Systems with parametric thread creation

Joint work with: M. Fortin, S. LaTorre, A. Muscholl, H. Seidl
Recursive programs with thread creation:

- Recursion (even at higher-orders)
- global variables
- local variables shared between a parent and its children
- spawn of new subprocesses
Programs with:

- Recursion (even at higher-orders)
- Global variables
- Local variables shared between a parent and its children
- Spawn of new subprocesses
Programs with:

- Recursion (even at higher-orders)
- Global variables
- Local variables shared between a parent and its children
- Spawn of new subprocesses

finite domain

parametric
Reachability problem:
can # appear in the register?

- Global variables: write(#)  
- Local variables: x  
  accessible by children: parent.x  
- spawn

```
root() {
    spawn(p);
    switch (x) {
        case 2: write(#);
    }
}

p() {
    switch (parent.x) {
        case 0 : spawn(p);
            if (*) parent.x = 1
            else switch (x) {
                case 1 : parent.x = 1; break;
                case 2 : break;
            }; break;
        case 1 : spawn(p);
            if (*) parent.x = 0
            else switch (x) {
                case 1: parent.x = 2; break;
                case 2: parent.x = 2; break;
            };
    }
}
```
We will define their semantics. The transition system is specified by a tuple \( (S, I, T, F) \) where:

- \( S \) is a (possibly infinite) set of states.
- \( I \) is a set of initial values.
- \( T \) is a set of labels.
- \( F \) is a finite set of input/output operations.

In this transition system the action names are uninterpreted. In Section 2.2 we present a formal definition of our parametric combination.

### 2.1 Transition systems

The states of the dynamic parametric process correspond to the lines in the program. The read/write operations of the children are denoted as input/output operations. Here is one possible execution.

Example 1.

Consider the program from Figure 1. The conditional \( \text{if} \) at line 19, which is the local variable of the root. Note that at least two children are required to write 2 or 3 into the local variable of the parent.

Example 2.

More interesting examples require more program states. Here, it is convenient to take the second branch of the nondeterministic choice and change the value of the root's local variable to 3. Only then, the root will write value 5 into the global variable.

A dynamic parametric process can, e.g., be defined by an explicitly given finite transition system:

\[
(S, I, T, F) = \left( \{ s_1, s_2, s_3 \}, \{ i \}, \{ \text{read}, \text{write} \}, \{ \text{add} \} \right)
\]

The root starts in state \( s_1 \), the latter can now spawn two sub-processes in state \( s_2 \). The root takes the same designated initial state. Making the spawn operation parametric is the main abstraction step that allows us to obtain decidability results.

In this example, the root starts in state \( s_1 \) and waits for some child to change the value of the root's local variable to 1. Only then, the root will write value 3 into the global variable. The states of the dynamic parametric process correspond to the lines in the program. The read/write operations of the children are denoted as input/output operations.

Example 3.

Once the variable \( x \) of some descendant is set to 2, it can take the second branch of the nondeterministic choice and change the value of the root's local variable to 3. Only then, the root will write value 5 into the global variable.

### 2.2 Parametric combination

Cooperating sub-processes are denoted as input/output operations. Here is one possible execution.

#### Example 2.

More interesting examples require more program states. Here, it is convenient to take the second branch of the nondeterministic choice and change the value of the root's local variable to 3. Only then, the root will write value 5 into the global variable.

A dynamic parametric process can, e.g., be defined by an explicitly given finite transition system:

\[
(S, I, T, F) = \left( \{ s_1, s_2, s_3 \}, \{ i \}, \{ \text{read}, \text{write} \}, \{ \text{add} \} \right)
\]

The root starts in state \( s_1 \), the latter can now spawn two sub-processes in state \( s_2 \). The root takes the same designated initial state. Making the spawn operation parametric is the main abstraction step that allows us to obtain decidability results.

In this example, the root starts in state \( s_1 \) and waits for some child to change the value of the root's local variable to 1. Only then, the root will write value 3 into the global variable.
When processes are finite state:

[Reasoning about systems with many processes, German, Sisla J ACM 92], or


Two pushdowns with a register $\rightarrow$ undecidability

Even two copies of the same pushdown.
Unspecified number of copies of a pushdown with one register: decidable. [Khalon]

Compute sets of values $V_1, V_2, \ldots$:

$V_1$: values that can be produced with $v_{init}$ in the register.
$V_2$: values that can be produced assuming any of $V_1$ can occur “magically” in the register.
Pushdown parametric processes

Decidable

spawn
Model

Hague, 2011

Esparza, Ganty, Majumdar, 2013
Durand-Gasselin, Esparza, Ganty, Majumdar, 2015
Register: reads/writes, no locks

