Fundamenta Informaticae 34 (1998) 1–20
IOS Press

Automating inductive Specification Proofs in NuPRL *

Brigitte Pientka
Department of Computer Science
Carnegie Mellon University
Pittsburgh, PA, USA
bp@cs.cmu.edu

Christoph Kreitz
Department of Computer Science
Cornell University
Ithaca, NY, USA
kreitz@cs.cornell.edu

Abstract. We present an automatic method which combines logical proof search and rippling heuristics to prove specifications. The key idea is to instantiate meta-variables in the proof with a *simultaneous match* based on rippling/*reverse rippling heuristic*. Underlying our rippling strategy is the *rippling distance* strategy which introduces a new powerful approach to rippling, as it avoids termination problems of other rippling strategies. Moreover, we are able to synthesize conditional substitutions for meta-variables in the proof. The strength of our approach is illustrated by discussing the specification of the integer square root and automatically synthesizing the corresponding algorithm. The described procedure is integrated as a tactic into the NuPRL system.

Keywords: theorem proving, program synthesis, induction

*This work has been carried out during the stay of the first author in the NuPRL group. It has been made possible through a fellowship of the Daimler Benz Foundation and the support of Robert Constable who provided the additional financial means and an intellectual challenging and encouraging environment for this work.

1. Introduction

The question, how can we trust software in safety-critical applications, has been a key issue in computer science from the start. In the age of the internet, a new dimension is added to this question: how can we trust programs we down load from the internet. This has caused a new widespread interest in formal methods. Describing programs and properties of programs involves understanding the problem, formalizing it, stating a conjecture, exploring possible solving strategies, writing down intermediate results, checking the results and assembling the solution.

A number of systems have been developed to support this formal reasoning process. The NuPRL system [9] offers a general framework for problem solving based on constructive type theory [14]. The setting of constructive type theory offers the unique advantage of total correctness of synthesized programs. A proof for a specification can be developed interactively by stepwise refining the goal into subgoals. *Tactics* are applied to perform a single proof step or several of these steps. The libraries provide an extensive collection of lemmata which assist the user during the proof process. The proofs are presented in a readable sequent proof style and the results can be checked by the user. By the proofs-as-programs principle [3] a program can be extracted from a proof of its specification, which is usually formulated as the statement that a solution to a given problem exists. However the application of NuPRL and other such systems is limited by its low degree of automation. In order to overcome this drawback, we suggest to incorporate and combine logical and inductive theorem proving techniques.

Logical proof search methods have been successfully used for proving formulas in first order logic. Otten [16] and Schmitt [17] show we can use efficient proof search methods like matrix methods in the framework of NuPRL. A major drawback of these methods is that they cannot deal with induction and inductive proof obligations. Inductive theorem proving techniques, on the other hand, have been very successful, in solving inductive proof obligations. *Rippling* [8, 7] is a powerful annotated rewriting technique which guides the rewriting process from the induction conclusion towards the application of the induction hypothesis. However, only little focus has been devoted to the automatic logical proof search part and instantiation of existentially quantified variables.

In this paper we combine the logic provided by constructive type theory with inductive theorem proving techniques such as rippling, in order to search for a specification proof. We use first-order meta-variables during proof search within the sequent calculus. We develop a matching procedure that is based on rippling and *reverse rippling* heuristic. Two expressions are said to be equal, if we can find a substitution for the meta-variables and a chain of rewriting steps, such that both expressions can be made equal. For a set of such "equalities" it might sometimes only be possible to find *conditional substitutions*. With this approach, we combine the benefits from logical and inductive theorem proving. 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 discuss the general idea of our approach and describe the *rippling distance* strategy which underlies the rippling and reverse rippling heuristic.

In Section 3.2 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 3.3 we present a more technical description of the matching method using NuPRL-ML-notation. In Section 4 we describe the integration into NuPRL. In Section 5 we show the flexibility of our technique by discussing the synthesis of a more efficient algorithm of the integer square root specification. The combination of the simultaneous matching procedure and matrix proof procedures is outlined in Section 6. Related work concerning program synthesis and rippling is discussed in Section 7. Finally in Section 8 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 correspond to universally quantified variables in the given and are marked by $\lfloor sink \rfloor$. We call the annotated rewrite rules *wave-rules*. To illustrate, consider the following wave-rule which is derived from the recursive definitions of $+$.

$$\boxed{\mathsf{s}(\underline{U})}^{\uparrow} + V \quad \stackrel{R}{\longmapsto} \quad \boxed{\mathsf{s}(\underline{U+V})}^{\uparrow} \qquad (1)$$

In this wave-rule $\mathsf{s}(\ldots)$ denotes the wave-front that is marked by a box. The underlined parts $\underline{U}$ resp. $\underline{U+V}$ mark the wave-holes. Intuitively, the position and orientation of the wave-fronts define the direction in which the wave-front has to move within the term tree. An up-arrow $\uparrow$ indicates that the wave-front has to move from a position within the term tree towards the root of the term tree (*rippling-out*). A down-arrow $\downarrow$ moves the wave-front inwards or sideways towards the sink in the term tree, i.e. the sink is filled with the wave-front (*rippling-in*). During the rippling process we might need to switch between different rippling strategies like rippling-out and rippling-in. If rippling succeeds in moving the annotations either to the root of the term tree or to a sink, then rippling terminates successfully and the induction hypothesis matches the induction conclusion. Rippling terminates unsuccessfully if the rewriting process is blocked, i.e. no wave-rule is applicable anymore and the induction hypothesis (given) does not match the induction conclusion (goal).

In Basin & Walsh [2], a calculus for rippling is presented and well-founded measure, called *wave measure* is defined, under which rippling terminates if no meta-variables occur in the goal. The wave measure associates weights to the wave-fronts to measure the width and the size of the wave-front. The *width* of a wavefront is defined by the number of nested function symbols between the root of the wave-front and the wave-hole. The *size* of a wave-front is the number of function symbols and constants in the wave-front. Rewriting is restricted to the application of wave-rules which are skeleton preserving and measure decreasing according to the defined wave measure. As the wave-rule itself is measure decreasing, also the application of it is measure decreasing. The defined measure is static and we can determine the measure merely by inspecting the wave-rules.

For instantiating existentially quantified variables via rippling, mainly two approaches have been suggested in the literature. In Bundy *et al.* [7] special existential wave-rules are suggested. Existential wave-rules can be derived from non-existential wave-rules. For example, the existential wave-rule corresponding to wave-rule (1) takes the following form:

$$\exists \boxed{U} : \mathbb{N}. \boxed{U} + V \quad \stackrel{R}{\longmapsto} \quad \exists \boxed{U'} : \mathbb{N}. \boxed{s(\underline{U' + V})}^{\uparrow} \tag{2}$$

Unfortunately, the search problem is exacerbated in the presence of existential quantifiers. Another disadvantage of this approach is that the process of existential rippling does not explicitly record the relationship between the non-existential wave-rule (1) and its existential analogue (2), more precisely the relation between U and U'. This does not matter if we are just interested in provability. In program synthesis, however, the identity of the existential witness plays a vital role of defining the program to be synthesized.

