

Matrix-based Inductive Theorem Proving

Christoph Kreitz¹ & Brigitte Pientka²

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 integrates rippling-based rewriting into matrix-based logical proof search. The selection of appropriate connections in a 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 combined approach by discussing several inductive proofs for the integer square root problem.

1 Introduction

Formal methods have 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 [12] or Coq [13] 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, 22, 28] 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 the logical proof search part 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 $\forall x \exists y y^2 \leq x \wedge x < (y+1)^2$, 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

$$\exists y y^2 \leq x \wedge x < (y+1)^2 \vdash \exists y y^2 \leq x+1 \wedge x+1 < (y+1)^2$$

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 aim at combining 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 [24] we have shown that a combination of a rippling-based extended matching procedure and simple sequent proof tactic can already be used to solve synthesis problems like the above automatically. The logical proof search technique in this approach, however, was not designed to be complete, as it focuses only on a top-down decomposition of formulae, and should therefore be replaced by a complete search procedure for constructive first-order logic.

Among the well-known proof procedures for intuitionistic first-order logic [22, 28] matrix-based proof techniques such as the *connection method* [4, 20] can be understood as very compact representations of sequent proof search. This makes them not only much more efficient but also allows to convert their results back into sequent proofs [27], which means that (constructive) matrix methods can be used to guide proof and program development in interactive proof assistants [7, 19]. These advantages suggest an integration of our extended rippling technique into matrix-based proof procedures.

In this paper we will present concepts for combining rippling techniques and matrix-based constructive theorem proving. In Section 2 we will review 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*. [*Sinks*] 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 $+$.

$$\underline{[U+1]}^\uparrow + V \xrightarrow{R} \underline{[(U+V)+1]}^\uparrow$$

In this rule, $\underline{[\dots+1]}$ 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* or its *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 within the framework of rippling, mainly two approaches have been suggested. Bundy et al. [10] proposes 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, 17, 25] use meta-annotations, which requires higher-order unification and may lead to non-terminating rippling sequences.

The approach presented in [24] addresses these drawbacks by combining rippling with first-order proof techniques and extending the matching procedure for instantiating existentially quantified variables. It proceeds in three steps.

First, the step case formula of an inductive proof is decomposed into several subgoals and meta-variables are introduced in place of existentially quantified variables. A straightforward sequent proof tactic for constructive logic is used for this purpose, which allows to integrate the results into the NuPRL proof development system [12] and to extract programs from the inductive proof.

Next a *simultaneous match* is used to compute substitutions for the meta-variables incrementally. Given a substitution σ and a term C from the induction conclusion there are three possibilities. One 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* illustrated below.

$$C \xrightarrow{R} C_0 \xrightarrow{R} \dots \xrightarrow{R} C_i \xrightarrow{R} \dots \xrightarrow{R} C_n \xrightarrow{R} H$$

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 $C_i \xrightarrow{R} \dots \xrightarrow{R} C_n \xrightarrow{R} H$ must be computed by reasoning backwards from the term H towards C_i . In this *reverse rippling* process the meta-variables in C are instantiated by a substitution σ' . If the generated rippling sequence is empty, then either H implies $\sigma(C)$ or $\sigma(C)$ and H can be unified.

The search for a rippling sequence is based on the *rippling-distance* strategy introduced in [21], 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 detailed analysis of this strategy we refer to [21, 7].

After extended matching has generated a rippling sequence and a substitution σ for one subgoal it is checked whether all remaining subgoals are true under σ as well. To restrict the search space, all subgoals are required to be provable by standard arithmetic, rippling, and matching. If a subgoal cannot be proven by these techniques, it is selected as constraint for σ . The approach then proves all subgoals under the negated constraint and continues until it has 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. This simple heuristic already turned out to be sufficient for many induction problems.

