Connection-Driven Inductive Theorem Proving

Christoph Kreitz (kreitz@cs.cornell.edu)
Department of Computer Science, Cornell-University, Ithaca, NY 14853-7501, USA

Brigitte Pientka (bp@cs.cmu.edu)
Department of Computer Science, Carnegie Mellon University Pittsburgh, PA 15213-3891, USA

Abstract. We present a method for integrating rippling-based rewriting into matrix-based theorem proving as a means for automating inductive specification proofs. The selection of connections in an inductive matrix proof is guided by symmetries between induction hypothesis and induction conclusion. Unification is extended by decision procedures and a rippling/reverse-rippling heuristic. Conditional substitutions are generated whenever a uniform substitution is impossible. We illustrate the integrated method by discussing several inductive proofs for the integer square root problem as well as the algorithms extracted from these proofs.

Keywords: Theorem Proving, Induction, Program Synthesis, Matrix Methods, Rippling

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 (Constable et al., 1986; Allen et al., 2000), Coq (Dowek and et. al, 1991), Alf (Altenkirch et al., 1994), or Isabelle (Paulson, 1990) have been developed to support the logical reasoning process involved in these methods. They present proofs in a very expressive and readable sequent or natural deduction style that enables a user to check the results and guide the development of proofs interactively. But their expressiveness is paid for by a low degree of automation, which limits the extent to which these systems can currently be applied in practice. 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 (Bibel, 1981; Otten and Kreitz, 1995; Tammet, 1996) have been used successfully for proving formulas in *constructive* first order logic and guiding the development of first order proofs in interactive proof assistants (Kreitz et al., 1996). 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* (Bundy et al., 1990; Bundy et al., 1993) 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 quantifiers, which is necessary for synthesizing programs from their specifications. In fact, many synthesis problems require first-order and inductive proof methods to be applied simultaneously, as neither technique is 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 following two subgoals*

1. $\vdash\; \exists y\; y^2{\leq}0 \wedge 0{<}(y{+}1)^2$ *and*
2. $\exists y\; y^2{\leq}x \wedge x{<}(y{+}1)^2 \;\vdash\; \exists y\; y^2{\leq}x{+}1 \wedge x{+}1{<}(y{+}1)^2$

While the base case of the induction (1) can be solved by standard arithmetical reasoning about the number zero, no single rippling sequence can rewrite the conclusion of the step case (2) into the induction hypothesis, as the choice of the variable y *in the conclusion* (Y_c) *strongly depends on the properties of (the constant)* 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*.*

On the other hand, 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 check variations of 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 (Pientka and Kreitz, 1999) we have shown that a combination of an extended rippling procedure and a simple sequent-based proof tactic can already solve synthesis problems like the above automatically. But the logical proof search technique in this approach was not designed to be complete, because it analyzes logical formulae only by top-down decomposition of connectives and quantifiers. The simple proof tactic should therefore be replaced by a complete proof search method for constructive first-order logic.

Among the well-known complete proof procedures for intuitionistic first-order logic (Otten and Kreitz, 1995; Tammet, 1996) matrix-based proof techniques such as the *connection method* (Bibel, 1981; Kreitz and Otten, 1999) can be understood as very compact representations of sequent proof search. A matrix proof is an abstract representation of a class of sequent proofs that are equivalent up to permutation. Proof search procedures operate on a single formula structure and construct derivations by inserting pointers into this structure. Furthermore they are driven by *connections*, i.e. possible leaves in a sequent proof, instead of logical connectives and thus proceed in a more goal-oriented fashion.

These properties make matrix methods much more efficient than sequent-based proof procedures but still allow to convert the resulting proofs back into sequent proofs (Kreitz and Schmitt, 2000). Therefore, (constructive) matrix methods can be used to guide proof and program development in interactive proof assistants (Bibel et al., 1998; Kreitz et al., 1996), which suggests an integration of our extended rippling technique into matrix-based proof procedures.

