?
    - Can express regular reachability:
      Is configuration with a yield from some regular language reachable

Properties

- Single processes: Control state reachability
  - Can process reach control state p?
  - Can express regular properties of stack
- Relation between processes:
  - n-state reachability:
    - Configuration containing distinct processes at $p_1, \ldots, p_n$ reachable?
  - all-state reachability
    - Configuration where all processes are at p reachable?
    - Can express regular reachability:
      Is configuration with a yield from some regular language reachable

Regular reachability

- yield(c) - Control-states in post-order traversal

$$\text{yield}((pw, X)\langle t_1 \dots t_n \rangle) = \text{yield}(t_1) \dots \text{yield}(t_n) p$$

- Let $A = (Q, \delta, q_0, F)$
- New control states: $\langle q, p, q' \rangle$
  - Idea: Process evaluates to tree t with $q \xrightarrow{\text{yield}(t)}_\delta q'$
- Start with initial guess: $p_0 \xrightarrow{\tau} \langle q_0, p_0, q' \rangle$ for $q' \in F$
- Non-spawn rules preserve guess
  - $pw \xrightarrow{\{\tau,\text{acq},\text{rel},\text{join}\}} p'w'$ becomes $\langle q, p, q' \rangle w \xrightarrow{\{\tau,\text{acq},\text{rel},\text{join}\}} \langle q, p', q' \rangle w'$ for all $q, q' \in Q$
- Spawn-rules distribute guess
  - $p\gamma \xrightarrow{p_1\gamma_1} p_2\gamma_2$ becomes $\langle q, p, q_2 \rangle \gamma \xrightarrow{\langle q,p_1,q_1 \rangle \gamma_1} \langle q_1, p_2, q_2 \rangle \gamma_2$ for all $q, q_1, q_2 \in Q$
- Can we reach configuration where all control states have form

$$(\langle q, p, q' \rangle w, X) \text{ such that } (q, p, q') \in \delta$$

Regular reachability

- yield(c) - Control-states in post-order traversal

$$\text{yield}((pw, X)\langle t_1 \ldots t_n \rangle) = \text{yield}(t_1) \ldots \text{yield}(t_n)p$$

- Let $A = (Q, \delta, q_0, F)$
- New control states: $\langle q, p, q' \rangle$
  - Idea: Process evaluates to tree t with $q \xrightarrow{\text{yield}(t)}_\delta q'$
- Start with initial guess: $p_0 \xrightarrow{\tau} \langle q_0, p_0, q' \rangle$ for $q' \in F$
- Non-spawn rules preserve guess
  - $pw \xrightarrow{\{\tau,\text{acq},\text{rel},\text{join}\}} p'w'$ becomes $\langle q, p, q' \rangle w \xrightarrow{\{\tau,\text{acq},\text{rel},\text{join}\}} \langle q, p', q' \rangle w'$ for all $q, q' \in Q$
- Spawn-rules distribute guess
  - $p\gamma \xrightarrow{p_1\gamma_1} p_2\gamma_2$ becomes $\langle q, p, q_2 \rangle \gamma \xrightarrow{\langle q, p_1, q_1 \rangle \gamma_1} \langle q_1, p_2, q_2 \rangle \gamma_2$ for all $q, q_1, q_2 \in Q$
- Can we reach configuration where all control states have form

$$(\langle q, p, q' \rangle w, X) \text{ such that } (q, p, q') \in \delta$$

Regular reachability

- yield(c) - Control-states in post-order traversal

$$\text{yield}((pw, X)\langle t_1 \dots t_n \rangle) = \text{yield}(t_1) \dots \text{yield}(t_n)p$$

- Let $A = (Q, \delta, q_0, F)$
- New control states: $\langle q, p, q' \rangle$
  - Idea: Process evaluates to tree t with $q \xrightarrow{\text{yield}(t)}_{\delta} q'$
- Start with initial guess: $p_0 \xrightarrow{\tau} \langle q_0, p_0, q' \rangle$ for $q' \in F$
- Non-spawn rules preserve guess
  - $pw \xrightarrow{\{\tau,\text{acq},\text{rel},\text{join}\}} p'w'$ becomes $\langle q, p, q' \rangle w \xrightarrow{\{\tau,\text{acq},\text{rel},\text{join}\}} \langle q, p', q' \rangle w'$ for all $q, q' \in Q$
- Spawn-rules distribute guess
  - $p\gamma \xrightarrow{p_1\gamma_1} p_2\gamma_2$ becomes $\langle q, p, q_2 \rangle \gamma \xrightarrow{\langle q, p_1, q_1 \rangle \gamma_1} \langle q_1, p_2, q_2 \rangle \gamma_2$ for all $q, q_1, q_2 \in Q$
- Can we reach configuration where all control states have form

$$(\langle q, p, q' \rangle w, X) \text{ such that } (q, p, q') \in \delta$$

Regular reachability

- yield(c) - Control-states in post-order traversal

$$\text{yield}((pw, X)\langle t_1 \dots t_n\rangle) = \text{yield}(t_1)\dots\text{yield}(t_n)p$$

