Matrix-based Inductive Theorem Proving

Christoph Kreitz[1] & Brigitte Pientka[2]

[1] *Department of Computer Science, Cornell-University*
Ithaca, NY 14853-7501, USA
kreitz@cs.cornell.edu

[2] *Department of Computer Science, Carnegie Mellon University*
Pittsburgh, PA, USA
bp@cs.cmu.edu

Abstract. We present an approach to inductive theorem proving that combines rippling-based rewriting and matrix-based logical proof search. The selection of appropriate connections for the matrix proof is guided by the symmetries between induction hypothesis and induction conclusion while unification is extended by a rippling/reverse-rippling heuristic. Conditional substitutions are generated whenever a uniform substitution is impossible. We illustrate the strength of the combined approach by discussing inductive proofs for the integer square root problem.

1 Introduction

Formal methods have become become increasingly important for the verification and synthesis of software for safety-critical applications. Over the past years, a number of proof assistants such as NuPRL [11] or Coq [12] have been developed to support the logical reasoning process involved in these methods. But they suffer from the low degree of automation, which is due to their expressive underlying calculi. It is therefore desirable to extend the reasoning power of proof assistants by integrating well-understood techniques from automated theorem proving.

Logical proof search methods [4, 19, 25] have been successfully used for proving formulas in constructive first order logic. But these methods cannot deal with the inductive proofs obligations that typically occur when reasoning about recursive programs. Inductive theorem proving techniques, on the other hand, have been very successful in solving such proof obligations. *Rippling* [9, 10] is a powerful annotated rewriting technique that guides the rewriting process from the induction conclusion towards the application of the induction hypothesis. However, little focus has been devoted to logical proof search and the automatic instantiation of existentially quantified variables, which is necessary for synthesizing programs from their specifications. In fact, many synthesis problems require both first-order and inductive proof methods to be applied simultaneously, as both techniques are not strong enough to solve the problem independently.

Example 1. Consider the formula $\forall x\,\exists y\; y^2{\le}x \,\wedge\, x{<}(y{+}1)^2$, which specifies an algorithm for computing the integer square root of a natural number x. A straightforward inductive proof for this formula will lead to the step case

$$\exists y\; y^2{\le}x \;\wedge\; x{<}(y{+}1)^2 \quad\vdash\quad \exists y\; y^2{\le}x{+}1 \;\wedge\; x{+}1{<}(y{+}1)^2$$

No single rippling sequence can rewrite the induction conclusion into the hypothesis, as the choice of y in the conclusion (Y_c) strongly depends on the properties of the y in the hypothesis (y_h). If $(y_h{+}1)^2{\leq}x{+}1$ then Y_c must be $y_h{+}1$ to make the first conjunct valid, while otherwise Y_c must be y_h to satisfy the second conjunct. Rippling would be able to rewrite the conclusion into the hypothesis in each of these two cases, but it can neither create the case analysis nor infer the value to be instantiated for Y_c.

Logical proof methods would not be able to detect the case distinction either, as they would have to prove the lemmata $x{<}(y_h{+}1)^2 \Rightarrow x{+}1{<}(y_h{+}1{+}1)^2$ for the first case, $y_h^2{\leq}x \Rightarrow y_h^2{\leq}x{+}1$ for the second, and also $(y_h{+}1)^2{\leq}x{+}1 \vee (y_h{+}1)^2{>}x{+}1$. Of course, it is easy to prove the induction step case if the crucial lemmata are already provided. But one would have to consult hundreds of lemmata about arithmetic to find the ones that complete the proof, which would make the proof search very inefficient.

To overcome these deficiencies we have combined logical proof search techniques with controlled rewrite techniques such as rippling in order to improve the degree of automation when reasoning inductively about program specifications. In [21] we have shown that a combination of a rippling-based extended matching procedure and a simple sequent prover can be used to generate the case analysis and to solve synthesis problems like the above automatically. For (constructive) first-order logic, however, there are well-known proof methods [19, 25], which are much more efficient than sequent-based proof search. Among those, matrix-based proof techniques such as the *connection method* [4, 20] 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. possible leaves in a sequent proof, instead of the logical connectives of a proof goal. On the other hand, there are algorithms for converting matrix proofs back into sequent proofs [24], which enables us to use matrix methods for guiding proof and program development in interactive proof assistants [7, 17]. These advantages suggest an integration of our extended rippling techniques into matrix-based proof search procedures in order to combine the strengths of goal-directed rewriting and logical theorem proving.

In this paper we will present concepts for combining rippling techniques and matrix-based theorem proving. In Section 2 we will resume our extended rippling techniques while Section 3 summarizes the theoretical foundations of matrix-based logical proof search. In Section 4 we extend these theoretical foundations according to our extended rippling techniques. In Section 5 we will describe an inductive proof method that is based on these extensions.

2 Rippling and Inductive Theorem Proving

Rippling is an annotated rewriting technique specifically tailored for inductive theorem proving. Differences between the induction hypothesis and the induction conclusion are marked by meta-level annotations, called *wave annotations*. Expressions that appear in both are called *skeleton*. Expressions that appear only in the conclusion are called *wave fronts*. The induction variable that is surrounded by a wave front is called *wave hole*. $\lfloor$*Sinks*$\rfloor$ are parts of the goal that correspond to universally quantified variables in the hypothesis. We call the annotated rewrite rules *wave rules*. To illustrate, consider a wave rule that is derived from the recursive definitions of $+$.

$$\boxed{\mathsf{s}(\underline{U})}^{\uparrow} + V \overset{R}{\longmapsto} \boxed{\mathsf{s}(\underline{U+V})}^{\uparrow}$$

In this rule, $\boxed{\mathsf{s}(\ldots)}$ marks a wave front. The underlined parts $\underline{U}$ and $\underline{U+V}$ mark 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 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 (*rippling-in*). If the annotations can be moved 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 no wave rule is applicable anymore and the hypothesis does not match the conclusion.