In this paper we will present concepts for combining rippling techniques with matrix-based constructive theorem proving and show how to use the combined approach for the inductive development of programs from specification theorems. In Section 2 we will review our combination of (extended) rippling and sequent-based proof search, while the foundations of matrix-based logical proof search are summarized in Section 3. 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. Integrating Rippling and Sequent-based Proof Search

Rippling is an annotated rewriting technique specifically tailored for inductive theorem proving. Differences between the induction hypothesis and its conclusion (or *goal*) are marked by meta-level annotations, called *wave annotations*. Expressions that appear in both are called *skeletons*, while expressions that appear only in the conclusion are called *wave fronts*. The induction variable 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. Rewrite rules that are annotated accordingly are called *wave rules*. To illustrate, consider a wave rule that is derived from the recursive definitions of $+$.

$$\boxed{\underline{U}+1}^{\uparrow} + V \xmapsto{R} \boxed{(\underline{U+V})+1}^{\uparrow}$$

In this rule, $\boxed{\ldots+1}$ 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 all the way to the root 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 (Basin and Walsh, 1996) have developed a calculus for rippling and defined a well-founded *wave measure* under which rippling terminates provided that 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.

Based on Basin & Walsh's rippling calculus Kurucz (Kurucz, 1997; Bibel et al., 1998) developed a *rippling-distance* strategy. It introduces a new *distance 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 like *rippling-in* or *rippling-out*, a more goal-oriented proof search, and avoids non-termination problems in the presence of unknown wave fronts.

For instantiating existentially quantified variables within the framework of rippling, Bundy (Bundy et al., 1993) 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 (Kraan et al., 1993; Smaill and Green, 1995) use meta-annotations, which often require higher-order unification and may lead to non-terminating rippling sequences. Proof planning in (Armando et al., 1999) is based on *middle-out reasoning* and on domain-independent wave-rules for propositional reasoning. The latter, however, increase the search space for rippling and – due to a guidance by annotations – are not guaranteed to be logically complete.

The approach presented in (Pientka and Kreitz, 1999) addresses these drawbacks. By combining an extension of rippling with first-order

1. *Decompose* F by applying sequent calculus rules, introducing meta-variables for existentially quantified variables.
2. *Simultaneously match atomic subgoals* to compute a substitution for all the meta-variables using (1) a decision procedure, (2) unification, or (3) the *rippling/reverse rippling heuristic*.
   If a substitution can be found, the formula is valid.
3. If no uniform substitution for all the meta-variables can be found, select an unproven subgoal as *constraint* c for the validity of F.
4. Prove all subgoals under the constraint $\neg c$ by continuing with step 1.
5. Generate a sequent proof from the rippling sequences and the set of conditional substitutions.

Figure 1. Sequent-based proof method for an inductive step case formula F

proof techniques it separates domain-specific from domain-independent reasoning. *Logical decomposition* splits a conjecture into smaller goals, which are then handled by rippling. A *reverse rippling* strategy based on the distance measure is used to instantiate existentially quantified variables. *Conditional substitutions* (i.e. constraints) are *synthesized* if the goal cannot be solved uniformly. An overview of the integrated approach, originally designed for specifications of the form $\forall x \exists y P[x, y]$, is presented in Figure 1. It proceeds in five steps.

First, after applying the appropriate induction rule, the step case formula of an inductive proof is *decomposed* into subgoals by applying (top-down) sequent rules. Existentially quantified variables in the goal are replaced by meta-variables. A straightforward sequent proof tactic for constructive logic can be used for this purpose.

Next a *simultaneous match* is used to incrementally compute substitutions for all the meta-variables in the atomic subgoals. Given a substitution σ and a subgoal C (i.e. a sub-formula of 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 an extension σ' of σ such that $\sigma'(C)$ can be rippled to H.

The key technique used in alternative (3) is the *rippling/reverse rippling heuristic* developed in (Pientka and Kreitz, 1999, Section 3). It generates a rippling sequence from C to H and the substitution σ' in two phases: First the formula C is rewritten to some formula C_i. If $\sigma(C_i)$ does not match the corresponding term H in the induction hypothesis then a rippling sequence $C_i \overset{R}{\longmapsto} \ldots \overset{R}{\longmapsto} C_n \overset{R}{\longmapsto} H$ 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.

$$\underbrace{C \overset{R}{\longmapsto} C_0 \overset{R}{\longmapsto} \dots \overset{R}{\longmapsto}}_{\textit{rippling}} C_i \underbrace{\overset{R}{\longmapsto} \dots \overset{R}{\longmapsto} C_n \overset{R}{\longmapsto} H}_{\textit{reverse rippling}}$$

The search for a rippling sequence is based on the *rippling-distance* strategy introduced in (Kurucz, 1997), which ensures its termination.

After a rippling sequence and a substitution σ have been generated 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 decision procedures, rippling, and matching.

If a subgoal cannot be proven by these techniques, no uniform substitution for all the meta-variables in the subgoals can be found. In this case the unproven subgoal is selected as constraint c for the substitution σ and the remaining subgoals are checked under the condition c.

