# Lemma Discovery in Automating Induction

**Deepak Kapur** **M. Subramaniam**

Computer Science Department
State University of New York
Albany, NY 12222
kapur@cs.albany.edu, subu@cs.albany.edu

**Abstract.** Speculating intermediate lemmas is one of the main reason of user interaction/guidance while mechanically attempting proofs by induction. An approach for generating intermediate lemmas is developed, and its effectiveness is demonstrated while proving properties of recursively defined functions. The approach is guided by the paradigm of attempting to generate a proof of the conclusion subgoal in an induction step by the application of an induction hypothesis(es). Generation of intermediate conjectures is motivated by attempts to find appropriate instantiations for non-induction variables in the main conjecture. In case, the main conjecture does not have any non-induction variables, such variables are introduced by attempting its generalization. A constraint based paradigm is proposed for guessing the missing side of an intermediate conjecture by identifying constraints on the term schemes introduced for the missing side. Definitions and properties of functions are judiciously used for generating instantiations and intermediate conjectures. Heuristics are identified for performing such analysis. The approach fails if appropriate instantiations of non-induction variables cannot be generated. Otherwise, proofs of intermediate conjectures are attempted and the proposed method is recursively applied. The method has proven to be surprisingly very effective in speculating intermediate lemmas for tail-recursive programs. The method is demonstrated using a number of examples on numbers and lists.

## 1 Introduction

Induction is one of the crucial inference rules that has been extensively used in verifying properties of both software and hardware. Because of its importance, many theorem provers including *Nqthm*, *RRL*, *INKA*, *LP*, *CLAM-Oyster*, *SPIKE*, support methods for mechanizing proofs by induction. In Nqthm, RRL and INKA, terminating recursive function definitions are typically used to generate induction schemes. The induction conclusion is obtained from the left hand sides of the definition, and recursive calls to the function are used to obtain the hypotheses. An inductive proof attempt may fail as the available induction hypotheses cannot be applied to simplify the conclusion. Most of these theorem provers do not provide any support for analyzing and fixing failed proof attempts.[1] Consequently, a user of such systems spends most of the time in the

[1] The only exceptions are the theorem proving systems *CLAM-Oyster* and *SPIKE*. For details, refer to section 2 on related work.

analysis and remedy of failed proof attempts. This is a nontrivial activity that requires considerable ingenuity in terms of (i) analyzing the failed proof attempts, (ii) conjecturing and establishing additional lemmas, (iii) suggesting alternate forms of induction, (iv) modifying the available hypothesis and (v) may even involve coming up with alternate conjectures entailing the given conjecture.

This paper is an attempt to address such weaknesses in induction theorem provers. An approach for speculating intermediate lemmas needed for mechanically attempting proofs by induction is developed. The approach is based on the assumption that for an induction proof attempt to succeed, it must be possible to use the induction hypothesis in every induction case. Hypotheses and conclusion subgoals of failed step cases are analyzed. Instantiations for non-induction variables in the hypotheses are searched, and intermediate conjectures that can render the hypothesis instances applicable to the conclusion, are speculated.

An induction hypothesis could potentially be used at many positions in the conclusion subgoal of a failed induction step case. All such possible positions have to be considered. Based on structural relationships among function symbols in the definitions and lemmas, many of these possibilities can be ruled out by relating the hypothesis to the subterm of the conclusion where the hypothesis may be applicable. A set of equations called the *difference set* is identified. Each difference equation relates the hypothesis and that particular subterm of the conclusion. By decomposing difference equations as well as using definitions and lemmas of function symbols appearing in the conjecture, instantiations for the non-induction variables in the hypothesis are speculated. A set of intermediate conjectures are then generated from the difference equations using the speculated instantiations of non-induction variables. A procedure for speculating intermediate lemmas and hypothesis instances is discussed in sections 3 and 4.

In many cases, the procedure fails to produce appropriate instances of the hypothesis since the conjecture is too specific, for instance, the conjecture may not have any non-induction variables. In such cases it is necessary to obtain generalized forms of the conjecture that entail the given conjecture, and can be automatically proved. The generalized conjectures are obtained by introducing non-induction variables in the given conjecture. Positions where such variables should be introduced are determined by analyzing failed proof attempts. Often such analysis leads to introduction of extra variables on one side of the conjecture in which case the other side of the generalized conjecture must be synthesized. *Term schemes* are introduced as placeholders for the missing side of the generalized conjecture, and a constraint based paradigm is proposed for synthesizing appropriate terms by generating constraints on them. The procedure for discovering such generalized conjectures is discussed in section 5.

The effectiveness of the proposed approach for speculating intermediate lemmas and discovering generalized conjectures is illustrated through several examples on lists and numbers.