Other approaches [1, 12, 18] use meta-annotations; the existentially quantified variable x is replaced by $\boxed{F(\underline{X})}$ where capital letters indicate meta-variables. Middle-out reasoning [10] is used in order to instantiate the function F and its argument X. This problem requires a computationally expensive higher-order unification. The presence of higher-order variables also leads to non-termination of rippling, as the width and the size of the wave-front cannot be determined.

Our approach overcomes these drawbacks by combining first–order theorem proving techniques and matching extended by rippling heuristics for finding the witness for the existentially quantified variables. Logical proof search methods give us the flexibility and possibility to concentrate on atomic subgoals. Rippling heuristics rewrite the conclusion part towards the induction hypothesis part and guide the search for the instantiation of the meta-variables. As we explain in section 3.1, we use a dynamic distance measure to ensure termination of our rippling strategy which treats different rippling techniques uniformly and avoids termination problems in the presence of meta-variables.

3. Automatic Instantiation of Meta-variables

Our research interest is to automate key steps such as the instantiation of the existentially quantified variable in sequent proofs. By the proofs–as–programs paradigm we are then able to extract a program from the proof of a specification. Our automization efforts focus on the step case of an inductive proof.

In Figure 1 an overview of our approach is presented. In the first step we split the step case formula into several subgoals and introduce meta-variables in place of existentially quantified variables. For this purpose we use a sequent prover in intuitionistic logic. Main reason for this

1. A *Logical Decomposition* of the step case formula in several subgoals and uses meta-variables in place of existentially quantified variables
2. The meta-variables are instantiated by *simultaneous matching* on atomic subgoals: a match between an element of the hypothesis H_i and the conclusion C succeeds, if
   (i) there exists a substitution σ that $\sigma \circ C = H_i$
   (ii) there exists a substitution σ and a rippling sequence between an element of the hypothesis H_i and the conclusion C, i.e. $\exists \sigma.(\sigma \circ C) \stackrel{R}{\longmapsto} \ldots \stackrel{R}{\longmapsto} H_i$. The *rippling / reverse rippling* heuristic is used to compute the rippling sequence and valid substitutions for meta-variables.
   (iii) the subgoal $\ldots H_i \ldots \vdash (\sigma \circ C)$ is provable under the existing substitutions by an arithmetic decision procedure.
3. If there is not a unique substitution for the meta-variables, we synthesize *conditional substitutions*, i.e. we first compute substitutions under the condition Cond and then compute substitutions under the condition $\neg\mathsf{Cond}$.

Figure 1. Automatic instantiation of meta-variables – 3 steps

choice has been that the results can be easily backtranslated into NuPRL. In general however our technique can be combined with any logical proof search method. In Section 6 we outline how to combine our technique with more efficient proof search methods like matrix methods.

The simultaneous match computes substitutions for the meta-variables incrementally[1]. This match consists of three disjunctive matches. Either the goal C under the existing substitutions σ can be proven by a decision procedure, or a matching procedure finds a substitution for the meta-variables such that the goal $\sigma \circ C$ is equal to the corresponding hypothesis term H_i. Key part of this match is the ripping/reverse rippling heuristic, which computes a rippling sequence and a substitution σ such that the term C from the induction conclusion can be rippled to the term H_i from the induction hypothesis. The following figure captures the intuition behind this heuristic.

$$\underbrace{C \stackrel{R}{\longmapsto} C_0 \stackrel{R}{\longmapsto} \ldots \stackrel{R}{\longmapsto} C_i}_{\text{rippling}} \underbrace{\stackrel{R}{\longmapsto} \ldots \stackrel{R}{\longmapsto} C_n \stackrel{R}{\longmapsto} H}_{\text{reverse rippling}}$$

This rippling sequence is synthesized in two steps: First the part of the induction conclusion is rewritten to some formula C_i. Either C_i matches directly with the corresponding induction hypothesis term H or the rippling sequence $C_i \stackrel{R}{\longmapsto} \ldots \stackrel{R}{\longmapsto} C_n \stackrel{R}{\longmapsto} H$ is computed by backwards reasoning from the induction hypothesis towards C_i. This process is called *reverse rippling*. If this sequence is empty, then either the term C implies H under the given substitution or there exists a substitution such that the terms are equal.

[1]As we work on inductive proof obligations, the hypothesis term is the mirror image of the conclusion, i.e. the mirror image of meta-variable Y in the hypothesis term is a term $F(y)$ in the conclusion and vice versa. Hence it suffices to restrict this instantiation procedure to matching.

If no unique substitution for the meta-variables can be found, a complementary set of constraints is synthesized. These constraints form a case analysis in the final proof.

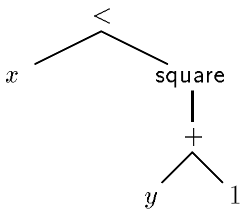
Our approach combines the advantages of the guided rewriting and the flexibility of logical proof search. Hence our approach is powerful in dealing with complex formulas which need both techniques for proving. If it succeeds, the rippling proof is translated back into sequent style.

3.1. Rippling Distance Strategy

The *rippling distance* strategy [13] is the concept underlying our rippling approach. The rippling distance strategy uses a new measure $\mathcal{MD}$ to ensure that rippling is measure decreasing, i.e. the application of wave-rules is terminating. In each rippling step $\mathcal{MD}$, the distance between the wave-front and the sink, must decrease. Main advantage of this method is the uniform treatment of the different rippling techniques and more goal-oriented proof search. Switching between different rippling techniques (e.g. between rippling-out and rippling-in) and backtracking over the different switching possibilities is superfluous. For a more detailed complexity analysis see [13, 5].

The distance between a wave-front and a sink is defined as the distance between the term position p_{wf} of the wave-front and the term position p_{sink} of the sink. The position of a sink p_{sink} in an annotated term is the term position of the sink in the skeleton of the annotated term. The position of the wave-front, p_{wf} in the annotated term is the position of the wave-hole in the skeleton of the annotated term. The distance δ between p_{sink} and p_{wf} is the length of the path from p_{sink} to p_{wf} in the term tree. An example is shown in figure 2.

We classify all the wave-fronts with the same distance δ_i to the sink together. The final measure $\mathcal{MD}$ is a vector where each position m_i in the vector corresponds to the number wave-fronts in the term whose distance to the sink is δ_i. We use the reverse lexicographic ordering on the vector which describes the distance measure. For two annotated terms t_1 and t_2, the measure is decreasing iff $\mathcal{MD}_{t_1} \leq \mathcal{MD}_{t_2}$ according to the reverse lexicographic ordering.



term position of y:	[2;1;1]
term position of wave-front:	[2]
distance δ between y and wave-front:	2
distance measure $\mathcal{MD}$ $(m_5 \ldots m_1)$:	[0 1 0 0]

Figure 2. Term tree for skeleton of annotated term $x < \boxed{\underline{\mathsf{square}(\lfloor y \rfloor + 1)} + 2(y+1)+1}$

Sometimes we want to allow not only wave-rule applications which move the wave-front closer to the sink position, but also wave-rules which decrease the size of the wave-front. In this case, we can extend the measure by associating weights w_k to each wave-front wf_k. Typically these weights describe the size of the wave-front. Each distance δ_i is weighted such that δ_i is the sum over the weights w_k associated with wave-front wf_k whose distance to the sink is i.