Afterwards, the method proves all subgoals under the negated constraint $\neg c$ and continues until it has found a *complementary set of constrained substitutions* that solve the induction step. This heuristic already turned out to be sufficient for many induction problems.

In a final step the conditional substitutions and rippling sequences that prove the induction step are translated into a sequent proof. This proof begins by applying the same sequent rules as in step 1, now instantiating existentially quantified variables according to the constructed substitution. A case analysis is inserted before decomposing the quantifier if a set of constrained substitutions was generated. Atomic sequences are proven by decision procedures, the *axiom* rule ($C \vdash C$), or converting a rippling sequence into series of (reverse) implications whose steps are shown by calling the lemmata that correspond to the wave rules as described in (Bibel et al., 1998, Section 3.3) and (Kurucz, 1997). Following the *proofs-as-programs* principle (Bates and Constable, 1985) one can then extract a conditional recursive proof expression that describes an algorithm for the given specification.

EXAMPLE 2. *Consider 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:*

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

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

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:

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

$$\boxed{\underline{U}+1} + V \quad \overset{R}{\longmapsto} \quad \boxed{\underline{U+V}+1} \tag{4}$$

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

Starting with subgoal (2) the method then tries to match the conclusion $C = \boxed{\underline{x}+1} < (Y_c+1)^2$ *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* $\boxed{\underline{x}+W} < \boxed{\underline{(y_h+1)^2}+W)}$*. This formula can then be rippled by wave rule (5), which instantiates* W *with* $2(y_h+1)+1$*. The inserted wave front moves closer to the sink variable* y_h*.*

Rippling towards the sink variable y_h *is now straightforward. By rule (4) the wave front* $\boxed{\ldots+1}$ *is moved to a position where it surrounds* y_h*, which leads to the formula* $C_0 = \boxed{\underline{x}+2(y_h+1)+1} < (\lfloor \boxed{\underline{y_h}+1} \rfloor +1)^2$*. As rippling has terminated successfully,* Y_c *can be instantiated by the sink value, which results in* $\sigma_1 := \{Y_c \backslash y_h+1\}$*. To prove that* C_0 *implies* $\sigma_1(C)$*, a decision procedure for induction-free arithmetic (*Chan, 1982*) is used. This proves that* σ_1 *is a valid substitution.*

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* $x+1<(y_h+1)^2$*. Matching* $x+1<(Y_c+1)^2$ *from subgoal (2) and* $x+1<(y_h+1)^2$ *yields* $\sigma_2 := \{Y_c \backslash y_h\}$*, which also makes the subgoal (1) true under the given constraint. Thus the induction step is provable under the set of conditional substitutions*

$$\{\ [\neg(x+1<(y_h+1)^2), \{Y_c \backslash y_h+1\}],\ [x+1<(y_h+1)^2, \{Y_c \backslash y_h\}]\ \}.$$

After converting the above proof into a sequent proof, an algorithm for computing an integer square root y *for each input* x*, together with its correctness proof, can be extracted. After stripping away the proof content, we arrive at an inductive program term, which in* Standard ML *notation reads as follows.*

```
fun sqrt x = if x=0 then 0
             else let val y = sqrt (x-1)
                  in
                    if x < (y+1)^2  then y
                    else y+1
                  end
```

The extended rippling technique, which is described in detail in (Pientka and Kreitz, 1999), has been implemented as proof tactic of the NuPRL proof development system (Constable et al., 1986) and applied to several program specifications such as quotient remainder, append, last, $\log_2$, and unification. It has also been extended further to instantiate 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. But the sequent proof tactic underlying this approach is neither complete nor very efficient, because it cannot deal with alternating quantifiers in deeply nested formulae. Such formulae require an in-depth analysis that goes beyond the capabilities of a top-down decomposition with meta-variables.

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 (Andrews, 1981; Bibel, 1981; Bibel, 1987) can be understood as compact representations of tableaux or sequent proof techniques. By characterizing a derivation as finding "admissible" pointers within a single formula structure they avoid the usual redundancies contained in these calculi. This allows a matrix proof to represent a whole class of sequent proofs that are equivalent up to permutation and facilitates the specification of proof search procedures that 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 (Otten and Kreitz, 1995), modal logics (Kreitz and Otten, 1999), and fragments of linear logic (Kreitz et al., 1997). In this section we will briefly summarize its essential concepts.