- Let $A = (Q, \delta, q_0, F)$
- New control states: $\langle q, p, q'\rangle$
  - Idea: Process evaluates to tree t with $q \xrightarrow{\text{yield}(t)}_{\delta} q'$
- Start with initial guess: $p_0 \xrightarrow{\tau} \langle q_0, p_0, q'\rangle$ for $q' \in F$
- Non-spawn rules preserve guess
  - $pw \xrightarrow{\{\tau,\text{acq},\text{rel},\text{join}\}} p'w'$ becomes $\langle q, p, q'\rangle w \xrightarrow{\{\tau,\text{acq},\text{rel},\text{join}\}} \langle q, p', q'\rangle w'$ for all $q, q' \in Q$
- Spawn-rules distribute guess
  - $p\gamma \xrightarrow{p_1\gamma_1} p_2\gamma_2$ becomes $\langle q, p, q_2\rangle\gamma \xrightarrow{\langle q, p_1, q_1\rangle\gamma_1} \langle q_1, p_2, q_2\rangle\gamma_2$ for all $q, q_1, q_2 \in Q$
- Can we reach configuration where all control states have form

$$(\langle q, p, q'\rangle w, X) \text{ such that } (q, p, q') \in \delta$$

Regular reachability

- yield(c) - Control-states in post-order traversal

$$\text{yield}((pw, X)\langle t_1 \dots t_n\rangle) = \text{yield}(t_1) \dots \text{yield}(t_n)p$$

- Let $A = (Q, \delta, q_0, F)$
- New control states: $\langle q, p, q' \rangle$
  - Idea: Process evaluates to tree t with $q \xrightarrow{\text{yield}(t)}_\delta q'$
- Start with initial guess: $p_0 \xrightarrow{\tau} \langle q_0, p_0, q' \rangle$ for $q' \in F$
- Non-spawn rules preserve guess
  - $pw \xrightarrow{\{\tau,\text{acq},\text{rel},\text{join}\}} p'w'$ becomes $\langle q, p, q' \rangle w \xrightarrow{\{\tau,\text{acq},\text{rel},\text{join}\}} \langle q, p', q' \rangle w'$ for all $q, q' \in Q$
- Spawn-rules distribute guess
  - $p\gamma \xrightarrow{p_1\gamma_1} p_2\gamma_2$ becomes $\langle q, p, q_2 \rangle \gamma \xrightarrow{\langle q,p_1,q_1\rangle\gamma_1} \langle q_1, p_2, q_2 \rangle \gamma_2$ for all $q, q_1, q_2 \in Q$
- Can we reach configuration where all control states have form

$$(\langle q, p, q' \rangle w, X) \text{ such that } (q, p, q') \in \delta$$

Regular reachability

- yield(c) - Control-states in post-order traversal

$$\text{yield}((pw, X)\langle t_1 \dots t_n\rangle) = \text{yield}(t_1)\dots\text{yield}(t_n)p$$

- Let $A = (Q, \delta, q_0, F)$
- New control states: $\langle q, p, q'\rangle$
  - Idea: Process evaluates to tree t with $q \xrightarrow{\text{yield}(t)}_\delta q'$
- Start with initial guess: $p_0 \xrightarrow{\tau} \langle q_0, p_0, q'\rangle$ for $q' \in F$
- Non-spawn rules preserve guess
  - $pw \xrightarrow{\{\tau,\text{acq},\text{rel},\text{join}\}} p'w'$ becomes $\langle q, p, q'\rangle w \xrightarrow{\{\tau,\text{acq},\text{rel},\text{join}\}} \langle q, p', q'\rangle w'$ for all $q, q' \in Q$
- Spawn-rules distribute guess
  - $p\gamma \xrightarrow{p_1\gamma_1} p_2\gamma_2$ becomes $\langle q, p, q_2\rangle\gamma \xrightarrow{\langle q,p_1,q_1\rangle\gamma_1} \langle q_1, p_2, q_2\rangle\gamma_2$ for all $q, q_1, q_2 \in Q$
- Can we reach configuration where all control states have form

$$(\langle q, p, q'\rangle w, X) \text{ such that } (q, p, q') \in \delta$$

Regular reachability

- yield(c) - Control-states in post-order traversal

$$\text{yield}((pw, X)\langle t_1 \dots t_n \rangle) = \text{yield}(t_1) \dots \text{yield}(t_n) p$$

