1 INTRODUCTION

Classical logics, such as first-order logic with least fixpoints or higher order logics, are often static in the sense that formulae describe the state of a single world. Verification conditions of imperative programs describe typically at least two different worlds: the pre-state and the post-state of a program. Consequently, expressing validity of verification conditions naturally involves the challenge of expressing the evolving worlds of program states in a static logic. The classical notion of strongest postcondition for programs with scalar variables does precisely this— it expresses the precondition, the intermediate states of the program, and the post-state using *auxiliary* first-order variables that capture the scalar variables at these states. The weakest precondition, again for programs with scalar variables, solves the same problem.

The focus of this paper is in generating logical formulations of verification conditions for program snippets that manipulate the heap. The pre- and post-conditions are written in quantifier-free FO + *lfp*. A lot of complex and interesting properties of heap manipulating programs can be expressed using recursively defined functions and predicates, such as 'x points to a linked list segment ending at z', 'the maximum element stored in the list pointed to by x'. In this setting, the heap consists, minimally, of a set of pointer fields that are modeled by *first-order functions*, and the program's execution alters these functions. Consequently, the translation of Hoare triples to validity of verification conditions is considerably harder, in comparison with programs with only scalar variables. This is further complicated by the presence of recursively defined functions/predicates that refer to data fields, which can express complex properties combining both shape and data on the heap.

There are two general ways of translating Hoare triples to verification conditions in this setting. The first is to accurately capture the Hoare triple; this typically involves two versions of the recursive functions, one for the pre-state and one for the post-state, where the two are parameterized by different sets of pointer fields. Furthermore, the verification conditions introduces quantification to precisely capture the heaplet that is changed by the program snippet. All of this makes automated reasoning of the verification condition extremely complex.

The second way is to model the heap change as an arbitrary change of the modified portion of the heap, and use *frame reasoning* to argue properties are conserved across the heap transformation. This is at the heart of the design of separation logic. However, frame reasoning alone is not complete.

For program snippets that involve function calls, we believe that frame reasoning is appropriate, and continue recommending it. However, for program snippets without function calls, we argue in this paper that an accurate encoding of the verification condition is possible that is also amenable to automated reasoning.

In this paper, we identify a new class of logics called *Delta-logics* for expressing verification conditions that addresses the above problem. Formulae in delta-logics are Boolean combinations of two kinds of formulae: one, *delta-specific* formulae, talk about the modified portion of the global heap, which we call Δ, without using any recursive definitions; the other kind, called *contextual/context-logic* formulae, strictly talk about the unbounded portion excluding Δ, and uses recursive definitions. A set of first-order interface variables (called *parameters*) are used to communicate information between Δ and the rest of the heap.

We motivate the desirable aspects of delta-logics for expressing verification conditions, and provide an overview of the mechanism of VC generation using delta-logics in Section **??**.

When translating Hoare triples to VCs in delta-logic, the program snippets translate naturally as delta-logic clearly delineates the modified heaplet from its context, and the program's transformation is a delta-specific formula. A technical challenge, however, is to express the pre- and post-conditions which are on both the modified and unmodified portions of the heap. We prove a key theorem, called the *Separability Theorem* that shows that any quantifier-free FO formula with recursive definitions can be expressed in delta-logics, i.e., as a boolean combination of delta-specific formulae and contextual formulae. We shall see that this separation is nontrivial, and requires the definition of new parameterized recursive functions called *ranks* for each parameterized recursive definition.

The simplicity of expressing VCs in delta logics enables us to design powerful *decidable* program logics. We define a delta-logic that expresses properties of list segments along with a varied collection of of measures on them, including their heaplets (for expressing separation properties), their lengths, the multisets of keys stored in them, the min/max keys stored in them, and their sortedness. We show that these delta-logic formulae can be translated to equisatisfiable quantifier-free formulae *without* recursive definitions. This leads us to a decision procedure for delta-logics for linked lists with all the six measures above. To the best of our knowledge, this is the most expressive decidable program logic for list manipulating programs.

We emphasize that though delta logics are particularly meant for program snippets that do not involve function calls, we show that it can be seamlessly combined with frame reasoning for function calls, leading us to define a comprehensive verification technique for programs with function calls.

Finally, we implement and evaluate our technique by expressing VCs using delta-logics and validating them using our decision procedure on a suite of programs, and show it to be effective both for verifying correct programs and finding bugs in incorrect ones.

The main contributions of this paper are:

- The construction of Delta-Logics, which are static and simple logics for expressing properties about heap-manipulating programs, and are program logics in which precise VCs for basic blocks without function calls can be generated.
- A nontrivial *Separability Theorem* that translates quantifier-free FO $+lfp$ formulae into delta-logic formulae, using ranks and only first-order communication variables.
- A VC generation technique for expressing VCs in delta-logic using the separability theorem that is amenable to automated reasoning.
- A powerful decidable delta-logic over lists and list-measures, and a novel quantifier-free encoding for a fragment of the contextual logic of lists and list-measures.
- An extension of the VC generation technique to incorporate frame reasoning with delta-logic, and consequently a decidability result for VCs of list-manipulating programs with function calls.
- Experimentation and evaluation of the VC generation using decidable delta-logics and their extension, on a suite of list-manipulating programs, that is effective both in the verification of correct programs and is meaningful in the indication of counterexamples for faulty programs.

The paper is organized as follows. In Section **??**, we motivate delta-logics, the separability theorem, an overview of VC generation using delta-logics, and provide an illustrative motivating example. In Section **??** we formally define delta-logics, delta-specific formulae, and contextual formulae. In Section **??** we state the separability theorem, which is a nontrivial theorem that is crucial in the generation of delta-logic VCs. Section **??** discusses the VC generation mechanism for delta-logics from Hoare triples for basic blocks without function calls. In Section **??** we define a powerful decidable delta-logic of lists and list-measures and provide a decision procedure. Combined with the VC generation technique, this provides a decision procedure for the program logic of list-manipulating programs without function calls. In Section **??** we extend the VC generation technique to verify list-manipulating programs with function calls by incorporating frame-reasoning with delta-logics. In Section **??** we discuss the implementation and evaluation of our technique on a suite of list-manipulating programs. Finally, in Section **??** we discuss and compare our work with existing literature.