Basin & Walsh [2] have developed a calculus for rippling and defined a well-founded *wave measure* under which rippling terminates if no meta-variables occur in the goal. The measure associates weights to the wave fronts to measure its *width* and *size*. Rewriting is restricted to the application of wave rule that are skeleton preserving and measure decreasing according to the defined wave measure. For instantiating existentially quantified variables via rippling, mainly two approaches have been suggested in the literature. Bundy et al. [10] suggests special existential wave rules that can be derived from non-existential ones. This, however, increases the search problem in the presence of existential quantifiers. Other approaches [1, 15, 22] use meta-annotations, which requires higher-order unification and may lead to non-terminating rippling sequences.

Our approach in [21] overcomes these drawbacks by combining rippling with first-order theorem proving techniques and extending the matching procedure for instantiating existentially quantified variables. It proceeds in three steps.

First, we decompose the step case formula of an inductive proof into several subgoals and introduce meta-variables in place of existentially quantified variables. We use a sequent prover for constructive logic for this purpose in order to be able to integrate the results into the NuPRL proof development system [11] and to extract programs from the inductive proof.

Next we apply a *simultaneous match* to compute substitutions for the meta-variables incrementally. There are three possibilities for a match. Given a substitution σ and a term C from the induction conclusion we can (1) prove the goal $\sigma(C)$ by a decision procedure, (2) unify $\sigma(C)$ and the corresponding hypothesis term H from the induction hypothesis, or (3) compute a rippling sequence and a substitution σ' such that $\sigma'(C)$ can be rippled to H. The key part of the third alternative is the rippling/reverse rippling heuristic, which is illustrated below.

$$\begin{tikzcd} C \arrow[r, mapsto, "R"] \arrow[rrr, bend right, "\textit{rippling}"'] & C_0 \arrow[r, mapsto, "R"] & \ldots \arrow[r, mapsto, "R"] & C_i \arrow[r, mapsto, "R"] & \ldots \arrow[r, mapsto, "R"] & C_n \arrow[r, mapsto, "R"] & H \arrow[lll, bend left, "\textit{reverse rippling}"] \end{tikzcd}$$

The rippling sequence is generated in two phases: First the term C in the induction conclusion is rewritten to some formula C_i. If C_i does not match the corresponding term H in the induction hypothesis then a rippling sequence $C_i \xmapsto{R} \ldots \xmapsto{R} C_n \xmapsto{R} H$ is computed by backwards reasoning from the term H towards C_i. This process, that we call *reverse rippling*, also instantiates the meta-variables in C. If the generated sequence is empty, then either C implies H under the given substitution or C and H can be unified. The search for a rippling sequence is based on the *rippling-distance* strategy introduced in [18], which uses a new measure to ensure termination. In each rippling step the distance between the wave front and the sink must decrease. This allows a uniform treatment of the different rippling techniques, a more goal-oriented proof search, and avoids non-termination problems in the presence of unknown wave fronts. For a more detailed analysis we refer to [18, 7].

After extended matching has generated a rippling sequence and a substitution σ for one subgoal we check whether all remaining subgoals are true under σ as well. In order to restrict the search space, we require the subgoals to be provable by standard arithmetic, rippling, and matching. If a subgoal cannot be proven by these techniques, we select this subgoal as a constraint for σ. We then prove all subgoals under the negated constraint and continue until we have found a complementary set of constrained substitutions that solve the induction step. The generated constraints form a case analysis in a sequent proof and lead to a conditional in the synthesized program. Our simple heuristic for introducing constraints could be replaced by more refined methods, but turned out to be sufficient for many induction problems.

Example 2. Consider again the integer square-root problem from example 1. To prove $\exists y\ y^2{\le}x \wedge x{<}(y{+}1)^2 \ \vdash\ \exists y\ y^2{\le}\mathsf{s}(x) \wedge \mathsf{s}(x){<}(y{+}1)^2$ we decompose the induction hypothesis and the conclusion.[1] The existentially quantified variable in the conclusion is replaced by a meta-variable Y_c. This gives us two subgoals:

$$\underline{y_h^2 \le x},\ x < (y_h{+}1)^2 \quad\vdash\quad Y_c^2 \le \boxed{\mathsf{s}(\underline{x})} \tag{1}$$

$$y_h^2 \le x,\ \underline{x < (y_h{+}1)^2} \quad\vdash\quad \boxed{\mathsf{s}(\underline{x})} < (Y_c{+}1)^2 \tag{2}$$

Next we annotate the conclusion such that its skeleton matches the corresponding part in the induction hypothesis. Corresponding parts are underlined. In the induction hypothesis y_h is marked as a sink variable because its mirror image in the conclusion is the meta-variable that we want to instantiate. The following wave rules are derived from the definitions of functions used in the specification:

$$\boxed{\underline{U}+W} < \boxed{\underline{V}+W} \quad\xmapsto{R}\quad U < V \tag{3}$$

$$\boxed{s(\underline{U})} + V \quad\xmapsto{R}\quad \boxed{s(\underline{U+V})} \tag{4}$$

$$(\boxed{s(\underline{A})})^2 \quad\xmapsto{R}\quad \boxed{\underline{A}^2 + 2A + 1} \tag{5}$$

We start with the second subgoal (2) and try to match $C = \boxed{\mathsf{s}(\underline{x})} < (Y_c{+}1)^2$ and the induction hypothesis $H = x < (y_h{+}1)^2$. The latter represents the final formula in the rippling sequence. As no wave rule is applicable, rippling leaves the subgoal

[1] For convenience we use $a{\le}b$ as abbreviation for $\neg(b{<}a)$ and $\mathsf{s}(x)$ for $x{+}1$.

