World Journal of Social Science Research
ISSN 2375-9747 (Print) ISSN 2332-5534 (Online)
Vol. 4, No. 4, 2017
www.scholink.org/ojs/index.php/wjssr

Reachability Analysis of Asynchronous Dynamic Pushdown Networks Based on Tree Semantics Approach

Guodong Wu[1*] & Junyan Qian[2]

[1] School of Information and Computer, Anhui Agricultural University, Hefei, China

[2] Guangxi Key Laboratory of Trusted Software, Guilin University of Electronic Technology, Guilin 541004, China

[*] Guodong Wu, E-mail: 1306160554@qq.com

Received: October 6, 2017 Accepted: October 15, 2017 Online Published: October 17, 2017

doi:10.22158/wjssr.v4n4p287 URL: http://dx.doi.org/10.22158/wjssr.v4n4p287

Abstract

ADPN (Asynchronous Dynamic Pushdown Networks) are an abstract model for concurrent programs with recursive procedures and dynamic thread creation. Usually, asynchronous dynamic pushdown networks are described with interleaving semantics, in which the backward analysis is not effective. In order to improve interleaving semantics, tree semantics approach was introduced. This paper extends the tree semantics to ADPN. Because the reachability problem of ADPN is also undecidable, we address the context-bounded reachability problem and provide an algorithm for backward reachability analysis with tree-based semantics Approach.

Keywords

DPN, ADPN, tree semantics, context-bounded, backward reachability

1. Introduction

With development of multi-core processor, the research on concurrent program has become the focus of programming. Usually, the concurrent program employs concurrent pushdown system or parallel procedure call for modeling, which two models, however, fail to well simulate program with dynamic thread creation. Bouajjani et al. suggested in 2005 the DPN which are applicable to modeling of concurrent program containing recursive procedures or dynamic thread creation, yet its execution semantics is interleaving semantics, in which the backward reachability analysis is inefficient. Lammich et al. suggested tree semantics for DPN to model execution of program as a tree, which more conforms to actual running of program, and they provided the reachability analysis means for dynamic pushdown networks under tree semantics yet without considering intercommunication of threads. Based on dynamic pushdown networks model, Bouajjani raised ADPN model, whose reachability problem, however, is undecidable and backward reachability on it is complicated. Faouzi modeled program

containing dynamic thread creation, presented context-bounded analysis algorithm for this model, and proved it is decidable. Wenner extended dynamic pushdown networks to weighted dynamic pushdown networks to enhance modeling capacity of model. The paper extends tree semantic analysis into ADPN model to make it more efficiently conduct backward reachability analysis, and solved undecidable problem using the context-bounded approach. The backward reachable patterns set $\mathrm{Pr}e_M^*$ of this model keeps regularity property under tree semantics, i.e., able to construct automator to receive a group of backward reachable patterns set $\mathrm{Pr}e_M^*$. The paper structures an intermediate model to simulate execution mode of tree semantics, calculates backward reachable patterns set of this intermediate model in context of finite times, and uses projection operation to derive the backward reachable patterns set on original model. Finally, the complexity analysis on this algorithm is presented.

2. Relate Knowledge

ADPN is extension of DPN to simulate pushdown, new process and asynchronous communication of inter-thread using of shared memory.

ADPN is a quintuple $M = (G, P, \Gamma, \Delta_l, \Delta_g)$, where G is global state set, P is local state set, Γ is stack symbol, Δ_l is set of local migration rules: (a) $\rho\gamma \xrightarrow{l} \rho_1 w_1$; (b) $\rho\gamma \xrightarrow{l} \rho_1 w_1 \triangleright \rho_2 w_2$, where $\rho, \rho_1, \rho_2 \in P$, $\gamma \in \Gamma$, $w_1, w_2 \in \Gamma^*$; Δ_g is set of global migration rules: (a) $(g, \rho\gamma) \xrightarrow{l} (g', \rho_1 w_1)$; (b) $(g, \rho\gamma) \xrightarrow{l} (g', \rho_1 w_1) \triangleright \rho_2 w_2$; where $g, g' \in G$, $\rho, \rho_1, \rho_2 \in P$, $\gamma \in \Gamma$, $w_1, w_2 \in \Gamma^*$.

ADPN's pattern is $(g, \alpha) \in G \times (P\ \Gamma^*)$, where g is global state, character string $\alpha = \rho_1 w_1 \rho_2 w_2 \ldots \rho_n\ w_n$, and each substring represents corresponding DPN pattern of ADPN pattern.