## 2 Related Work

Our work is similar in its motivation to that of [6, 5, 4] in which approaches for speculating intermediate lemmas and for discovering generalized forms of conjectures for fixing failed induction proof attempts are given using the *rippling* heuristic [2]. The non-induction variables of a conjecture (called *sinks*) are exploited in that approach also. In rippling, it is assumed that a common term structure called the *skeleton* is shared by both the induction hypothesis and the conclusion. Meta level annotations called *wave fronts* are used to mark the differences between the hypothesis and the conclusion with regards to the skeleton. Similar annotations are associated with the rewrite rules (called *wave rules*) representing definitions and lemmas. Intermediate lemmas are generated as annotated equations by individually speculating each of the sides of the equation. Speculation typically involves starting with the skeleton embedded in second order meta variables denoting the missing wave fronts and repeatedly simplifying using wave rules. The missing term structure is incrementally generated by unifying terms with second order meta variables after each simplification step. The approach uses higher-order unification, an expensive primitive operation, often leading to many useless paths. This approach is implemented in the prover *Oyster* and has been used in conjunction with proof planner *CLAM*.

Another related approach based on rippling is a critic [10] for handling diverging induction proof attempts. The critic is implemented in the theorem prover *SPIKE* and identifies accumulating term structures in successive induction subgoals by difference matching these subgoals [3], a technique for reconciling term annotations. Missing lemmas are heuristically speculated as wave rules that can aid in removal of this accumulating structure. The speculation of lemmas is based solely on the analysis of the proof attempts and does not exploit the structure of the rewrite rules.

The proposed approach, in contrast, is guided by heuristics to semantically match the hypothesis and the conclusion in a restricted fashion based on the structure of the available definitions and lemmas. We believe that it is much simpler as it does not require additional annotations to be associated with the rewrite rules and avoids higher order unification for generating instantiations for term schemas as done in [6, 5, 4]. Suitable instantiations are instead obtained by generating sufficiently many constraints on instantiations, and heuristically speculating ground instances using constraints. More constraints can be generated, if need be, depending upon how much resources a prover is interested in using in speculating conjectures. All of the examples reported in [5, 6] have been done using the proposed approach. For a detailed comparison of these approaches the reader is referred to [8].

## 3 Using Non-induction Variables to Fix Failures

The main idea exploited in the proposed approach is that if an induction proof attempt is not likely to succeed because of inapplicability of induction hypotheses, then in each induction step, intermediate conjectures should be generated to be applicable to the conclusion subgoal. However the application of these intermediate conjectures should lead to a simplified form of the conclusion subgoal on which the induction hypothesis becomes applicable. The search for intermediate conjectures is guided by looking for proper instantiations of non-induction variables (if any) in an induction hypothesis.

**NOTATION**: Primed variables are used to denote non-induction variables in a hypothesis throughout the paper. Induction variables are sometimes replaced by constants to emphasize that we are not seeking substitutions for them. For examples, if $x, y, u$, etc., are induction variables, they will be replaced by $xc, yc, uc$, etc., respectively, in conclusion and hypotheses.

For instance, consider proving the following conjecture,

$$(C_1): \quad rotate(len(x), app(x, y)) = app(y, x),$$

using a rewrite system $R_l$ with the following terminating rules defining *app, len, rotate* on lists generated by *nl,, cons.*:

$$\begin{array}{ll} app(nl, x) \rightarrow x, & app(cons(x, y), z) \rightarrow cons(x, app(y, z)). \\ len(nl) \rightarrow 0, & len(cons(x, y)) \rightarrow s(len(y)). \\ rotate(0, x) \rightarrow x & rotate(x, nl) \rightarrow nl, \\ \multicolumn{2}{l}{rotate(s(x), cons(y, z)) \rightarrow rotate(x, app(z, cons(y, nl))).} \end{array}$$

An inductive proof attempt on lists with $x$ as the induction variable, leads to a basis and an induction step case. The basis case: $(C_{1.1}): rotate(len(nl), app(nl, y)) = app(y, nl)$, reduces by the definitions of $app, len$ and $rotate$ to: $y = app(y, nl)$, which is easily proved by induction on $y$ using the definition of $app$.

In the induction step, the conclusion is

$$rotate(len(cons(x1c, xc)), app(cons(x1c, xc), yc)) = app(yc, cons(x1c, xc)),$$

with the hypothesis being,

$$rotate(len(xc), app(xc, y')) = app(y', xc).$$

The variable $y$ in the conjecture is a non-induction variable as it is not used for induction. In the hypothesis, it is primed to distinguish from $yc$ in the conclusion.

The conclusion simplifies by the definitions of $app, len$ and $rotate$ to,

$$(C_{1.2}): rotate(len(xc), app(app(xc, yc), cons(x1c, nl))) = app(yc, cons(x1, x)).$$

In theorem provers such Nqthm, RRL, a hypothesis is typically generated by instantiating $y'$ to be $yc$. (Even if $y'$ is instantiated to be some other value, the instantiated hypothesis is still not applicable until a lemma about the associativity of $app$ is speculated, and applied on the conclusion.) As a result, neither of the sides of the hypothesis is applicable to the conclusion in this case.

However, if an intermediate lemma stating the associativity of $app$

$$(L_1): \quad app(app(x, y), z) = app(x, app(y, z)),$$

can be speculated, then $(C_{1.2})$ simplifies using $(L_1)$ to

$$rotate(len(xc), app(xc, app(yc, cons(x1c, nl)))) = app(yc, cons(x1c, xc)).$$

It is obvious now that instantiating $y'$ to be: $app(yc, cons(x1c, nl))$, in the hypothesis would lead to a proof of the conjecture $(C_1)$ using lemma $(L_1)$.

