Context-bounded Model Checking

Om Swostik Mishra

IIT Bombay

- The interaction between concurrently executing threads of a program might lead to programming errors that are difficult to reproduce and fix.
- In general, the problem of verifying a concurrent Boolean program is undecidable [3].
- We will show that the analysis is decidable if the number of context switches is bounded by an arbitrary constant.
- In general, restricting the number of context switches is not sound as there could be dynamic systems with multiple processes and threads where context switches happen at a high frequency .
- However, the analysis is sound up till the context bound as every thread is explored for an unbounded stack depth.

The Problem

- We will analyze the following problem,
  Given a Boolean program P and an integer k, does P go wrong by failing an assertion with at most k contexts?
- A *context* is an uninterrupted set of actions by a single thread.
- An execution with k contexts invloves $k - 1$ *context switches* i.e execution switches from one thread to another exactly $k - 1$ times.
- The above problem is decidable and there exists an algorithm which decides it, that is polynomial in the size of P and exponential in the size of k.

Domains

γ	$\in$	Γ	*Stack alphabet*
w	$\in$	Γ^*	*Stack*
g	$\in$	G	*Global state*
Δ	$\subseteq$	$(G \times \Gamma) \times (G \times \Gamma^*)$	*Transition relation*
c	$\in$	$G \times \Gamma^*$	*Configuration*
$\longrightarrow_\Delta$	$\subseteq$	$(G \times \Gamma^*) \times (G \times \Gamma^*)$	*Pds transition*

- A single-stack pushdown system is a 5-tuple,

$$P = \ (G, \Gamma, \Delta, g_{in}, w_{in})$$

- G and Γ refer to the set of *global states* and *stack alphabet* respectively.
- A *stack* w is an element of Γ^*.
- A *configuration* is a member of $G \times \Gamma^*$, such that any $c \in G \times \Gamma^*$ can be written as $c = \langle g, w \rangle$, where $g \in G$ and $w \in \Gamma^*$.

Pushdown Systems (continued)

- The *transition relation* Δ over G and Γ is a finite subset of $(G \times \Gamma) \times (G \times \Gamma^*)$.
- $c_{in} = \langle g_{in}, w_{in} \rangle$ refers to the initial configuration of the pushdown system P.
- Given Δ, we can define a transition system on configurations $\rightarrow_\Delta$ as follows:

$$\langle g, \gamma w' \rangle \rightarrow_\Delta \langle g', w w' \rangle, \text{ for all } w' \in \Gamma^* \text{ iff } (\langle g, \gamma \rangle, \langle g', w \rangle) \in \Delta$$

- Let $\rightarrow_\Delta^*$ denote the reflexive-transitive closure of $\rightarrow_\Delta$
- By definiton of Δ above, there are no transitions from a configuration whose stack is empty. Hence, the PDS would halt in this case.

Reachable configurations

- A configuration c is called *reachable* iff $c_{in} \rightarrow^*_\Delta c$, where c_{in} is the initial configuration.
- A *pushdown store automaton* $A = (Q, \Gamma, \delta, I, F)$ is a finite automaton with states Q, alphabet Γ, $\delta \subseteq Q \times \Gamma \times Q$ as the transition function, initial states I and final states F.
- The sets Q and I satisfy $G \subseteq Q$ and $I \subseteq G$.
- A accepts a pushdown configuration $\langle g, w \rangle$ iff A accepts w when started in the state g.
- A subset $S \subseteq G \times \Gamma^*$ is called regular if there exists a pushdown store automaton A such that $S = L(A)$.
- The reachability problem for pushdown systems is decidable because the set of reachable configurations from any initial configuration forms a regular set.

Reachable configurations (continued)

- For a pushdown system $P = (G, \Gamma, \Delta, g_{in}, w_{in})$ and a set of configurations $S \subseteq G \times \Gamma^*$, $Post^*_\Delta(S)$ is the set of configurations reachable from S i.e.

$$Post^*_\Delta(S) = \{c \mid \exists c' \in S,\ \ c' \to^*_\Delta c\}$$

- The set of configurations reachable from a regular set forms a regular set. More precisely,

Theorem 1

Let $P = (G, \Gamma, \Delta, g_{in}, w_{in})$ be a pushdown system and let A be a regular pushdown store automaton. There exists a regular pushdown store automaton A' such that $Post^*_\Delta(L(A)) = L(A')$. The automaton A' can be constructed from A in time $O(|P|^2|A| + |P|^3)$ [4].