Let C represent ADPN pattern set, migration relation is defined as: $(g, u) \xrightarrow{l} (g', v)$ is true, then $\rho\gamma \xrightarrow{l} p_1\ w_1$ belongs to rules set Δ_l, and $u = u_1\ \rho\ \gamma\ u_2$, $v = u_1\ \rho_1\ w_1\ u_2$, $g = g'$, or $\rho\gamma \xrightarrow{l} p_1\ w_1 \triangleright \rho_2\ w_2$ belongs to rules set Δ_l, or $(g, \rho\gamma) \xrightarrow{l} (g', \rho_1\ w_1)$ belongs to rules set Δ_g, and $u = u_1\ \rho\ \gamma\ u_2$, $v = u_1\ \rho_1\ w_1\ u_2$, or $(g, \rho\gamma) \xrightarrow{l} (g', \rho_1 w_1) \triangleright \rho_2 w_2$, and $u = u_1 \rho\gamma u_2$, $v = u_1\ \rho_2\ w_2\ \rho_1\ w_1\ u_2$;

Where $u_1 \in (P\Gamma^*)^*$, $u_2 \in \Gamma^*(P\Gamma^*)^*$. Global migration and local migration rules sets constitute the whole migration system.

State is reachable: There is ADPN M and patterns set $S \subseteq C$, define $post_M^*(S)$ and $pre_M^*(S)$ as respectively representing forward and backward reachable patterns sets that start from pattern S. The forward and backward reachability problems can respectively be defined as: patterns set I and F are respectively initial and a certain given pattern sets. To judge whether the state in given pattern F is reachable just

needs to judge whether $post_M^*(I) \cap F = \phi$ or $pre_M^*(F) \cap I = \phi$ is true. However, these two reachability problems are both undecidable.

K-context-bounded state is reachable: Qadeer et al. Put forward context-bounded reachability analysis. The calculation is switched from one thread to another thread, which is defined as context switching. *K*-boundary means the corresponding migration sequence at most contains K contexts. K-context-bounded state being reachable just means the set of state reachable by ADPN in K contexts. $post_{k,M}^*(S)$ and $pre_{k,M}^*(S)$ are respectively defined as set of forward and backward reachable patterns that start from S and reachable to K-context-boundary. The K-context-bounded state reachability problem is decidable. Musuvathi et al. proved that majority of states can be detected as long as K is set as a small constant.

3. Tree Semantics of ADPN

This section extends tree semantics to ADPN, and migration sequence is partial order, which is similar to a tree, as shown in Figure 1. The initial state of the migration sequence only contains a thread, and new threads are created dynamically. The migration sequence of newly created threads correspond to execution tree on the left of node S, and the migration sequence on the right of node S is original thread. The form is defined as T_M: = $N\ L\ T_M/S\ L\ T_MT_M/L < G,PI^* >$, where *Nlt* corresponds to non-dynamic creation rule *l*, and what follows it is t; $S\ l\ t_st$ corresponds to dynamic creation rule *l*, and what follows it is a dynamically created new thread node t_s and original thread's node t; $L <g, pw>$ corresponds to the pattern when thread running terminates. The advantage of tree semantics is immediately judging which progress the migration belongs to via execution model of tree form.

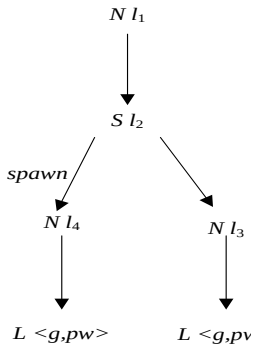

Figure 1. ADPN Execution Tree

Definition 1 (Execution tree): The execution tree is defined as migration sequence of model under tree semantics. Migration rule$\Rightarrow_M \subseteq <g,\ PI^*> \times T_M \times <g',\ Conf_N>$ means the initial state only contains execution tree of single thread, where $Conf_N$ is pattern α of corresponding DPN. When the initial state

contains multiple threads, definition $c \overset{h}{\Rightarrow}_M c'$ means migration from pattern c to pattern *c'* via execution tree h, where $h = t_1 \ldots t_n$.