In this section we develop a procedure for fixing such failures as guided by an application of an induction hypothesis. The key steps of the procedure are first illustrated using the above example. A complete description is given in the next section. A successful execution of the procedure has two outcomes:

1. a candidate instantiation of non-induction variable(s), and
2. intermediate conjectures leading to the application of the hypothesis using the candidate instantiation to the conclusion subgoal.

Whenever an induction proof attempt fails due to inapplicability of the hypothesis, we first identify a subterm in the conclusion where either of the sides of the hypothesis could be potentially applied, and equate the subterm with the appropriate side of the hypothesis. For the example, since *rotate* appears at the outermost level of the conclusion subgoal, the left hand side of the hypothesis is applicable only at the top-most level. (In general, this is a choice-point since there can be many such subterms in the conclusion subgoal.) The resulting equation is

1. $rotate(len(xc), app(app(xc, yc), cons(x1c, nl))) = rotate(len(xc), app(xc, y'))$.

If the left hand side of the hypothesis can be applied at this position, then the right sides of the conclusion and hypothesis give an additional equation.

$$2.\ app(yc, cons(x1c, xc)) = app(y', xc).$$

Such equations are called *difference equations*, denoted by $D$.

The second step is to simplify the difference equations for finding substitutions for non-induction variables. Whenever the two sides of a difference equation have the same root symbol, then one possible heuristic is to *decompose* the difference equation to generate smaller equations (much like unification and matching). If smaller equations lead to failure (see below), then backtrack.

For the above example, either of the two equations can be chosen. Decomposing the second difference equation into: $yc = y', \quad cons(x1c, xc) = xc$, results in failure since *cons* is a free constructor and $cons(x1c, xc)$ cannot be made equal to $xc$ for constants $x1c$ and $xc$. The first equation can be decomposed as:

$$len(xc) = len(xc), \quad app(app(xc, yc), cons(x1c, nl)) = app(xc, y').$$

The first of these equations is trivial and is discarded. We are thus left with two difference equations relating $y'$:

$$\begin{aligned} 2.\ app(yc, cons(x1c, xc)) &= app(y', xc), \\ 3.\ app(app(xc, yc), cons(x1c, nl)) &= app(xc, y'). \end{aligned}$$

The third step is to use the definitions and properties of function symbols to solve for the primed variables, if any, in the difference equations. For each non-induction variable, an instantiation scheme that depends upon all the induction variables is substituted. Constraints are generated on these instantiation schemes from the difference equations, based on interactions with definitions and properties of function symbols appearing in difference equations. For the above example, the instantiation scheme $I_1(xc, yc, x1c)$ is substituted for $y'$ in the difference equations, giving:

$$\begin{aligned} 2.\ app(yc, cons(x1c, xc)) &= app(I_1(xc, yc, x1c), xc), \\ 3.\ app(app(xc, yc), cons(x1c, nl)) &= app(xc, I_1(xc, yc, x1c)). \end{aligned}$$

Now constraints on $I_1$ are generated using the definition of *app*. From the first rule of the definition of *app* and the right side of equation 3 above, we get:

$$4.\ I_1(nl, yc, x1c) \ = \ app(yc, cons(x1c, nl)).$$

From the second rule of the definition of *app* and the right side of equation 3 above, we get:

$$cons(x2, app(app(z, yc), cons(x1c, nl))) \ = cons(x2, app(z, I_1(cons(x2, z), yc, x1c))),$$

which simplifies by the $free\text{-}ness$ of *cons* to:

$$app(app(z, yc), cons(x1c, nl)) \ = \ app(z, I_1(cons(x2, z), yc, x1c)).$$

Using the first rule defining *app*, one more time, we get:

$$5.\ I_1(con(x2, nl), yc, x1c) \ = \ app(yc, cons(x1c, nl)).$$

In the fourth step, these constraints are used to speculate candidate instantiations. From equations 4 and 5 (and more equations can be generated if need be to enhance our confidence in the speculation), a possible speculation is that the substitution for $I_1$ does not depend upon its first argument. So we get

$$y' = I_1(xc, yc, x1c) = app(yc, cons(x1c, nl))$$

as a candidate instantiation for $y'$ for $C_1$.

Step 5: Linked with candidate instantiations are intermediate conjectures obtained by substituting for $I_1(xc, yc, x1c)$ in the difference equations. This gives two intermediate conjectures to speculate:

$$L2.\ app(app(x, y), cons(x1, nl)) = app(x, app(y, cons(x1, nl)))$$
$$L3.\ app(y, cons(x1, x)) = app(app(y, cons(x1, nl), x)).$$

Conjecture $L2$ can be further generalized by abstracting the common subterm $cons(x1, nl)$ by a new variable $z$, a heuristic implemented in $RRL$ and the Boyer-Moore Prover, giving $L1$, the associativity of *app*. Before attempting proofs of these intermediate conjectures, they are first checked for consistency using test data. For the above two lemmas, substituting constructor ground terms for $x, y$ and normalizing leads to true, enhancing confidence in the conjectures. These intermediate conjectures are then attempted for proofs. If successful, the intermediate conjectures as well as the main conjecture are proved.