Intuitively, it is clear that this measure is terminating. In each rippling step either the size of the wave-front is decreasing of the distance between the sink and its associated wave-fronts. Finally the distance between the sink and its associated wave-fronts is zero, i.e. the sink is filled with the wave-front(s).

It is interesting to observe that we can easily deal with wave-fronts whose size is unknown. We can associate the same weight to every unknown wave-front; the measure will still be decreasing. The only wave-rules we rule out are wave-rules which would only decrease the size of the wave-front (if it would be known), but not the distance to the associated sink. This is not a real restriction, as we can freely choose the wave-front, hence we do not need these kind of wave-rule applications. Note that even if the size of the wave-front is unknown, termination is still guaranteed. This is one of the main advantages in comparison to the conventional rippling approaches by Bundy *et al* and Basin & Walsh.

Hence the rippling distance strategy is a more powerful strategy which dynamically checks, if the wave-rule application is measure decreasing. Moreover this strategy avoids non-termination problems in the presence of unknown wave-fronts.

3.2. Proving the Specification of the Integer Square Root

In this section we illustrate the automatic instantiation of existentially quantified variables by discussing the proof of the following integer square root specification:

$$\forall x : \mathbb{N}.\ \exists y : \mathbb{N}.\ y^2 \leq x \wedge x < (y+1)^2 \tag{3}$$

The top–down sequent proof starts by induction on x. We concentrate on the step case of the induction:

$$x : \mathbb{N},\ \exists y : \mathbb{N}.\ y^2 \leq x \wedge x < (y+1)^2 \quad \vdash \quad \exists y : \mathbb{N}.\ y^2 \leq \mathsf{s}(x) \wedge \mathsf{s}(x) < (y+1)^2$$

This sequent is proved by a procedure which searches for a rippling proof and an instantiation for the existentially quantified variable y. It proceeds as described in figure 1. In the first step, logical inference rules are used in order to decompose the induction hypothesis and the conclusion. The existentially quantified variable in the conclusion is replaced by a meta-variable Y. This gives us two subgoals, (4) and (5):

$$x : \mathbb{N},\ y : \mathbb{N},\ \underline{y^2 \leq x},\ x < (y+1)^2 \quad \vdash \quad Y^2 \leq \boxed{\mathsf{s}(\underline{x})} \tag{4}$$

and

$$x : \mathbb{N},\ y : \mathbb{N},\ y^2 \leq x,\ \underline{x < (y+1)^2} \quad \vdash \quad \boxed{\mathsf{s}(\underline{x})} < (Y+1)^2 \tag{5}$$

During the second step, the goal is annotated in such a way, that its skeleton matches the corresponding part in the induction hypothesis. Corresponding parts are underlined in this example. Note, that the meta-variable Y matches the variable y in the given. The following wave-rules are derived from the definitions of functions used in the specification:

$$\boxed{(\underline{U}+W)} < \boxed{(\underline{V}+W)} \overset{R}{\longmapsto} U < V \tag{6}$$

$$\boxed{s(\underline{U})} + V \overset{R}{\longmapsto} \boxed{s(\underline{U+V})} \tag{7}$$

$$(\boxed{s(\underline{A})})^2 \overset{R}{\longmapsto} \boxed{\underline{A^2} + 2A + 1} \tag{8}$$

Note, while in wave-rule (7) and (8) wave-fronts occur on both sides of the wave-rule, in wave-rule (6) the wave-fronts are dropped on the right hand side of the wave-rule. Wave-rules like (6) are usually used to complete proofs[2]. As no wave-rule is applicable, rippling leaves the subgoal unchanged.

The reverse rippling match reasons backwards from the induction hypothesis towards the (rippled) conclusion and extracts a rippling sequence and an instantiation for the meta-variable Y. In the induction hypothesis y is marked as a sink variable. During the first reverse wave-rule application wave-fronts are created by wave-rules of the same type as wave-rule (6). The inserted wave-front is refined step–by–step. This wave-front has to move towards the sink variable y by reverse rippling and results in the instantiation of the meta-variable Y. We start with the second subgoal (5) and try to match $\boxed{\mathsf{s}(\underline{x})} < (Y+1)^2$ (induction conclusion IC – goal) and $x < (y+1)^2$ (induction hypothesis IH – given). The induction hypothesis IH represents the final formula in the rippling sequence. In order to determine which formula preceded the induction hypothesis, the wave-rule set is inspected. The given must match the right hand side of a wave-rule. The left hand side of this rule constitutes the predecessor to IH, if further rippling towards the sink variable is possible. By wave-rule (6) it is suggested that the formula before reaching the induction hypothesis $x < (y+1)^2$ is $\boxed{\underline{x}+W} < \boxed{(\underline{(y+1)^2} + W)}$. This formula can be rippled by wave-rule (8) and W can be instantiated with $2(y+1)+1$. The inserted wave-front moves closer to the sink variable y. Rippling towards the sink variable y is straightforward and the generated rippling sequence is presented in Figure 3. By wave-rule (7) the wave-front s is moved to a position where it surrounds y; therefore our rippling sequence terminates successfully. As no wave-rule is available to justify the final step, we need to prove that the induction conclusion is implied by the last step. Typically these implications can be proven by decision procedure using standard arithmetic. In this case the proof is trivial. Therefore $\mathsf{s}(y)$ is a valid substitution for Y.

Note that the final step does not correspond to a wave-rule application step, even if the wave-rule would have been available. In order to allow such an application we would need to show that either the distance to the sink is decreasing or the size of the wave-fronts is decreasing. But neither is the case. The distance to the sink stays the same. The size of the wave-front however is even increasing, from $\boxed{s(\underline{\ldots})}$ to $\boxed{\underline{\ldots} + 2(y+1)+1}$. The proved implication is a reasonable proof step in order to strengthen the actual conjecture. However it cannot be justified by

[2]We consider here proofs by strong fertilization, which aim for total match between the induction conclusion term and induction hypothesis term.

$$
\begin{array}{ll}
\boxed{\mathsf{s}(\underline{x})} < (\boxed{\mathsf{s}(\underline{y})} + 1)^2 & \\
& \text{by decision proc.} \\
\qquad \longleftarrow \boxed{\underline{x} + 2(y+1) + 1} < (\boxed{\boxed{\mathsf{s}(\underline{y})}} + 1)^2 & \\
& \text{by wr (7)} \\
\overset{R}{\longmapsto} \boxed{\underline{x} + 2(y+1) + 1} < \boxed{\mathsf{s}((\underline{\lfloor y \rfloor + 1}))}^2 & \\
& \text{by wr (8)} \\
\overset{R}{\longmapsto} \boxed{\underline{x} + 2(y+1) + 1} < \boxed{(\underline{(\lfloor y \rfloor + 1)^2} + 2(y+1) + 1)} & \\
& \text{by wr (6)} \\
\overset{R}{\longmapsto} x < (\lfloor y \rfloor + 1)^2 &
\end{array}
$$

Figure 3. Rippling sequence generated by extended matching

rippling. Hence the combination of the rippling and the reverse rippling strategies gives us the flexibility to detect such wholes in the proof and fill them in a goal-oriented way.