- Let $A = (Q, \delta, q_0, F)$
- New control states: $\langle q, p, q' \rangle$
  - Idea: Process evaluates to tree t with $q \xrightarrow{\text{yield}(t)}_{\delta} q'$
- Start with initial guess: $p_0 \xrightarrow{\tau} \langle q_0, p_0, q' \rangle$ for $q' \in F$
- Non-spawn rules preserve guess
  - $pw \xrightarrow{\{\tau,\text{acq},\text{rel},\text{join}\}} p'w'$ becomes $\langle q, p, q' \rangle w \xrightarrow{\{\tau,\text{acq},\text{rel},\text{join}\}} \langle q, p', q' \rangle w'$ for all $q, q' \in Q$
- Spawn-rules distribute guess
  - $p\gamma \xrightarrow{p_1\gamma_1} p_2\gamma_2$ becomes $\langle q, p, q_2 \rangle \gamma \xrightarrow{\langle q,p_1,q_1 \rangle \gamma_1} \langle q_1, p_2, q_2 \rangle \gamma_2$ for all $q, q_1, q_2 \in Q$
- Can we reach configuration where all control states have form

$$(\langle q, p, q' \rangle w, X) \text{ such that } (q, p, q') \in \delta$$

Zoo of Results

	n-state			all-state/regular	
	2-proc.	n-proc.	dynamic	dynamic	join
nested	[KIG05]		[LMO08]	[LMOW09]	[GLMO$^+$11]
bounded lockchains	[Kah09]				
contextual	[CMV12]				
arbitrary	[KIG05]				

	n-state			all-state/regular	
	2-proc.	n-proc.	dynamic	dynamic	join
nested	=NP				$\geq$PSPACE
bounded lockchains	decidable				
contextual	= P/NP				
arbitrary	undecidable				

Zoo of Results

<table>
<tr><td></td><td colspan="3">n-state</td><td colspan="2">all-state/regular</td></tr>
<tr><td></td><td>2-proc.</td><td>n-proc.</td><td>dynamic</td><td>dynamic</td><td>join</td></tr>
<tr><td>nested</td><td colspan="2">[KIG05]</td><td>[LMO08]</td><td>[LMOW09]</td><td>[GLMO+11]</td></tr>
<tr><td>bounded lockchains</td><td colspan="2">[Kah09]</td><td colspan="3"></td></tr>
<tr><td>contextual</td><td>[CMV12]</td><td colspan="4">[LMOSW13]</td></tr>
<tr><td>arbitrary</td><td colspan="5">[KIG05]</td></tr>
</table>

<table>
<tr><td></td><td colspan="3">n-state</td><td colspan="2">all-state/regular</td></tr>
<tr><td></td><td>2-proc.</td><td>n-proc.</td><td>dynamic</td><td>dynamic</td><td>join</td></tr>
<tr><td>nested</td><td colspan="4">=NP</td><td>≥PSPACE</td></tr>
<tr><td>bounded lockchains</td><td colspan="2">decidable</td><td colspan="3"></td></tr>
<tr><td>contextual</td><td>= P/NP</td><td colspan="2">=PSPACE</td><td colspan="2">≥PSPACE</td></tr>
<tr><td>arbitrary</td><td colspan="5">undecidable</td></tr>
</table>

Dynamic Pushdown Networks

- DPN = Pushdown automata + Dynamic thread creation
- Configurations: Trees
  - Nodes: Process configuration (control, locks, stack)
  - Edges: Genealogy of process creation

$$\begin{array}{ccc} & (p_1, L_1)\, w_1 & \\ \swarrow & & \searrow \\ (p_2, L_2)\, w_2 & & (p_3, L_3)\, w_3 \\ & & \mid \\ & & (p_4, L_4)\, w_4 \end{array}$$

- Rules applied to nodes

$$\begin{aligned} (\mathit{base}) \quad & p\,\gamma \xrightarrow{a} p'\,\gamma' \quad \text{where } a \in \{\tau, \mathrm{acq}\ l, \mathrm{rel}\ l, \mathrm{join}\} \\ (\mathit{call}) \quad & p\,\gamma \xrightarrow{\tau} p'\,\gamma'\gamma'' \\ (\mathit{return}) \quad & p\,\gamma \xrightarrow{\tau} p'\,\varepsilon \\ (\mathit{spawn}) \quad & p\,\gamma \xrightarrow{(p'',\gamma'')} p'\,\gamma' \end{aligned}$$

Dynamic Pushdown Networks

- DPN = Pushdown automata + Dynamic thread creation
- Configurations: Trees
  - Nodes: Process configuration (control, locks, stack)
  - Edges: Genealogy of process creation

$$
\begin{array}{ccc}
 & (p_1, L_1)\, w_1 & \\
 \diagup & & \diagdown \\
(p_2, L_2)\, w_2 & & (p_3, L_3)\, w_3 \\
 & & | \\
 & & (p_4, L_4)\, w_4
\end{array}
$$