## 4 Speculating Intermediate Conjectures

Below, we give a procedure called *speculate* for speculating intermediate conjectures when an inductive proof attempt of the main conjecture does not succeed because of inapplicability of the induction hypothesis to the conclusion subgoal in an induction step of the conjecture. For each major step in *speculate*, there is a subsection justifying the step as well discussing heuristics used to constrain search. As more heuristics are identified, the search space can be further pruned. We would like to point out that many of the steps in *speculate* can be done by a procedure for matching modulo a rewrite system $R$ and/or using higher-order (second-order, to be precise) unification. Given that such procedures are prohibitively expensive, we have chosen to devise heuristics that are specific to the procedure *speculate*.

- **Input**:
  1. The conclusion $l_c = r_c$ and the hypothesis $l_h = r_h$.
  2. A rewrite system $R$.
- **Output**: A set of intermediate conjectures $D$ and a substitution $\psi$.
- **Method**:
  1. *Initialize*: Let $\psi = \{\}$. For each position $p$ in $l_c$ such that *possible-equal*$((l_c)|p, l_h)$ is true, generate a set of difference equations $D = \{(l_c)|p = l_h,\ l_c[p \leftarrow r_h] = r_c\}$.[2] Predicate *possible-equal* is defined below. If there are many such positions in $l_c$, each such position leads to a different branch below. (As a convention, for an equation $s = t$ in the difference set, only $t$ can have primed variables appearing.)
  2. Repeat until no more changes possible to $D$ and $\psi$:
     (a) *Eliminate*: $\langle\{t = y', \cdots\}, \psi\rangle \Rightarrow \langle\{\cdots\}, \psi \cup \{y' \leftarrow t\}\rangle$.
     (b) *Decompose*: $\langle\{f(s_1, \cdots, s_n) = f(t_1, \cdots, t_n), \cdots\}, \psi\rangle \Rightarrow$ $\langle\{s_1 = t_1, \cdots s_n = t_n, \cdots\}, \psi\rangle$ if whenever corresponding arguments that do not contain primed variables are identical (as there is no way to make arguments without primed variables equal).
  3. While there are uninstantiated primed variables, do: For each equation $s = t$ in $D$ and for each uninstantiated primed variable $y'$ in $t$, do[3]
     (a) *Abstract by term schemas* : Replace all occurrences of $y'$ in $t$ by a term schema $I_1(x_1, \cdots, x_n)$ where $\{x_1, \cdots, x_n\} = Vars(s)$.
     (b) *Generate Constraints on Instantiation*: Let $t = f(\cdots, I_1(\cdots), \cdots) = s$ be the abstracted equation.
        i. *Remove Context*: For any rule $l \rightarrow x$ in $R$ such that $\sigma(l) = \sigma(t)$ and $\sigma(x) = \sigma(I_1(\cdots))$ for a substitution $\sigma$, generate constraint: $\sigma(I_1(\cdots)) = \sigma(s)$ (see discussion below).
        ii. *Instantiate*: Consider a rule $l \rightarrow r$ in $R$ where $\sigma(l) = \sigma(t)$. Simplify $\sigma(s) = \sigma(t)$ to generate constraint on the instantiation scheme by removing the context.
     (c) *Speculate Instantiation*: Once enough constraints on an instantiation scheme have accumulated speculate a ground instance. See subsection below for heuristics. If speculation is not possible, generate more constraints or stop searching this branch.
     (d) *Consistency Check*: Substitute the instantiation speculated for the instantiation scheme (primed variable) into the difference equations in $D$. The resulting set is the set of candidate intermediate conjectures. Check for their consistency (see discussion below). If any candidate is not consistent, revise the instantiation for the primed variable or stop searching this branch.

[2] In the discussion here, we assume for simplicity that if the induction hypothesis is applicable, its left side will be used to match against a subterm of the left side of the conclusion subgoal. This assumption can be easily relaxed by considering both sides of the hypothesis to possibly apply on either side of the conclusion subgoal.

[3] In a later subsection, we discuss heuristics for selecting a difference equation in $D$ to use.

4. *Generate Conjectures*:
   - Among intermediate conjectures generated from candidate instantiations in $D$, discard equations that are reduced to true by the hypothesis. Prune $D$ by identifying a minimal subset from which others follow.
   - Output $D$ and $\psi$.

The *speculate* procedure *fails* if every branch leads to a difference set with an equation that cannot be solved (i.e. is inconsistent), and candidate substitutions for primed variables could not be generated.

### 4.1 Identifying positions where hypothesis may be applicable

The predicate *possible-equal* implements a heuristic to determine whether its two arguments can possibly be made equal from the definitions and properties of function symbols, by substituting for primed variables in its second argument. In the most general case, this is matching modulo $R$, the rewriting system consisting of definitions and properties. But that problem is undecidable, and even for simple rewriting system, the matching procedure can be expensive. Below we discuss some useful heuristics based on the function symbols appearing in the two arguments and their definitions to prune out many possibilities.