Concurrent Pushdown Systems

- A *concurrent pushdown system* is a tuple

$$P = (G, \Gamma, \Delta_0, \ldots, \Delta_N, g_{in}, w_{in})$$

- $\Delta_0, \ldots, \Delta_N$ are transition relations over G and Γ, g_{in} is an initial state and w_{in} is an initial stack.
- A configuration of a concurrent pushdown system is a tuple $c = \langle g, w_0, \ldots, w_N \rangle$ where $g \in G$ and $w_i \in \Gamma^*$.
- The initial configuration of P is $\langle g, w_{in}, \ldots, w_{in} \rangle$ where all $N + 1$ stacks are initialized to w_{in}.
- The transition system of P ($\rightarrow_P$) rewrites the global state while changing any one of the stacks, according to the transition relation of the PDS assciated to that stack.
- More formally,

$$\langle g, w_0, w_1 \ldots, w_i, \ldots w_N \rangle \rightarrow_i \langle g', w_0, w_1 \ldots, w_i', \ldots w_N \rangle \text{ iff}$$

$$\langle g, w_i \rangle \rightarrow_{\Delta_i} \langle g', w_i' \rangle$$

Concurrent Pushdown Systems (continued)

- $\rightarrow_P$ is defined on the configurations of P by the union of $\rightarrow_i$ i.e.

$$\rightarrow_P = \bigcup_{i=0}^{N} \rightarrow_i$$

- $\rightarrow_P^*$ denotes the reflexive, transitive closure of P.

Reachability

- A configuration c is called reachable iff $c_{in} \rightarrow_P^* c$, where c_{in} is the initial configuration.
- In general, checking reachability for concurrent pushdown systems is an undecidable problem [3].
- However, bounding the number of context switches leads to a decidable restriction of the problem.
- For $k \in \mathbb{N}$, we define $\xrightarrow{k}$ as the *k-bounded transition relation*,
  $c \xrightarrow{1} c'$ iff there exists i such that $c \rightarrow_i^* c'$
  $c \xrightarrow{k+1} c'$ iff there exists $c^{''}$ and i such that $c \xrightarrow{k} c^{''}$ and $c^{''} \rightarrow_i^* c'$
- A k-bounded transition contains atmost $k-1$ context switches in which a new relation $\rightarrow_i$ can be chosen.
- A configuration c is *k-reachable* if $c_{in} \xrightarrow{k} c$.
- We define the *k-bounded reachability* problem as follows,
  Given two configuration c_0 and c_1, is it the case that $c_0 \xrightarrow{k} c_1$?

Reachability (continued)

- We will show that the k-bounded reachability problem is decidable.
- To do this, we define a transition relation over *aggregate configurations* i.e. over configurations of the form $\langle\langle g, R_0, \ldots, R_N \rangle\rangle$ where R_i are regular subsets of Γ^*.
- For $g \in G$ and a regular set $R \subseteq \Gamma^*$,

$$\langle\langle g, R \rangle\rangle \;=\; \{\langle g, w \rangle \mid w \in R\}$$

- Let $G = \{g_1, \ldots, g_m\}$. Any regular set $S \subseteq G \times \Gamma^*$ can be written as

$$S \;=\; \biguplus_{i=1}^{m} \langle\langle g_i, R_i \rangle\rangle \tag{1}$$

  where each set $R_i \subseteq \Gamma^*$ is regular.
- By Theorem 1, the set $Post^*_\Delta(S)$ can be written in the form (1), if S is regular.

