# Tree Automata for Analyzing Dynamic Pushdown Networks

Peter Lammich

Institut für Informatik, Westfälische Wilhelms-Universität Münster, Germany

12.10.2009
KPS 2009

# Motivation

- DPNs: Abstract model for concurrent programs
  - Dynamic thread Creation
- Original: Interleaving Semantics, analysis by $\text{pre}^*_M$
- Recently: True-Concurrency semantics, analysis by $\text{pre}^*_M$
- Here: True Concurrency semantics, analysis by tree-automata techniques

# Overview

Given:

DPN $\Delta$ with executions $e_\Delta \subseteq E$

Property $\Phi \subseteq E$

Want to know: $e_\Delta \cap \Phi = \emptyset$

Solution:

Have $r_\Delta \subseteq R$ and $\alpha : R \rightarrow E$

such that: $\alpha(r_\Delta) = e_\Delta$

Now decide: $r_\Delta \cap \alpha^{-1}(\Phi) = \emptyset$

Good News:

$r_\Delta$ and $\alpha^{-1}(\Phi)$ are regular sets of trees

Use standard tree-automata techniques

# Dynamic Pushdown Networks (DPNs)

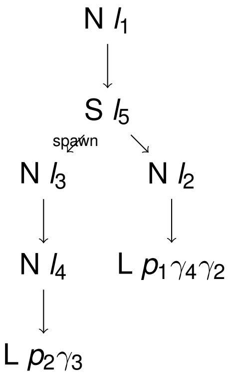
- Pushdown processes that can spawn new processes:
  Rules of type: $p\gamma \overset{a}{\hookrightarrow} p'w$ and $p\gamma \overset{a}{\hookrightarrow} p'w \rhd p_s w_s$
- Interleaving semantics: $c \overset{l}{\rightarrow}^* c'$
  $c, c' \in (P\Gamma^*)^*$ start and end configuration
  Words over alphabet $P \cup \Gamma$, with $P \cap \Gamma = \emptyset$
  $l \in L^*$ sequence of executed labels
- Predecessor set: $\text{pre}^*_M(C) := \{c \mid \exists c' \in C, l.\ c \overset{l}{\rightarrow}^* c'\}$
  Preserves regularity, computable in polynomial time
  [Bouajjani et al., 2005]

# Tree-Based Semantics

DPN rules:

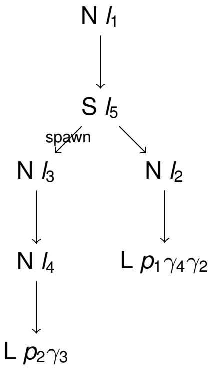
$$p_1\gamma \xhookrightarrow{l_1} p_1\gamma_1\gamma_2$$
$$p_1\gamma_3 \xhookrightarrow{l_2} p_1\gamma_4$$
$$p_2\gamma \xhookrightarrow{l_3} p_2\gamma_2\gamma_3$$
$$p_2\gamma_2 \xhookrightarrow{l_4} p_2$$
$$p_1\gamma_1 \xhookrightarrow{l_5} p_1\gamma_3 \triangleright p_2\gamma$$

Execution tree $t$:
(from $p_1\gamma$)




Schedules:
$\text{sched}(t) = \{$
$l_1 l_5 l_2 l_3 l_4,$
$l_1 l_5 l_3 l_2 l_4,$
$l_1 l_5 l_3 l_4 l_2$
$\}$

# Execution Trees




Information contained in execution tree:

- Total ordering of steps of each process
- Causality induced by process creation
- Reached configuration
- (Implicitly) Process IDs

# Regular Execution Trees

DPN rules:

$p_1\gamma \overset{l_1}{\hookrightarrow} p_1\gamma_1\gamma_2$

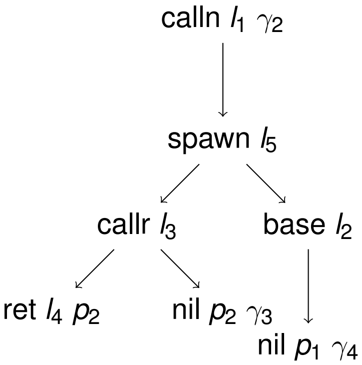
$p_1\gamma_3 \overset{l_2}{\hookrightarrow} p_1\gamma_4$