Model

Hague, 2011

Esparza, Ganty, Majumdar, 2013
Durand-Gasselin, Esparza, Ganty, Majumdar, 2015
Semantics

\[ C = \langle S, \delta \subseteq S \times \Sigma_C \times S, s_{\text{init}} \rangle \quad D = \langle T, \Delta \subseteq T \times \Sigma_D \times T, t_{\text{init}} \rangle. \]

\( S \) and \( T \) may be infinite, for example: \( S = Q \times \Gamma^* \)

\( G \): a finite set of register values

A configuration is \((M, t, g)\), where \( M \in \mathbb{N}^S \), \( t \in T \), \( g \in G \).

\[
\begin{align*}
(M, t, g) \xrightarrow{w(h)} (M, t', h) & \quad \text{if } t \xrightarrow{w(h)} t' \text{ in } \Delta, \\
(M, t, g) \xrightarrow{r(h)} (M, t', h) & \quad \text{if } t \xrightarrow{r(h)} t' \text{ in } \Delta \text{ and } h = g, \\
(M, t, g) \xrightarrow{\bar{w}(h)} (M', t, h) & \quad \text{if } M \xrightarrow{\bar{w}(h)} M' \text{ in } \delta, \\
(M, t, g) \xrightarrow{\bar{r}(h)} (M', t, h) & \quad \text{if } M \xrightarrow{\bar{r}(h)} M' \text{ in } \delta \text{ and } h = g.
\end{align*}
\]

where

\[ M \xrightarrow{a} M' \text{ in } \delta \quad \text{if } s \xrightarrow{a} s' \text{ in } \delta \text{ and } M' = M - [s] + [s'], \text{ for some } s, s' \in S. \]
(M, t, g) \xrightarrow{w(h)} (M, t', h) \quad \text{if } t \xrightarrow{w(h)} t' \text{ in } \Delta ,
(M, t, g) \xrightarrow{r(h)} (M, t', h) \quad \text{if } t \xrightarrow{r(h)} t' \text{ in } \Delta \text{ and } h = g ,
(M, t, g) \xrightarrow{\bar{w}(h)} (M', t, h) \quad \text{if } M \xrightarrow{\bar{w}(h)} M' \text{ in } \delta ,
(M, t, g) \xrightarrow{\bar{r}(h)} (M', t, h) \quad \text{if } M \xrightarrow{\bar{r}(h)} M' \text{ in } \delta \text{ and } h = g .

where

\[ M \xrightarrow{a} M' \text{ in } \delta \quad \text{if } s \xrightarrow{a} s' \text{ in } \delta \text{ and } M' = M - [s] + [s'] , \text{ for some } s, s' \in S.\]
Set semantics

\[(B, t, g) \xrightarrow{w(h)} (B, t', h) \quad \text{if } t \xrightarrow{w(h)} t' \text{ in } \Delta, \]
\[(B, t, g) \xrightarrow{r(h)} (B, t', h) \quad \text{if } t \xrightarrow{r(h)} t' \text{ in } \Delta \text{ and } h = g, \]
\[(B, t, g) \xrightarrow{\bar{w}(h)} (B', t, h) \quad \text{if } B \xrightarrow{\bar{w}(h)} B' \text{ in } \delta, \]
\[(B, t, g) \xrightarrow{\bar{r}(h)} (B', t, h) \quad \text{if } B \xrightarrow{\bar{r}(h)} B' \text{ in } \delta \text{ and } h = g. \]

\[B \xrightarrow{a} B' \text{ in } \delta \quad \text{if } s \xrightarrow{a} s' \text{ in } \delta \text{ and } B' = B \cup \{s'\}, \text{ for some } s, s' \in S. \]

\[(M, t, g) \xrightarrow{w(h)} (M, t', h) \quad \text{if } t \xrightarrow{w(h)} t' \text{ in } \Delta, \]
\[(M, t, g) \xrightarrow{r(h)} (M, t', h) \quad \text{if } t \xrightarrow{r(h)} t' \text{ in } \Delta \text{ and } h = g, \]
\[(M, t, g) \xrightarrow{\bar{w}(h)} (M', t, h) \quad \text{if } M \xrightarrow{\bar{w}(h)} M' \text{ in } \delta, \]
\[(M, t, g) \xrightarrow{\bar{r}(h)} (M', t, h) \quad \text{if } M \xrightarrow{\bar{r}(h)} M' \text{ in } \delta \text{ and } h = g. \]