Reachability (continued)

- We say $\langle\langle g', R'\rangle\rangle \in Post^*_\Delta(S)$ if $Post^*_\Delta(S) = \biguplus_{i=1}^{m}\langle\langle g_i, R_i\rangle\rangle$, where $\langle\langle g', R'\rangle\rangle = \langle\langle g_j, R_j\rangle\rangle$ (for some j such that $1 \leq j \leq m$).
- We define relations $\Rightarrow_i$ on aggregate configurations,

$$\langle\langle g, R_0, \ldots, R_i, \ldots, R_N\rangle\rangle \Rightarrow_i \langle\langle g', R_0, \ldots R'_i, \ldots, R_N\rangle\rangle$$

  if $\langle\langle g', R'_i\rangle\rangle \in Post^*_\Delta(\langle\langle g, R_i\rangle\rangle)$.
- We define the transition relation on aggregate configurations $\Rightarrow$ as $\bigcup_{i=0}^{N} \Rightarrow_i$.
- For aggregate configurations a_1 and a_2, $a_1 \overset{k}{\Rightarrow} a_2$, if a_2 can be reached from a_1 using at most k transitions.
- Using the notations described above, the following reduces k-bounded reachability problem to sequential appications of $Post^*_\Delta$ operator,

Reachability (continued)

Theorem 2

Let $P = (G, \Gamma, \Delta_0, \ldots, \Delta_N, g_{in}, w_{in})$ *be a concurrent pushdown system. Then, for any* k*, we have* $\langle g, w_0, \ldots, w_N \rangle \xrightarrow{k} \langle g', w'_0, \ldots, w'_N \rangle$ *iff* $\langle\langle g, \{w_0\}, \ldots, \{w_N\} \rangle\rangle \overset{k}{\Rightarrow} \langle\langle g', R'_0, \ldots, R'_N \rangle\rangle$ *for some* $R'_0, \ldots, R'_N$ *such that* $w'_i \in R'_i$ *for all* i *such that* $1 \leq i \leq N$*.*

Algorithm

Using Theorem 1 and Theorem 2, we can generate an algorithm for solving the context-bounded reachability problem for concurrent pushdown systems.

Input: Concurrent pushdown system $(G, \Gamma, \Delta_0, \ldots, \Delta_N, g_{in}, w_{in})$ *and bound* k

let $A_{in} = (Q, \Gamma, \delta, \{g_{in}\}, F)$ such that $L(A_{in}) = \{\langle g_{in}, w_{in}\rangle\}$;

WL := $\{(\langle g, A_{in}, \ldots, A_{in}\rangle, 0)\}$; // There are N copies of A_{in}
$Reach$:= $\{\langle g, A_{in}, \ldots, A_{in}\rangle\}$;

while (WL **not empty**)
  let $(\langle g, A_0, \ldots, A_N\rangle, i) =$ REMOVE(WL) **in**
    if $(i < k)$
      forall $(j = 0 \ldots N)$
        let $A'_j = Post^*_{\Delta_j}(A_j)$ **in**
          forall $(g' \in G(A'_j))$ {
            let $x = \langle g',$ RENAME$(A_0, g'), \ldots,$ ANONYMIZE$(A'_j, g'), \ldots,$ RENAME$(A_N, g')\rangle$ **in**
              ADD(WL, $(x, i+1)$);
              $Reach$:= $Reach \cup \{x\}$;
          }
Output : $Reach$

Figure 1: The algorithm

Algorithm (continued)

- The algorithm processes a worklist WL which contains a set of items of the form $(\langle g, A_0, \ldots, A_N \rangle, i)$, where $g \in G$ is a global state, A_j are pushdown store automata and $i \in \{0, 1, \ldots, k-1\}$.
- Remove(WL) removes an item from the worklist and returns it
- Add(WL, item) adds item to the worklist
- The inital pushdown store automata A_{in} has initial state g_{in} and accepts only w_{in}.
- Given $A'_j = Post^*_\Delta(A_j)$, we construct A'_j according to Theorem 1, such that $L(A'_j) = Post^*_\Delta(L(A_j))$.
- $G(A'_j) = \{g' \mid \exists w. \langle g', w \rangle \in L(A'_j)\}$.
- All pushdown store automata involved in the algorithm have atmost one start state.
- Given an automata A and a state $g \in G$, RENAME(A, g) returns the result of renaming the start state (if any) of A to g
- ANONYMIZE(A, g) returns the result of renaming all states of G (except g) to fresh states which are not in G.