A *formula tree* is the representation of a formula F as a syntax tree. Each node corresponds to exactly one sub-formula of F and is marked with a unique name $a_0, a_1, \ldots$, its *position*. The *label* of a position u

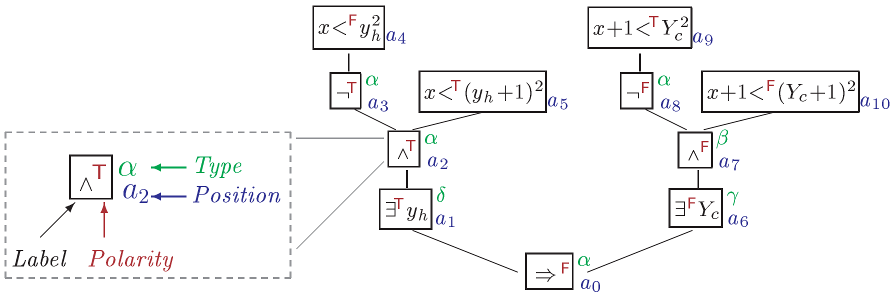


Figure 2. Formula tree for F_0

denotes the connective of the corresponding sub-formula or the sub-formula 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 where the *root* is the smallest position with respect to this tree ordering.

Each position in a formula tree is associated with a polarity and a principal type. The *polarity* (F or T) of a position is determined by the label and polarity of its parent. The root position has polarity F. The *principal type* of a position (α, β, γ, or δ) is determined by its polarity and its label. Atomic positions have no principal type. Polarities and types of positions are defined in the table below. For example, a position labelled with $\Rightarrow$ and polarity T has principal type β and its successor positions have polarity F and T, respectively. For a given formula we denote the sets of γ- and δ-positions[1] by Γ and Δ.

principal type α	$(A\wedge B)^{\mathsf{T}}$	$(A\vee B)^{\mathsf{F}}$	$(A\Rightarrow B)^{\mathsf{F}}$	$(\neg A)^{\mathsf{T}}$	$(\neg A)^{\mathsf{F}}$
successor polarity	$A^{\mathsf{T}}, B^{\mathsf{T}}$	$A^{\mathsf{F}}, B^{\mathsf{F}}$	$A^{\mathsf{T}}, B^{\mathsf{F}}$	A^{F}	A^{T}
principal type β	$(A\wedge B)^{\mathsf{F}}$	$(A\vee B)^{\mathsf{T}}$	$(A\Rightarrow B)^{\mathsf{T}}$		
successor polarity	$A^{\mathsf{F}}, B^{\mathsf{F}}$	$A^{\mathsf{T}}, B^{\mathsf{T}}$	$A^{\mathsf{F}}, B^{\mathsf{T}}$		
principal type γ	$(\forall x A)^{\mathsf{T}}$	$(\exists x A)^{\mathsf{F}}$	*principal type* δ	$(\forall x A)^{\mathsf{F}}$	$(\exists x A)^{\mathsf{T}}$
successor polarity	A^{T}	A^{F}	successor polarity	A^{F}	A^{T}

Formulae of type γ can be used in a proof several times in different ways. A quantifier *multiplicity* $\mu{:}\Gamma\to\mathbb{N}$ is an encoding of the number of distinct instances of γ-sub-formulae 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 the step case formula

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

[1] For technical reasons, matrix methods usually choose variable names to be identical to the *position* of the binding quantifier. In this article we choose less technical names instead, because they are better suited for illustration purposes, and denote γ-variables by capital letters.

$$\begin{bmatrix} \begin{bmatrix} x <^{\mathsf{F}} y_h^2 & x <^{\mathsf{T}} (y_h+1)^2 \end{bmatrix} & \begin{bmatrix} x+1 <^{\mathsf{T}} Y_c^2 \\ \\ x+1 <^{\mathsf{F}} (Y_c+1)^2 \end{bmatrix} \end{bmatrix}$$

Figure 3. Matrix of the formula F_0

from Example 1 (after unfolding $\leq$), together with polarities and principal types is presented in Figure 2.

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*), denoted $u{\sim_\alpha}v$ ($u{\sim_\beta}v$), if the greatest common ancestor of u and v wrt. the tree ordering $<$ is of principal type α (β). The matrix-representation of F_0 is shown in Figure 3.

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 atoms. A connection is *complementary* if the connected atoms can be unified by some *admissible* substitution. With these definitions the following matrix-characterization of validity has been shown.