- Rules applied to nodes

$$
\begin{array}{rll}
(\mathit{base}) & p\,\gamma \xrightarrow{a} p'\,\gamma' & \text{where } a \in \{\tau, \mathrm{acq}\ l, \mathrm{rel}\ l, \mathrm{join}\} \\
(\mathit{call}) & p\,\gamma \xrightarrow{\tau} p'\,\gamma'\gamma'' & \\
(\mathit{return}) & p\,\gamma \xrightarrow{\tau} p'\,\varepsilon & \\
(\mathit{spawn}) & p\,\gamma \xrightarrow{(p'',\gamma'')} p'\,\gamma' &
\end{array}
$$

Dynamic Pushdown Networks

- DPN = Pushdown automata + Dynamic thread creation
- Configurations: Trees
  - Nodes: Process configuration (control, locks, stack)
  - Edges: Genealogy of process creation

$$\begin{tikzcd}[column sep=small]
 & (p_1, L_1)\, w_1 \arrow[dl, no head] \arrow[dr, no head] & \\
(p_2, L_2)\, w_2 & & (p_3, L_3)\, w_3 \arrow[d, no head] \\
 & & (p_4, L_4)\, w_4
\end{tikzcd}$$

- Rules applied to nodes

$$\begin{array}{rl}
(\mathit{base}) & p\,\gamma \xrightarrow{a} p'\,\gamma' \quad \text{where } a \in \{\tau, \mathrm{acq}\; l, \mathrm{rel}\; l, \mathrm{join}\} \\
(\mathit{call}) & p\,\gamma \xrightarrow{\tau} p'\,\gamma'\gamma'' \\
(\mathit{return}) & p\,\gamma \xrightarrow{\tau} p'\,\varepsilon \\
(\mathit{spawn}) & p\,\gamma \xrightarrow{(p'', \gamma'')} p'\,\gamma'
\end{array}$$

Lower Bound

n-state reachability for 3 processes with contextual locking is PSPACE-hard, even for a constant number of locks.

- Idea: Emulate linearly bounded Turing machine
  - Regard run $c_0\overline{c_1}c_1\overline{c_2}c_2\dots\overline{c_n}$
  - Important: Configuration has constant length n
  - Process 1: Check that $c_i\overline{c_{i+1}}$ is valid step
  - Process 2: Check that c_i is reverse of $\overline{c_i}$
- Problem: How to synchronize the processes

Lower Bound

n-state reachability for 3 processes with contextual locking is PSPACE-hard, even for a constant number of locks.

- Idea: Emulate linearly bounded Turing machine
  - Regard run $c_0 \overline{c_1} c_1 \overline{c_2} c_2 \ldots \overline{c_n}$
  - Important: Configuration has constant length n
  - Process 1: Check that $c_i \overline{c_{i+1}}$ is valid step
  - Process 2: Check that c_i is reverse of $\overline{c_i}$
- Problem: How to synchronize the processes

Lower Bound

n-state reachability for 3 processes with contextual locking is PSPACE-hard, even for a constant number of locks.

- Idea: Emulate linearly bounded Turing machine
  - Regard run $c_0\overline{c_1}c_1\overline{c_2}c_2\ldots\overline{c_n}$
  - Important: Configuration has constant length n
  - Process 1: Check that $c_i\overline{c_{i+1}}$ is valid step
  - Process 2: Check that c_i is reverse of $\overline{c_i}$
- Problem: How to synchronize the processes

Lower Bound

n-state reachability for 3 processes with contextual locking is PSPACE-hard, even for a constant number of locks.

- Idea: Emulate linearly bounded Turing machine
  - Regard run $c_0 \overline{c_1} c_1 \overline{c_2} c_2 \dots \overline{c_n}$
  - Important: Configuration has constant length n
  - Process 1: Check that $c_i \overline{c_{i+1}}$ is valid step
  - Process 2: Check that c_i is reverse of $\overline{c_i}$
- Problem: How to synchronize the processes

Lower Bound

n-state reachability for 3 processes with contextual locking is PSPACE-hard, even for a constant number of locks.

- Idea: Emulate linearly bounded Turing machine
  - Regard run $c_0\overline{c_1}c_1\overline{c_2}c_2\ldots\overline{c_n}$
  - Important: Configuration has constant length n
  - Process 1: Check that $c_i\overline{c_{i+1}}$ is valid step
  - Process 2: Check that c_i is reverse of $\overline{c_i}$
- Problem: How to synchronize the processes

Lower Bound

n-state reachability for 3 processes with contextual locking is PSPACE-hard, even for a constant number of locks.