The operations RENAME and ANONYMIZE are necessary for applying Theorem 1 repeatedly as the construction of the pushdown store automata uses elements of G as states [4]. In order to avoid errors in the construction, renaming on states of G is necessary.

Theorem 3

Let $P = (G, \Gamma, \Delta_0, \ldots, \Delta_N, g_{in}, w_{in})$ be a concurrent pushdown system. For any k, the algorithm terminates on input P and k, and $\langle\langle g_{in}, w_{in}, \ldots, w_{in}\rangle\rangle \overset{k}{\Rightarrow} \langle\langle g', R'_0, \ldots, R'_N\rangle\rangle$ iff the algorithm outputs Reach with $\langle g', A'_0, \ldots, A'_N\rangle \in$ Reach such that $L(A'_i) = \langle\langle g', R'_i\rangle\rangle$ for all $i \in \{0, 1, \ldots, N\}$.

Theorem 2 together with Theorem 3, imply that the algorithm solves the k-bounded reachability problem. The next natural step is to look at the running time of the algorithm.

Theorem 4

Given a concurrent pushdown system $P = (G, \Gamma, \Delta_0, \ldots, \Delta_N, g_{in}, w_{in})$ and a bound k, the algorithm in Figure 1 decides the k-bounded reachability problem in time $O(k^3(N|G|)^k|P|^5)$.

We will now look at dynamic concurrent pushdown systems with additional operations, fork and join.

- To allow for dynamic fork-join parallelism, thread identifiers are to be stored as program variables. They are members of the set $Tid = \{0, 1, 2, \ldots\}$.
- A dynamic concurrent PDS is a tuple $(GBV, GTV, LBV, LTV, \Delta, \Delta_F, \Delta_J, g_{in}, \gamma_{in})$.
- GBV is the set of all global variables containing boolean values and GTV is the set of all global variables containing thread identifiers.
- G is the (infinite) set of all valuations to the global variables.
- LBV is the set of all local variables containing boolean values LTV is the set of all local variables containing thread identifiers.
- Γ is the (infinite) set of all valuations to the local variables.
- $\Delta \subseteq (G \times \Gamma) \times (G \times \Gamma^*)$ is the transition relation describing the single step of any thread.
- $\Delta_F \subseteq Tid \times (G \times \Gamma) \times (G \times \Gamma^*)$ is the fork transition relation.

Dynamic Concurrent Pushdown Systems (continued)

- If $(t, \langle g, \gamma \rangle, \langle g', w \rangle) \in \Delta_F$, then with the global store being g, a thread with γ at the top of its stack may fork a thread with identifier t, changing the global store to g' and rewriting the top of the stack to w.
- $\Delta_J \subseteq LTV \times (G \times \Gamma) \times (G \times \Gamma^*)$ is the join transition relation.
- If $(x, \langle g, \gamma \rangle, \langle g', w \rangle) \in \Delta_J$, then with the global store being g, a thread with γ at the top of its stack blocks until the thread with identifier $\gamma(x)$ finishes execution. On getting unblocked, the global store is modified to g' and the top of the stack is rewritten with w.
- g_{in} is the initial valuation to the set of all global variables such that $g_{in}(x) = 0$ for all $x \in GTV$.
- γ_{in} is the initial valuation to the set of all local variables such that $\gamma_{in}(x) = 0$ for all $x \in LTV$.

Dynamic Concurrent Pushdown Systems (continued)

$$ss \in Stacks = Tid \rightarrow (\Gamma \cup \{\$\})^*$$
$$c \in C = G \times Tid \times Stacks \qquad \textit{Configuration}$$
$$\rightsquigarrow \subseteq C \times C$$

- Every dynamic concurrent PDS is equipped with a special symbol $\$ \notin \Gamma$ to mark the bottom of each stack.
- A configuartion of the system is $\langle g, n, ss \rangle$, where g is the global state, n is the identifier of the last thread to be forked and $ss(t)$ is the stack for the thread with identifier $t \in Tid$.
- The execution of the dynamic concurrent PDS starts in the configuration $\langle g_{in}, 0, ss_0 \rangle$, where $ss_0(t) = \gamma_{in}\$$ for all $t \in Tid$.
- The execution follows a certain set of rules which define the transitions that may be performed starting from any thread t in the configuration $\langle g, n, ss \rangle$.