unchanged and reverse rippling reasons backwards from H to C. In order to determine which formula may precede the induction hypothesis, we inspect the right hand sides of the wave rules. Wave rule (3) suggests that the predecessor has the form $\boxed{\underline{x}+W} < \boxed{\underline{(y_h+1)^2}+W}$. This formula can then be rippled by wave rule (5) which instantiates W with $2(y_h+1)+1$. Rippling towards the sink variable y_h is now straightforward. By wave rule (4) the wave front s is moved to a position where it surrounds y_h, which leads to $C_0 = \boxed{\underline{x}+2(y_h+1)+1} < (\boxed{\boxed{\mathsf{s}(\underline{y_h})}}+1)^2$. As rippling has terminated successfully we can instantiate Y_c by the sink value and get the substitution $\sigma_1 := \{Y_c \backslash \mathsf{s}(y_h)\}$. To prove that C_0 implies $\sigma_1(C)$ we use a decision procedure for elementary arithmetic.

As subgoal (1) is not valid under σ_1, we use its instantiated conclusion $(\mathsf{s}(y_h))^2 \leq \mathsf{s}(x)$ as constraint and try to prove both subgoals for the case $\neg(\mathsf{s}(y_h))^2 \leq \mathsf{s}(x)$. We unfold the abbreviations s and $\leq$ and use normal matching to derive a substitution σ_2. Matching $\mathsf{s}(x) < (Y_c+1)^2$ from subgoal (2) and $\mathsf{s}(x) < (y_h+1)^2$ yields the substitution $\sigma_2 := \{Y_c \backslash y_h\}$, which also makes the subgoal (1) true under the given constraint.

Thus the step case of the induction is provable under the set of conditional substitutions $\{[(\mathsf{s}(y_h))^2 \leq \mathsf{s}(x), \{Y_c \backslash \mathsf{s}(y_h)\}], [\neg((\mathsf{s}(y_h))^2 \leq \mathsf{s}(x)), \{Y_c \backslash y_h\}]\}$.

After conditional substitutions and rippling sequences that prove the induction step have been found, the results are translated straightforwardly into a sequent proof. Following the *proofs-as-programs* principle [3] we can then extract a proof expression that describes an algorithm for the given specification.

Our method has been implemented as NuPRL-tactics (see [21] for details) and applied to several program specifications. It can also be used for the instantiation of universally quantified variables in the hypothesis list, which allows us to select non-standard induction schemes that enable us to synthesize more efficient while loops. But although our method is universal, the sequent prover that we have used so far is not the most efficient logical proof method. In the rest of this paper we will show how to combine our approach with more efficient proof procedures.

3 Matrix-based Theorem Proving

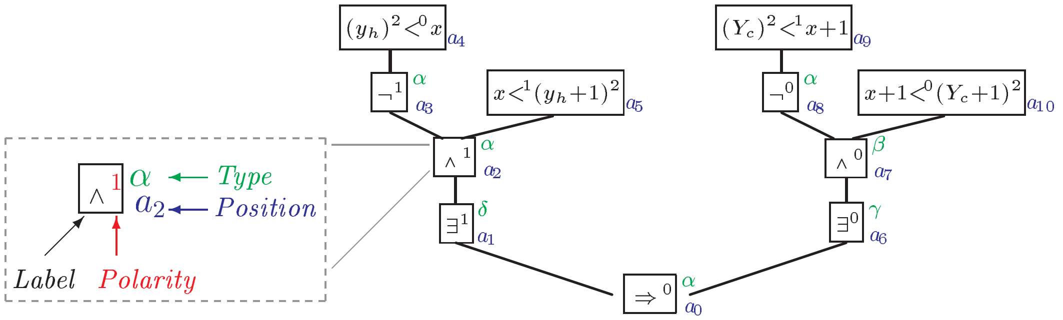
Matrix-based proof search procedures [4, 6] can be understood as compact representations of tableaux or sequent proof techniques. They avoid the usual redundancies contained in these calculi and are driven by *complementary connections*, i.e. pairs of atomic formulae that may become leaves in a sequent proof, instead of the logical connectives of a proof goal. Although originally developed for classical logic, the *connection method* has recently been extended to a variety of non-classical logics such as intuitionistic logic [19], modal logics [20], and fragments of linear logic [16]. In this section we will briefly summarize the theoretical foundations of this method

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 formula tree for

$$F_0 \;\equiv\; \exists y\,\neg(x<y^2) \wedge x<(y+1)^2 \;\;\Rightarrow\;\; \exists y\,\neg(x+1<y^2) \wedge x+1<(y+1)^2$$

from the example 1 (after unfolding the abbreviations $\leq$ and s), together with polarities and principal types is presented below. As usual we denote γ-variables by capital letters.



The *matrix(-representation)* of a formula F is a two-dimensional representation of its atomic formulae without connectives and quantifiers. In it 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_0 is shown in Figure 1.

$$\left[\;\left[\; x<^1(y_h+1)^2 \qquad x<^0 y_h^2 \;\right] \quad \begin{bmatrix} x+1<^1 Y_c^2 \\[2em] x+1<^0(Y_c+1)^2 \end{bmatrix}\;\right]$$

Fig. 1. Matrix of the formula F_0

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.[2] With these definitions the following characterization of logical validity has been shown.

[2] The complementarity conditions for *constructive* logic are 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 [26, 19].

