Model Checking Dynamic Pushdown Networks

Fu Song\(^1\) and Tayssir Touili\(^2\)

\(^1\) Shanghai Key Laboratory of Trustworthy Computing, East China Normal University
fsong@sei.ecnu.edu.cn
\(^2\) LIAFA, CNRS and Université Paris Diderot
touili@liafa.univ-paris-diderot.fr

Abstract. A Dynamic Pushdown Network (DPN) is a set of pushdown systems (PDSs) where each process can dynamically create new instances of PDSs. DPNs are a natural model of multi-threaded programs with (possibly recursive) procedure calls and thread creation. Thus, it is important to have model-checking algorithms for DPNs. We consider in this work model-checking DPNs against single-indexed LTL and CTL properties of the form $\bigwedge f_i$ s.t. $f_i$ is a LTL/CTL formula over the PDS $i$. We consider the model-checking problems w.r.t. simple valuations (i.e., whether a configuration satisfies an atomic proposition depends only on its control location) and w.r.t. regular valuations (i.e., the set of the configurations satisfying an atomic proposition is a regular set of configurations). We show that these model-checking problems are decidable. We propose automata-based approaches for computing the set of configurations of a DPN that satisfy the corresponding single-indexed LTL/CTL formula.

1 Introduction

Multithreading is a commonly used technique for modern software. However, multithreaded programs are known to be error prone and difficult to analyze. Dynamic Pushdown Networks (DPN) \cite{4} are a natural model of multi-threaded programs with (possibly recursive) procedure calls and thread creation. A DPN consists of a finite set of pushdown systems (PDSs), each of them models a sequential program (process) that can dynamically create new instances of PDSs. Therefore, it is important to investigate automated methods for verifying DPNs. While existing works concentrate on the reachability problem of DPNs \cite{4,18,17,15,24}, model checking for the Linear Temporal Logic (LTL) and the Computation Tree Logic (CTL) which can describe more interesting properties of program behaviors has not been tackled yet for DPNs.

In general, the model checking problem is undecidable for double-indexed properties, i.e., properties where atomic propositions are interpreted over the control states of two or more threads \cite{11}. This undecidability holds for pushdown networks even without thread creation. To obtain decidable results, in this paper, we consider single-indexed LTL and CTL model checking for DPNs, where a single-index LTL or CTL formula is a formula of the form $\bigwedge f_i$ such that $f_i$ is a LTL/CTL formula over the PDS $i$. 

\(*\) This work is partially funded by ANR grant ANR-08-SEG1-006, Shanghai Knowledge Service Platform for Trustworthy Internet of Things No. ZF1213, NSFC Project No.91118007, Civil Aerospace Project 125 and NSFC Project No.61021004.
A DPN satisfies \( \bigwedge f_i \) iff every PDS \( i \) that runs in the network satisfies the subformula \( f_i \). We first consider LTL model-checking for DPNs with simple valuations where whether a configuration of a PDS \( i \) satisfies an atomic proposition depends only on the control state of the configuration. Then, we consider LTL model-checking for DPNs with regular valuations where the set of configurations of a PDS satisfying an atomic proposition is a regular set of configurations. Finally, we consider CTL model-checking for DPNs with simple and regular valuations. We show that these model-checking problems are decidable. We propose automata-based approaches for computing the set of configurations of a DPN that satisfy the corresponding single-indexed LTL/CTL formula.

It is non-trivial to do LTL/CTL model checking for DPNs, since the number of instances of PDSs can be unbounded. Checking independently whether all the different PDSs satisfy the corresponding subformula \( f_i \) is not correct. Indeed, we do not need to check whether an instance of a PDS \( j \) satisfies \( f_j \) if this instance is not created during a run. To solve this problem, we extend the automata-based approach for standard LTL/CTL model-checking for PDSs \([2,8,7,20]\). For every process \( i \), we compute a finite automaton \( A_i \) recognizing all the configurations from which there exists a run \( \sigma \) of the process \( i \) that satisfies \( f_i \). \( A_i \) also memorizes the set of all the initial configurations of the instances of PDSs that are dynamically created during the run \( \sigma \). Then, to check whether a DPN satisfies a single-indexed LTL/CTL formula, it is sufficient to check whether the initial configurations of the processes are recognized by the corresponding finite automata and whether the set of generated instances of PDSs that are stored in the automata also satisfy the formula. This condition is recursive. To solve it, we compute the largest set \( D_{fp} \) of the dynamically created initial configurations that satisfy the formula \( f \). Then, to check whether a DPN satisfies \( f \), it is sufficient to check whether the initial configurations of the different processes are recognized by the corresponding finite automata and whether the dynamically created initial configurations that are stored in the automata are in \( D_{fp} \).

To compute the finite automata \( A_i \)s, we extend the automata-based approaches for standard LTL \([2,7,8]\) and CTL \([20]\) model-checking for PDSs. For every \( i, 1 \leq i \leq n \), we construct a Büchi Dynamic PDS (resp. alternating Büchi Dynamic PDS) which is a synchronization of the PDS \( i \) and the LTL (resp. CTL) formula \( f_i \). Büchi Dynamic PDS (resp. alternating Büchi Dynamic PDS) is an extension of Büchi PDS (resp. alternating Büchi PDS) with the ability to create new instances of PDSs during the run. The finite automata \( A_i \)s we are looking for correspond to the languages accepted by these Büchi Dynamic PDSs (resp. alternating Büchi Dynamic PDSs). Then, we show how to solve these language problems and compute the finite automata \( A_i \)s.

**Related Work.** The DPN model was introduced in \([4]\). Several other works use DPN and its extensions to model multi-threaded programs \([4,9,17,18,24]\). All these works only consider reachability issues. Ground Tree Rewrite Systems \([10]\) and process rewrite systems \([5,19]\) are two models of multi-threaded programs with procedure calls and threads creation. However, \([19]\) only considers reachability problem and \([10,5]\) only consider subclasses of LTL. We consider LTL and CTL model checking problems.

Pushdown networks with communication between processes are studied in \([3,6,1,22]\). These works consider systems with a fixed number of threads. \([15,16]\) use