Example 2. Consider again the integer square-root problem from example 1. To prove $\exists y y^2 \leq x \wedge x < (y+1)^2 \vdash \exists y y^2 \leq x+1 \wedge x+1 < (y+1)^2$ the induction hypothesis and the conclusion are decomposed, which also unfolds the abbreviation $y^2 \leq x$ to $\neg(x < y^2)$. The existentially quantified variable in the conclusion is replaced by a meta-variable Y_c . This results in two subgoals:

$$\underline{\neg(x < y_h^2)}, x < (y_h+1)^2 \vdash \neg(\underline{x+1} < Y_c^2) \quad (1)$$

$$\underline{\neg(x < y_h^2)}, \underline{x < (y_h+1)^2} \vdash \underline{x+1} < (Y_c+1)^2 \quad (2)$$

Next the conclusion is annotated 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:

$$\underline{U+W} < \underline{V+W} \xrightarrow{R} U < V \quad (3)$$

$$\underline{U+1} + V \xrightarrow{R} \underline{U+V+1} \quad (4)$$

$$(\underline{A+1})^2 \xrightarrow{R} \underline{A^2 + 2A + 1} \quad (5)$$

Starting with subgoal (2) the method then tries to match $C = \underline{x+1} < (Y_c+1)^2$ 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 unchanged and reverse rippling reasons backwards from H to C . To determine which formula may precede H , the right hand sides of the wave rules are inspected. Rule (3) suggests that the predecessor of H has the form $\underline{x+W} < \underline{(y_h+1)^2+W}$. 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 rule (4) the wave front $\lfloor \dots + 1 \rfloor$ is moved to a position where it surrounds y_h , which leads to $C_0 = \lfloor x + 2(y_h + 1) + 1 \rfloor < (\lfloor y_h + 1 \rfloor + 1)^2$. As rippling has terminated successfully, Y_c can be instantiated by the sink value, which results in $\sigma_1 := \{Y_c \setminus y_h + 1\}$. To prove that C_0 implies $\sigma_1(C)$ a decision procedure for induction-free arithmetic [11] is used.

As subgoal (1) is not valid under σ_1 , its instantiated conclusion $\neg(x + 1 < (y_h + 1)^2)$ is added as constraint. Next both subgoals must be proved for the case $x + 1 < (y_h + 1)^2$. Normal matching $x + 1 < (Y_c + 1)^2$ from subgoal (2) and $x + 1 < (y_h + 1)^2$ yields the substitution $\sigma_2 := \{Y_c \setminus y_h\}$, 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 $\{\{\neg(x + 1 < (y_h + 1)^2), \{Y_c \setminus y_h + 1\}\}, \{x + 1 < (y_h + 1)^2, \{Y_c \setminus y_h\}\}\}$.

In a final step the conditional substitutions and rippling sequences that prove the induction step are translated into a sequent proof. Following the *proofs-as-programs* principle [3] one can then extract a proof expression that describes an algorithm for the given specification.

The extended rippling technique, which is described in detail in [24], has been implemented as NUPRL-tactic and applied to several program specifications such as quotient remainder, append, last, \log_2 , and unification. It has also been used for the instantiation of universally quantified variables in the hypothesis list, which is necessary for dealing with non-standard induction schemes that enable a synthesis of more efficient while loops. However, as the sequent proof tactic underlying this approach is neither complete nor very efficient, because it is driven only by a top-down decomposition of formulae. In the rest of this paper we will show how to integrate extended rippling into a complete, and more efficient proof procedure for constructive first-order logic.

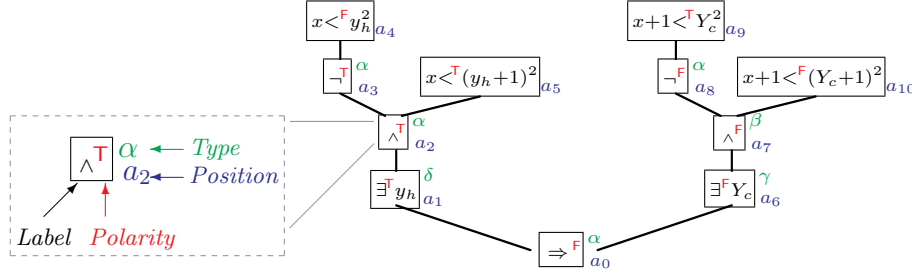
3 Matrix-based Constructive Theorem Proving

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 been extended to a variety of non-classical logics such as intuitionistic logic [22], modal logics [20], and fragments of linear logic [18]. In this section we will briefly summarize its essential concepts.

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. As in the tableaux calculus each position is associated with a *polarity* (F or T) and a *principal type* (α , β , γ , or δ). 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 \leq), together with polarities and principal types is presented below. As usual we denote γ -variables by capital letters.¹



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.