THEOREM 3. *A formula F is valid iff there is a multiplicity μ, an admissible substitution σ, and a set of complementary connections $\mathcal{C}$ such that every path through F^μ contains a connection from this set.*

The above characterization theorem holds uniformly for a classical, intuitionistic, modal, and linear logics (Bibel, 1981; Bibel, 1987; Wallen, 1990; Kreitz et al., 1997; Mantel and Kreitz, 1998). However the precise definition of complementarity depends on the logic under consideration, as the dependencies between the rules in the corresponding sequent calculi need to be expressed by substitutions and their admissibility conditions.

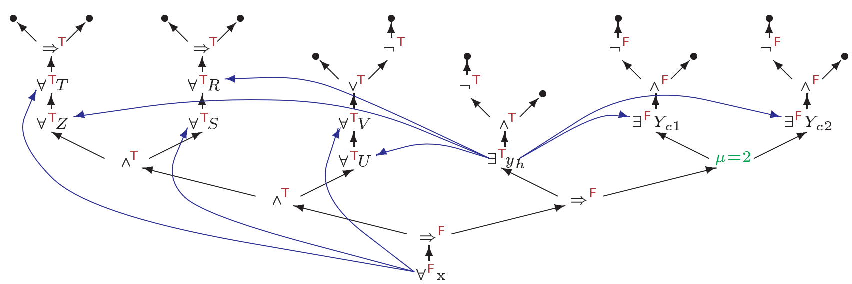
In *classical* logic, a substitution only has to express dependencies between quantifiers. A *(first-order) substitution* σ_Q is a mapping from positions of type γ to terms over $\Delta \cup \Gamma$, i.e. over γ- and δ-positions. It induces a relation $\sqsubset_Q \subseteq \Delta \times \Gamma$ in the following way: if $\sigma_Q(u) = t$, then $v \sqsubset_Q u$ for all $v \in \Delta$ occurring in t. This encodes the *eigenvariable condition* of the sequent calculus, where eigenvariables $v \in \Delta$ have to be introduced before they can be used in δ-quantifiers. Together with the tree ordering $<$ the relation $\sqsubset_Q$ determines a *reduction ordering* $\lhd := (< \cup \sqsubset_Q)^+$. σ_Q is *admissible* if this relation is irreflexive. A connection is *σ_Q-complementary* if σ_Q unifies the labels of the connected atoms.

EXAMPLE 4. *The formula* $F_0 \equiv \exists y\ \neg(x{<}y^2) \wedge x{<}(y{+}1)^2 \Rightarrow \exists y\ \neg(x{+}1{<}y^2) \wedge x{+}1{<}(y{+}1)^2$ *from Example 1 can be proven in first-order logic if we add to it the crucial lemmata* $\forall z \forall t\ t{+}1{<}z \Rightarrow t{<}z$, $\forall s \forall r\ s{<}r^2 \Rightarrow s{+}1{<}(r{+}1)^2$, *and the case analysis* $\forall u \forall v\ v{<}u \vee \neg(v{<}u)$, *as the following matrix proof with* $\mu(Y_c){=}2$ *shows.*

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

$$\sigma_Q = \{Z\backslash y_h^2,\ T\backslash x,\ Y_{c1}\backslash y_h,\ V\backslash x{+}1,\ U\backslash(y_h{+}1)^2,\ Y_{c2}\backslash y_h{+}1,\ S\backslash x,\ R\backslash y_h\}$$

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 σ_Q *given below the matrix. The resulting reduction ordering* $\lhd$ *(diagram below), which consists of the formula tree for* F_0 *extended by the three arithmetical lemmata (straight arrows) and the induced relation* $\sqsubset_Q$ *(curved arrows) is irreflexive.*



Thus the formula F_0*, extended by the three lemmata, is valid.*

In *constructive* logic, an admissible substitution also has to capture the dependencies between $\forall$, $\Rightarrow$, and $\neg$ to ensure that the connected atoms can be reached by applying the same sequence of intuitionistic sequent rules. Technically this can be achieved by unifying *prefixes*, i.e. strings of special positions between the root and the atom.

For this purpose the positions labelled with atoms, $\forall$, $\Rightarrow$, or $\neg$ receive an additional *intuitionistic type* ϕ if they have polarity T and ψ otherwise. ψ-positions represent non-permutable sequent rules and are considered constants whereas ϕ-positions are variables. Distinct instances of ϕ-sub-formulae are encoded by an *intuitionistic multiplicity* μ_J. The notion of an *indexed formula* F^μ is extended accordingly.