This substitution and its corresponding rippling sequence (cp. Figure 3) constitute a successful match if the remaining subgoals are true under the found substitution (step 3 in Figure 1). We use a heuristic to check the subgoals and synthesize case splits if necessary. In order to restrict search space, we require that the subgoals can be proven by standard arithmetic, rippling and equational reasoning. If one of the remaining subgoals is not provable by these techniques, we use this subgoal as a constraint of the substitutions. In order to be consistent, we then prove all the remaining subgoals under the negated constraint. In this example, (4) is the only remaining subgoal. We use the subgoal $(\mathsf{s}(y))^2 \leq \mathsf{s}(x)$ as a constraint, and therefore try to prove both cases (4) and (5) for the case $\neg(\mathsf{s}(y))^2 \leq \mathsf{s}(x)$. We use unfolding of the successor and $\neg(\ldots \leq \ldots)$ function s and normal matching, to derive a substitution. We start again by inspecting subgoal (5). Matching between $\mathsf{s}(x) < (Y+1)^2$ and $\mathsf{s}(x) < (y+1)^2$ returns the substitution $[y/Y]$. The subgoal (4) is trivially true under this substitution.

By combining logical proof search with rippling and extended matching, we are able to generate automatically a proof for the step case and to synthesize a set of conditional substitutions: $[(\mathsf{s}(y))^2 \leq \mathsf{s}(x), \mathsf{s}(y)/Y], [\neg((\mathsf{s}(y))^2 \leq \mathsf{s}(x)), y/Y]$.

3.3. An Algorithm for Simultanous Matching

In this section we present a technical description of the steps performed during the automatic instantiation of existentially quantified variables (see Figure 1). We use NuPRL-ML-notation to describe the algorithm. To automate step 1 standard theorem proving methods can be used. We concentrate on the simultanous matching procedure (step 2) which is the core of the automatic instantiation of existentially quantified variables. The algorithmic description for the simultanous match is given in Figure 4.

The function simultaneous_match is applied to all the open subgoals, incrementally building up the list of constraints and the list of substitutions. If not a unique substitution can be found, then a simultaneous match under the negation of the generated constraints is started. This simultaneous match is not allowed to generate any new constraints, but rather has to use the given constraints and synthesize a second substitution list. If all the remaining subgoals can be proven under the substitution, then the match is successful. The function simultanous_match returns (conditional) substitutions and proofs.

```
let simultanous_match conc ind_hyp (constraints,subst_list) =
let inst_conc = instantiate conc subst_list in
let inst_hyp = instantiate ind_hyp subst_list in
  if decidable inst_conc inst_hyp constraints then
    (true, (constraints,subst_list))
  else
    let subst = simple_match inst_conc inst_hyp subst_list in
    (true,(constraints, subst::subst_list))
    ?
    rippling_sequence conc ind_hyp (constraints,subst_list)
    ?
    (true, (conc::constraints,subst_list))
```

Figure 4. Algorithm for simultanous match

First the conclusion conc and the corresponding term from the induction hypothesis ind_hyp are instantiated with possibly already existing substitutions from subst_list. Second, we check if under the given constraints ind_hyp implies conc is provable by an arithmetic decision procedure. This decision procedure is a combination of NuPRL's tactics Unfold, SupInf and Auto which uses standard arithmetic. If we succeed then we are finished. If this is not possible, then we try to find a match between the instantiated terms inst_conc and inst_hyp. If this fails too, we try to synthesize a rippling sequence between these two terms. If everything fails, then we just add the conclusion term to our list of constraints and defer solving this subgoal.

The function rippling_sequence, presented in Figure 5, computes a rippling sequence conc $\xmapsto{R} \ldots \xmapsto{R}$ ind_hyp and a substitution for the meta-variable. Before calling the algorithm rippling_sequence, the wave-rule set wrs contains potential wave-rules. The wave-rules are annotated dynamically during the rippling and reverse rippling process. rip_conc denotes the rippled conclusion and ind_hyp the induction hypothesis. First the variable in the induction hypothesis that corresponds to the meta-variable in the induction conclusion is marked with a sink. The function poss_predecessors computes possible predecessors C_{i-1} for a given formula C_i in the rippling sequence by inspecting the wave-rule set wrs. It simulates the backwards application of one rippling rule. The function reverse_rippling_in triggers the reverse rippling process. It proceeds recursively by depth first search if there are several predecessors to a formula. It reasons backwards from the induction hypothesis ind_hyp towards the rippled conclusion rip_conc. In each recursion, the next possible predecessors are computed. It is successful if the sink variable is surrounded by a wave-front, i.e. the sink is filled. The reverse rippling sequence is recorded in

```
let rippling_sequence conc ind_hyp wrs =
let ann_conc = annotate conc ind_hyp in
let rip_conc = ripple ann_conc wrs in
let predecessors = poss_predecessors ind_hyp rip_conc wrs in
letrec reverse_rippling_in path predecessors =
        if predecessors = [ ] & filled_sink (hd path) & (hd path) → rip_conc
          then path
          else select p ∈ predecessors
              let new_predecessors = poss_predecessors p wrs in
                  reverse_rippling_in p::path new_predecessors in
reverse_rippling_in [ ] predecessors
```

Figure 5. Algorithm for synthesizing rippling sequence $C \stackrel{R}{\longmapsto} \ldots \stackrel{R}{\longmapsto}$ ind_hyp

path. The head of the computed sequence path corresponds to the last term C_{i+1} of the reverse rippling sequence $C_{i+1} \stackrel{R}{\longmapsto} \ldots \stackrel{R}{\longmapsto}$ ind_hyp. Reverse Rippling is successful, if in the final step a decision procedure proves conc that this formula C_{i+1} implies rip_conc. Otherwise backtracking is initiated.

The rippling / reverse rippling heuristics is embedded in the broader framework of a matching procedure. Therefore, it is independent of the actual logical proof search method and can easily be combined with any of them. The simultaneous matching procedure is able to instantiate existentially quantified variables and solve the step case of an inductive proof automatically. Moreover, we are able to synthesize conditional substitutions. These conditions form a case split in the proof.

4. Integrating into NuPRL

The simultaneous matching procedure based on rippling / reverse rippling heuristics automate key steps in an inductive specification proof. It fits into the broader efforts to empower NuPRL, an interactive, tactic based theorem prover, by automatic tools. Figure 6 illustrates the general structure of NuPRL and its relation to external proof procedures like the presented simultaneous matching procedure based on rippling/reverse rippling heuristic.

The described proof procedure is implemented as an external proof procedure which is executed as the tactic TReverseRippling. It is one of the automatic proof procedure which can be used in NuPRL. All the logical proof search and the rewriting steps are carried out outside of NuPRL. This allows us to use conventional proof search techniques and rippling techniques efficiently. It is also possible for the different automatic proof procedures to interact with each other. This is outlined further in Section 6. The proof procedure receives a proof task from NuPRL, indicated by the horizontal arrows which is extracted from a given synthesis task.