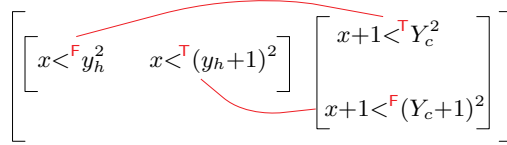


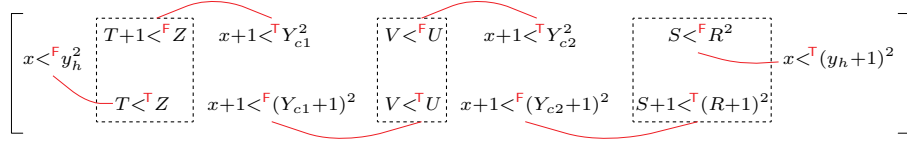
Fig. 1. Matrix of the formula F_0

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 *admissible* substitution σ to γ -variables, where admissibility encodes the eigenvariable condition on the substituted terms. With these definitions the following *matrix-characterization of logical validity* has been shown.

Theorem 1. *A formula F is valid iff there is a multiplicity μ , an admissible substitution σ , and a set of complementary connections such that every path through F^μ contains a connection from this set.*

Example 3. The formula F_0 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.

¹ Note that extended rippling considers γ -variables to be meta-variables.



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 $\sigma = \{Z \setminus y_h^2, T \setminus x, Y_{c1} \setminus y_h, V \setminus x+1, U \setminus (y_h+1)^2, Y_{c2} \setminus y_h+1, S \setminus x, R \setminus y_h+1\}$. Thus the formula F_0 has become valid after being extended by the three arithmetical lemmata.

So far we have only characterized validity in classical logic. In *constructive* logic one not only has to check whether the terms of connected atoms can be unified by a *quantifier substitution* σ_Q but also that both atoms be reached by applying the same sequence of sequent rules, since only then they can form a leaf in a sequent proof. Technically, this can be done by computing an additional *intuitionistic substitution* σ_J , which unifies the prefixes of the connected atoms. A *prefix* is a string of positions between the root and the atom that are labelled with \forall , \Rightarrow , \neg , or atomic formulae, where positions of polarity F are considered as constants and the others as variables. Thus, a connection is *constructively complementary* if there is an admissible *combined substitution* $\sigma := (\sigma_Q, \sigma_J)$ that unifies both the terms and the prefixes of the connected atoms. Admissibility now involves a few additional conditions (see [29]). With these modifications theorem 1 holds accordingly for constructive logic.

The *connection method* for constructive first-order logic describes an efficient, connection-driven technique for checking 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 the algorithms for checking paths and unifying prefixes as well as their justifications can be found in [23, 20].

4 Extending the Matrix-Characterization

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

A first extension comes from the observation that a pair $\{A^T, \bar{A}^F\}$ of connected atoms 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 left side of the leaf sequent, i.e. A , *implies* the right side \bar{A} under σ , where the implication can be proven logically, by decision procedures, or by rewriting \bar{A} into A . The corresponding extension of the matrix-characterization requires us to consider *directed connections* (u^T, v^F) and a notion of *implication with respect to a given theory \mathcal{T}* , written as $\Rightarrow_{\mathcal{T}}$. In principle, $\Rightarrow_{\mathcal{T}}$ denotes any implication that can be proven within the theory \mathcal{T} . Usually, however, we restrict ourselves to implications that can be proven by standard decision procedures or

by rippling sequences generated by extended matching. In particular, we consider *arithmetical implication* $A \Rightarrow_{\mathcal{A}} \bar{A}$ to denote that either $A \Rightarrow \bar{A}$ is provable by a decision procedure for some sub-theory of arithmetic² or that there is a rippling sequence $\bar{A} \xrightarrow{R} \dots \xrightarrow{R} A$ that rewrites \bar{A} into A using arithmetical wave rules. With these notions the concept of complementarity is extended as follows:

Definition 1. A directed connection (u^{\top}, v^{F}) 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 $A = \text{label}(u)$ and $\bar{A} = \text{label}(v)$.