$p_2\gamma \overset{l_3}{\hookrightarrow} p_2\gamma_2\gamma_3$

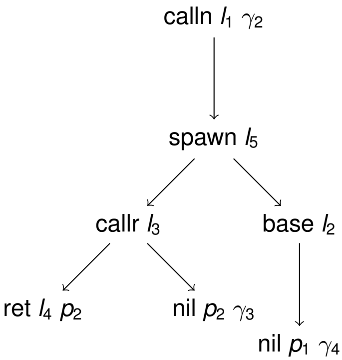
$p_2\gamma_2 \overset{l_4}{\hookrightarrow} p_2$

$p_1\gamma_1 \overset{l_5}{\hookrightarrow} p_1\gamma_3 \rhd p_2\gamma$

Regular execution tree $\tau$:
(from $p_1\gamma$)



# Regular Execution Trees




- Idea: Make Call/Return structure visible in execution tree
- Set of regular execution trees of DPN is tree-regular
  - Automata can be generated from DPN
- (Tree-) regular properties transfer from standard execution trees
  - Done by hand: Reachability of configuration
  - Indication: $\alpha$ is macro-tree transducer

# Summary

- True-Concurrency Semantics for DPN
- Regular execution trees
- Tree-automata techniques for model-checking

- Results verified with Isabelle/HOL
- Future Work
    - Properties with intermediate configurations
    - Symbolic techniques to speed-up computation
      Horn-Clauses, BDDs, ...
    - Compare with automata-based techniques

# Tree Automata for Executions

$$e_\Delta = N[p_0, \gamma_0]$$

$$
\begin{array}{lll}
& [\text{n-nil}] \quad \text{nil } p\gamma \in N[p,\gamma] & \\
\text{for } p\gamma \overset{l}{\hookrightarrow} p' \in \Delta : & & \\
& [\text{r-ret}] \quad \text{ret } l\ p' \in R[p,\gamma,p'] & \\
\text{for } p\gamma \overset{l}{\hookrightarrow} p'\gamma' \in \Delta, \tilde{p} \in P : & & \\
& [\text{n-base}] \quad \text{base } l\ \tau \in N[p,\gamma] & \Leftarrow \quad \tau \in N[p',\gamma'] \\
& [\text{r-base}] \quad \text{base } l\ \tau \in R[p,\gamma,\tilde{p}] & \Leftarrow \quad \tau \in R[p',\gamma',\tilde{p}] \\
\text{for } p\gamma \overset{l}{\hookrightarrow} p'\gamma_1\gamma_2 \in \Delta, \tilde{p}, \hat{p} \in P : & & \\
& [\text{n-calln}] \quad \text{calln } l\ \tau\ \gamma_2 \in N[p,\gamma] & \Leftarrow \quad \tau \in N[p',\gamma_1] \\
& [\text{n-callr}] \quad \text{callr } l\ \tau_c\ \tau \in N[p,\gamma] & \Leftarrow \quad \tau_c \in R[p',\gamma_1,\hat{p}] \wedge \tau \in N[\hat{p},\gamma_2] \\
& [\text{r-callr}] \quad \text{callr } l\ \tau_c\ \tau \in R[p,\gamma,\tilde{p}] & \Leftarrow \quad \tau_c \in R[p',\gamma_1,\hat{p}] \wedge \tau \in R[\hat{p},\gamma_2,\tilde{p}] \\
\text{for } p\gamma \overset{l}{\hookrightarrow} p'\gamma' \rhd p_s\gamma_s \in \Delta, \tilde{p} \in P : & & \\
& [\text{n-spawn}] \quad \text{spawn } l\ \tau_s\ \tau \in N[p,\gamma] & \Leftarrow \quad \tau_s \in N[p_s,\gamma_s] \wedge \tau \in N[p',\gamma'] \\
& [\text{r-spawn}] \quad \text{spawn } l\ \tau_s\ \tau \in R[p,\gamma,\tilde{p}] & \Leftarrow \quad \tau_s \in N[p_s,\gamma_s] \wedge \tau \in R[p',\gamma',\tilde{p}]
\end{array}
$$