If the roots of the two arguments of terms to *possible-equal* are not the same, then it should be possible to get from the second argument, after primed variables have been substituted and simplification, a term that has the same root function symbol as that of the first argument. This requirement is captured in the notion of reachable functions from a term $t$ using a rewrite system $R$ introduced in [7]. The set of functions *reachable* from a term $t$ using a rewrite system $R$ can be iteratively constructed by starting with the function symbols in $t$. If $f$ occurs in $t$, and $R$ includes a rewrite rule of the form $f(...) \rightarrow r$, then a conservative estimate is every function symbol in $r$ can be reached from $f$ in one step. The transitive closure of this relation gives the function symbols reachable from $f$ in many steps. For instance, for the conjecture $(C_1)$, for the subterm $app(y', x)$ the reachable symbols with respect to $R_l$ are: $\{nl, cons, app\}$.

The above relation can be refined further utilizing more information from the structure of $t$, and prune out function symbols that may not be reachable. The set of function symbols which can possibly appear as the root of a term after normalization can also be exploited.

Another heuristic useful for determining whether the two arguments can become equal is based on using the notion of incompatible functions introduced in [7], which can be used to determine the function symbols in the term retained in any of its normal forms.

A function symbol $f$ is incompatible with a function $g$ in its $k^{th}$ argument in $R$ if there is no rule $l \rightarrow r$ in $R$ of the form $f(..., g(...), ...) \rightarrow r$, where $g(...)$ appears as the $k^{th}$ argument. The function symbol *rotate*, for instance, is incompatible with *len* in its first argument with respect to $R_l$.

Let a function symbol $f$ be incompatible with $g$ at its $k^{th}$ argument. It is easy to see that if $s = f(s_1, \cdots, g(...), \cdots, s_n)$ is already simplified and primed variables do not occur in $g(...)$, then any simplification of $s$ by $R$ would lead to a term whose outermost function symbol is $f$ and the $k^{th}$ argument is still the same $g(...)$. Function symbol $f$ and function symbols in $g(...)$ are said to be *invariant* function symbols of $s$ with respect to $R$. For instance, the *rotate* and *len* are invariant in $rotate(len(x), app(x, y'))$ with respect to $R_l$.

*possible-equal*$(l_c, l_h)$ is false if either the invariant function symbols in $l_c$ and $l_h$ are different, or an invariant function symbol in $l_c$ is not reachable from $l_h$.

In $(C_1)$, the reachable function symbols of $app(y', x)$ are: $\{nl, cons, app\}$; *rotate* is an invariant function symbol of the left hand side of the conclusion. So, *possible-equal*$(rotate(len(x), app(app(x, y), cons(x1, nl))), app(x, y'))$, denoting the application of the right hand side of the hypothesis to the left hand side of the conclusion, is false. Similarly, the possible application of the left hand side of the hypothesis to the right hand side conclusion need not be considered.

### 4.2 Decomposition

A difference equation $f(s_1, \cdots, s_n) = f(t_1, \cdots, t_n)$ should be decomposed, whenever possible, into simpler difference equations $s_i = t_i$, for all $i$, since these may give candidate substitutions for non-induction variables. A difference equation holds either because the arguments are equal or because of some property of $f$. The latter case is covered by steps 3b.i and 3b.ii of *speculate*, in which the root symbol of two terms in the equation may or may not be different.

A difference equation should be decomposed only if the result of decomposition does not include a nontrivial equation without any primed variable; by a nontrivial equation, we mean the one in which the two sides are not identical. A nontrivial equation without any primed variables cannot be satisfied.

For the conjecture $(C_1)$, in the first equation: $rot(len(x), app(app(x, y), cons(x1, nl))) = rot(len(x), app(x, y'))$, is a candidate for decomposition since the first argument on both sides of the equation is $len(x)$. The second equation cannot be decomposed since the second argument to *app* on the two sides, $cons(x1, x)$ and $x$, are different and do not contain any non-induction variables.

Consider another difference set generated during analyzing $(C_1)$:

$$\{app(app(x, y), cons(x1, nl)) = app(x, y'), app(y, cons(x1, x)) = app(y', x)\}.$$

The first equation cannot be decomposed since the first argument of the two sides, $app(x, y)$ and $x$, that do not involve non-induction variables are different.

### 4.3 Choosing equations with removable context

Step 3b.ii in *speculate* is used to generate constraints on instantiations of primed variables. Consider a difference equation $s = t$, in which root symbols of $s$ and $t$ could be either same or different. A primed variable is replaced by the associated instantiation scheme by step 3a, and it appears inside $t$. Our goal is to generate as many constraints on the instantiation scheme as possible so as to enable a good guess later, i.e., by instantiating variables in the instantiation scheme, it may become equal to other terms. This gives constraints on the instantiations for different values of induction variables.

An important heuristic is to apply rewrite rules which can remove contexts around its subterms. Applications of rules of the form $l \rightarrow x$ whose left hand side unifies with $t$ by unifying the variable $x$ with the instantiation scheme in $t$ lead to the immediate removal of the context surrounding the instantiation scheme. The context surrounding an instantiation scheme can also be removed by a sequence of rewrites leading to the application of such a rule.

An equation in the difference set can be annotated to identify instantiation schemes (primed variables) such that its right hand side leads to a removable context with respect to the instantiation schemes. Equations whose right hand sides do not lead to a removable context with respect to instantiation schemes are not preferred. The above criterion often suffices to choose an equation from the difference set in the step 3.(a) for further processing. For instance, in case of $(C_1)$, the equation: $app(app(x, y), cons(x1, nl)) = app(x, y')$ is picked since its right hand side leads to a removable context with respect to $y'$.