Operational Semantics

Operational Semantics

(SEQ)
$$\frac{t \le n \quad ss(t) = \gamma w' \quad (\langle g, \gamma \rangle, \langle g', w \rangle) \in \Delta}{\langle g, n, ss \rangle \rightsquigarrow_t \langle g', n, ss[t := ww'] \rangle}$$

(SEQEND)
$$\frac{t \le n \quad ss(t) = \$}{\langle g, n, ss \rangle \rightsquigarrow_t \langle g, n, ss[t := \epsilon] \rangle}$$

(FORK)
$$\frac{t \le n \quad ss(t) = \gamma w' \quad (n+1, \langle g, \gamma \rangle, \langle g', w \rangle) \in \Delta_F}{\langle g, n, ss \rangle \rightsquigarrow_t \langle g', n+1, ss[t := ww'] \rangle}$$

(JOIN)
$$\frac{t \le n \quad ss(t) = \gamma w' \quad x \in LTV \quad (x, \langle g, \gamma \rangle, \langle g', w \rangle) \in \Delta_J \quad ss(\gamma(x)) = \epsilon}{\langle g, n, ss \rangle \rightsquigarrow_t \langle g', n, ss[t := ww'] \rangle}$$

- All rules have the condition $t \leq n$, which indicates that thread t must have already been forked.
- The rule SEQ allows thread t to perform a move according to the transition relation Δ.
- The rule SEQEND is enabled if the only symbol of the stack of thread t is \$ which is popped from the stack without changing the global state.

Operational Semantics (continued)

- The rule FORK creates a new thread with identifier $n + 1$.
- The rule JOIN is enabled if thread $\gamma(x)$ has terminated, where γ is the top of the stack of thread t. Termination of a thread is indicated by an empty stack.

Assumptions

- In realistic concurrent programs, the usage of thread identifiers is restricted.
- In view of this, we introduce some assumptions.
- A *renaming function* is a partial function from Tid to Tid.
- When a renaming function is applied to a global store g, it returns another store in which every variable of type Tid is transformed by an application of f.
- If f is undefined on some global variable in g, it is undefined on g.
- A renaming function can also be applied to a local store or a sequence of local stores by a pointwise application to each element of the sequence.

Assumptions (continued)

A1. For all $g \in G, \gamma \in \Gamma$ and renaming functions f such that $f(g)$ and $f(\gamma)$ are defined,

1. If $(\langle g, \gamma \rangle, \langle g', w \rangle) \in \Delta$, then there exists $fg' \in G$ and $fw \in \Gamma^*$ such that $fg' = f(g')$, $fw = f(w)$, and $(\langle f(g), f(\gamma) \rangle, \langle fg', fw \rangle) \in \Delta$.
2. If $(\langle f(g), f(\gamma) \rangle, \langle fg', fw \rangle) \in \Delta$, then there exists $g' \in G$ and $w \in \Gamma^*$ such that $fg' = f(g')$, $fw = f(w)$ and $(\langle g, \gamma \rangle, \langle g', w \rangle) \in \Delta$.

A2. For all $t \in Tid$, $g \in G$, $\gamma \in \Gamma$ and renaming functions f such that $f(t)$, $f(g)$ and $f(\gamma)$ are all defined,

1. If $(t, \langle g, \gamma \rangle, \langle g', w \rangle) \in \Delta_F$, then there exists $fg' \in G$ and $fw \in \Gamma^*$ such that $fg' = f(g')$, $fw = f(w)$ and $(f(t), \langle f(g), f(\gamma) \rangle, \langle fg', fw \rangle) \in \Delta_F$.
2. If $(f(t), \langle f(g), f(\gamma) \rangle, \langle fg', fw \rangle) \in \Delta_F$, then there exists $g' \in G$ and $w \in \Gamma^*$ such that $fg' = f(g')$, $fw = f(w)$ and $(t, \langle g, \gamma \rangle, \langle g', w \rangle) \in \Delta_F$.