Within a theory \mathcal{T} equality and implication are not the only means to close a branch in a sequent proof. Sequents like $\cdot \vdash \bar{A}$ and $A \vdash \cdot$ are provable if \bar{A} can be proven in \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^{\top} and v^{F} be unary connections, $A = \text{label}(u)$, and $\bar{A} = \text{label}(v)$.
 u^{\top} is *σ -complementary with respect to a theory \mathcal{T}* iff $\sigma(A) \Rightarrow_{\mathcal{T}} \text{False}$.
 v^{F} is *σ -complementary with respect to \mathcal{T}* iff $\text{True} \Rightarrow_{\mathcal{T}} \sigma(\bar{A})$.

In a similar way the concept of complementary with respect to a theory \mathcal{T} , which resembles Bibel's theory-connections [6], could also be extended to n-ary connections w.r.t some theory \mathcal{T} . It enables us to formulate an extended matrix-characterization of logical validity exactly like Theorem 1 and to use the same general proof technique as for constructive first-order logic. The only modification is the test for complementarity: we can now use decision procedures to prove a negative literal A to be false or a positive literal \bar{A} to be true in \mathcal{T} , unify two connected literals A and \bar{A} via standard unification, prove the implication $A \Rightarrow \bar{A}$ with a decision procedure, or find a substitution σ and a rippling sequence $\sigma(\bar{A}) \xrightarrow{R} \dots \xrightarrow{R} \sigma(A)$ with our rippling / reverse rippling heuristic. Decision procedures can even be used to close the gap between forward and reverse rippling (c.f. example 2), which makes our method more flexible and enables it to find more proofs.

A second modification comes from the insight that there is a strong relation between individual subformulae of the induction hypothesis H and the conclusion C . As C is usually identical to $H[x \setminus \rho(x)]$, where x is the induction variable and ρ an inductive constructor like $s = \lambda x.x+1$ or a destructor like the $p = \lambda x.x-1$, a rippling proof usually links a subformula of H to the corresponding subformula of C , which means that proof-relevant connections run between corresponding subformulae of H and C . By checking such *orthogonal* connections first, we can drastically reduce the search space of an inductive matrix-proof.

Definition 3. A formula F is *orthogonal* with respect to a variable x if $F \equiv H \Rightarrow C$ and $C = H[x \setminus \rho(x)]$ for some substitution ρ . A connection (u^{\top}, v^{F}) (or (v^{\top}, u^{F})) 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 .

² A common decidable sub-theory of arithmetic is elementary arithmetic as defined in [11], which includes the common axioms of $+$, $-$, $*$, $/$, $=$, $<$ but no induction.

Orthogonal connections are easy to identify, as they connect literals with the same relative positions in the two major subtrees of F . In an inductive proof it is sufficient to consider only orthogonal connections.

Theorem 2. *An orthogonal formula F is constructively 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 formula $F \equiv H \Rightarrow C$ contains an orthogonal connection.

If H is atomic the statement is trivially true. If H has the form $H_1 \text{ op } H_2$ where op is of type α in F then $C = C_1 \text{ op } C_2$ where 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

In an orthogonal formula it is not necessary to check the unifiability of the prefixes of orthogonal connections, as a classical proof implies constructive validity.

Lemma 1. *In an orthogonal formula F the prefixes of all orthogonal connections are simultaneously unifiable.*

Proof. Let (u, v) be an orthogonal connection in F and $a_0 a_1 \dots a_n$ be the prefix of u . Then the prefix of v is $a_0 b_1 \dots b_n$ where b_i is a variable iff a_i is a constant. Substituting the variables by the corresponding constant unifies the two prefixes. Since this construction is the same for all orthogonal connections, it leads to a common unifier.

Checking the complementarity of orthogonal connections, which resembles Bibel's linear connection method [5] although used for a different purpose, is sufficient but not necessary for validity. If a proof attempt with orthogonal connections fails, the formula may be valid for reasons that have nothing to do with induction and a conventional (constructive!) matrix proof has to be tried.