The execution tree is regular. The paper adopts hedge-automator for receiving. Hedge-automator $T = (S, A_0, D)$, where S is set of a group of finite states, A_0 is an initial automator meeting $L(A_0) \in S^*$. Rules set $D = D_L \cup D_N \cup D_S$: The rule $s \to A_1 \in D_L$ simulates leaf node, rule $s \xrightarrow{l} s' \in D_N$ simulates non-dynamically created node, where, *s, s', $s_s \in S$, $l \in L$, automator A_1* receives all patterns sets ($L(A_1) \in (G, (P\Gamma^*)^+)$).

Execution tree $h = t_1 \ldots t_n \in T_M^*$ can be received by hedge-automator when and only when execution tree $t_1 \ldots t_n$ can be simulated by rules set D and initial states set of execution tree respectively correspond to initial states set $s_1 \ldots s_n \in L(A_0)$ of automator. Rule $lab_T \subseteq S \times T_M$ represents the association between hedge-automator and execution tree, as shown below in form:

[Leaf node] $lab_T(s, L\langle g, pw\rangle) \Leftrightarrow s \to A_1 \in D_L \wedge \langle g, pw\rangle \in L(A_1)$

[Non-dynamic creation] $lab_T(s, N\ l\ t) \Leftrightarrow s \xrightarrow{l} s' \in D_N \wedge lab_T(s', t)$

[Dynamic creation] $lab_T(s, S\ l\ t_s\ t) \Leftrightarrow$

$$s \xrightarrow{l} s' \triangleright s_s \in D_S \wedge lab_T(s', t) \wedge lab_T(s_s, t_s)$$

Where $lab_T(s, t)$ means the state of execution tree t corresponds to state s of automator. It is assumed that $lab_T(s_1 \ldots s_n, t_1 \ldots t_n) \Leftrightarrow lab_T(s_1, t_1) \wedge \ldots \wedge lab_T(s_n, t_n)$, then the language received by hedge-automator *T is* $L(T) := \{h \mid \exists \overline{s} \in L(A_0).lab_T(\overline{s}, h)\}$.

4. Backward Reachable Algorithm Based on Tree Semantics

It is assumed that pattern of ADPN is $\langle g, \alpha\rangle$, where α is corresponding DPN pattern. Receiving pattern α of automator A is structured according to literature. The paper will structure automator Z receiving ADPN pattern $\langle g, \alpha\rangle$. For any character string $u_1 p u_2 \in L(A)$, automator Z receives character string $w = u_1(g,p)u_2$, where $u_1 \in (P\Gamma^*)^*$, $p \in P$, $u_2 \in \Gamma^*(P\Gamma^*)^*$.

Theorem 1: ADPN M is given, and automator Z receives character string $w = u_1(g, p)u_2$, based on this, the paper structures automator Z_{pre^*} as meeting $L(Z_{pre^*}) = pre^*(L(Z))$. Time complexity for structuring Z_{pre^*} automator is $O(|Q^3| \times |\Delta|)$.

Prove: There is ADPN M whose pattern is $\langle g, \alpha\rangle$, and automator A receives character string $\alpha = u_1 p u_2$, automator $Z = (Q, \Sigma, \delta, q_0, F)$ receiving character string $w = u_1(g, p)u_2$ can be easily structured, then automator $Z_{pre^*} = (Q, \Sigma, \delta', q_0, F)$ is structured, wherein the rule of δ is as follows:

1. If $(g, p\gamma) \xrightarrow{l} (g', p_1w_1) \in \Delta$, and $q \xrightarrow{gp_1w_1}_{\delta} q'$, wherein q, $q \in Q$, then $(q_p, \gamma, q') \in \delta'$ is true, where q_p means the new state arrived after q receives character p.

2. If $(g, p\gamma) \xrightarrow{l} (g', p_1w_1) \triangleright \rho_2w_2 \in \Delta$, and $q \xrightarrow{gp_2w_2p_1w_1}_{\delta} q'$, wherein q, $q \in Q$, then $(q_p, \gamma, q') \in \delta'$ is true, wherein, q_p means the new state arrived after q receives character p.