### 4.4 Speculating ground instances from constraints

Once enough definitional constraints are generated for a term scheme, a candidate instantiation for the associated primed variable is speculated. The speculated instantiation must satisfy all the constraints. A useful heuristic in speculation is to identify *unchangeable* [1] arguments of the term schema from the definitional constraints. If terms appearing at the unchangeable argument of different definitional constraints are different, they can be *abstracted* by new variables in the definitional constraints, the main rationale being that the instantiation scheme is made dependent on all the variables of the left hand side in step 3.i, a very conservative guess. Often, a candidate instantiation need not depend on all these variables. It is also useful to abstract common subexpressions by a variable. Other heuristics can also be easily integrated.

### 4.5 Consistency check

A candidate substitution for a primed variable leads to generating candidate intermediate conjecture(s) from difference equations. Before attempting a proof of such an intermediate conjecture, simple checks can be performed to enhance confidence in its validity. Firstly, the two sides of the conjecture can be normalized to check whether the result of normalization is an inconsistency such as two free constructors being made equal. Secondly, constructor ground terms can be generated for variables, and conjecture tried on these specific values. If any of the conjectures is false, then the candidate substitution must be discarded.

### 4.6 More examples

In this section we illustrate the *speculate* procedure on an additional example involving multiplication and exponentiation defined recursively as well as tail-recursively. Several other examples illustrating how speculate can be recursively employed to generate a series of intermediate lemmas are discussed in [8].

Consider proving the conjecture,

$$(C_2): \quad qm(m, n, u + v) = u + qm(m, n, v),$$

from the rewrite system $R_N$ with rules

$$\begin{array}{llll}
x+0 \rightarrow x & & x+s(y) \rightarrow s(x+y) \\
x*0 \rightarrow 0 & & x*s(y) \rightarrow x+(x*y) \\
exp(x,0) \rightarrow 1 & & exp(x,s(y)) \rightarrow x*exp(x,y) \\
qm(x,0,y) \rightarrow y & & qm(x,s(y),z) \rightarrow qm(x,y,x+z) \\
qe(x,0,z) \rightarrow z & & qe(x,s(y),z) \rightarrow qe(x,y,x*z).
\end{array}$$

Induction on $n$ leads to a basis case and a step case. The basis case: $qm(m,0,u+v) = u + qm(m,0,v)$ follows from the definition of $qm$.

In the induction step case, the conclusion is,

$$qm(mc, s(nc), uc + vc) = uc + qm(mc, s(nc), vc),$$

with the hypothesis being,

$$qm(m', nc, u' + v') = u' + qm(m', nc, v').$$

The conclusion reduces by the definition of $qm$ to,

$$qm(mc, nc, mc + (uc + vc)) = uc + qm(mc, nc, mc + vc).$$

The proof attempt fails since neither of the sides of the hypothesis can be applied to either of the sides of the conclusion. Speculating intermediate conjectures leads to the following steps.

1. *Initialize:* The predicate *possible-equal* is true only for the left sides of the conclusion and hypothesis. The left hand side of the hypothesis is not likely to be applicable to any proper subterm on the left hand side of the conclusion since the root function symbol $qm$ is invariant and is not reachable from any such subterm. Other possible applications of the hypothesis can be similarly ruled out since the root function symbol $+$ of the right hand sides of the hypothesis and the conclusion, and its second argument are invariant. Step 1 of *speculate* leads to a single branch with the difference set: $\{qm(mc, nc, mc+(uc+vc)) = qm(m', nc, u'+v'), uc+qm(mc, nc, mc+vc) = u' + qm(m', nc, v')\}$.
2. *Decompose and Eliminate*: Both the equations can be decomposed. By decomposition and subsequent elimination we get the substitution: $\psi = \{m' \leftarrow mc, u' \leftarrow uc, \}$ and the difference set : $\{mc+(uc+vc) = uc+v', qm(mc, nc, mc +vc) = qm(mc, nc, v')\}$. Even if $+$ is known to be commutative, only the above decompositions are valid.
   The last difference equation in the above set can be again decomposed, giving the substitution: $\psi = \{m' \leftarrow mc, u' \leftarrow uc, v' \leftarrow mc + vc\}$ and the the difference set is: $\{mc + (uc + vc) = uc + (mc + vc)\}$.
3. *Generate Conjectures*: Lemmas are generated since all the primed variables are instantiated. The equation in the difference set is not an equational consequence of the hypothesis instance. The lemma generated is:

   $$L3' : \quad m + (u + v) = u + (m + v),$$

   a combination of commutativity and associativity, with the hypothesis instance being,

   $$qm(mc, nc, uc + (mc + vc)) = uc + qm(mc, nc, mc + vc).$$

Lemma $L3'$ can be established by induction on $v$. And, the conjecture, $(C_2)$ follows by induction using this lemma, and the hypothesis instance generated.