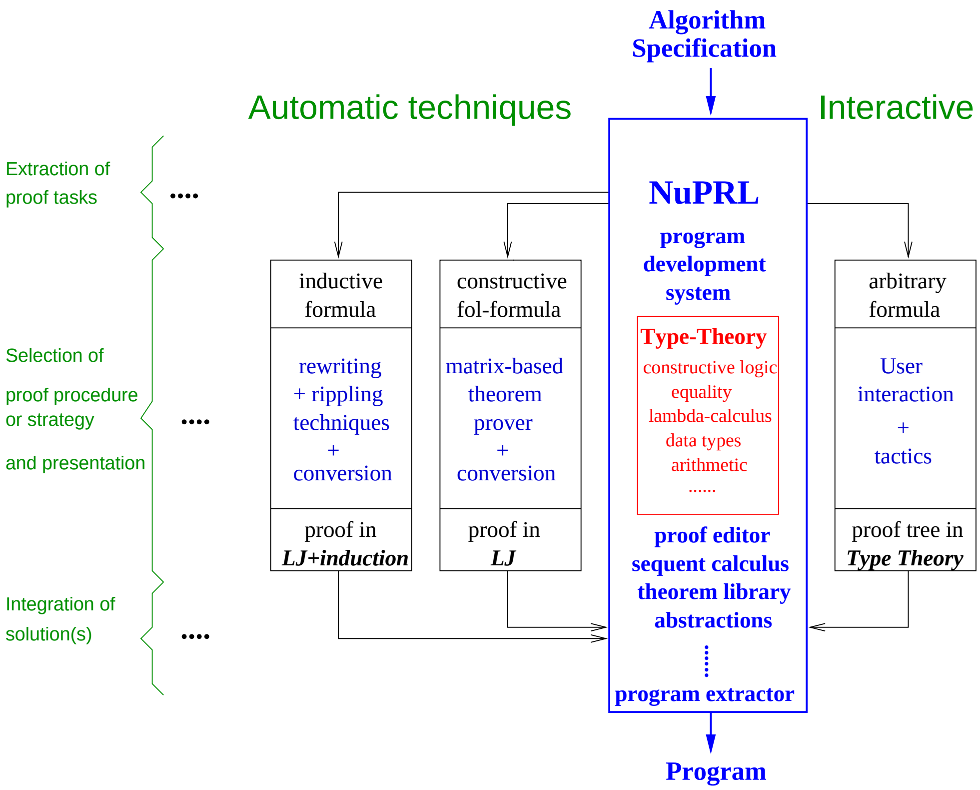


Figure 6. Integrating inductive reasoning techniques into NuPRL

During the rippling/reverse rippling process the theorems from the theorem library are used as wave-rules. Standard arithmetic decision procedures are used to solve trivial proof tasks. The presented proof technique solves automatically a substantial step in the proof of a specification and supports the multi-level approach to program synthesis as discussed in [5]. Other steps which cannot be handled automatically can be proven interactively.

In the previous section we have discussed how to synthesize automatically conditional substitutions and rippling sequences in order to solve a proof task handed by NuPRL. We will now describe how the results from the simultaneous match and rippling are translated back into a valid NuPRL-proof and how a program can be extracted from this proof.

Proof Backtranslation. After solving the subgoals and generating conditional substitutions and rippling sequence(s), these results are translated into sequent proof style. The translation of the logical proof search within the sequent calculus is straight forward. Due to the nature of reverse rippling, the eigenvariablen condition is observed and does not cause any problems.

The synthesized rippling sequence $IC \overset{\mathsf{wr}_0}{\longmapsto} C_1 \overset{\mathsf{wr}_1}{\longmapsto} \ldots \overset{\mathsf{wr}_{n-1}}{\longmapsto} C_n \overset{\mathsf{wr}_n}{\longmapsto} IH$ is translated into a sequence of cut-substitution-lemma applications [13]. The cut formula corresponds to the rippled formula C_i, the lemma corresponds to the applied wave-rule wr_i and the substitution σ describes under which instantiation the wave-rule wr_i is applicable. By this mapping, we can translate the rippling sequence into the following sequent proof.

```
:
\
...  IH ⊢ IC
BY Cut Cn
|\
| ...   IH ⊢ Cn
2 BY Substitution σn THEN Lemma wave-rule  wrn
 \
  ... , IH, Cn ⊢ IC

      :
     ... , IH, Cn , Cn−1 ...   , C0 ⊢ IC
    BY Cut IC
    |\
    | ... , IH, Cn, Cn−1 ...   , C0 ⊢ IC
    2 BY Substitution σ0 THEN Lemma wave-rule  wr0
     \
      8. ... , IH, Cn, Cn−1 ...   , C0 , IC ⊢ IC
      |
      BY Hypothesis
```

The sequent-style proof in NuPRL for the integer square root specification is presented in Figure 7. In this proof the user specified the induction scheme. First, the universal quantifier on the right hand side is decomposed by tactic allR. The tactic TNatInd 'x' then splits the conjecture into base and step case. Our automation efforts concentrate on the step case of the induction. The step cases of an induction proof are challenging for mainly two reasons: 1) It is harder than in the base case to find the witness for the existentially quantified variable(s). 2) Sometimes, a case split is required and the existentially quantified variable is instantiated according to the different cases. These case splits are not immediately obvious, and often require user insight.

The tactic TReverseRippling synthesizes a conditional substitution set for the existentially quantified variable and translates this information into a sequent proof. The proof displays the subtactics which were applied by TReverseRipple. A case split is performed by tactic Decide based on the conditional substitution set before instantiating the existential quantifier on the right hand side. The tactic exL decomposes the existential quantifier on the left hand side. Applications of tactic andL resp.andR eliminate the conjunction on the left resp. right hand side.

Synthesis of the Algorithm. After the proof has been generated we can extract a proof expression from it by using NuPRL's standard extraction mechanism. This expression describes an algorithm that computes an integer square root y for each input x as well as a proof for the correctness of this computation. To synthesize the desired algorithm, we eliminate the proof component of this expression: we insert a projection function and symbolically evaluate the resulting term, which also takes care of the abstractions that were generated by the cuts. These steps lead to the following inductive program term.

```
λx.I(x) where I(α) =
     when α < 0, y = I(α+1). ()
     when α = 0.              0
     when α > 0, y = I(α-1). if α <(y+1)² then y  else y+1
    end
```