As state Q is constant, structuring of automator Z_{pre^*} is always terminable. According to rule of migrating δ', *efficient automator structuring method is adopted to structure* Z_{pre^*}, with time complexity of structuring being $O(|Q^3| \times |\Delta|)$. Let $\langle g, \alpha \rangle \in pre^*(L(Z))$, and $\langle g, \alpha \rangle \xrightarrow{l} \langle g, \alpha' \rangle$, then $\langle g, \alpha' \rangle \in L(Z)$. Automator Z receives pattern $\langle g, \alpha' \rangle$. It is derived from structuring rule of automator Z_{pre^*} that $\langle g, \alpha \rangle \in L(Z_{pre^*})$. Let $\langle g, \alpha \rangle \in L(Z_{pre^*})$, and $\langle g, \alpha \rangle \xrightarrow{l} \langle g, \alpha' \rangle$, it is derived from structuring rule of Z_{pre^*} that $\langle g, \alpha' \rangle \in L(Z)$, and it is concluded that $\langle g, \alpha \rangle \in pre^*(L(Z))$. So $L(Z_{pre^*}) = pre^*(L(Z))$ is true. q.e.d.

To combine the state of hedge-automator and corresponding ADPN's local state into a new state, and realize execution mode of tree semantics for migration sequence, frist a special ADPN model $M \times T = (G, P \times S, \Gamma, L, \Delta_l, \Delta_g)$ is structured, whose pattern is $C' \times T \subseteq Conf_{M \times T}$, and form is as follows:

$C' \times T: = \{(g,(p_1, s_1)w_1 \ldots (p_n,s_n)w_n) | (g,p_1w_1 \ldots p_nw_n) \in C' \wedge \forall i.1 \leq i \leq n. \exists A_1.s_i \rightarrow A_1 \in D_L \wedge \langle g,p_iw_i \rangle \in L(A_1)\}$

The new pattern $C' \times T$ combines state in hedge-automator H with the local state of each of its corresponding model M, with form being $(g,(p,s)\gamma)$.

Global migration rule Δ_g of ADPN model $M \times T$ is defined as follows:

[Non-dynamic creation rule]

$(g,(p,s)\gamma) \xrightarrow{l} (g',(p',s')w) \in \Delta'_g$ is true, when and only when $(g, p\gamma) \xrightarrow{l} (g', pw) \in \Delta_g \wedge s \rightarrow s' \in D_N$ is true.

[Dynamic creation rule] $(g,(p,s)\gamma) \xrightarrow{l} (g',(p',s')w) \triangleright (p_s,s_s)w \in \Delta'_g$ is true,

when and only when $(g, p\gamma) \xrightarrow{l} (g, p'w) \triangleright p_sw_s \in \Delta_g \wedge s \rightarrow s' \triangleright s_s \in D_S$ is true.

Local migration rule Δ_l is defined as follows:

[Non-dynamic creation rule]

$(p,s)\gamma \xrightarrow{l} (p',s')w \in \Delta'_l$ is true, when and only when $p\gamma \xrightarrow{l} pw \in \Delta_l \wedge s \xrightarrow{l} s' \in D_N$ is true.

[Dynamic creation rule] $(p,s)\gamma \xrightarrow{l} (p',s')w \triangleright (p_s,s_s)w \in \Delta'_g$ is true, when and only when $p\gamma \xrightarrow{l} p'w \triangleright p_s w_s \in \Delta_g \wedge s \xrightarrow{l} s' \triangleright s_s \in D_s$ is true.

Definition 2 (backward reachable pattern under tree semantics). There is given ADPN M, whose pattern $C' = (g, \alpha)$, $C' \in Conf_M$, where α is corresponding DPN pattern. As pattern α can be received by automator A, the form can also be $C' = (g, A)$. It is known that execution tree $H \subseteq T_M^*$, and backward reachable patterns set based on tree semantics is defined as follows:

$pre_M[H](C') := \{(g,\alpha) \in Conf_M \mid \exists (g',\alpha') \in C', h \in H.(g,\alpha) \overset{h}{\Rightarrow}_M (g',\alpha')\}$ where H is regular execution tree. As C' is regular set, so $pre_M[H](C')$ is also kept regular, and $pre_M[H](C')$ can be received by structuring automator.

However, reachability problem of ADPN under tree sementics remain undecidable, and context-bounded technology is adopted to solve this problem. Definition $pre_{K,M}[H](C')$ represents the K-context-bounded backward reachable patterns set that starts from pattern C' under model M.

Calculation of $pre_{K,M}[H](C')$ can be divided into K steps: first calculate $pre_{1,M}[H](C')$, i.e., calculate backward patterns set starting from pattern C' and with context switching not occurring. For all patterns set $pre_{1,M}[H](C')$ calculated from previous context, and for all $g' \in G$, if $(g', u') \in pre_{1,M}[H](C')$, then calculate $pre_{1,M}[H](g', u')$. Calculate recursively like this until finishing calculating K contexts, to get K-context-bounded backward reachable patterns set. Following is analysis on how to calculate $pre_{1,M}[H](g, u)$.