The *prefix pre(u)* of an atomic position u is a string $u_1 u_2 \dots u_n$ where $u_1{<}u_2{<}\dots{<}u_n{=}u$ are the elements of $\Psi{\cup}\Phi$ (the sets of ψ- and ϕ-positions) that dominate u in the formula tree. An *intuitionistic substitution* σ_J is a mapping from positions of type ϕ to strings over $\Psi{\cup}\Phi$. It induces a relation $\sqsubset_J \subseteq \Psi \times \Phi$ in the following way: if $\sigma_J(u){=}p$, then $v{\sqsubset_J}u$ for all characters $v \in \Psi$ occurring in p.

The *combined substitution* $\sigma := (\sigma_Q, \sigma_J)$ is *admissible* iff the induced *reduction ordering* $\lhd{:=}({<}{\cup}{\sqsubset_Q}{\cup}{\sqsubset_J})^+$ is irreflexive and $|\sigma_J(\mathrm{pre}(v))| \leq |\sigma_J(\mathrm{pre}(u))|$ holds for all $u \in \Gamma$ and all $v \in \Delta$ in $\sigma_Q(u)$. A connection is *σ-complementary* if σ_Q unifies the labels of the connected atoms and σ_J their prefixes. With these definitions, Theorem 3 holds accordingly.

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.

The key notions of this algorithm are active paths, active subgoals, and open goals. During search the *active path* $\mathcal{P}$ specifies those paths that are currently investigated for complementarity. All paths that contain $\mathcal{P}$ and an element u of the *active subgoal* $\mathcal{C}$ (a set of mutually β-related atoms that are α-related to all elements of of $\mathcal{P}$) have already been proven to contain a complementary connection. All paths that contain $\mathcal{P}$ and an element v of the *open goal* $\mathcal{E}$ must still be tested for complementarity. If the latter can be proven complementary as well then the *active goal* $(\mathcal{P},\mathcal{C})$ is *provable*, since all paths containing the active path are complementary. The proof search procedure is essentially described by the following two theorems.

THEOREM 5. *A formula F is valid iff there is a multiplicity μ and an admissible substitution σ such that the active goal $(\emptyset,\emptyset)$ w.r.t. F^μ is provable.*

THEOREM 6. *Let $(\mathcal{P},\mathcal{C})$ be an active goal and $\mathcal{E}$ the open goal w.r.t. $(\mathcal{P},\mathcal{C})$. The active goal $(\mathcal{P},\mathcal{C})$ is provable iff*

1. *the open goal $\mathcal{E}$ is empty, or*
2. *there is a complementary connection $\{A,\bar{A}\}$ with $A \in \mathcal{E}$, such that the active goal $(\mathcal{P},\mathcal{C}{\cup}\{A\})$ is provable and*
   $\bar{A} \in \mathcal{P}$ *or* $\bar{A}{\sim_\alpha}(\mathcal{P}{\cup}\{A\})$ *and* $(\mathcal{P}{\cup}\{A\},\{\bar{A}\})$ *is provable.*

Using Theorem 6 a path-checking algorithm will recursively check whether all paths containing the empty active path are complementary. According to Theorem 5 this is the case if and only if the given formula is valid. A precise definition of the above concepts and the proof search procedure can be found in (Kreitz and Otten, 1999).

The algorithm presented in (Kreitz and Schmitt, 2000) converts matrix proofs back into sequent proofs. For this purpose, it essentially traverses a formula tree F^{μ} in an order that respects the induced reduction ordering $\lhd{=}(<\cup\sqsubset_Q\cup\sqsubset_J)^+$ generated during proof search. It selects the appropriate sequent rule for each visited node and instantiates quantifiers according to the substitution σ_Q. The technical details of the conversion procedure are quite subtle, as it tries to avoid additional search during the conversion process. The conversion algorithm enables us to use matrix-based proof procedures to guide the development of proofs in interactive proof assistants.

4. Extending the Matrix-Characterization

In order to integrate our extended rippling techniques described in Section 2 into connection-based proof procedures we have to modify the matrix-characterization of logical validity given in Theorem 3 in several ways. It needs to provide a theoretical foundation for incorporating rippling sequences and decision procedures besides the usual term unification, for taking advantage of the specific symmetries in most inductive proofs, and for generating conditional substitutions.

A first extension of the characterization is due to the observation that a pair $\{A^{\mathsf{T}}, \bar{A}^{\mathsf{F}}\}$ of connected atoms does not necessarily have to be *equal* under a substitution σ_Q. Since 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_Q. This implication may be based on logical properties or, as in the case of inductive proofs, on the axioms of some theory $\mathcal{T}$ and can be proven by unification, by decision procedures, or by rewriting $\bar{A}$ into A.

