1 INTRODUCTION

–what is the problem being addressed
–how is it being addresed: structure of the paper
–focus on decidability
– main contributions

INTRO STARTS HERE

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 *auxilliary* 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*, for a lot of interesting and important properties of heap manipulating programs can be expressed using recursively defined functions and predicates. 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 prescence of recursively defined functions/predicates that refer to data fields, which can express complex properties combining both shape and data on the heap. However, traditional ways of reducing validation of verification conditions to logic embed the verifcation in logics that have much more expressive power than necessary, and harder to reason with; we think it is a reduction from an easier problem to a harder problem!

We identify a new class of logics that can express the verification conditions for such programs precisely, while at the same time being amenable to automated reasoning: we call these *Delta-logics*. We motivate these logics and provide a broad overview of our method in Section **??**. We also give definitions and discuss the formalism in detail in Section **??**. We argue that VCs must be generated in delta-logic. To this end we prove a crucial theorem in Section **??**, which we call the *Separability Theorem*, that helps us generate these delta-logic VCs from Hoare Triples of basic blocks without function calls (where the pre- and post-conditions are written in quantifier-free FO+*lfp*). To complete our goal, we show how to generate VCs for such basic blocks using the separability theorem in Section **??**.

We then turn to specific delta-logics and explore decidability results for them in Section **??**. We define a delta-logic that expresses properties of list segments along with a variety 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 consider the contextual logic corresponding to this delta-logic, and by exploiting the simplicity of delta-logics, we show that they can be transformed to equivalent 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, and hence a decidable logic for verification conditions of programs manipulating linked lists with pre- and post-conditions expressed using the above measures.

For this delta-logic of lists and list-measures, we also bridge the gap to verifying full programs and not just basic blocks without function calls by illustrating a more general technique for VC generation in Section **??**. We do this in particular for this delta-logic, because the decidability result and the consequent method for efficient verification is one of the main contributions of this paper, and this generalised technique explains the VC generation mechanism for full-fledged list-manipulating programs with function calls (our experiments include such programs!). In addition, while the technique is certainly amenable to the general case, it is better illustrated via this particular logic.

Lastly, 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. This is discussed in Section **??**. To the best of our knowledge, this is the most expressive decidable logic over lists in existing literature.

The main contributions of this paper are: –**LIST MAIN CONTRIBUTIONS**