As DPN automator A receives character string $u_1(p, s)u_2$, automator Z receiving a character string $u_1(g,(p,s))u_2$ can be structured, where $u_1(p, s)u_2 \in L(A)$, $u_1 \in ((P, S)\Gamma^*)^*, p \in P, u_2 \in \Gamma^*((P, S)\Gamma^*)^*$. According to theorum 1, the automator Z_{pre^*} can be structured, hence, following is derived:

$$pre_{1,M}[H](g,u) = \bigcup_{g' \in G} (g', \{w \in (P\Gamma^*)^+ : w = upu' \cap \exists u(g', p)u' \in L(Z_{pre^*})\})$$

The patterns set $pre_{1,M}[H](C')$ calculated each time is of $(G,((P, S)\Gamma^*)^*)$ form. In local state, S represents execution tree represented by hedge-automator in calculating reachable pattern. To get the patterns set of original model, a projection operation $proj_T: 2^{Conf_{M \times T}} \to 2^{Conf_M}$ is defined.

$proj_T(C) = \{(g,p_1w_1 \ldots p_nw_n) \mid \exists s_1,\ldots s_n \in S.\ s_1,\ldots s_n \in L(A_0) \wedge (g,(p_1, s_1)w_1 \ldots (p_n, s_n)w_n) \in C\}$

A backward reachable algorithm can be derived from above analysis, as shown in Figure 2.

```
Input: ADPN M，tuple M = (g, A) and integer k and pattern C
Output: Backward reachable patterns set pre*(C)
1. Structure hedge-automator T according to the rule as stated in section 3;
2. M'←M×T; //Structure a new model according to model M and automator T of M's execution tree.
3. reachable←ϕ, level = 0; //initialize reachable set and context-boundary identifier.
4. reachable←pre 1,M[H](g, A); //Calculate all reachable patterns in the first context.
5. C0←pre1, M[H](g, A); //intermediate result is temporarily saved in C0 set.
6. level++;
7. A = local (C0∩(g', (PΓ*)+)); //Update local state pattern.
8. while(level≠k) // Time of context-boundary is K.
10. {for all g'∈G.
11. {A = local (Clevel -1 ∩ (g', (PΓ*)+)); //Update local state pattern.
12. Clevel←pre1, M[H](g', A); } //intermediate result is temporarily saved in Clevel set.
13. reachable←Clevel; //Put all reachable patterns in each context into reachable.
14. level++;}.
15. projT(reachable); //Conduct projection operation for patterns set.
```

Figure 2. Reverse Reachability Algorithm Based on Tree Semantics

The first line simulates execution mode of tree semantics for migration sequence by structuring an intermediate model *M'*. *while* cyclically controls *K* context-boundaries. *local*() represents the operation of projecting ADPN patterns to DPN pattern. C_{level} represents calculating reachable pattern under the *level*th context. $pre_{1,M}[H]()$ operation represents all possible reachable patterns sets in each context. As the patterns sets from reach able sets are patterns set of intermediate model *M*', the pattern of reach able pattern set *reachable* is subjected to projection to get reachable patterns set under model *M*.

5. Analysis on Algorithm

The complexity of algorithm suggested herein is $O(|G|^{k-1}\times|Q^3|\times|\Delta|)$. The complexity of new model structured via given model *M* and execution tree H is linear complexity. According to theorem 1, the time complexity for structuring of automator Z_{pre^*} each time is $|Q^3|\times|\Delta|$, as the time for context-boundary is k, the execution sequence $g_1 \ldots g_{k-1}\in G^{k-1}$ for *for* loop in the worst case has $|G|^{k-1}$ possibilities. $proj_T()$ function is linear complexity, hence the complexity of algorithm is $O(|G|^{k-1}\times|Q^3|\times|\Delta|)$, which has certain advantages over the algorithms such as k-delimited DPN reachability analysis, etc.