- Idea: Emulate linearly bounded Turing machine
  - Regard run $c_0 \overline{c_1} c_1 \overline{c_2} c_2 \ldots \overline{c_n}$
  - Important: Configuration has constant length n
  - Process 1: Check that $c_i \overline{c_{i+1}}$ is valid step
  - Process 2: Check that c_i is reverse of $\overline{c_i}$
- Problem: How to synchronize the processes

Lower Bound

n-state reachability for 3 processes with contextual locking is PSPACE-hard, even for a constant number of locks.

- Idea: Emulate linearly bounded Turing machine
  - Regard run $c_0\overline{c_1}c_1\overline{c_2}c_2\dots\overline{c_n}$
  - Important: Configuration has constant length n
  - Process 1: Check that $c_i\overline{c_{i+1}}$ is valid step
  - Process 2: Check that c_i is reverse of $\overline{c_i}$
- Problem: How to synchronize the processes

Handshake-Synchronization via locks

Let use(l) = acq(l)rel(l) and free(l) = rel(l)acq(l)

Process 1				
Process 2				
Controller				

- Controller frees l_1 to signal begin of handshake
- Another lock avoids multiple iterations
- Reverse pattern avoids not entering the synchronization
    - Problem: Only possible if process in initial context
    - Solution: Only check after constantly many handshakes
    - Processes 1 and 2 return to initial context after $2n$ handshakes

Handshake-Synchronization via locks

Let use(l) = acq(l)rel(l) and free(l) = rel(l)acq(l)

Process 1	use(l_1)			
Process 2	use(l_1)			
Controller	free(l_1)			

- Controller frees l_1 to signal begin of handshake
- Another lock avoids multiple iterations
- Reverse pattern avoids not entering the synchronization
  - Problem: Only possible if process in initial context
  - Solution: Only check after constantly many handshakes
  - Processes 1 and 2 return to initial context after $2n$ handshakes

Handshake-Synchronization via locks

Let use(l) = acq(l)rel(l) and free(l) = rel(l)acq(l)

Process 1	use(l_1)	use(l_2)	
Process 2	use(l_1)	use(l_2)	
Controller	free(l_1)	free(l_2)	

- Controller frees l_1 to signal begin of handshake
- Another lock avoids multiple iterations
- Reverse pattern avoids not entering the synchronization
  - Problem: Only possible if process in initial context
  - Solution: Only check after constantly many handshakes
  - Processes 1 and 2 return to initial context after $2n$ handshakes

Handshake-Synchronization via locks

Let use(l) = acq(l)rel(l) and free(l) = rel(l)acq(l)

Process 1	use(l_1)	use(l_2)	free(l_3)free(l_4)
Process 2	use(l_1)	use(l_2)	free(l_5)free(l_6)
Controller	free(l_1)	free(l_2)	use(l_3)use(l_4) use(l_5)use(l_6)

- Controller frees l_1 to signal begin of handshake
- Another lock avoids multiple iterations
- Reverse pattern avoids not entering the synchronization
  - Problem: Only possible if process in initial context
  - Solution: Only check after constantly many handshakes
  - Processes 1 and 2 return to initial context after $2n$ handshakes

Handshake-Synchronization via locks

Let use(l) = acq(l)rel(l) and free(l) = rel(l)acq(l)

Process 1	use(l_1)	use(l_2)	free(l_3)free(l_4)
Process 2	use(l_1)	use(l_2)	free(l_5)free(l_6)
Controller	free(l_1)	free(l_2)	use(l_3)use(l_4) use(l_5)use(l_6)

- Controller frees l_1 to signal begin of handshake
- Another lock avoids multiple iterations
- Reverse pattern avoids not entering the synchronization
  - Problem: Only possible if process in initial context
  - Solution: Only check after constantly many handshakes
  - Processes 1 and 2 return to initial context after $2n$ handshakes

Handshake-Synchronization via locks

Let use(l) = acq(l)rel(l) and free(l) = rel(l)acq(l)

Process 1	use(l_1)	use(l_2)	free(l_3)free(l_4)
Process 2	use(l_1)	use(l_2)	free(l_5)free(l_6)
Controller	free(l_1)	free(l_2)	use(l_3)use(l_4) use(l_5)use(l_6)

- Controller frees l_1 to signal begin of handshake
- Another lock avoids multiple iterations
- Reverse pattern avoids not entering the synchronization
  - Problem: Only possible if process in initial context
  - Solution: Only check after constantly many handshakes
  - Processes 1 and 2 return to initial context after $2n$ handshakes

Handshake-Synchronization via locks

Let use(l) = acq(l)rel(l) and free(l) = rel(l)acq(l)

Process 1	use(l_1)	use(l_2)	free(l_3)free(l_4)
Process 2	use(l_1)	use(l_2)	free(l_5)free(l_6)
Controller	free(l_1)	free(l_2)	use(l_3)use(l_4) use(l_5)use(l_6)

- Controller frees l_1 to signal begin of handshake
- Another lock avoids multiple iterations
- Reverse pattern avoids not entering the synchronization
    - Problem: Only possible if process in initial context
    - Solution: Only check after constantly many handshakes
    - Processes 1 and 2 return to initial context after $2n$ handshakes

Upper Bound — n-state reachability

n-state reachability for DPN with contextual locking is in PSPACE.