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 is the observation that a formula F is valid if and only if there is a complete set $\{c_1, \dots, c_n\}$ of logical constraints (usually decidable formulae) such that each $F_i = c_i \Rightarrow F$ is 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 $\{c_1, \dots, c_n\}$ of constraints such that $C = c_1 \vee \dots \vee c_n$ is valid and for all i there is a multiplicity μ_i , a substitution σ_i , and a set C_i of σ_i -complementary connections such that every path through $c_i \Rightarrow F^{\mu_i}$ contains a connection from C_i .*

Conditional substitutions also lead to an extension of theorem 2, as our proof method in [24] synthesizes constraints for orthogonal formulas whenever a connection cannot be shown to be complementary. In this case, all paths through the formula are closed either by an orthogonal connection or a connection between a

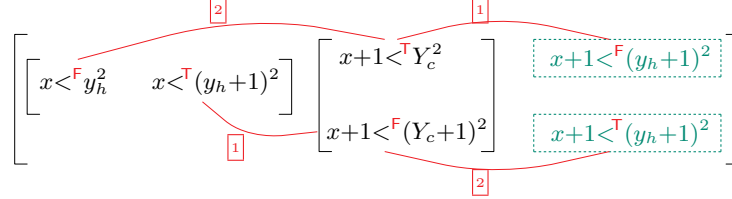
constraint and one literal of an orthogonal connection. To formulate this insight, we call a connection (u, v) *complementary under the constraint c* if c and either u or v form a complementary connection.

Theorem 4. *A formula $(c^1 \wedge \dots \wedge c^k) \Rightarrow F$, 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 .*

Again, we do not have to check the unifiability of prefixes to prove σ -complementarity under c^i , since the prefixes always unify. Thus a classical proof is sufficient.

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 $(x <^T (y_h+1)^2, x+1 <^F (Y_c+1)^2)$ and $(Y_c^2 <^T x+1, y_h^2 <^F x)$. Applying extended matching to the first leads to the substitution $\sigma_1 = \{Y_c \setminus y_h+1\}$, as described in example 2. Because the instantiated second connection is not complementary, we add $x+1 < (y_h+1)^2$ as condition c_1 (i.e. with opposite polarity F) 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 1)



To complete the proof we now try to establish the validity of F_0 under the negated condition $c_2 = x+1 <^T (y_h+1)^2$, which we also add to the matrix. By connecting it to $x+1 <^F (Y_c+1)^2$ we get $\sigma_2 = \{Y_c \setminus y_h\}$ through conventional unification. As the instantiated second connection $(y_h^2 <^T x+1, y_h^2 <^F x)$ is complementary in the sense of definition 1, we have completed the second subproof 2. F_0 is valid because of theorem 3.

5 A Matrix-based Inductive Proof Method

Based on the concepts and characterizations introduced in the previous section we will now describe an inductive proof method that combines both matrix-based logical proof search and rippling techniques and is summarized in Figure 2. It uses the same basic steps as [24] but in a more refined and goal-directed way due to the use of a matrix-representation and an emphasis on connections.

In the first step *we decompose the step case formula F* by constructing its matrix-representation. This means that δ -quantified variables will become constants and γ -variables will become meta-variables. Since all existential quantifier

1. *Logical Decomposition* of F by constructing its orthogonal matrix-representation.
2. Simultaneously *check the complementarity of orthogonal connections* and generate a substitution σ for all γ -variables by (1) unifying the connected terms, (2) applying a decision procedure, or (3) applying the rippling / reverse rippling heuristic. If all orthogonal connections are complementary the formula is valid.
3. If the complementarity test fails, *synthesize a condition c* for the validity of F .
4. *Prove the validity of $\neg c \Rightarrow F$* with the standard connection method.
If the proof fails generate more constraints by continuing with step 2.

Fig. 2. Matrix-based proof method for an inductive step case formula F

in the conclusion and universal quantifiers in the induction hypothesis are represented by γ -variables, our proof method can be applied to inductive proof problems with arbitrarily nested quantifiers.

In the second step we try to *prove the complementarity of all orthogonal connections* and generate a substitution σ for all γ -variables in the process. As in the usual connection method [20] we investigate the connections in an order given by *active subgoals*, i.e. a set of β -related atoms that must be covered by complementary connections, and construct σ incrementally. Given a partial substitution σ' and a connection (u^T, v^F) with $A=label(u)$ and $\bar{A}=label(v)$ we (1) try to unify $\sigma'(A)$ and $\sigma'(\bar{A})$, (2) apply a decision procedure to prove $\sigma'(A) \Rightarrow_T \sigma'(\bar{A})$, or (3) apply the rippling / reverse rippling heuristic to generate a substitution σ'' and a rippling sequence $\sigma''(\sigma'(\bar{A})) \xrightarrow{R} \dots \xrightarrow{R} \sigma''(\sigma'(A))$. The complementarity of unary connections must be provable with a decision procedure. If all orthogonal connections are complementary, the formula F is valid according to Theorem 2 and the proof is complete.