6. Conclusion

The tree semantics simulates the running of concurrent program more accurately than interleaving semantics and extends modeling capacity of model (such as being able to show which thread the migration belongs to), and can more efficiently conduct backward reachability analysis than interleaving semantics. To make the reachability problem decidable, the algorithm adopts context-bounded approach to make backward reachability analysis on ADPN, and structures an intermediate model to realize

execution mode of semantics. Besides, it combines the state of hedge-automator and corresponding model state into a new state to calculate reachable pattern of this intermediate model. Lastly, projection operation is conducted to get the patterns set on original model, and complexity of algorithm is presented to well solve the reachability problem under this model. The key content in future research will be following two aspects: 1) Combine summarization technology and optimize the algorithm by dint of summary course being abstract and able to be reused for many times; 2) As the algorithm herein can only be used in abstract models with limited global variables, next it is considered to extend it to abstract models with infinite global variables.

References

Abdulla, P. A. et al. (2014). Budget-bounded model-checking pushdown systems. *Formal Methods in System Design*, *2014*, 273-301. https://doi.org/10.1007/s10703-014-0207-y

Atig, M. F., Abdulla, P. A., Kumar, K. N., & Saivasan, P. (2012). Linear time model-checking for multithread programs under scope-bounding//proceedings of the 10th international symposium on automated technology for verification and analysis. *Thiruvananthapuram*, *2012*, 152-166.

Bouajjani, A., Esparza, J., Schwoon, S., & Strejcek, J. (2005). Reachability analysis of multithreaded software with asynchronous communication. In *Proc. of FSTTCS* (Vol. 2005, pp. 348-359). Hyderabad: Springer. https://doi.org/10.1007/11590156_28

Bouajjani, A., Müller-Olm, M., & Touili, T. (2005). Regular symbolic analysis of dynamic networks of pushdown systems. In *Proceedings of the 16th International Conference on Concurrency Theory* (pp. 473-487). https://doi.org/10.1007/11539452_36

Emmi, M. et al. (2015). Analysis of Asynchronous Programs with Event-Based Synchronization. In *Programming Languages and Systems* (pp. 535-559). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-662-46669-8_22

Faouzi Atig, M., Bouajjani, A., & Qadeer, S. (2009). Context-bounded analysis for concurrent programs with dynamic creation of threads. In *Proceedings of the 15th International Conference on TACAS, LNCS 5505* (Vol. 2009, pp. 107-123). Univ York: European Assoc.

Gawlitza, T. M., Lammich, P., Müller-Olm, M., Seidl, H., & Wenner, A. (2011). Join-Lock-Sensitive forward reachability analysis for concurrent programs with dynamic process creation. In *Proceeding of the 12th international conference on verification, model checking, LNCS 6538* (pp. 199-213). https://doi.org/10.1007/978-3-642-18275-4_15

Irigoin, F., Jouvelot, P., & Triolet, R. (2014). Semantical interprocedural parallelization: An overview of the PIPS project. In *25th Anniversary International Conference on Supercomputing Anniversary* (Vol. 2014, pp. 143-150). https://doi.org/10.1145/2591635.2667163

Kahlon, V., & Gupta, A. (2006). An automata-theoretic approach for model checking threads for LTL properties. In *Proceedings of the 21st Annual IEEE Symposium on Logic in Computer Society* (pp. 101-110). Swattle: LICS 2006.

Lammich, P., Müller-Olm, M., & Wenner, A. (2009). Predecessor sets of dynamic pushdown networks with tree-regular constraints. In *Proceedings of CAV* (pp. 525-539). Grenoble: Artist Design. https://doi.org/10.1007/978-3-642-02658-4_39

Li, X., & Qian, J. Y. (2016). Research on k-delimited accessibility analysis for dynamic pushdown network. *Journal of Guilin university of electronic technology*, *36*(1), 48-51.

Musuvathi, M., & Qadeer, S. (2007). Iterative context bounding for systematic testing of multithreaded programs. *Conference on PLDI, 2007*, 446-455. https://doi.org/10.1145/1250734.1250785

Pun, K. I., Steffen, M., & Stolz, V. (2014). Effect-polymorphic behaviour inference for deadlock checking. In *Software Engineering and Formal Methods* (pp. 50-64). Springer International Publishing. https://doi.org/10.1007/978-3-319-10431-7_5

Qian, J. Y. et al. (2016). Verification of real-time systems with time multi-pushdown net. *Chinese journal of computers*, *39*(11), 2253-2569.

Wenner, A. (2010). Weighted dynamic pushdown networks. In *Programming Languages and Systems* (Vol. 2010, pp. 590-609). Heidelberg: Springer. https://doi.org/10.1007/978-3-642-11957-6_31