Theorem 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.*

Example 3. $F_0 \;\equiv\; \exists y\;\neg(x{<}y^2) \wedge x{<}(y{+}1)^2 \;\Rightarrow\; \exists y\;\neg(x{+}1{<}y^2) \wedge x{+}1{<}(y{+}1)^2$ from example 1 can be proven in first-order logic if we add to it the crucial lemmata $\forall z\forall t\; t{+}1{<}z \Rightarrow t{<}z$ and $\forall s\forall r\; s{<}r^2 \Rightarrow s{+}1{<}(r{+}1)^2$ as well as the case analysis $\forall u\forall v\; v{<}u \vee \neg(v{<}u)$, as the following matrix proof with $\mu{=}2$ in the conclusion shows.

$$\left[\begin{array}{ccccccc} x{<}^0 y_h^2 & \boxed{\begin{matrix} T{+}1{<}^0 Z \\[2ex] T{<}^1 Z \end{matrix}} & \begin{matrix} x{+}1{<}^1 Y_{c1}^2 \\[2ex] x{+}1{<}^0 (Y_{c1}{+}1)^2 \end{matrix} & \boxed{\begin{matrix} V{<}^0 U \\[2ex] V{<}^1 U \end{matrix}} & \begin{matrix} x{+}1{<}^1 Y_{c2}^2 \\[2ex] x{+}1{<}^0 (Y_{c2}{+}1)^2 \end{matrix} & \boxed{\begin{matrix} S{<}^0 R^2 \\[2ex] S{+}1{<}^1 (R{+}1)^2 \end{matrix}} & x{<}^1 (y_h{+}1)^2 \end{array}\right]$$

There are 32 paths through the extended formula, each containing one of the six connections depicted above. The terms of these connections can be unified by the substitution $\sigma = \{Z\backslash y_h^2, T\backslash x, Y_{c1}\backslash y_h, V\backslash x{+}1, U\backslash(y_h{+}1)^2, Y_{c2}\backslash y_h{+}1, S\backslash x, R\backslash y_h{+}1\}$. Thus the formula F_0 has become valid after being extended by the three arithmetical lemmata.

The connection method for (constructive) first-order logic describes an efficient, connection-driven technique to check the complementarity of all paths through a formula F. 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. A detailed description of this method and its justification can be found in [19, 20].

4 Extending the Matrix-Characterization

In order to integrate our extended rippling techniques described in Section 2 into matrix-based proof search procedures we have to modify the matrix-characterization of logical validity in several ways.

A first extension of the matrix-characterization 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 non-logical methods such as decision procedures or rewriting $\bar{A}$ into A. In order to modify the matrix-characterization accordingly we have to consider *directed connections* (u^1, v^0), introduce a notion of *implication with respect to a given theory* $\mathcal{T}$, denoted by $\Rightarrow_{\mathcal{T}}$, and extend the concept of complementarity.

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

If we allow reasoning within a theory $\mathcal{T}$ then equality and implication are not the only means to close a branch in a sequent proof. In addition to that the sequents $\cdot \vdash \bar{A}$ and $A \vdash \cdot$ are provable, if $\bar{A}$ can be proven in the theory $\mathcal{T}$ and A can be proven to be false in $\mathcal{T}$. For the sake of uniformity of notation we call the corresponding atomic positions *unary connections*.

Definition 2. Let u^1 and v^0 be unary connections, $A{=}label(u)$, and $\bar{A}{=}label(v)$.
u^1 is *σ-complementary with respect to a theory $\mathcal{T}$* iff $\sigma(A) \Rightarrow_{\mathcal{T}} \mathsf{False}$.
v^0 is *σ-complementary with respect to $\mathcal{T}$* iff $\mathsf{True} \Rightarrow_{\mathcal{T}} \sigma(\bar{A})$.

Our concept of complementary with respect to a theory $\mathcal{T}$, which resembles Bibel's theory-connections [6], is quite general and could also be extended to n-ary connections w.r.t some theory $\mathcal{T}$. But as we are mainly interested in rippling-based inductive theorem proving we restrict ourselves to implications that can be proven by standard decision procedures and to rippling sequences generated by extended matching. In the context arithmetic we fix the notion of implication with respect to the theory $\mathcal{A}$ of elementary arithmetic as follows.

Definition 3. A formula A *implies a formula $\bar{A}$ with respect to the theory $\mathcal{A}$* of elementary arithmetic, denoted by $A \Rightarrow_{\mathcal{A}} \bar{A}$ if

1. $A \Rightarrow \bar{A}$ is provable by an arithmetic decision procedure, or
2. there is a rippling sequence $\bar{A} \overset{R}{\longmapsto} \ldots \overset{R}{\longmapsto} A$ that rewrites $\bar{A}$ into A by using arithmetical wave rules.

With the above extensions of the notion of complementarity the matrix-characterization of validity can be formulated exactly as in theorem 1 and we can use the same proof search method as before. To show complementarity, we can prove a negative literal A to be false or a positive literal $\bar{A}$ to be true with an arithmetical decision procedure, unify two connected literals A and $\bar{A}$ via standard unification procedures, prove the implication $A \Rightarrow \bar{A}$ with decision procedures, or find a substitution σ and a rippling sequence $\sigma(\bar{A}) \overset{R}{\longmapsto} \ldots \overset{R}{\longmapsto} \sigma(A)$ with our rippling / reverse rippling heuristic.[3] To close the gap between forward rippling and reverse rippling we use a decision procedure (c.f. example 2). This makes our method more flexible and enables it to find more proofs.

A second efficiency improvement during proof search comes from the fact that there is a strong relation between individual subformulae of the induction hypothesis H and the conclusion C. As C is usually identical to H, except for the induction variable x being replaced by $x{+}1$ or some other induction term, a rippling proof usually links a subformula of H to the corresponding subformula of C. Thus the proof-relevant connections are usually the ones between the corresponding subformulae of H and C. We can therefore reduce the search space of a matrix-proof by investigating these connections first. In order to make use of this observation we introduce the concept of *orthogonal* matrices and connections, which is motivated by the fact that in the matrix representation of an inductive step $H \Rightarrow C$ (see e.g. Figures 1 and 2) the sub-matrices for H and C are orthogonal to each other. To generalize beyond simple induction we require $C = H[x \backslash \rho(x)]$, where ρ is a constructor like s or a destructor like p for the inductive domain.