Otherwise, we *synthesize a condition c* for the validity of F . For each connection that fails the complementarity test, we add the negation c^i of the current active subgoal to a set of constraints in order to *create* a complementary connection that covers this subgoal. As a result we get a condition $c \equiv c^1 \wedge \dots \wedge c^k$, a substitution σ_1 , and a matrix-proof for the validity of $c \Rightarrow F$ according to Theorem 4. Note that c is usually decidable as it is composed of atomic literals.

Using Theorem 3 we now have to *prove the validity of $\neg c \Rightarrow F$* to complete the proof. We extend the matrix into a representation of $(c \vee \neg c) \Rightarrow F$ while increasing the multiplicity of F . This allows us to generate different substitutions for each condition and to reuse the matrix-proof for $c \Rightarrow F$. Only paths through $\neg c$ still need to be tested for complementarity. Since the condition $\neg c$ very likely characterizes a special case not to be proved by induction, we start with investigating $\neg c$ and use the standard connection method [20] with unification enhanced by decision procedures. No constraints will be introduced in this step.

If the conventional matrix proof for $\neg c \Rightarrow F$ fails, we attempt another inductive proof based on orthogonal connections, extended rippling, and possibly additional constraints by continuing with the second step again. In inductive proofs of program specifications each constraint will correspond to a case split in the program. The examples we tested so far required at most one such case split.

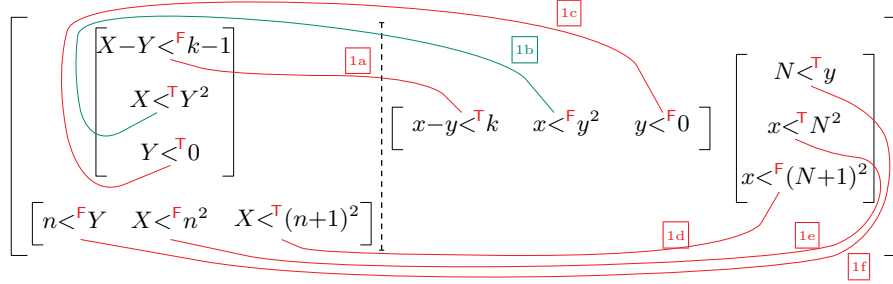


Fig. 3. Matrix of F_1 with orthogonal connections

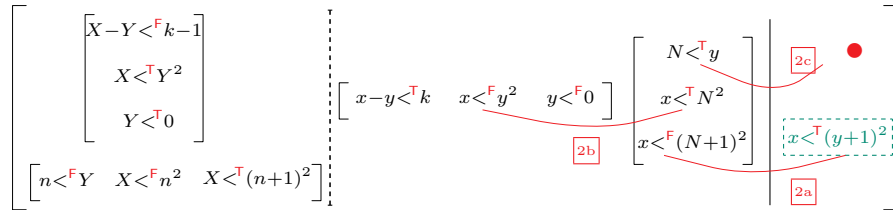
Example 5. To illustrate our technique, we prove the integer square root specification $\forall x \exists y \ y^2 \leq x \wedge x < (y+1)^2$ with a different induction scheme, which will result in a more efficient program. In each iteration we increment y instead of x and do so until $(y+1)^2 > x$. The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

$$F_1 \equiv \forall x, y. x-y < k-1 \wedge y^2 \leq x \wedge 0 \leq y \Rightarrow \exists n. y \leq n \wedge n^2 \leq x \wedge x < (n+1)^2 \\ \Rightarrow \forall x, y. x-y < k \wedge y^2 \leq x \wedge 0 \leq y \Rightarrow \exists n. y \leq n \wedge n^2 \leq x \wedge x < (n+1)^2$$

After unfolding \leq we generate the matrix-representation of F_1 and the six orthogonal connections, as depicted in Figure 3. 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 $\boxed{1a} \dots \boxed{1f}$ using a decision procedure **Arith** for arithmetic, unification, and the rippling / reverse rippling heuristic.