- Sufficient to consider executions with polynomially bounded stack
  - Inline non-returning calls
  - Contract too deeply nested calls
    - If stack is deeper than $|P|^2|\Gamma|$, there are two nested calls that start and end in the same situation
    - Replace outer by inner one
- Extension to dynamic thread creation
  - Larger stack bound (must not drop important spawns)
  - Drop unimportant spawns
  - Contract too long chains of spawns (Similar to contracting calls)

Upper Bound — n-state reachability

n-state reachability for DPN with contextual locking is in PSPACE.

- Sufficient to consider executions with polynomially bounded stack
  - Inline non-returning calls
  - Contract too deeply nested calls
    - If stack is deeper than $|P|^2|\Gamma|$, there are two nested calls that start and end in the same situation
    - Replace outer by inner one
- Extension to dynamic thread creation
  - Larger stack bound (must not drop important spawns)
  - Drop unimportant spawns
  - Contract too long chains of spawns (Similar to contracting calls)

Upper Bound — n-state reachability

n-state reachability for DPN with contextual locking is in PSPACE.

- Sufficient to consider executions with polynomially bounded stack
  - Inline non-returning calls
  - Contract too deeply nested calls
    - If stack is deeper than $|P|^2|\Gamma|$, there are two nested calls that start and end in the same situation
    - Replace outer by inner one
- Extension to dynamic thread creation
  - Larger stack bound (must not drop important spawns)
  - Drop unimportant spawns
  - Contract too long chains of spawns (Similar to contracting calls)

Upper Bound — n-state reachability

n-state reachability for DPN with contextual locking is in PSPACE.

- Sufficient to consider executions with polynomially bounded stack
  - Inline non-returning calls
  - Contract too deeply nested calls
    - If stack is deeper than $|P|^2|\Gamma|$, there are two nested calls that start and end in the same situation
    - Replace outer by inner one
- Extension to dynamic thread creation
  - Larger stack bound (must not drop important spawns)
  - Drop unimportant spawns
  - Contract too long chains of spawns (Similar to contracting calls)

Upper Bound — n-state reachability

n-state reachability for DPN with contextual locking is in PSPACE.

- Sufficient to consider executions with polynomially bounded stack
  - Inline non-returning calls
  - Contract too deeply nested calls
    - If stack is deeper than $|P|^2|\Gamma|$, there are two nested calls that start and end in the same situation
    - Replace outer by inner one
- Extension to dynamic thread creation
  - Larger stack bound (must not drop important spawns)
  - Drop unimportant spawns
  - Contract too long chains of spawns (Similar to contracting calls)

Upper Bound — n-state reachability

n-state reachability for DPN with contextual locking is in PSPACE.

- Sufficient to consider executions with polynomially bounded stack
  - Inline non-returning calls
  - Contract too deeply nested calls
    - If stack is deeper than $|P|^2|\Gamma|$, there are two nested calls that start and end in the same situation
    - Replace outer by inner one
- Extension to dynamic thread creation
  - Larger stack bound (must not drop important spawns)
  - Drop unimportant spawns
  - Contract too long chains of spawns (Similar to contracting calls)

Upper Bound — n-state reachability

n-state reachability for DPN with contextual locking is in PSPACE.

- Sufficient to consider executions with polynomially bounded stack
  - Inline non-returning calls
  - Contract too deeply nested calls
    - If stack is deeper than $|P|^2|\Gamma|$, there are two nested calls that start and end in the same situation
    - Replace outer by inner one
- Extension to dynamic thread creation
  - Larger stack bound (must not drop important spawns)
  - Drop unimportant spawns
  - Contract too long chains of spawns (Similar to contracting calls)

Upper Bound — n-state reachability

n-state reachability for DPN with contextual locking is in PSPACE.

- Sufficient to consider executions with polynomially bounded stack
  - Inline non-returning calls
  - Contract too deeply nested calls
    - If stack is deeper than $|P|^2|\Gamma|$, there are two nested calls that start and end in the same situation
    - Replace outer by inner one
- Extension to dynamic thread creation
  - Larger stack bound (must not drop important spawns)
  - Drop unimportant spawns
  - Contract too long chains of spawns (Similar to contracting calls)

Upper Bound — n-state reachability

n-state reachability for DPN with contextual locking is in PSPACE.

