Science of Computer Programming 95 (2014) 320–342

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Static analysis of lists by combining shape and numerical abstractions

Liqian Chen *, Renjian Li, Xueguang Wu, Ji Wang

National Laboratory for Parallel and Distributed Processing, National University of Defense Technology, Changsha 410073, China

HIGHLIGHTS

- We propose a light-weight shape abstraction for lists based on bit vectors.
- We use numerical abstractions to infer relations among sizes of list segments.
- We combine shape and numerical abstractions to analyze programs with lists.
- We extend our approach to fit for analyzing circular lists.
- Experiments show that our approach find intricate length properties for lists.

ARTICLE INFO

Article history:
Received 6 July 2013
Received in revised form 29 May 2014
Accepted 2 June 2014
Available online 12 June 2014

Keywords:
Static analysis
Abstract interpretation
Lists
Abstract domains
Shape analysis

ABSTRACT

We present an approach in the framework of abstract interpretation to analyze list-manipulating programs by combining shape and numerical abstractions. The analysis automatically divides a list into non-overlapping list segments according to the reachability property of pointer variables to list nodes. The list nodes in each segment are abstracted by a bit-vector wherein each bit corresponds to a pointer variable and indicates whether the nodes can be reached by that pointer variable. Moreover, for each bit-vector, we introduce an auxiliary integer variable, namely a counter variable, to record the number of nodes in the segment abstracted by that bit-vector. On this basis, we leverage the power of numerical abstractions to discover numerical relations among counter variables, so as to infer relational length properties among list segments. Furthermore, we show how our approach works for circular lists. Our approach stands out in its ability to find intricate properties that involve both shape and numerical information, which are important for checking program properties such as memory safety. A prototype is implemented and preliminary experimental results are encouraging.

1. Introduction

Invariants involving both shape and numerical information are crucial for checking nontrivial program properties in heap manipulating programs, such as memory safety, termination, bounded size of heap memory. However, automatically inferring such invariants is challenging, especially for programs manipulating dynamic linked data structures. In this paper, we consider the problem of analyzing programs manipulating lists. And inferring such invariants over lists requires considering both the shape of lists and the numerical information over the number of the list nodes.

* Corresponding author.
E-mail address: lqchen@nudt.edu.cn (L. Chen).

http://dx.doi.org/10.1016/j.scico.2014.06.004
0167-6423/

Shape analysis provides a powerful approach to generate shape invariants, and much progress has been achieved in shape analysis in the past two decades [1,2]. However, shape analysis itself has limited ability in inferring non-trivial properties involving numerical information such as "sum of traversed and remained list sizes equals to the input list size". On the other hand, numerical static analysis by abstract interpretation [3] is widely adopted to automatically generate numerical invariants for programs. However, most of existing abstract domains focus on purely numerical properties and thus are specific for analyzing numerical programs. A recent interesting trend is to combine these two techniques, using shape analysis to generate shape invariants and using numerical abstract domains to track numerical relationships [4–7]. The key technical issue here is how to interact effectively between the shape aspect and the numerical aspect. Although several generic frameworks for the combination have been proposed [4,6], tighter bidirectional coupling between the two aspects still needs further considerations for the selected shape abstraction and numerical abstraction. For instance, shape abstraction needs to be enhanced to support numerical aspects while numerical abstraction also needs to be adapted with respect to the semantics of shape abstraction. And transfer functions for the combined domain should be designed by taking into account both the shape and numerical information at the same time.

In this paper, we present an approach in the framework of abstract interpretation to combine shape and numerical abstractions for analyzing programs manipulating lists. First, for the shape of a list, we propose a lightweight shape abstraction based on bit-vectors, upon the insight that list nodes in a list can be naturally grouped into non-overlapping list segments according to the reachability property of pointer variables to list nodes. Each list segment which includes those list nodes that can be reached by the same set of pointer variables, is abstracted by one bit-vector wherein each bit corresponds to a pointer variable in the program. From the numerical aspect, in order to track the number of list nodes in each list segment, we introduce an auxiliary (nonnegative) integer counter variable for each bit-vector. And we apply numerical abstract domains to infer numerical relations among counter variables. Then, transfer functions for the combined domain are constructed in terms of transfer functions of the numerical domain upon the semantics of the shape abstraction. Specifically, in this paper we instantiate our approach by using a combination of intervals [8] and affine equalities [9] to conduct numerical abstractions. For the sake of better understanding, we first focus on non-circular lists and then consider extensions for circular lists. On this basis, a prototype is implemented and preliminary experimental results are presented on benchmark programs.

This paper is an extended version of our SAC 2013 paper in the Software Verification and Testing track [10]. On top of [10], we have added formal descriptions of maintaining points-to sets for pointer variables (Section 5.2). We have extended our approach to fit for circular lists (Section 6). We have conducted more experiments over programs manipulating circular lists and programs that may cause memory errors (Section 7). We have also provided details of proofs (in Appendix A).

The rest of the paper is organized as follows. Section 2 describes a simple list-manipulating programming language. Section 3 presents a shape abstraction approach for non-circular lists based on bit-vectors. Section 4 presents a combined domain of intervals and affine equalities to conduct numerical abstractions over counter variables. Section 5 shows how to perform analysis of programs manipulating non-circular lists based on the proposed abstractions. Section 6 extends the approach to fit for circular lists. Section 7 presents our prototype implementation together with preliminary experimental results. Section 8 discusses related work before Section 9 concludes.

2. List-manipulating programming language

We first present a small language that manipulates lists. The syntax of our language is depicted in Fig. 1. It is a simple procedure-less sequential language with dynamic allocation and deallocation but no recursion. There is only one type of variables, i.e., pointer variables of LIST type, denoted as *PVar*. For the sake of simplicity, we first focus on non-circular singly-linked list.

The structure for list nodes contains a *next* field pointing to the successive list node, while all other fields are considered as data fields. The data fields are ignored in this paper, since we assume that operations over data fields have no influence on the shape of lists. We assume that there is at most one *next* operator in a statement and a pointer variable appears at most once in an assignment statement. All other cases could be transformed into this form by introducing temporary variables.

3. Shape abstraction for non-circular lists

First, we recall the definition of classic shape graph that is a graph used to represent the allocated memory in heap.

Definition 1. A shape graph for lists is a tuple $SG = \langle N, V, E \rangle$, where:

- N denotes the set of pointer variables and list nodes, and we utilize N_{nil} to denote $N \cup \{NULL\}$,
- $V \subseteq N$ denotes the set of pointer variables in the program,
- $E \subseteq N \times (N_{nil} - V)$ denotes the set of edges, which describes the points-to relations of pointer variables as well as successive relations between list nodes through the "*next*" field.