## 5 Speculating Generalized Conjectures

If a conjecture is quite specific (e.g., it does not include any extra variables on which induction does not have to be performed), then *speculate* cannot generate intermediate conjectures if an induction proof attempt does not succeed. For instance, consider proving the following conjecture relating the tail recursive definition, $qrev$, of the reverse function on lists to the recursive definition $rev$:

$$(C_3): \quad rev(qrev(x, nl)) = x,$$

using the definitions, denoted $R_r$:

$$\begin{array}{lllllll} rev(nl) & \rightarrow & nl, & \quad & rev(cons(x,y)) & \rightarrow & app(rev(y), cons(x, nl)), \\ qrev(nl, y) & \rightarrow & y, & \quad & qrev(cons(x1, x), y) & \rightarrow & qrev(x, cons(x1, y)), \end{array}$$

along with the definition of $app$ given earlier. A proof attempt based on the induction variable $x$ leads to the basis case: $rev(qrev(nl, nl)) = nl$ which follows from the definitions of $qrev$ and $rev$. In the induction step, the conclusion is,

$$rev(qrev(xc, cons(x1c, nl))) = cons(x1c, xc),$$

with the hypothesis,

$$rev(qrev(xc, nl)) = xc.$$

Since the hypothesis cannot be used for simplification, the proof attempt fails. And, since the conjecture does not have any extra variable, *speculate* does not work either. This is evident when *speculate* is attempted on the hypothesis and the conclusion subgoal. In the difference set generated, there are no primed variables and failure is reported as the following equations cannot be solved.

$$\{rev(qrev(xc, cons(x1c, nl))) = rev(qrev(xc, nl)), cons(x1c, xc) = xc\}.$$

In this section, we show how *speculate* can be extended to speculate and generalize from the conjecture being attempted, with the main objective of introducing extra variables in the generalized conjecture.

### 5.1 Generalization by introducing non-induction variables

When the *speculate* fails in step 2 of every branch arising from step 1, we identify subterms in the conjecture for possible generalization. Equations in the difference set which could not be decomposed can help in identifying subterms in the conjecture which can be candidates for generalization and introducing a new variable. A candidate subterm may appear only in one side of the conjecture, in which case it is necessary to search for the appropriate right hand side of the generalized conjecture.

Given an equation $s = t$ in the difference set $D$ of step 2, a subterm $v$ of $t$ is a *candidate* for generalization if abstracting $v$ by a primed variable enables its successful decomposition.[4] For instance, consider the first equation $rev(qrev(x, cons(x1, nl))) = rev(qrev(x, nl))$ in the above difference set of $(C_3)$. The equation cannot be decomposed since the second argument of $qrev$–$nl$ and $cons(x1, nl)$–are different and cannot be made equal. However, if the second argument, $nl$, of $qrev$ in the conjecture is abstracted to be a primed variable, it would be possible to decompose the first equation as then its right side (contributed from the hypothesis) would have a primed variable. The subterm $nl$ is thus a candidate for generalization in the conjecture. There are no other candidate subterms for generalization in this case.

The generalized conjecture is:

$$(GC_3): \quad rev(qrev(x, z)) = T(x, z),$$

where $z$ generalizes $nl$; since the right side of the generalized conjecture has to be generated, we place a term scheme $T(x, z)$ which depends upon all the variables on the left side. To relate $GC_3$ to $C_3$, $T(x, nl)$ is equated to $x$.

We now attempt to do the proof of the generalized conjecture, and on its failure, apply *speculate*. For $GC_3$, a proof is attempted again by doing on induction on $x$. The basis case leads to:

$$T(nl, z) = rev(z).$$

In the induction step, the conclusion is:

$$rev(qrev(x, cons(x1, z))) = T(cons(x1, x), z),$$

with the hypothesis,

$$rev(qrev(x, z')) = T(x, z').$$

Note the presence of a primed variable $z'$ in the hypothesis. The proof attempt fails. So *speculate* is applied, giving rise to the difference set:

$$\{rev(qrev(x, cons(x1, z))) = rev(qrev(x, z')), T(cons(x1, x), z) = T(x, z')\}.$$

In this case, the second equation cannot be decomposed but the first can be. Typically, even if the second equation can be decomposed we do not do so since $T$ there is not a function symbol. $T$ is a term scheme, and depending upon its instantiation, the two sides of the equation could simplify by rewrite rules.

The first equation is decomposed to giving a candidate instantiation for $z'$, which is $cons(x1, z)$. This results in another constraint on $T$: 1. $T(cons(x1, x), z) = T(x, cons(x1, z))$. In addition, we have the above two properties: 2. $T(x, nl) = x$, 3. $T(nl, z) = rev(z)$, which can be used in step 3.b.i and 3.b.ii to generate

[4] We have assumed in this section that all variables in a conjecture are used in an induction scheme. The proposed heuristic works also when a conjecture has extra non-induction variables. In case *speculate* fails to find suitable instantiations of non-induction variables, a generalization of the conjecture can be attempted by introducing more non-induction variables. In such case, it suffices to consider only those subterms of $t$ that do not contain any primed variables since abstracting subterms with such variables does not enable the decomposition step.

additional constraints on $T$. For example, in the *Instantiate* step, 3.b.ii, the left side of 3 above unifies with the right side of 1, to give, after simplification:

$$T(cons(x1, nl), z) = rev(cons(x1, z)) = app(rev(z), cons(x1, nl)).$$

The subterm $cons(x1, nl)$ can be generalized to a variable $u$ to give:

$$T(u, z) = app(rev(z), u).$$

This definition of $T$ satisfies the three constraints above (of course, it is assumed that $app(x, nl) = x$ is available as a lemma). Hence $T$ is synthesized as well as a proper instantiation for the non-induction variable in the generalized conjecture for hypothesis application is generated.