- Sufficient to consider executions with polynomially bounded stack
  - Inline non-returning calls
  - Contract too deeply nested calls
    - If stack is deeper than $|P|^2|\Gamma|$, there are two nested calls that start and end in the same situation
    - Replace outer by inner one
- Extension to dynamic thread creation
  - Larger stack bound (must not drop important spawns)
  - Drop unimportant spawns
  - Contract too long chains of spawns (Similar to contracting calls)

Upper Bound — all-state reachability

All-state reachability for DPN with contextual locking and join is decidable.

- Also bound stack
- Let $\preceq$ be homeomorphic embedding on configurations
  - $\preceq$ is a well-quasic ordering
  - All-state reachability is downward closed: $d \preceq c \wedge A_p(c) \implies A_p(d)$
    - $A_p(c)$ — All processes in c are at p
  - System (w/o join) is downward compatible:
    $d \preceq c \wedge c \rightarrow^* c' \implies \exists d' \preceq c'.\ d \rightarrow^* d'$
- Well structured transition system
  - Decide by saturating set of minimal reachable states

Upper Bound — all-state reachability

All-state reachability for DPN with contextual locking and join is decidable.

- Also bound stack
- Let $\preceq$ be homeomorphic embedding on configurations
  - $\preceq$ is a well-quasic ordering
  - All-state reachability is downward closed: $d \preceq c \wedge A_p(c) \implies A_p(d)$
    - $A_p(c)$ — All processes in c are at p
  - System (w/o join) is downward compatible: $d \preceq c \wedge c \rightarrow^* c' \implies \exists d' \preceq c'.\ d \rightarrow^* d'$
- Well structured transition system
  - Decide by saturating set of minimal reachable states

Upper Bound — all-state reachability

All-state reachability for DPN with contextual locking and join is decidable.

- Also bound stack
- Let $\preceq$ be homeomorphic embedding on configurations
  - $\preceq$ is a well-quasic ordering
  - All-state reachability is downward closed: $d \preceq c \wedge A_p(c) \implies A_p(d)$
    - $A_p(c)$ — All processes in c are at p
  - System (w/o join) is downward compatible: $d \preceq c \wedge c \to^* c' \implies \exists d' \preceq c'.\, d \to^* d'$
- Well structured transition system
  - Decide by saturating set of minimal reachable states

Upper Bound — all-state reachability

All-state reachability for DPN with contextual locking and join is decidable.

- Also bound stack
- Let $\preceq$ be homeomorphic embedding on configurations
  - $\preceq$ is a well-quasic ordering
  - All-state reachability is downward closed: $d \preceq c \wedge A_p(c) \implies A_p(d)$
    - $A_p(c)$ — All processes in c are at p
  - System (w/o join) is downward compatible: $d \preceq c \wedge c \rightarrow^* c' \implies \exists d' \preceq c'.\, d \rightarrow^* d'$
- Well structured transition system
  - Decide by saturating set of minimal reachable states

Upper Bound — all-state reachability

All-state reachability for DPN with contextual locking and join is decidable.

- Also bound stack
- Let $\preceq$ be homeomorphic embedding on configurations
  - $\preceq$ is a well-quasic ordering
  - All-state reachability is downward closed: $d \preceq c \wedge A_p(c) \implies A_p(d)$
    - $A_p(c)$ — All processes in c are at p
  - System (w/o join) is downward compatible: $d \preceq c \wedge c \rightarrow^* c' \implies \exists d' \preceq c'.\, d \rightarrow^* d'$
- Well structured transition system
  - Decide by saturating set of minimal reachable states

Upper Bound — all-state reachability

All-state reachability for DPN with contextual locking and join is decidable.

- Also bound stack
- Let $\preceq$ be homeomorphic embedding on configurations
  - $\preceq$ is a well-quasic ordering
  - All-state reachability is downward closed: $d \preceq c \wedge A_p(c) \implies A_p(d)$
    - $A_p(c)$ — All processes in c are at p
  - System (w/o join) is downward compatible: $d \preceq c \wedge c \to^* c' \implies \exists d' \preceq c'.\ d \to^* d'$
- Well structured transition system
  - Decide by saturating set of minimal reachable states

Upper Bound — all-state reachability

All-state reachability for DPN with contextual locking and join is decidable.

- Also bound stack
- Let $\preceq$ be homeomorphic embedding on configurations
  - $\preceq$ is a well-quasic ordering
  - All-state reachability is downward closed: $d \preceq c \wedge A_p(c) \implies A_p(d)$
    - $A_p(c)$ — All processes in c are at p
  - System (w/o join) is downward compatible: $d \preceq c \wedge c \rightarrow^* c' \implies \exists d' \preceq c'.\ d \rightarrow^* d'$
- Well structured transition system
  - Decide by saturating set of minimal reachable states

Upper Bound — all-state reachability

All-state reachability for DPN with contextual locking and join is decidable.