Assumptions (continued)

A3. For all $x \in LTV$, $g \in G$, $\gamma \in \Gamma$ and renaming functions f such that $f(g)$ and $f(\gamma)$ are defined,

1. If $(x, \langle g, \gamma \rangle, \langle g', w \rangle) \in \Delta_J$, then there exists $fg' \in G$ and $fw \in \Gamma^*$ such that $fg' = f(g')$, $fw = f(w)$ and $(x, \langle f(g), f(\gamma) \rangle, \langle fg', fw \rangle) \in \Delta_J$.
2. If $(x, \langle f(g), f(\gamma) \rangle, \langle fg', fw \rangle) \in \Delta_J$, then there exists $g' \in G$ and $w \in \Gamma^*$ such that $fg' = f(g')$, $fw = f(w)$ and $(x, \langle g, \gamma \rangle, \langle g', w \rangle) \in \Delta_J$.

Henceforth, these are the assumptions we will be making on Δ, Δ_F and Δ_J.
By making these asssumptions, we are exploiting the restricted usage of thread identifiers in concurrent systems.

Reduction to Concurrent PDS

- We will reduce the *k-bounded reachability* problem on a dynamic concurrent PDS to a concurrent PDS with $k+1$ threads.
- Given a dynamic concurrent PDS P and a positive integer k, we will construct a concurrent PDS P_k containing $k+1$ threads with identifiers in $\{0, 1, \ldots, k\}$ such that it suffices to verify the k-bounded executions of P_k.
- In a k-bounded execution, at most k different threads may perform a transition.
- The last thread in P_k doesn't perform a transition, it only exists to simulate the remaining threads in P.
- Let $Tid_k = \{0, 1, \ldots, k\}$ be the set of thread identifiers bounded by k.
- $AbsG_k$ and $Abs\Gamma_k$ are finite sets which consist of all valuations to global and local variables, where the variables of thread identifier type are assigned values from Tid_k.

Reduction to Concurrent PDS (continued)

- Given a dynamic concurrent PDS $P = (GBV, GTV, LBV, LTV, \Delta, \Delta_F, \Delta_J, g_{in}, \gamma_{in})$, we define,

  $$P_k = (AbsG_k \times Tid_k \times \mathscr{P}(Tid_k), Abs\Gamma_k \cup \{\$\}, \Delta_0, \ldots, \Delta_k, (g_{in}, 0, \emptyset), \gamma_{in}\$)$$

- P_k has $k+1$ threads.
- A global state of P_k is a three-tuple (g, n, α), where g is a valuation to the global variables, n is the largest thread identifier whose corresponding thread is allowed to make a transition and α is the set of thread identifiers whose stacks are empty.
- The initial state is $(g_{in}, 0, \emptyset)$ which means that only thread 0 is allowed to make a transition and no thread has finished execution.
- There are rules for every transition relation Δ_t, for each thread $t \in Tid_k$.

Reduction to Concurrent PDS (continued)

Definition of Δ_t

(ABSSEQ)
$$\frac{t \leq n \quad (\langle g, \gamma \rangle, \langle g', w \rangle) \in \Delta}{(\langle (g, n, \alpha), \gamma \rangle \;,\; \langle (g', n, \alpha), w \rangle) \in \Delta_t}$$

(ABSSEQEND)
$$\frac{t \leq n}{(\langle (g, n, \alpha), \$ \rangle \;,\; \langle (g, n, \alpha \cup \{t\}), \epsilon \rangle) \in \Delta_t}$$

(ABSFORK)
$$\frac{t \leq n \quad n+1 < k \quad (n+1, \langle g, \gamma \rangle, \langle g', w \rangle) \in \Delta_F}{(\langle (g, n, \alpha), \gamma \rangle \;,\; \langle (g', n+1, \alpha), w \rangle) \in \Delta_t}$$