[3] Note that we have only extended the *term-unification* between two connected literals. If we want to use the inductive proof for program synthesis, we still need to unify the *prefixes* of the connected atoms in order to prove constructive validity.

Definition 4. A formula F is *orthogonal with respect to the variable x* if $F \equiv H \Rightarrow C$ and $C = H[x\backslash\rho(x)]$ for some substitution ρ. A connection (u^1, v^0) (or (v^1, u^0)) where $u \in H$ and $v \in C$ is *orthogonal in F* if the relative position of u in H is identical to the relative position of v in C.

Orthogonal connections are easy to find, as we connect literals with the same relative positions in the two major subtrees of F. The following theorem shows that in an inductive proof it is sufficient to consider orthogonal connections.[4]

Theorem 2. *An orthogonal matrix F is valid if all orthogonal connections in F are complementary under some substitution σ*

Proof. Using theorem 1 we show by induction on the structure of H that every path through an orthogonal matrix $F \equiv H \Rightarrow C$ contains an orthogonal connection.
If H is atomic the statement is trivially true. If H has the form $H_1\, op\, H_2$ where op is of type α in F then $C = C_1\, \bar{op}\, C_2$ where $\bar{op}$ has type β in F. Consequently each path $\mathcal{P}$ through F goes through both H_1 and H_2 and either C_1 or C_2. By the induction hypothesis $\mathcal{P}$ must contain an orthogonal connection. The argument is the same if op is of type β in F, while γ- and δ-quantifiers do not affect the set of paths. $\square$

It should be noted that checking the complementarity of orthogonal connections, which resembles Bibel's linear connection method [5] but is used for a different purpose, gives only a sufficient but not a necessary condition for validity. If a proof based on orthogonal connections fails, it may not be an inductive one and we may still attempt a conventional matrix proof.

A third extension of the matrix-characterization comes from the need for conditional substitutions in some inductive proofs. In a sequent proof the conditions generated by our extended matching procedure will lead to a case analysis as a first step, while each case can be proven by the proof techniques discussed so far. A justification of this case distinction comes from the observation that a formula F is valid if and only if there is a complete set $\{c_1,..,c_n\}$ of logical constraints such that each of the formulae $F_i = c_i \Rightarrow F$ are logically valid. This observation, which in the context of sequent calculi can be proven easily by applying the cut rule, leads to the following extended version of theorem 1.

Theorem 3. *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 $\mathcal{C}_i$.*

Conditional substitutions also lead to an extension of theorem 2, as our proof method in [21] synthesizes constraints for orthogonal matrices whenever a connection cannot be shown to be complementary. In this case, all paths through the matrix are closed either by an orthogonal connection or a connection between a constraint and one literal of an orthogonal connection. To formulate this insight, we call a connection (u, v) *complementary under the constraint c* if c and either u or v form a complementary connection.

[4] In an orthogonal matrix the prefixes of orthogonal connections are always unifiable.

Theorem 4. *A formula* $(c^1 \wedge \ldots \wedge c^k) \Rightarrow F$*, where* F *is orthogonal, is valid if all orthogonal connections* (u, v) *are* σ*-complementary or* σ*-complementary under one of the* c^i *for some substitution* σ*.*

The above extensions of the matrix-characterization of logical validity enable us to prove inductive specification theorems with existential quantifiers without having to provide domain lemmata and case-splits beforehand. Instead we extend the notion of unification by rippling and other theory-based reasoning techniques, consider orthogonal connections first, and generate conditional substitutions if necessary. The following example shows the advantages of such a proof method.

Example 4. Consider $F_0 \equiv \exists y \, \neg(x{<}y^2) \wedge x{<}(y{+}1)^2 \;\Rightarrow\; \exists y \, \neg(x{+}1{<}y^2) \wedge x{+}1{<}(y{+}1)^2$ and its matrix-representation in Figure 1. The matrix contains two orthogonal connections $(x{<}^1(y_h{+}1)^2,\; x{+}1{<}^0(Y_c{+}1)^2)$ and $(Y_c^2{<}^1 x{+}1,\; y_h^2{<}^0 x)$. Applying extended matching to the first leads to the substitution $\sigma_1{=}\{Y_c \backslash y_h{+}1\}$, as described in example 2. As the instantiated second connection is not complementary we add $x{+}1{<}(y_h{+}1)^2$ as condition c_1 (i.e. with opposite polarity 0) to the matrix. According to theorem 4 we have thus established the validity of F_0 under c_1, which completes the first subproof (marked by $\boxed{1}$)

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

To complete the proof we now try to establish the validity of F_0 under the negated condition $c_2 \;=\; x{+}1{<}^1(y_h{+}1)^2$, which we also have to add to the matrix. By connecting this condition to $x{+}1{<}^0(Y_c{+}1)^2$ we get $\sigma_2{=}\{Y_c \backslash y_h\}$ through conventional unification. As the instantiated second connection $(y_h^2{<}^1 x{+}1,\; y_h^2{<}^0 x)$ is also complementary in the sense of definitions 1 and 3, we have completed the second subproof $\boxed{2}$ as well. Thus F_0 is valid according to theorem 3.

5 A Matrix-based Inductive Proof Method

Based on the concepts and characterizations introduced in the previous section we will now describe a proof method for inductive theorem proving that combines both matrix-based logical proof search and rippling techniques. The basic methodology is the same as the one we used in [21], but is is more goal-oriented because of its strong emphasis on connections.