- Also bound stack
- Let $\preceq$ be homeomorphic embedding on configurations
  - $\preceq$ is a well-quasic ordering
  - All-state reachability is downward closed: $d \preceq c \wedge A_p(c) \implies A_p(d)$
    - $A_p(c)$ — All processes in c are at p
  - System (w/o join) is downward compatible:
    $d \preceq c \wedge c \rightarrow^* c' \implies \exists d' \preceq c'.\, d \rightarrow^* d'$
- Well structured transition system
  - Decide by saturating set of minimal reachable states

Upper Bound — join

- Problem: Joins violate downward compatibility!
  - Children may have terminated, while grand-children still running
- Solution
  - Configurations are multisets of trees
  - On spawn, guess if process will be joined later
    - joined: Add as new child
    - not joined: Add as new root
  - Now: On join, all successors have terminated

Upper Bound — join

- Problem: Joins violate downward compatibility!
    - Children may have terminated, while grand-children still running
- Solution
    - Configurations are multisets of trees
    - On spawn, guess if process will be joined later
        - joined: Add as new child
        - not joined: Add as new root
    - Now: On join, all successors have terminated

Upper Bound — join

- Problem: Joins violate downward compatibility!
  - Children may have terminated, while grand-children still running
- Solution
  - Configurations are multisets of trees
  - On spawn, guess if process will be joined later
    - joined: Add as new child
    - not joined: Add as new root
  - Now: On join, all successors have terminated

Upper Bound — join

- Problem: Joins violate downward compatibility!
  - Children may have terminated, while grand-children still running
- Solution
  - Configurations are multisets of trees
  - On spawn, guess if process will be joined later
    - joined: Add as new child
    - not joined: Add as new root
  - Now: On join, all successors have terminated

Upper Bound — join

- Problem: Joins violate downward compatibility!
  - Children may have terminated, while grand-children still running
- Solution
  - Configurations are multisets of trees
  - On spawn, guess if process will be joined later
    - joined: Add as new child
    - not joined: Add as new root
  - Now: On join, all successors have terminated

Upper Bound — join

- Problem: Joins violate downward compatibility!
  - Children may have terminated, while grand-children still running
- Solution
  - Configurations are multisets of trees
  - On spawn, guess if process will be joined later
    - joined: Add as new child
    - not joined: Add as new root
  - Now: On join, all successors have terminated

Upper Bound — join

- Problem: Joins violate downward compatibility!
  - Children may have terminated, while grand-children still running
- Solution
  - Configurations are multisets of trees
  - On spawn, guess if process will be joined later
    - joined: Add as new child
    - not joined: Add as new root
  - Now: On join, all successors have terminated

Upper Bound — join

- Problem: Joins violate downward compatibility!
  - Children may have terminated, while grand-children still running
- Solution
  - Configurations are multisets of trees
  - On spawn, guess if process will be joined later
    - joined: Add as new child
    - not joined: Add as new root
  - Now: On join, all successors have terminated

Conclusion

- Regular reachability for DPN + contextual locking + join is decidable
- PSPACE-hard even for 3 processes and constant number of locks
- Important sub-classes are PSPACE-complete

References

Rohit Chadha, P. Madhusudan, and Mahesh Viswanathan.
Reachability under contextual locking.
In *TACAS*, volume 7214 of *LNCS*, pages 437–450. Springer, 2012.

Thomas Martin Gawlitza, Peter Lammich, Markus Müller-Olm, Helmut Seidl, and Alexander Wenner.
Join-lock-sensitive forward reachability analysis for concurrent programs with dynamic process creation.
In *VMCAI*, volume 6538 of *LNCS*, pages 199–213. Springer, 2011.

Vineet Kahlon.
Boundedness vs. unboundedness of lock chains: Characterizing decidability of pairwise cfl-reachability for threads communicating via locks.
In *LICS*, pages 27–36. IEEE Computer Society, 2009.

Vineet Kahlon, Franjo Ivancic, and Aarti Gupta.
Reasoning about threads communicating via locks.
In *CAV*, volume 3576 of *LNCS*, pages 505–518. Springer, 2005.

Peter Lammich and Markus Müller-Olm.
Conflict analysis of programs with procedures, dynamic thread creation, and monitors.
In *SAS*, volume 5079 of *LNCS*, pages 205–220. Springer, 2008.

Peter Lammich, Markus Müller-Olm, Helmut Seidl, and Alexander Wenner.
Contextual locking for dynamic pushdown networks.
In *SAS*, volume 7935 of *LNCS*. Springer, 2013.

Peter Lammich, Markus Müller-Olm, and Alexander Wenner.
Predecessor sets of dynamic pushdown networks with tree-regular constraints.
In *CAV*, volume 5643 of *LNCS*, pages 525–539. Springer, 2009.