- $\boxed{1a}$ 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 $\sigma = \{X \setminus x, Y \setminus y+1\}$
- $\boxed{1b}$ As $x < (y+1)^2 \Rightarrow_{\mathcal{A}} x < y^2$ is not valid in arithmetic, we add the inverse of the instantiated active subgoal, i.e. $c_1 \equiv x <^F (y+1)^2$ as condition to the matrix.
- $\boxed{1c}$ The instantiated third connection requires us to prove $y+1 < 0 \Rightarrow_{\mathcal{A}} y < 0$, which is shown by **Arith**.
- $\boxed{1d}$ The instantiated fourth connection can be solved by unifying $x < (n+1)^2$ and $x < (N+1)^2$. This extends the substitution to $\sigma = \{X \setminus x, Y \setminus y+1, N \setminus n\}$.
- $\boxed{1e}$ The terms in the instantiated fifth connection are now equal.
- $\boxed{1f}$ The sixth connection requires us to prove $n < y \Rightarrow_{\mathcal{A}} n < y+1$, which again is shown by **Arith**.

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



There are several $<^F$ -literals in the matrix that could be connected to c_2 . Among those, all but $x <^T (N+1)^2$ are immediately ruled out because of a constant mismatch. Thus we connect c_2 to $x <^F (N+1)^2$ 2a and get the substitution $\sigma = \{N \setminus y\}$. This leaves $x <^T y^2$ and $y <^T y$ as active subgoals. To solve the first, we connect to $x <^F y^2$ 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 2c.

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

After a matrix-proof has been generated, it has to be converted into a sequent proof, which enables us to extract a verifiably correct algorithm if the proven formula represents a program specification. For this purpose one can combine the algorithms for transforming logical matrix proof into sequent proofs developed in [26, 27] with the methods for transforming rippling sequences into sequent proofs described in [21, 7].

Depending on the chosen induction and the matrix proof we will get different algorithms for the same specification. The matrix proof for the integer square root specification in Example 4, for instance, will result in a linear algorithm that iterates the input x , while the algorithm extracted from the proof in Example 5 will iterate the output y , which is much more efficient.

6 Conclusion

We have presented a method for integrating rippling-based rewriting into matrix-based constructive theorem proving as a means for generating inductive specification proofs. The proof search is guided by a connection-driven path-checking algorithm that gives preference to connections between the induction hypothesis and the induction conclusion. The complementarity of the connected atoms is checked by unification, decision procedures, and an extended matching procedure based on a rippling / reverse rippling heuristic. Constrained substitutions are generated if there is no unique substitution for the quantified variables. The resulting proof can be converted into a verifiably correct algorithm for a given inductive program specification.

There have been other approaches to automate the instantiation of quantified 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. [17] refines this idea by using rippling and middle-out reasoning [14] 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 [15] introduces a technique to synthesize induction orderings that is related to our reverse rippling heuristic [24]. 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.

The INKA [16] and SPASS [30] provers integrate induction techniques into resolution-based theorem proving. However, they cannot synthesize case distinctions and resolution is generally non-constructive, which makes it difficult to use these tools for handling inductive program specifications.

The advantage of our approach is that it combines the strengths of inductive reasoning and constructive logical theorem proving 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). Decision procedures and rippling-based rewriting are used on the level of the connections and thus extend the usual unification, while the systematic generation of case distinctions adds flexibility to the proof search procedure. The combination of these features allows us to deal with complex induction schemes, like the one used in example 5, which cannot be handled by either first-order and inductive proof methods individually.

Our approach is only a first step towards a full integration of first-order and inductive theorem proving. But most of the individual components used in our method, i.e. matrix-based proof search for constructive logic [20], the rippling / reverse rippling heuristic [24], and decision procedures for arithmetic [11] have already been implemented independently. We intend to combine these components as described in Section 5 but also aim at preserving the uniformity of the existing matrix-based theorem prover. We also intend to integrate transformation algorithms for converting matrix proofs and rippling sequences into sequent proofs [26, 27, 21, 7] 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. Smail & I. Green. Automatic synthesis of recursive programs: The proof-planning paradigm. *12th IEEE International Automated Software Engineering Conference*, pp. 2–9, 1997.
2. D. Basin & T. Walsh. A calculus for and termination of rippling. *Journal of Automated Reasoning*, 16(2):147–180, 1996.
3. J. Bates & 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 & G. Stolpmann. A multi-level approach to program synthesis. *7th International Workshop on Logic Program Synthesis and Transformation*, LNAI 1463, pp. 1–25, 1998.
8. S. Biundo. Automated synthesis of recursive algorithms as a theorem proving tool. *8th European Conference on Artificial Intelligence*, 1988.