In the first step we decompose the step case formula F of an inductive proof by constructing its matrix-representation. In the process δ-quantified variables will become constants and γ-variables will remain to be variables. As γ-variables refer to both (positive) existential and (negative) universal quantifiers, our extended matching procedure becomes applicable not only to existentially quantified variables in the conclusion, but also to universally quantified variables in the induction hypothesis and to other inductive proof problems that contain more deeply nested quantifiers.

In the second step we try to prove that the inductive conclusion follows from the induction hypothesis. Using theorem 4 we check the complementarity of the orthogonal connections in the matrix and generate constraints if necessary. For this purpose we apply our simultaneous matching procedure to the set of orthogonal connection and follow the usual connection method [20] to determine the order in which *open subgoals*, i.e. literals that have not yet been closed by a complementary connection, will be investigated. As in [21], simultaneous match attempts to prove the complementarity of a connection first by applying a decision procedure, then by unification, and finally by rippling / reverse rippling match. The first test has to rely entirely on a given set of constraints and a given substitution σ while the latter two may extend σ. If all three fail we add the inverse of the current open subgoal to the set of constraints in order to *create* a complementary connection. The rationale behind the generation of constraints is the assumption that induction has to be the guiding principle in this part of the proof and that a failing induction hints only at a special case which needs to be proven differently.

After testing the complemementarity of the orthogonal connections we have a conjunction of constraints $c_1 \equiv c_1^1 \wedge ... \wedge c_1^k$, a substitution σ_1, and a matrix-proof for the validity of $c_1 \Rightarrow F$ according to theorem 4. To complete the proof we now have to prove the validity of $\neg c_1 \Rightarrow F$. For this purpose we add the formula $c_1 \vee \neg c_1$ to the matrix in order to get a representation of $(c_1 \vee \neg c_1) \Rightarrow F$. From a technical point of view we have now synthesized a case distinction, added it to the matrix, and increased the multiplicity in order to allow two different substitutions for the two cases.

In the resulting matrix all paths through c_1 are already proven to be complementary and we only have to investigate the paths through $\neg c_1$. As this subproof does not necessarily have to be an inductive one, we will consider all possible connections instead of only the orthogonal ones and will not introduce constraints. Instead we use the standard path-checking algorithm of the connection method and show complementarity by applying decision procedures, unification, and our rippling / reverse rippling procedure. As a starting point we use the newly introduced literal $\neg c_1$ as it will very likely play an important role in the proof.

If the conventional matrix proof for $\neg c_1 \Rightarrow F$ fails, we attempt a new proof based on orthogonal connections, generate further constraints, and continue until we have found a proof. Because of this our proof method may not terminate if the proof does not follow by induction. But in the programming examples that we tested so far the inductive proof requires at most one case split.

Because of the combination of efficient path-checking mechanisms with extended rippling mechanisms, our method is able to handle complex logical formulas in inductive proofs and to solve problems where conventional rippling techniques [9] are not expressive enough to instantiate quantified variables properly. The compact matrix-representation of a formula helps us to focus on the essential structure of a given formula and to find a proof more effectively.

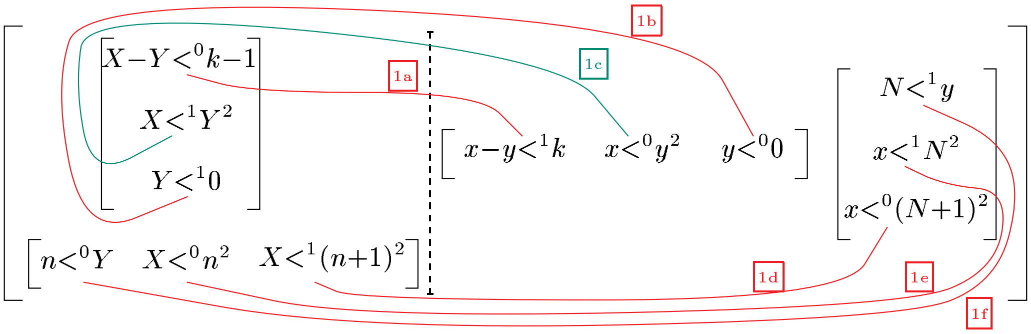


Fig. 2. Matrix of F_1 with orthogonal connections

Example 5. To illustrate the flexibility and strength of our technique, we prove the integer square root specification $\forall x\, \exists y\; y^2 \leq x \wedge x<(y+1)^2$ with a different induction scheme, which will result in a more efficient program. In each iteration we increment y instead of x and do so until $(y+1)^2 > x$. The (destructor) induction has to proceed over an auxiliary variable k and yields the following step case formula.

$$\begin{aligned} F_1 \;\equiv\; & \quad\; \forall x, y.\; x-y<k-1 \wedge y^2 \leq x \wedge 0 \leq y \;\Rightarrow\; \exists n.\; y \leq n \wedge n^2 \leq x \wedge x<(n+1)^2 \\ & \Rightarrow \forall x, y.\; x-y<k \quad\;\;\; \wedge y^2 \leq x \wedge 0 \leq y \;\Rightarrow\; \exists n.\; y \leq n \wedge n^2 \leq x \wedge x<(n+1)^2 \end{aligned}$$

After unfolding $\leq$ we generate the matrix-representation of F_1 and the six orthogonal connections, as depicted in Figure 2. We separate the two orthogonal submatrices by a dashed line and denote γ-variables by capital letters. We select the induction hypothesis as starting point and try to prove the complementarity of the connections $\boxed{\text{1a}}$ ... $\boxed{\text{1f}}$

$\boxed{\text{1a}}$ To prove the complementarity of the connection our extended matching procedure has to find a substitution σ and a rippling sequence from $\sigma(X-Y<k-1)$ to $\sigma(x-y<k)$. Using wave rules derived from monotonicity laws and the definition of $-$, reverse rippling succeeds and generates the substitution $\sigma = \{X \backslash x,\ Y \backslash y{+}1\}$

$\boxed{\text{1b}}$ Instantiating the second connection requires us to prove $y{+}1<0 \Rightarrow_{\mathcal{A}} y<0$, which is shown by an arithmetical decision procedure.