### 5.2 Extending Speculate

Step 2 is modified. If for every branch, the associated difference set cannot be decomposed, then attempt to introduce extra variables by abstracting a subterm in the conjecture. From equations in difference sets that can be decomposed if certain subterms in the hypothesis (which is a proxy for the conjecture) are abstracted by primed variables, identify candidate subterms for generalization. Using each such subterm, generalize the conjecture by introducing a new variable for the subterm, and making the other side (in which the subterm does not appear) to be a term schema $T(x_1, \cdots, x_n)$ which potentially depends upon all the variables in the other side. From the generalized conjecture, generate the property that $T(x_1, \cdots, x_n)$ must satisfy in order for the original conjecture to follow from the generalized conjecture.

Given a conjecture $l = r$, let $l|p$ be a candidate subterm for generalization. Let $l'$ be obtained by replacing $l|p$ by a new variable. The generalized conjecture is: $l' = T(x_1, \cdots, x_n)$ with $\{x_1, \cdots, x_n\} = Vars(l')$. If $\sigma(l') = l$, then $\sigma(T(x_1, \cdots, x_n)) = r$ is a desired property of $T$.

Attempt an induction proof of the generalized conjecture. Identify additional properties of $T(x_1, \cdots, x_n)$ from the basis step(s). Every induction step will fail. Apply *speculate*, with an exception that a difference equation whose both sides have $T$ as the root is not decomposed. Solve for equations in the difference set, attempting to generate a candidate instantiation for primed variables. Use properties already identified of the replacement scheme $T$ as rewrite rules to generate additional constraints. In general, constraints may be generated on both instantiations of the primed variables as well as the terms being synthesized for one side of the generalized conjecture.

Once sufficiently many constraints on the replacement scheme $T$ and its properties are identified, attempt to speculate $T$ using heuristics discussed in subsections 4.3 and 4.4.

## 6 Concluding Remarks

Induction proof attempts often fail due to unavailability of appropriate induction hypothesis as well as due to the lack of appropriate intermediate lemmas.

Handling such failures is a nontrivial task that involves considerable user intervention. This paper is an attempt to provide automated support for handling such failures. A simple approach is proposed for speculating intermediate conjectures by semantically matching the induction hypothesis and the conclusion of a failed induction proof attempt in a restricted fashion. The applicability of the approach to generate nontrivial generalized versions of conjectures to fix failures is also illustrated. The initial experimentation with the proposed approach is extremely encouraging. The approach has proven to be surprisingly effective in speculating intermediate lemmas for tail-recursive programs. However, it is applicable in general in induction proof attempts based on recursive definitions. For instance, the proposed approach can be employed to automatically establish a conjecture such as $rev(app(rev(x), y)) = app(rev(y), x)$ from the recursive definitions of $app$ and $rev$. An implementation of the proposed method in $RRL$ is in progress. It is planned to integrate the method for fixing failures of inductive proof attempts predicted based on several criteria proposed in [7].

## References


1. R.S. Boyer and J S. Moore, *A Computational Logic.* ACM Monographs in Computer Science, 1979.
2. A. Bundy, A. Stevens, A. Ireland, and A. Smaill, "Rippling: A heuristic for guiding inductive proofs", *Artificial Intelligence,* 62:185-253, 1993.
3. D.Basin and T.Walsh "Difference Matching", *In Proc. of CADE 11,* D. Kapur (editor), LNAI 607, Springer Verlag 1992.
4. J.T.Hesketh "Using middle out reasoning to guide inductive theorem proving", Ph.D thesis. University of Edinburgh 1991.
5. A. Ireland and A. Bundy, "Productive Use of Failure in Inductive Proof", Edinburgh DAI Research Report No: 716. To appear on the *Special Issue on Automation of Proof by Mathematical Induction, JAR.*
6. A. Ireland "The use of planning critics in mechanizing inductive proofs", *In Proc. LPAR 92*, A. Voronkov (editor), LNAI 624 Springer Verlag, 1992.
7. Deepak Kapur and M. Subramaniam " Failure Analysis and its role in Generalization", Manuscript under Preparation.
8. Deepak Kapur and M.Subramaniam "Lemma Discovery in Automated Induction", Technical Report, CS Dept., SUNY April, 1995(an expanded version of this paper).
9. D. Kapur and H. Zhang, "An overview of Rewrite Rule Laboratory (RRL)," *J. of Computer and Mathematics with Applications,* 29, 2, 1995, 91-114. Earlier descriptions in CADE-88 and RTA-89.
10. T. Walsh "A divergence critic", *In Proc. of CADE 12,* A. Bundy (editor), LNAI 814, Springer Verlag 1994.
11. H. Zhang, D. Kapur, and M.S. Krishnamoorthy, "A mechanizable induction principle for equational specifications," Proc. *Ninth International Conference on Automated Deduction*, E. Lusk and R. Overbeek (eds.), LNCS, Vol. 310, Springer-Verlag, Berlin, 1988, 250–265.