9. A. Bundy, F. van Harmelen, A. Smaill *et al.*. Extensions to the rippling-out tactic for guiding inductive proofs. *10th Conference on Automated Deduction*, LNCS 449, pp. 132–146, 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. T. Chan. A decision procedure for checking PL/CV arithmetic inferences. In *Introduction to the PL/CV2 Programming Logic*, LNCS 135, pp. 227–264, 1982.
12. R. Constable, S. Allen, M. Bromley, *et al.*. *Implementing Mathematics with the NuPRL proof development system*. Prentice Hall, 1986.
13. G. Dowek *et al.*. *The Coq proof assistant user's guide*. Institut National de Recherche en Informatique et en Automatique, Report RR 134, 1991.
14. J. Hesketh. *Using Middle-Out Reasoning to Guide Inductive Theorem Proving*. PhD thesis, Dept. of Artificial Intelligence, University of Edinburgh, 1991.
15. D. Hutter. *Synthesis of Induction Orderings for Existence Proofs*. *12th International Conference on Automated Deduction*, LNAI 814, 1994.
16. D. Hutter & C. Sengler. INKA, the next generation. *13th Conference on Automated Deduction*, LNAI 1104, pp. 288–292, 1996.
17. I. Kraan, D. Basin & A. Bundy. Logic program synthesis via proof planning. *4th International Workshop on Logic Program Synthesis and Transformation*, pp. 1–14, 1993.
18. C. Kreitz, H. Mantel, J. Otten & S. Schmitt. Connection-Based Proof Construction in Linear Logic. *14th Conference on Automated Deduction*, LNAI 1249, pp. 207–221, 1997.
19. C. Kreitz, J. Otten & S. Schmitt. Guiding Program Development Systems by a Connection Based Proof Strategy. *Fifth International Workshop on Logic Program Synthesis and Transformation*, LNCS 1048, pp. 137–151, 1996.
20. C. Kreitz & J. Otten. Connection-based theorem proving in classical and non-classical logics. *Journal of Universal Computer Science*, 5(3):88–112, 1999.
21. F. Kurucz. Realisierung verschiedener Induktionsstrategien basierend auf dem Rippling-Kalkül. Diplomarbeit, Technische Universität Darmstadt, 1997.
22. J. Otten & C. Kreitz. A connection based proof method for intuitionistic logic. *4th Workshop on Theorem Proving with Analytic Tableaux and Related Methods*, LNAI 918, pp. 122–137, 1995.
23. J. Otten & C. Kreitz. T-String-Unification: Unifying Prefixes in Non-Classical Proof Methods. *5th Workshop on Theorem Proving with Analytic Tableaux and Related Methods*, LNAI 1071, pp. 244–260, 1996.
24. B. Pientka & C. Kreitz. Automating inductive specification proofs. *Fundamenta Informatica*, 39(1–2):189–209, 1999.
25. A. Smaill & I. Green. Automating the synthesis of functional programs. Research paper 777, Dept. of Artificial Intelligence, University of Edinburgh, 1995.
26. S. Schmitt & C. Kreitz. On transforming intuitionistic matrix proofs into standard-sequent proofs. *4th Workshop on Theorem Proving with Analytic Tableaux and Related Methods*, LNAI 918, pp. 106–121, 1995.
27. S. Schmitt & C. Kreitz. Converting non-classical matrix proofs into sequent-style systems. *13th Conference on Automated Deduction*, LNAI 1104, pp. 418–432, 1996.
28. T. Tammet. A resolution theorem prover for intuitionistic logic. *13th Conference on Automated Deduction*, LNAI 1104, pp. 2–16, Springer, 1996.
29. L. Wallen. *Automated deduction in nonclassical logic*. MIT Press, 1990.
30. C. Weidenbach. SPASS: Combining superposition, sorts and splitting. *Handbook of Automated Reasoning*. Elsevier, 1999.