$\boxed{\text{1c}}$ As $x<(y+1)^2 \Rightarrow_{\mathcal{A}} x<y^2$ is not a valid implication, we must add the inverse of the instantiated open subgoal, i.e. $c_1 \equiv x <^0 (y{+}1)^2$ as condition to the matrix.

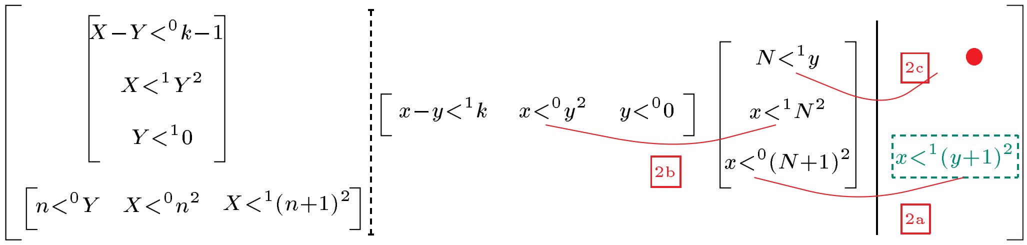
$\boxed{\text{1d}}$ The instantiated fourth connection can be solved by unifying $x<(n+1)^2$ and $x<(N+1)^2$. This extends the substitution to $\sigma = \{X \backslash x,\ Y \backslash y{+}1,\ N \backslash n\}$.

$\boxed{\text{1e}}$ The terms in the instantiated fifth connection are now equal.

$\boxed{\text{1f}}$ The sixth connection requires us to prove $n<y \Rightarrow_{\mathcal{A}} n<y{+}1$, which again is shown by a decision procedure.

This concludes the first subproof. We now have to prove the validity of F under the inverse of c_1, i.e. $c_2 \equiv x <^1 (y{+}1)^2$. We add c_2 to the matrix and select this literal as starting point for the second sub-proof.

There are several $<^0$-literals in the matrix that could be connected to c_2. Among those, all but $x <^1 (N{+}1)^2$ are immediately ruled out because of a constant mismatch. Thus we connect c_2 to $x <^0 (N{+}1)^2$ $\boxed{\text{2a}}$ and get the substitution $\sigma = \{N \backslash y\}$. This leaves $x<^1 y^2$ and $y<^1 y$ as open subgoals. To solve the first, we connect to $x<^0 y^2$ $\boxed{\text{2b}}$, which is already a complementary connection. The second open subgoal is solved by applying a decision procedure, which proves $y<y$ to be false. We indicate this unary connection by an arc to a bullet $\boxed{\text{2c}}$.



This concludes the second subproof as well. F_1 is valid under the set of conditional substitutions $\{[x<(y+1)^2,\ \{X\backslash x,\ Y\backslash y+1,\ N\backslash n\}],\ [\neg(x<(y+1)^2),\ \{N\backslash y\}]\}$.

After a matrix-proof has been generated, it has to be converted into a sequent proof. This enables us to extract a verifiably correct algorithm from the proof if the proven formula represents a program specification. Depending on the chosen induction and the matrix-proof we may get different algorithms for the same specification. The matrix proof for the integer square root specification in example 4, for instance, will result in an algorithm that iterates the input x and results in a linear algorithm, while the algorithm extracted from the proof in example 5 will iterate the output y, which is much more efficient.

Converting inductive matrix proofs into sequent proofs is by no means trivial. We intend to combine the algorithms for transforming logical matrix proof into sequent proofs developed in [23, 24] with methods from [18, 7], which transform rippling sequences into sequent proofs.

6 Conclusion

We have presented a proof method for inductive theorem proving that integrates rippling-based rewriting and matrix-based logical proof search. The proof search is guided by a connection-driven path-checking algorithm, which gives preferences to connections between the induction hypothesis and the induction conclusion. The complementarity of the connected literals is checked by arithmetical decision procedures, a unification algorithm, and an extended matching procedure based on a rippling / reverse rippling heuristic. Our method combines the strengths of inductive reasoning and logical theorem proving and gives us the flexibility to deal with complex logical formulas in inductive proofs.

There have been other approaches to automate the instantiation of quantifiers variables in inductive proofs. Biundo [8] replaces existentially quantified variables by Skolem functions, which represent programs to be synthesized, and proceeds by *clause-set translations* like rewriting and case splitting. The work of Kraan et al. [15] refines this idea by using rippling and middle-out reasoning [13] to control the search space. However, as both approaches are not integrated into a logical proof environment, they cannot guarantee that the synthesized program is correct and have to verify it afterwards.

Hutter [14] introduces a technique to synthesize induction orderings that is related to our reverse rippling heuristic [21]. To find an appropriate induction scheme, it looks at the function symbols surrounding existentially quantified variable and consults the applicable wave rules to guess an instantiation of this variable. This localized approach, however, causes difficulties when function symbols are defined by two-step or even more complex recursions.

All these approaches face difficulties when dealing with complex logical formulas as they do not have a clean methodology for integrating goal-directed rewriting and logical proof search. In contrast to that our approach combines both techniques in an elegant way. Matrix methods provide a uniform foundation for instantiating existentially and universally quantified variables, as they abstract from the notational peculiarities and focus on their type (i.e. γ or δ). Rippling-based rewriting is used on the level of the connections and thus extends the usual unification. This allows us to deal with complex induction schemes, like the one used in example 5, where other approaches are not expressive enough anymore. As matrix-methods can be viewed as compact representation of sequent provers, our matrix-based inductive proof method also improves our previous approach [21], since it can handle inductive proofs more efficiently.