This term can easily be translated into an ML program for computing integer square roots.

```
⊩ ∀x:ℕ.∃y:ℕ.  y²≤x ∧ x<(y+1)²
BY allR
|
1.  x:  ℕ
⊩ ∃y:ℕ.  y²≤x ∧ x<(y+1)²
BY TNatInd ‘x‘
|\
| ⊩ ∃y:ℕ.  y²≤0 ∧ 0<(y+1)²
1 BY exR ⌈0⌉
| |
| ⊩ 0*0 ≤ 0 ∧ 0 < (0+1)*(0+1)
1 BY Auto
\
 2.  ∃y:ℕ.  y²≤x ∧ x<(y+1)²
 ⊩ ∃y:ℕ.  y²≤s(x) ∧ s(x)<(y+1)²
 BY exL 2 THEN andL 3
 |
 2.  y:  ℕ
 3.  y² ≤ x
 4.  x < (y+1)²
 BY Decide ⌈(y+1)² ≤ s(x)⌉ THENW Auto
 |\
 | 5.  (y+1)² ≤ s(x)
 1 BY exR ⌈s(y)⌉
 | |
 | ⊩ (s(y))²≤s(x) ∧ s(x)<(s(y)+1)²
 1 BY andR
 | |\
 | | ⊩ (s(y))²≤s(x)
 1 2 BY Auto
 | \
 | ⊩ s(x) < (s(y)+1)²
 1 BY Cut ⌈x + 2*(y+1) + 1 < (y+1)² + 2*(y+1) + 1⌉
 | |\
 | | ⊩ x + 2*(y+1) + 1 < (y+1)² + 2*(y+1) + 1
 1 2 BY Substitution THEN Lemma wave-rule 2
 | \
 | 6.  x + 2*(y+1) + 1 < (y+1)² + 2*(y+1) + 1
 1 BY Cut ⌈x + 2*(y+1) + 1 < (s(y + 1))²⌉
 | |\
 | | ⊩ x + 2*(y+1) + 1 < (s(y + 1))²
 1 2 BY Substitution THEN Lemma wave-rule 4
 | \
 | 7.  x + 2*(y+1) + 1 < s((y + 1))²
 1 BY Cut ⌈x + 2*(y+1) + 1 < (s(y)+1)²⌉
 | |\
 | | ⊩ x + 2*(y+1) + 1 < (s(y)+1)²
 1 2 BY Substitution THEN Lemma wave-rule 3
 | \
 | 8.  x + 2*(y+1) + 1 < (s(y)+1)²
 1 BY Unfold ‘s‘ 0 THEN SupInf THEN Auto
 |   5.  ¬( (y+1)² ≤ s(x) )
 BY exR ⌈y⌉
 |
 ⊩ y² ≤ s(x) ∧ s(x) < (y+1)²
 BY andR
 |\
 | ⊩ y² ≤ s(x)
 | |
 1 BY Unfold ‘s‘ 0 THEN Auto
 \
  ⊩ s(x) < (y+1)²
  BY Auto
```

Figure 7. Proof of the integer square root in NuPRL

5. Extensions and Experimental Results

Experience with the presented technique shows that it substantially increases the degree of automation for specification proofs. Not only are currently no automatic logical proof search methods available, but also there is no support for automatic rewriting. By integrating the discussed methods into NuPRL we are able to overcome these drawbacks. For results of the rippling heuristic for universally quantified statements see [13]. We concentrate here on specifications which can be proven automatically. The examples are drawn from the literature (see [6, 12, 18]) and contain specifications of quotient remainder, append, half, or last. We also applied our technique to the specification of unification algorithm. These examples do not require any case splits, but existentially quantified variables need to be instantiated. We also can prove the specification for $\log_2$ which results in a similar proof to the integer square root example. Moreover, we used the extended matching procedure to instantiate universally quantified variables in the hypothesis list. With this extension we are also able to prove the specification of the integer square root and $\log_2$ by using non-standard induction schemes allowing us to synthesize while loops for these two specifications.

To illustrate the flexibility and strength of our technique, we prove the integer square root specification by a different induction scheme[3]. In the step case (induction proceeds over k) we yield the following conjecture:

$$k:\mathbb{N}, \forall x,y:\mathbb{N}.\ x-y<\mathsf{p}(k) \wedge y^2 \leq x \wedge 0 \leq y \rightarrow \exists n:\mathbb{N}.\ y \leq n \wedge n^2 \leq x \wedge x < (n+1)^2$$
$$\vdash \forall x,y:\mathbb{N}.\ x-y<k \wedge y^2 \leq x \wedge 0 \leq y \rightarrow \exists n:\mathbb{N}.\ y \leq n \wedge n^2 \leq x \wedge x < (n+1)^2$$

Rippling would annotate the term $x-y<k$ to give $\lfloor x \rfloor - \lfloor y \rfloor < \boxed{\mathsf{s}(\mathsf{p}(\underline{k}))}$ as it operates on the induction conclusion. However, no rippling proof for the left hand side of the sequent can be found. Our approach first decomposes the right and left hand side and then uses extended matching to find a match between $x-y<k$ in the hypothesis list and $X-Y<\mathsf{p}(k)$ on the conclusion side. By reverse rippling starting from $x-y<k$ we try to generate a rippling sequence and an instantiation for X and Y. The following additional wave-rules are derived from monotonicity laws and the definition of $-$ is provided:

$$V>0 \wedge U>0 \rightarrow \boxed{\mathsf{p}(\underline{U})} < \boxed{\mathsf{p}(\underline{V})} \quad \stackrel{R}{\longmapsto} \quad U<V \tag{9}$$

$$\boxed{\mathsf{p}(\underline{U-V})} \quad \stackrel{R}{\longmapsto} \quad U - \boxed{\mathsf{s}(\underline{V})} \tag{10}$$

By wave-rule (9) and wave-rule (10) the extended matching procedure generates the following rippling sequence:

$$x - \boxed{\mathsf{s}(\underline{y})} < \boxed{\mathsf{p}(\underline{k})} \quad \underset{(10)}{\stackrel{R}{\longmapsto}} \quad \boxed{\mathsf{p}(\underline{x-y})} < \boxed{\mathsf{p}(\underline{k})} \quad \underset{(9)}{\stackrel{R}{\longmapsto}} \quad x-y<k$$

This example illustrates that the conventional rippling approach [8] to instantiate universally quantified variables in the induction hypothesis by rippling-in is not expressive enough. Moreover, it supports the strength of our approach. The combination of logical proof search and rippling gives us the flexibility to deal with complex logical formulas. More compact logical proof search methods which avoid backtracking and unnecessary simultaneous matching lead to a more efficient procedure.

6. Rippling Match and other Proof Search Methods

In our proof method described so far we have used a sequent prover for the logical decomposition of a given formula. For (constructive) first-order logic, however, there are well-known proof search methods [15, 19] which are much more efficient than sequent-based proof search. Among those, matrix-based proof techniques such as the *connection method* [4, 16] can be understood as a very compact representation of sequent proof techniques. They avoid the usual redundancies contained in the sequent calculus and are driven by *complementary connections*, i.e. pairs of atomic formulae that may become leaves in a sequent proof, instead of the logical connectives of

[3] This induction scheme will result in a more efficient program, namely a while loop. In each iteration y is incremented until $(y+1)^2 > x$.

$$\left[\begin{array}{ccc} x<^1(y_h+1)^2 & x<^0 y_h^2 & \begin{array}{c} x+1<^1 Y_c^2 \\ x+1<^0(Y_c+1)^2 \end{array} \end{array}\right]$$

Figure 8. Matrix of the formula F

a proof goal. This suggests an integration of our rippling-based extended matching procedure into matrix-based proof search procedures in order to combine the strengths of inductive and logical theorem proving. In this section we will summarize the fundamental concepts of the connection method and illustrate the basic ideas of such an integration.