\[M \xrightarrow{a} M' \text{ in } \delta \quad \text{if } s \xrightarrow{a} s' \text{ in } \delta \text{ and } M' = M - [s] + [s'], \text{ for some } s, s' \in S. \]
**Multiset**

\[
\begin{array}{cccc}
  t & \xrightarrow{a} & t' & t' \\
  g & \xrightarrow{g'} & g'' & g'''
\end{array}
\]

\[
\begin{array}{cccc}
  s_1 & \xrightarrow{b} & s_1' & s_1' \\
  \vdots & \vdots & \vdots & \vdots \\
  s_i & \xrightarrow{c} & s_i' & s_i' \\
  \vdots & \vdots & \vdots & \vdots \\
  s_n & \xrightarrow{} & s_n' & s_n'
\end{array}
\]

**Set**

\[
\begin{array}{cccc}
  t & \xrightarrow{a} & t' & t' \\
  g & \xrightarrow{g'} & g'' & g'''
\end{array}
\]

\[
\begin{array}{cccc}
  s_1 & \xrightarrow{b} & s_1' & s_1' \\
  \vdots & \vdots & \vdots & \vdots \\
  s_i & \xrightarrow{c} & s_i' & s_i' \\
  \vdots & \vdots & \vdots & \vdots \\
  s_n & \xrightarrow{} & s_n' & s_n'
\end{array}
\]
Every contributor proposes a value
Leader chooses one of these values
The rest of the protocol uses the chosen value

Example of a system:
- Every contributor proposes a value
- Leader chooses one of these values
- The rest of the protocol uses the chosen value

Example properties:
(for every n, for every run)
- Leader eventually decides on a value
- If the leader decides on the value, contributors use only this value.
- On runs where only one value is used i.o. the protocol is correct
Example of a system:

* Contributors proposes values.
* Leader chooses one of these values.
* The rest of the protocol uses the chosen value.

Example properties:
(for every n, for every run)

* Leader eventually decides on a value
* If the leader decides on the value, contributors use only this value.
* On runs where only one value is used i.o. the protocol is correct

There is a run where the leader has decided on some value and afterwards a contributor is using a different value.
Example of a system:

- Contributors proposes values.
- Leader chooses one of these values.
- The rest of the protocol uses the chosen value.

Example properties:
(for every $n$, for every run)

- Leader eventually decides on a value
- If the leader decides on the value, contributors use only this value.
- On runs where only one value is used i.o. the protocol is correct

There is a maximal run where the leader does not decide on a value.
Example of a system:

- Contributors proposes values.
- Leader chooses one of these values.
- The rest of the protocol uses the chosen value.

Example properties: (for every n, for every run)

- Leader eventually decides on a value
- If the leader decides on the value, contributors use only this value.
- On runs where only one value is used i.o. the protocol is correct

reachability

safe run

more general liveness property
Pushdown parametric processes

Decidable

reachability, PSPACE-compl

repeated reachability, PSPACE-compl
Reachability in pushdown parametric systems:

Given a pushdown parametric system, decide if # can be written into the register.

Repeated reachability in pushdown parametric systems:

...decide if # can be written into the register infinitely often.
Thm [Hague; Durand-Gasselin, Esparza, Ganty, Majumdar; us]

The reachability problem is decidable in PSPACE. The same for repeated reachability.
Pushdown parametric processes

Decidable
reachability, PSPACE-compl
repeated reachability, PSPACE-compl
safety, NEXPTIME-compl, maximal safe runs
Safety in pushdown parametric systems:
Given a pushdown parametric system, decide if it has a maximal run that does not write # into the register.

Thm:
The safety problem is NEXPTIME-complete.
Safety in pushdown parametric systems:
Given a pushdown parametric system, decide if it has a **maximal** run that **does not** write # into the register.

**Thm:**
Deciding if there is some **infinite** safe run is PSPACE-complete.
Deciding if there is some **maximal finite** safe run is NEXPTIME-complete.
Prop:
Deciding if there is some maximal finite safe run is NEXPTIME-hard.

Reduction of a tiling problem:
Find a tiling with letters from $\Sigma$ of a $2^n \times 2^n$ square.
The tiling should respect neighbourhood relations $H, V \subseteq \Sigma \times \Sigma$.

Leader writes: $A_{1,1}, \overline{A_{1,1}}, A_{1,2}, \overline{A_{1,2}}, \ldots, A_{1,2^n}, \overline{A_{1,2^n}}, \ldots, A_{2^n,2n}, \overline{A_{2^n,2n}} (\$$)_{2^n} \Diamond$.

and checks the horizontal dependencies.
Prop:
Deciding if there is some maximal finite safe run is NEXPTIME-hard.

