Instantiation of Existentially Quantified Variables in Inductive Specification Proofs

Brigitte Pientka* and Christoph Kreitz

Department of Computer Science, Cornell University
Ithaca, NY 14853-7501, U.S.A.
{pientka, kreitz}@cs.cornell.edu

Abstract. We present an automatic approach for instantiating existentially quantified variables in inductive specifications proofs. Our approach uses first-order meta-variables in place of existentially quantified variables and combines logical proof search with rippling techniques. We avoid the non-termination problems which usually occur in the presence of existentially quantified variables. Moreover, we are able to synthesize conditional substitutions for the meta-variables. We illustrate our approach by discussing the specification of the integer square root.

1 Introduction

Constructive type theory [12] offers the unique advantage of total correctness of synthesized programs. In this setting a specification is of the form

\[ \forall \text{input}. \exists \text{output}. \ spec(\text{input, output}) \]

where \text{input} is a vector of arguments, \text{output} is a result and \text{spec} is a proposition describing the required relation between them. A program meeting this specification can be extracted from its proof via the proofs-as-programs principle [3]. This style is widely advocated [13] and supported in a number of implementations such as NuPRL [8]. The application of such systems however is limited by its low degree of automation. In order to overcome this drawback, we suggest incorporating techniques from inductive theorem proving.

The first difficult step within a proof is the choice of the appropriate induction scheme. Different induction schemes result in algorithms which differ in their complexity. In this paper we focus on the second crucial step during the induction step, the instantiation of existentially quantified variables. The witness for an existentially quantified variable corresponds to the recursive calls in the program. Sometimes a case split is necessary before decomposing the existential quantifier. The existentially quantified variables are then instantiated according to the cases.

A standard technique to deal with existentially quantified variables is to use meta-variables in place of the existential witness and allow the application of

* The research reported is supported by the Gottlieb Daimler and Karl Benz Foundation with a fellowship to the first author.
logical rules that refine the goal. To complete the proof, a unification procedure provides the instantiation of the meta-variables. In inductive specification proofs, standard unification techniques are not sufficient; we need to rewrite the expression from the induction conclusion towards the application of the corresponding expression in the induction hypothesis. Both expressions have to be equal after some rewriting steps. The crucial question is how can we find a chain of rewriting steps, such that both expressions can be made equal by rewriting in the presence of meta-variables.

In inductive theorem proving, an annotated rewriting technique, called rippling [7,6], has been used successfully in order to control the rewriting process. However, only little focus has been devoted to the automatic instantiation of existentially quantified variables. In this paper we suggest combining the logic provided by constructive type theory with inductive theorem proving techniques such as rippling, in order to compute valid instantiations for the existentially quantified variables. We use first-order meta-variables during proof search within the sequent calculus. During rippling the meta-variables are treated in the same way as potential “sink variables”. We develop a reverse rippling match that matches the induction conclusion with the induction hypothesis. If this match is successful, it returns an instantiation for the meta-variables and a rippling sequence, that rewrites the instantiated induction conclusion to the induction hypothesis. With this approach we avoid non-termination problems that usually occur in the presence of existentially quantified variables. Moreover, we check the consistency of the remaining subgoals under the synthesized substitution. During this consistency check, we are able to synthesize constraints that form a case split during the proof. We demonstrate the strength of our approach by discussing the proof of the integer square root specification.

In Section 2, we give a brief introduction to rippling and discuss the rippling approaches for dealing with meta-variables. In Section 3 we describe the general idea of our approach and in 4 we consider the proof of the integer square root. We show step-by-step how we can derive conditional substitutions for the existentially quantified variables. In Section 5 we present a more technical description of our method using ML-notation. In Section 6 we describe the formalization in NuPRL. In Section 7 an extension to our technique is presented. We discuss related work concerning program synthesis in Section 8 and finally in Section 9 we outline future work and draw some conclusions.

2 A Brief Introduction to Rippling

Rippling is an annotated rewriting technique that has been successfully applied in inductive theorem proving. Differences between the induction hypothesis (given) and the induction conclusion (goal) are marked by meta-level annotations, called wave annotations. Expressions that appear both in the goal and in the given are called skeleton. Expressions that appear in the goal, but not in the given are called wave-fronts. The induction (or recursive) variable that is surrounded by a wave-front is called wave-hole. Sinks are parts of the goal which