In the terminology of matrices this means that we have to consider *directed connections* of the form $(u^{\mathsf{T}}, v^{\mathsf{F}})$, because these capture the concept of a sequence $A \vdash \bar{A}$ better than undirected connections. We also have to relax the unifiability condition to a notion of *implication with respect to a given theory* $\mathcal{T}$, written as $\Rightarrow_{\mathcal{T}}$. With these notions the concept of σ_Q-complementarity is extended as follows:

DEFINITION 7. A directed connection $(u^{\mathsf{T}}, v^{\mathsf{F}})$ is *σ_Q-complementary with respect to a theory $\mathcal{T}$* iff $\sigma_Q(A){=}\sigma_Q(\bar{A})$ or $\sigma_Q(A) \Rightarrow_{\mathcal{T}} \sigma_Q(\bar{A})$, where $A{=}\mathit{label}(u)$ and $\bar{A}{=}\mathit{label}(v)$.

In principle, $\Rightarrow_{\mathcal{T}}$ can denote any implication that is valid within the theory $\mathcal{T}$. Usually we will limit ourselves to implications that can be proven by standard decision procedures or by the rippling / reverse rippling heuristic. In our examples we consider *arithmetical implication*: $A \Rightarrow_{\mathcal{A}} \bar{A}$ denotes that either (1) $A \Rightarrow \bar{A}$ is provable by an (intuitionistic) arithmetical decision procedure[2] or that (2) there is a rippling sequence $\bar{A} \stackrel{R}{\longmapsto} \ldots \stackrel{R}{\longmapsto} A$ that rewrites $\bar{A}$ into A using arithmetical wave rules.

Equality and implication are not the only means to close a branch in a sequent proof within a theory $\mathcal{T}$. A sequent of the form $\cdot \vdash \bar{A}$ is provable if $\bar{A}$ can be proven to hold in $\mathcal{T}$. Likewise, $A \vdash \cdot$ is provable if A can be proven to be false in $\mathcal{T}$. To preserve the notation uniform, we consider the corresponding atomic positions to be *unary connections*.

DEFINITION 8.
Let u^{T} and v^{F} be unary connections with $label(u){=}A$ and $label(v){=}\bar{A}$.
u^{T} is *σ_Q-complementary with respect to a theory $\mathcal{T}$* iff $\sigma_Q(A) \Rightarrow_{\mathcal{T}} \mathsf{False}$.
v^{F} is *σ_Q-complementary with respect to $\mathcal{T}$* iff $\mathsf{True} \Rightarrow_{\mathcal{T}} \sigma_Q(\bar{A})$.

The concept of σ-complementary with respect to a theory $\mathcal{T}$ is defined accordingly for a combined substitution $\sigma = (\sigma_Q, \sigma_J)$ It resembles Bibel's theory-connections (Bibel, 1987) and could easily be extended to *n-ary directed connections* as well. It enables us to preserve the formulation of the matrix-characterization of logical validity given in Theorem 3 while broadening its meaning and range of application.

THEOREM 9. *A formula F is valid with respect to a theory $\mathcal{T}$ iff there is a multiplicity μ, an admissible substitution σ, and a set of possibly unary connections $\mathcal{C}$ that are σ-complementary with respect to $\mathcal{T}$ such that every path through F^{μ} contains a connection from this set.*

Accordingly, Theorems 5 and 6 remain valid for the extended notion of complementarity, which makes it possible to apply the general path-checking algorithm for constructive first-order logic even when dealing with inductive problems.

The only modification of the procedure in (Kreitz and Otten, 1999) is the test for complementarity. Besides the usual unification procedure for making two connected literals A and $\bar{A}$ equal it now incorporates the rippling / reverse rippling heuristic to find a substitution σ and a rippling sequence $\sigma(\bar{A}) \stackrel{R}{\longmapsto} \ldots \stackrel{R}{\longmapsto} \sigma(A)$ and decision procedures to prove a negative literal A to be false, a positive literal $\bar{A}$ to be true in $\mathcal{T}$, or an implication $A \Rightarrow \bar{A}$ to be valid. Furthermore, 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.

[2] A common procedure deals with *elementary arithmetic* as defined in (Chan, 1982), which includes the common axioms of `+`, `-`, `*`, `/`, `=`, `<` but no induction.