Reduction of a tiling problem:
Find a tiling with letters from $\Sigma$ of a $2^n \times 2^n$ square.
The tiling should respect neighbourhood relations $H, V \subseteq \Sigma \times \Sigma$.

Leader writes: $A_{1,1}, \overline{A}_{1,1}, A_{1,2}, \overline{A}_{1,2}, \ldots, A_{1,2^n}, \overline{A}_{1,2^n}, \ldots, A_{2^n,2n}, \overline{A}_{2^n,2n}$ ($\$\$)^{2^n} \Diamond$.

and checks the horizontal dependencies.

Contributors check vertical dependencies.

We can ensure that contributors:
- read all the symbols, and
- every vertical dependency is checked by some contributor.
Decidable

reachability, PSPACE-compl

repeated reachability, PSPACE-compl

safety, NEXPTIME-compl, maximal safe runs

stutter-expanding properties, NEXPTIME
A trace is a sequence of register operations during a run. A maximal trace comes from a maximal run (finite or infinite).

A property of traces is \( P \subseteq (\Sigma_D \cup \Sigma_C)^\infty \).
A property is \textbf{C-stutter-expanding} if it is closed under duplicating actions of contributors.

\[
\text{If } x \bar{w}(g) y \in P \text{ then } x \bar{w}(g)\bar{w}(g) y \in P
\]

Verification for arbitrary regular properties is undecidable, as with a property we can require that there is only one copy of a contributor.
Verification of properties of pushdown parametric systems:
Given a pushdown parametric system and a regular C-stutter-expanding property P, decide if it has a **maximal trace** in P.

**Thm:**
Verification of properties given by Buchi automata is NEXPTIME-complete.
Pushdown parametric processes

Decidable
reachability, PSPACE-compl
repeated reachability, PSPACE-compl
safety, NEXPTIME-compl, maximal safe runs
stutter-expanding properties, NEXPTIME

spawn
Reachability problem:
can # appear in the register?

- Global variables: write(#)
- Local variables: x
  accessible by children: parent.x
- spawn
Pushdown dynamic parametric processes
Pushdown **dynamic** parametric processes

Each box is a pushdown process that can spawn other pushdown processes with an operation `spawn(q)`.
Pushdown parametric processes

- Decidable reachability, PSPACE-completeness
- Repeated reachability, PSPACE-completeness
- Safety, NEXPTIME-completeness, maximal safe runs
- Stutter-expanding properties, NEXPTIME

spawn

reachability decidable
External behaviour of subprocesses can be summarised by a language:

\[ Ext \subseteq \Sigma^*_{ext} \]

\[ \Sigma_{ext} = \{ i(x, v), o(x, v), r(g, v), w(g, v) : x \in X, g \in G, v \in V \} \]
External behaviour of subprocesses can be summarized by a language:

\[
Ext \subseteq \Sigma^*_{ext} \quad \quad Ext = \bigcup_{k=1,2,...} Ext_k
\]

where \( Ext_k \) is the behaviour of processes of depth \( \leq k \).
Semantics under hypothesis

External transitions under hypothesis:

\[
\begin{align*}
(q_1, \lambda, B) & \xrightarrow{a} L (q_2, \lambda, B) \quad \text{if } q_1 \xrightarrow{a} q_2 \text{ if } a \in \Sigma_{\text{ext}} \\
(q, \lambda, B) & \xrightarrow{w(g,v)} L (q, \lambda, B \cup B' \cdot \{w(g,v)\}) \quad \text{if } \emptyset \neq B' \subseteq B, \ B' \cdot \{w(g,v)\} \subseteq \text{pref}(L) \\
(q, \lambda, B) & \xrightarrow{r(g,v)} L (q, \lambda, B \cup B' \cdot \{r(g,v)\}) \quad \text{if } \emptyset \neq B' \subseteq B, \ B' \cdot \{r(g,v)\} \subseteq \text{pref}(L)
\end{align*}
\]
Semantics under hypothesis