(ABSFORKNONDET)
$$\frac{t \leq n \quad (k, \langle g, \gamma \rangle, \langle g', w \rangle) \in \Delta_F}{(\langle (g, n, \alpha), \gamma \rangle \;,\; \langle (g', n, \alpha), w \rangle) \in \Delta_t}$$

(ABSJOIN)
$$\frac{t \leq n \quad x \in LTV \quad (x, \langle g, \gamma \rangle, \langle g', w \rangle) \in \Delta_J \quad \gamma(x) \in \alpha}{(\langle (g, n, \alpha), \gamma \rangle \;,\; \langle (g', n, \alpha), w \rangle) \in \Delta_t}$$

- All transitions are guarded by the condition $t \leq n$ to ensure that a thread t cannot make a transition if $t > n$.
- The rule ABSSEQ adds transitions from Δ to Δ_t.
- The rule ABSSEQEND adds thread t to the set of terminated threads.
- The rules ABSFORK and ABSFORKNONDET handle thread creation in P.

Reduction to Concurrent PDS (continued)

- The rule ABSFORK increments the counter n allowing thread $n+1$ to simulate the newly forked thread which participates in the k-bounded execution.
- The rule ABSFORKNONDET leaves the counter unchanged and the newly forked thread doesn't participate in the k-bounded execution.
- The rule ABSJOIN adds rules from Δ_J to Δ by using the fact that the identifiers of all previously terminated threads are present in α.
- We will now state the correctness theorems for the transformation.
  A configuration of P_k, $\langle (g', n', \alpha), w_0, w_1, \ldots, w_k \rangle$, can be written as $\langle (g', n', \alpha), ss' \rangle$, where ss' is a map from Tid_k to $(Abs\Gamma_k \cup \$)^*$.

Theorem 5 (Soundness)

Let P be a dynamic concurrent pushdown system and k be a positive integer. Let $\langle g, n, ss \rangle$ be a k-reachable configuration of P. Then there is a total renaming function $f : Tid \rightarrow Tid_k$ and a k-reachable configuration $\langle (g', n', \alpha), ss' \rangle$ of the concurrent pushdown system P_k such that $g' = f(g)$ and $ss'(f(j)) = f(ss(j))$ for all $j \in Tid$.

Theorem 6 (Completeness)

Let P be a dynamic concurrent pushdown system and k be a positive integer. Let $\langle (g', n', \alpha), ss' \rangle$ be a k-reachable configuration of the concurrent pushdown system P_k. Then there is a total renaming function $f : Tid \rightarrow Tid_k$ and a k-reachable configuration $\langle g, n, ss \rangle$ of P such that $g' = f(g)$ and $ss'(f(j)) = f(ss(j))$ for all $j \in Tid$.

With Theorems 5 and 6, we can conclude that our reduction from a dynamic concurrent pushdown system to a concurrent pushdown system with $k + 1$ threads is correct.

- The classical notion of context bounding is too restrictive.
- For dynamic systems, bounding the number of context switches bounds the number of threads involved.
- K-bounded computation: Each thread can be interrupted and resumed at most K times.
- Decidability also holds under this new restriction.
- Given a program P and an integer K, the problem of checking whether a program fails some assertion in a K-bounded computation is EXPSPACE-complete [1].

Mohamed Faouzi Atig, Ahmed Bouajjani, and Shaz Qadeer.
Context-bounded analysis for concurrent programs with dynamic creation of threads.
Logical Methods in Computer Science, 7, 2011.

Shaz Qadeer and Jakob Rehof.
Context-bounded model checking of concurrent software.
In *International conference on tools and algorithms for the construction and analysis of systems*, pages 93–107. Springer, 2005.

Ganesan Ramalingam.
Context-sensitive synchronization-sensitive analysis is undecidable.
ACM Transactions on Programming languages and Systems (TOPLAS), 22(2):416–430, 2000.

Stefan Schwoon.
Model-checking pushdown systems.
PhD thesis, Technische Universität München, 2002.

THANK YOU!