A *formula tree* is the tree-representation of a formula F. Each *position* u in the tree is marked with a unique name and a *label* that denotes the connective of the corresponding subformula or the subformula itself, if it is atomic. In the latter case, u is called an *atom*. The *tree ordering* $<$ of F is the partial ordering on the positions in the formula tree. Each position is associated with a *polarity* (0 or 1) and a *principal type* (α, β, γ, or δ) as in the tableaux calculus. Formulae of type γ can be used in a proof several times in different ways. A quantifier *multiplicity* μ encodes the number of distinct instances of γ-subformulae that need to be considered during the proof search. By F^μ we denote an *indexed formula*, i.e. a formula and its multiplicity.

The *matrix(-representation)* of a formula F is a two-dimensional representation of its atomic formulae without connectives and quantifiers. In it that α-related positions appear side by side and β-related positions appear on top of each other, where two positions u and v are *α-related* (*β-related*) if the greatest common ancestor of u and v wrt. $<$ is of principal type α (β). The matrix-representation of $F \equiv \exists y\ \neg(x<y^2) \wedge x<(y+1)^2 \Rightarrow \exists y\ \neg(x+1<y^2) \wedge x+1<(y+1)^2$ from Section 3.2 (after unfolding the abbreviations $\leq$ and s), where variables of γ-quantifiers are denoted by capital letters, is shown in Figure 8.

The matrix-characterization of logical validity depends on the concepts of paths, connections, and complementarity. A *path* through a formula F is a horizontal path through its matrix-representation, i.e. a maximal set of mutually α-related atomic positions. A *connection* is a pair of atomic positions labelled with the same predicate symbol but with different polarities. In a matrix we indicate connections by arcs between two atoms. A connection is *complementary* if the connected atoms can be made equal by applying some substitution σ to γ-variables.[4] With these definitions the following characterization of logical validity has been shown.

Theorem 6.1. *A formula F is valid iff there is a multiplicity μ, a substitution σ, and a set of complementary connections such that every path through F^μ contains a connection from this set.*

The connection method for (constructive) first-order logic [15, 16] describes an efficient, connection-driven technique to check the complementarity of all paths through a formula F.

[4]The complementarity conditions for *constructive* logic are a bit more complex, since we have to show that positions labelled with $\forall$, $\Rightarrow$, and $\neg$ can be decomposed in a particular order. Formally this can be done by unifying the *prefixes* of the connected atoms. Details can be found in [20, 15]

Starting with all possible paths through F it eliminates step by step all paths that contain a given connection, provided it can be shown to be complementary. In order to extend this method to inductive theorem proving we need to modify the matrix-characterization in two ways.

A first extension comes from the observation that a pair $\{A^1, \bar{A}^0\}$ of connected atomic formulae does not necessarily have to be equal under a substitution σ. As complementary connections correspond to leaves in a sequent proof we only have to require that the negative atom A, i.e. the left side of the leaf sequent, *implies* $\bar{A}$ under σ, where the implication can be proven by standard decision procedures or by rewriting $\bar{A}$ into A. Thus we consider *directed connections* $(A^1, \bar{A}^0)$ and require A to imply $\bar{A}$ with respect to a given theory $\mathcal{T}$, denoted by $A \Rightarrow_{\mathcal{T}} \bar{A}$.

Definition 6.1. A directed connection (u^1, v^0) is *σ-complementary* with respect to a theory $\mathcal{T}$ iff $\sigma\circ A = \sigma\circ\bar{A}$ or $\sigma\circ A \Rightarrow_{\mathcal{T}} \sigma\circ\bar{A}$, where $A = label(u)$ and $\bar{A} = label(v)$.

A second extension comes from the need for *conditional substitutions*. The conditions generated by our extended matching procedure will lead to a case analysis in the first step of a top-down sequent proof such that each of the resulting subgoals can be proven by "conventional" proof techniques. A justification of this approach comes from the observation that a formula F is valid if and only if there is a set $\{\mathrm{c}_1,..,\mathrm{c}_n\}$ of logical constraints such that $C = \mathrm{c}_1 \vee .. \vee \mathrm{c}_n$ and each of the formulae $F_i = \mathrm{c}_i \Rightarrow F$ are logically valid, which can easily be proven by the cut rule.

Theorem 6.2. *A formula F is valid iff there is a set $\{c_1,..,c_n\}$ of constraints such that $C = c_1 \vee .. \vee c_n$ is valid and for all i there is a multiplicity μ_i, a substitution σ_i, and a set $\mathcal{C}_i$ of σ_i-complementary connections such that every path through $c_i \Rightarrow F^{\mu_i}$ contains a connection from $\mathcal{C}_i$.*

Extending the connection method according to the above modifications enables us to prove inductive specification theorems with existential quantifiers in a straightforward way. Essentially we only have to integrate our rippling-based matching procedure into the usual unification process and to add a mechanism for expanding a given matrix by a case distinction whenever our matching procedure generates a conditional substitution. We conclude this section with a small example that shows the advantages of these extensions.

Consider the matrix of $F \equiv \exists y\ \neg(x<y^2) \wedge x<(y+1)^2 \Rightarrow \exists y\ \neg(x+1<y^2) \wedge x+1<(y+1)^2$ in Figure 8. As connections should mainly run between atoms of the induction hypothesis and the corresponding axioms of the conclusion, the matrix contains only two useful connections $(x<^1(y_h+1)^2,\ x+1<^0(Y_c+1)^2)$ and $(Y_c^2<^1 x+1,\ y_h^2<^0 x)$. Since x and y_h are constants neither of the two connections can be unified and we have to use the rippling techniques described above to show their complementarity. In the first case $\boxed{1}$ this eventually leads to the substitution $\sigma_1=\{Y_c \backslash y_h+1\}$. As the instantiated second connection does not describe an arithmetically valid implication, which can easily be checked by arithmetical decision procedures, we add the open subgoal $x+1<(y_h+1)^2$ as prerequisite c_1 (i.e. with opposite polarity 0) to the matrix and have thus established the validity of the sequent under this condition. To complete the proof we now try to establish the validity of the sequent under the negated condition $c_2 = x+1<^1(y_h+1)^2$. By connecting this condition to $x+1<^0(Y_c+1)^2$ we get $\sigma_2=\{Y_c \backslash y_h\}$ which also makes the connection $\boxed{2}$ an arithmetically valid implication. Thus F is valid according to theorem 6.2.

Matrix methods, like our our extended matching procedure, can treat quantified variables in the conclusion and universally quantified variables in the hypotheses uniformly, as they are both of type γ. Thus a combined procedure will be able to handle examples like the one given in section 5 easily. Details of such a procedure will be elaborated in the future.

7. Related Work

One of the first approaches to automate the instantiation of existentially quantified variables has been by Biundo [6]. Existentially quantified variables are replaced by Skolem functions which describe the program which is to be synthesized. After induction the formula in the step case is put into clausal form. The synthesis proceeds by *clause–set translations* (e.g. rewriting and case splitting) which induce an AND/OR search space. The work of Kraan *et al.*[12] builds upon the idea to replace existentially quantified variables by skolem functions in order to synthesize logical programs. In order to control better the search space within the inductive step, rippling and middle–out reasoning [10] are used to construct predicate definitions from specifications in classical logic. However, both approaches do not guarantee that the synthesized program is correct, it has to be verified after the synthesis. We believe that constructive type theory provides a firmer mathematical foundation than is found in these systems.