Internal transitions under hypothesis:

\[
\begin{align*}
(q_1, \lambda, B) \xrightarrow{\tau}^L (q_2, \lambda, B) & \quad \text{if } q_1 \xrightarrow{\tau} q_2 \\
(q_1, \lambda, B) \xrightarrow{\text{spawn}(p)}^L (q_2, \lambda, B \cup \{\text{spawn}(p)\}) & \quad \text{if } q_1 \xrightarrow{\text{spawn}(p)} q_2 \text{ and } \text{spawn}(p) \in \text{pref}(L) \\
(q_1, \lambda, B) \xrightarrow{w(x,v)}^L (q_2, \lambda', B) & \quad \text{if } q_1 \xrightarrow{w(x,v)} q_2 \text{ and } \lambda' = \lambda[v/x] \\
(q_1, \lambda, B) \xrightarrow{r(x,v)}^L (q_2, \lambda, B) & \quad \text{if } q_1 \xrightarrow{r(x,v)} q_2 \text{ and } \lambda(x) = v \\
(q, \lambda, B) \xrightarrow{\text{w}(x,v)}^L (q, \lambda', B \cup B' \cdot \{o(x,v)\}) & \quad \text{if } \emptyset \neq B' \subseteq B, \ B' \cdot \{o(x,v)\} \subseteq \text{pref}(L) \text{ and } \\
& \lambda' = \lambda[v/x] \\
(q, \lambda, B) \xrightarrow{\tau(x,v)}^L (q, \lambda, B \cup B' \cdot \{i(x,v)\}) & \quad \text{if } \emptyset \neq B' \subseteq B, \ B' \cdot \{i(x,v)\} \subseteq \text{pref}(L), \\
& \lambda(x) = v
\end{align*}
\]
Lemma:
$\rightarrow_{Ext_k}$ describes the behaviour of the process of height $\leq k + 1$. 
For $\alpha \in \Sigma_{Ext}^*$,

$$\text{sig}(\alpha) = \text{the subsequence of the first occurrences of letters in } \alpha.$$  

$$\alpha = b_0 \alpha_1 b_1 \alpha_2 b_2 \cdots \alpha_k b_k \alpha_{k+1} \quad \text{sig}(\alpha) = b_0 b_1 \cdots b_k$$

$\beta \in \text{lift}(\alpha)$ if

$$\alpha = b_0 \alpha_1 b_1 \alpha_2 b_2 \cdots \alpha_k b_k \alpha_{k+1}$$

$$\beta = b_0 \beta_1 b_1 \beta_2 b_2 \cdots \beta_k b_k \beta_{k+1}$$

and $\alpha_i$ is a subword of $\beta_i$.

**Lemma:** $\rightarrow_L$ is the same as $\rightarrow_{\text{lift}(L)}$
Def: $\alpha \preceq \beta$ if $\beta \in \text{lift}(\alpha)$.

Def: $\text{core}(L)$ is the set of $\preceq$ minimal words in $L$.

Obs: $\text{core}(L)$ is always finite since $\preceq$ is WQO.

Lemma: $\rightarrow L$ is the same as $\rightarrow_{\text{core}(L)}$.

\[
\rightarrow_{\text{core}(L)} \subseteq \rightarrow L \\
\rightarrow L \subseteq \rightarrow_{\text{lift}(\text{core}(L))} \\
\rightarrow_{\text{core}(L)} \equiv \rightarrow_{\text{lift}(\text{core}(L))}
\]

because $\text{core}(L) \subseteq L$

because $L \subseteq \text{lift}(\text{core}(L))$

by the previous lemma.
Def: $\alpha \preceq \beta$ if $\beta \in \text{lift}(\alpha)$.

Def: $\text{core}(L)$ is the set of $\preceq$ minimal words in $L$.

Obs: $\text{core}(L)$ is always finite since $\preceq$ is WQO.

Lemma: $\rightarrow^*_L$ is the same as $\rightarrow^*_{\text{core}(L)}$.

Lemma: One can compute $\text{core}(\text{Ext}_i)$ for all $i$.

Thm:
Reachability for pushdown parametric processes with spawn is decidable.
Pushdown parametric processes

Decidable
reachability, PSPACE-compl
repeated reachability, PSPACE-compl
safety, NEXPTIME-compl, maximal safe runs
stutter-expanding properties, NEXPTIME

reachability decidable
PSPACE-compl if no local vars
DEXPTIME if no writes to own local
NP-compl if no writes to own local, and no reads from parent.
Thm:
Reachability for pushdown parametric processes with spawn is decidable.

The result holds also for other models than pushdowns

The model should support:

- Constructively decidable emptiness test
- Synchronized product with finite-state automata
- Alphabet extension

We do not require computable downwards closure.

These conditions are satisfied by:

- Higher-order pushdown automata with collapse
- VAS with action labels
Recursion together with concurrency make verification undecidable, even for finite state systems.

Parametrized concurrency is an abstraction step making the problem decidable, and even manageable.

NEXPTIME-hardness for safety shows that nontrivial behaviors can be encoded even with parametric spawn.

Verification of properties may be useful, particularly to cut out undesired behaviors.

We do not know if verification of properties is decidable for systems with spawn.