Synthesizing Invariants via Iterative Learning of Decision Trees

Pranav Garg, Daniel Neider, P. Madhusudan, and Dan Roth

University of Illinois at Urbana-Champaign, USA

Automatically synthesizing inductive invariants (i.e., statements about the configurations of a program that remain true during its execution) lies at the heart of automated program verification. Once an inductive invariant is found or given by the user, the problem of checking whether the program satisfies a specification can be reduced to logical validity of so-called verification conditions, which is increasingly tractable with the advances in automated logic solvers.

In recent years, the *black-box* or *learning* approach to finding invariants has gained popularity. In this data-driven approach, the synthesizer of invariants is split into two components. One component is a *teacher*, which is essentially a program verifier that can verify the program using a conjectured invariant; in case that the conjecture is inadequate to verify the program against the specification, the teacher can return a counterexample (i.e., a concrete program configuration) as a witness to why this is the case. The other component is a *learner*, which learns a candidate invariant from counterexamples returned by the teacher. Both components are run in alternation until the learner proposes an invariant that is adequate to verify the program.

One of the biggest advantages of the black-box learning paradigm is the possible usage of *machine learning techniques* to synthesize invariants. The learner, being completely agnostic to the program (its programming language, semantics, etc.), can be seen as a machine learning algorithm that learns a Boolean classifier of program configurations. However, Garg et al. [2] argue that learning from positively or negatively classified program configurations alone does not form a robust paradigm for learning invariants. Instead, the authors propose to allow the teacher to return implications $x \Rightarrow y$, where both x and y are program configurations, with the constraint that the learner must propose a classifier such that if x is classified as true, so is y (intuitively, implications capture the semantics of a program). The resulting learning setting is called *ICE learning* (short for implication-counterexamples) and entails that the learner now additionally needs to handle implications.

The goal of our paper [1] is to adapt Quinlan's decision tree learning algorithms to the ICE learning model in order to synthesize invariants. Our key contributions are: (i) a generic top-down decision tree learning algorithm that learns from training data with implications; (ii) several novel "information gain" measures that are used to determine the best attribute to split on the current collection of examples and implications; (iii) a convergence mechanism that guarantees learning an invariant if one exists; and (iv) an extensive experimental evaluation.

1. Pranav Garg, P. Madhusudan, Daniel Neider, and Dan Roth. *Learning Invariants using Decision Trees and Implication Counterexamples.* In POPL 2016, pages 499–512. ACM, 2016.
2. Pranav Garg, Christof Löding, P. Madhusudan, and Daniel Neider. *ICE: A Robust Framework for Learning Invariants.* In CAV 2014, volume 8559 of LNCS, pages 69–87. Springer, 2014.