In Smaill & Green [18], an approach for the synthesis of functional programs within the framework of constructive type theory is suggested. This approach builds on higher–order embeddings and higher–order rippling. Middle-out reasoning and higher order embeddings have the disadvantage of a big search space, as rippling in the presence of higher-order function variables does not terminate.

The rippling approaches rely exclusively on this technique and encode logical inference rules as wave-rules. The whole induction conclusion is rippled and these systems aim for a match of the whole induction conclusion with the entire induction hypothesis. The underlying logical calculus is not used to decompose the step case during proof search. This causes major problems when we deal with specifications of more complex formulas, as we illustrated in section 5.

Related to the reverse rippling heuristic is is a technique to synthesize induction orderings for existence proofs by Hutter [11]. In order to find automatically the appropriate induction scheme for a specification, it looks at the surrounding function symbols of the existentially quantified variable, and guesses the instantiation of this variable based on the applicable wave-rules. However, analyzing the immediately surrounding functions in order to find an appropriate instantiation for the existentially quantified variable might be too restrictive, for example analyzing $\mathsf{even}(X+y)$ would suggest to instantiate X with $s(x)$ as addition $+$ is recursively defined over the first argument. However after this step rippling is stuck, as the function even is defined in a two step recursion. Our technique proceeds not from the existentially variable to the outside but rather the other way round. This has the advantage that it avoids these stuck-states, but it needs to guess and incrementally refine wave-fronts. The incrementally refining of the wave-fronts is restricted to one level which reduces search space dramatically. Moreover reverse rippling is combined with rippling which gives our rippling strategy additional flexibility. The power of our rippling / reverse rippling method stems from its integration into the simultaneous match which can be easily combined with any logical prover.

8. Conclusion and Future Work

We have presented an approach for the instantiation of existentially quantified variables which provides a significant degree of automation to proofs in constructive type theory. The key idea is to use first-order meta-variables in place of the existential witness during proof search and rippling and instantiate this meta-variable by an extended matching procedure. Because we reason backwards from the induction hypothesis towards the rippled conclusion by reverse rippling, our approach is highly goal directed and we are able to synthesize lemmata during reverse rippling. The only additional proof search involved is the incremental instantiation of the wave-front during reverse rippling. This search is constrained in such a way that the wave-front has to be refined in each step, i.e. we do not introduce new meta-variables. Moreover the rippling-distance strategy ensures that the reverse rippling process is terminating. By embedding rippling heuristics into the simultaneous matching procedure, we are able to synthesize case splits which cannot be derived by other comparable systems. The power of this simultanous match stems from the easy integration into logical proof search methods, like the connection method. By this approach we merge the advantages of logical proof search, like efficiency and compactness with the goal-directed rewriting power of rippling. In the future we plan to elaborate further details of the integration of matrix methods and our simultanous matching procedure. Part of this work will involve a more sophisticated backtranslation algorithm of the matrix proof and simultanous matching. In this regard we will extend backtranslation methods as discussed in [17] to incorporate the additional features.

References

[1] Armando, A. and Smaill, A and Green, I.: Automatic synthesis of recursive programs: The proof-planning paradigm. In *12th IEEE International Automated Software Engineering Conference* Incline Village, Nevada, IEEE Computer Society, 1997, 2–9.

[2] Basin, D. and Walsh, T.: A calculus for and termination of rippling. *Journal of Automated Reasoning*, **16**(2), 1996, 147–180.

[3] Bates, J. L. and Constable, R. L.: Proofs as programs. *ACM Transactions on Programming Languages and Systems*, **7**(1), 1985, 113–136.

[4] Bibel, W.: On matrices with connections. *Journal of the Association for Computing Machinery*, **28**, 1981, 633–645.

[5] Bibel, W., Korn, D., Kreitz, C., Kurucz, F., Otten, J., Schmitt, S. and Stolpmann, G.: A multi-level approach to program synthesis. In N.E. Fuchs, ed. *7th International Workshop on Logic Program Synthesis and Transformation (LOPSTR'97)*, Leuven, Belgium, 1997, LNAI 1463, 1998, 1–25.

[6] Biundo, S.: Automated synthesis of recursive algorithms as a theorem proving tool. In *8th European Conference on Artificial Intelligence*, München, Germany, 1988.

[7] Bundy, A., Stevens, A., van Harmelen, F., Smaill, A. and Ireland, A.: Rippling: A heuristic for guiding inductive proofs. *Artificial Intelligence*, **62**(2), 1993, 185–253.

[8] Bundy, A., van Harmelen, F., Smaill, A. and Ireland, A.: Extensions to the rippling–out tactic for guiding inductive proofs. In *10th International Conference on Automated Deduction*, Kaiserslautern, Springer, 1990, 132–146.

[9] Constable, R. L. and et al. *Implementing Meta-Mathematics with the Nuprl Proof Development System.* Prentice-Hall, 1986.

[10] Hesketh, J. T.: *Using Middle-Out Reasoning to Guide Inductive Theorem Proving.* PhD thesis, Dept. of Artificial Intelligence, University of Edinburgh, 1991.

[11] Hutter, D.: *Synthesis of Induction Orderings for Existence Proofs.* In A. Bundy (ed.),*12th International Conference on Automated Deduction*, Nancy, France, LNAI 814, Springer, 1994, .

[12] Kraan, I., Basin, D. and Bundy,A.: Logic program synthesis via proof planning. In K. K. Lau and T. Clement, eds., *Logic Program Synthesis and Transformation*, pages 1–14. Springer, 1993.

[13] Kurucz, F.: Realisierung verschiedender Induktionsstrategien basierend auf dem Rippling-Kalkül. Master's thesis, Darmstadt University of Technology, 1997.

[14] Martin-Löf,P.: Constructive mathematics and computer programming. In *6-th International Congress for Logic, Methodology and Philosophy of Science, 1979*, North-Holland, 1982, 153–175.

[15] Otten, J. and Kreitz, C.: A connection based proof method for intuitionistic logic. In P. Baumgartner, R. Hähnle & J. Posegga, eds., *4^{th} Workshop on Theorem Proving with Analytic Tableaux and Related Methods*, LNAI 918, Springer, 1995, 122–137.

[16] Otten, J. and Kreitz, C.: A Uniform Proof Procedure for Classical and Non-classical Logics. In G. Görz & S. Hölldobler, eds., *KI-96: Advances in Artificial Intelligence*, LNAI 1137, Springer, 1996, 307–319.

[17] Schmitt, S. and Kreitz, C.: Converting non-classical matrix proofs into sequent-style systems. In M. McRobbie and J. Slaney, eds., *13th International Conference on Automated Deduction*, New Brunswick, NJ, LNAI 1104, Springer, 1996, 418–432.

[18] Smaill, A. and Green, I.: Automating the synthesis of functional programs. Research paper 777, Dept. of Artificial Intelligence, University of Edinburgh, 1995.

[19] Tammet, T.: A resolution theorem prover for intuitionistic logic. In M. McRobbie and J. Slaney, eds., *13th International Conference on Automated Deduction*, New Brunswick, NJ, LNAI 1104, Springer, 1996, 2–16.

[20] Wallen, L.: *Automated deduction in nonclassical logic.* MIT Press, 1990.