Although we have independent implementations of all the individual components used in our method, i.e. matrix-based proof search, the rippling / reverse rippling heuristic, and decision procedures for elementary arithmetic, we do not have an integrated implementation yet. We aim at an implementation of our combined proof method that retains the uniformity of the existing matrix-based theorem prover [20] and can deal with non-inductive proofs equally well. We also intend to integrate transformation algorithms for converting matrix proofs [23, 24] and rippling sequences [18, 7] into sequent proofs and use them to guide the development of proofs in interactive proof assistants and for the synthesis of conditional recursive programs.

References

1. A. Armando, A. Smaill, and I. Green. Automatic synthesis of recursive programs: The proof-planning paradigm. In *12^{th} IEEE International Automated Software Engineering Conference*, pp. 2–9. IEEE Computer Society, 1997.
2. D. Basin and T. Walsh. A calculus for and termination of rippling. *Journal of Automated Reasoning*, 16(2):147–180, 1996.
3. J. Bates and R. Constable. Proofs as programs. *ACM Transactions on Programming Languages and Systems*, 7(1):113–136, 1985.
4. W. Bibel. On matrices with connections. *Journal of the Association for Computing Machinery*, 28:633–645, 1981.
5. W. Bibel. A deductive solution for plan generation. *New Generation Computing*, 4:115–132, 1986.
6. W. Bibel. *Automated Theorem Proving*. Vieweg, 1987.
7. W. Bibel, D. Korn, C. Kreitz, F. Kurucz, J. Otten, S. Schmitt, and G. Stolpmann. A multi-level approach to program synthesis. In N.E. Fuchs, ed., *7^{th} International Workshop on Logic Program Synthesis and Transformation*, LNAI 1463, pp. 1–25, Springer, 1998.
8. S. Biundo. Automated synthesis of recursive algorithms as a theorem proving tool. In *8^{th} European Conference on Artificial Intelligence*, Elsevier, 1988.

9. A. Bundy, F. van Harmelen, A. Smaill *et al.*. Extensions to the rippling–out tactic for guiding inductive proofs. In M. Stickel, ed. *10^{th} Conference on Automated Deduction*, LNCS 449, pp. 132–146, Springer 1990.
10. A. Bundy, A. Stevens, F. van Harmelen *et al.*. Rippling: A heuristic for guiding inductive proofs. *Artificial Intelligence*, 62(2):185–253, 1993.
11. R. Constable, S. Allen, M. Bromley, et. al. *Implementing Mathematics with the* NuPRL *proof development system.* Prentice Hall, 1986.
12. G. Dowek et. al. *The Coq proof assistant user's guide.* Institut National de Recherche en Informatique et en Automatique, Report RR 134, 1991.
13. J. Hesketh. *Using Middle-Out Reasoning to Guide Inductive Theorem Proving.* PhD thesis, Dept. of Artificial Intelligence, University of Edinburgh, 1991.
14. D. Hutter. *Synthesis of Induction Orderings for Existence Proofs.* In A. Bundy, ed., *12^{th} International Conference on Automated Deduction*, LNAI 814, Springer, 1994.
15. I. Kraan, D. Basin, and A. Bundy. Logic program synthesis via proof planning. In *4^{th} International Workshop on Logic Program Synthesis and Transformation*, pp. 1–14. Springer, 1993.
16. C. Kreitz, H. Mantel, J. Otten, and S. Schmitt. Connection-Based Proof Construction in Linear Logic. In W. McCune, ed., *14^{th} Conference on Automated Deduction*, LNAI 1249, pp. 207–221. Springer, 1997.
17. C. Kreitz, J. Otten, and S. Schmitt. Guiding Program Development Systems by a Connection Based Proof Strategy. In M. Proietti, ed., *Fifth International Workshop on Logic Program Synthesis and Transformation*, LNCS 1048, pp. 137–151. Springer, 1996.
18. F. Kurucz. Realisierung verschiedender Induktionsstrategien basierend auf dem Rippling-Kalkül. Diplomarbeit, Technische Universität Darmstadt, 1997.
19. J. Otten and C. Kreitz. 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, pp. 122–137. Springer, 1995.
20. J. Otten and C. Kreitz. A Uniform Proof Procedure for Classical and Non-classical Logics. In G. Görz & S. Hölldobler, eds., *KI-96: Advances in Artificial Intelligence*, LNAI 1137, pp. 307–319. Springer, 1996.
21. B. Pientka and C. Kreitz. Instantiation of existentially quantified variables in inductive specification proofs. In *4^{th} International Conference on Artificial Intelligence and Symbolic Computation*, LNAI 1476. Springer, 1998.
22. A. Smaill and I .Green. Automating the synthesis of functional programs. Research paper 777, Dept. of Artificial Intelligence, University of Edinburgh, 1995.
23. S. Schmitt and C. Kreitz. On transforming intuitionistic matrix proofs into standard-sequent proofs. In P. Baumgartner, R. Hähnle & J. Posegga, eds., *4^{th} Workshop on Theorem Proving with Analytic Tableaux and Related Methods*, LNAI 918, pp. 106–121. Springer, 1995.
24. S. Schmitt and C. Kreitz. Converting non-classical matrix proofs into sequent-style systems. In M. McRobbie & J. Slaney, eds, *13^{th} Conference on Automated Deduction*, LNAI 1104, pp. 418–432. Springer, 1996.
25. T. Tammet. A resolution theorem prover for intuitionistic logic. In M. McRobbie & J. Slaney, eds, *13^{th} Conference on Automated Deduction*, LNAI 1104, pp. 2–16, Springer, 1996.
26. L. Wallen. *Automated deduction in nonclassical logic.* MIT Press, 1990.