A second modification of the matrix-characterization comes from the insight that there is a strong relation between individual sub-formulae of the induction hypothesis H and those of the conclusion C. The latter is usually a variation of the former with the induction variable x replaced by the application $\rho(x)$ of an inductive constructor like $\mathsf{s} = \lambda x.x{+}1$ or a destructor like $\mathsf{p} = \lambda x.x{-}1$. That is, depending on the particular form of the induction rule, either C has the form $H[x\backslash\rho(x)]$ or H has the form $C[x\backslash\rho(x)]$.

Because of this symmetry, a rippling proof usually links a sub-formula of H to the corresponding sub-formula of C, which means that the proof-relevant connections in the matrix of an inductive formula run between corresponding sub-formulae of H and C. By checking these *orthogonal* connections first instead of proceeding in the order given by the generic path checking algorithm, we can significantly reduce the search space of inductive matrix-proofs.

DEFINITION 10. A formula F is *orthogonal with respect to a variable x* if F has the form $H \Rightarrow C$ and $C = H[x\backslash\rho(x)]$ (or $H = C[x\backslash\rho(x)]$) for some substitution ρ. A position $u \in H$ is *orthogonal to* a position $v \in C$ if the relative position of u in the formula tree of H is identical to the relative position of v in the formula tree of C. A connection $(u^{\mathsf{T}}, v^{\mathsf{F}})$ (or $(v^{\mathsf{T}}, u^{\mathsf{F}})$) is an *orthogonal connection* if the atomic positions $u \in H$ and $v \in C$ are orthogonal to each other.

If a formula F is orthogonal with respect to some variable x we also call F an *orthogonal formula* and its matrix-representation an *orthogonal matrix*. The name *orthogonal* becomes obvious if one considers the two-dimensional representation of an inductive step case formula $H \Rightarrow C$ (see e.g. Figures 3 and 5), in which the sub-matrices for H and C are orthogonal to each other.

Orthogonal connections are easy to identify, as they 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 only orthogonal connections.

THEOREM 11.
An orthogonal formula F is constructively valid if all orthogonal connections in F are complementary under some substitution σ.

Proof. By induction on the structure of H we show that every path through an orthogonal formula $F \equiv H \Rightarrow C$ contains an orthogonal connection. If these connections are complementary then F is valid according to Theorem 3.

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 must have the form $C_1\, \bar{op}\, C_2$ where $\bar{op}$ has type β in F and the matrices $F_i \equiv H_i \Rightarrow C_i$ are orthogonal. In this case each path $\mathcal{P}$ through F must go 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, since $\bar{op}$ must have type α in this case. If H has the form qH_1 where q is a quantifier of type γ (or δ) then C must have the form $\bar{q}C_1$ with $\bar{q}$ being of type δ (or γ). In this case the paths through F are identical to those through $H_1 \Rightarrow C_1$. $\square$

Considering orthogonal connections does not only improve the efficiency of path checking but also simplifies the complementarity test. Because of the symmetric structure of orthogonal formulae it is not necessary to check the unifiability of the prefixes of orthogonal connections.

LEMMA 12. *In an orthogonal formula F the prefixes of all orthogonal connections are simultaneously unifiable.*

Proof. Let (u, v) be an orthogonal connection in $F \equiv H \Rightarrow C$ and $a_0 a_1 ... a_n$ be the prefix of u. Since H and C are structurally identical every position in H that is labelled with $\forall$, $\Rightarrow$, $\neg$, or an atom has a corresponding position in C with the same label but opposite polarity. According to the definition of intuitionistic types and prefixes, the prefix of v must have the form $a_0 b_1 ... b_n$ where b_i is a variable (type ϕ) if and only if a_i is a constant (type ψ). Substituting the variables by the corresponding constant unifies the two prefixes.

Since this construction method is the same for all orthogonal connections, it leads to a common unifier for the prefixes of all orthogonal connections. $\square$

The intuitionistic substitution σ_J constructed in the above proof replaces each ϕ-position u by a string consisting of a single ψ-position v, which is orthogonal to u. Intuitively, this means that the induced relation $\sqsubset_J$ adds only horizontal arrows from ψ-positions to ϕ-positions in the tree ordering of F.

This property enables us to simplify the admissibility test for the combined substitution (σ_Q, σ_J) to the requirement that the induced relation $\sqsubset_Q$ does not introduce any down-going arrows from δ-positions to γ-positions. This guarantees the irreflexivity of the reduction relation $\lhd = (<\cup\sqsubset_Q\cup\sqsubset_J)^+$ and that the prefix of a δ-position in $\sigma_Q(u)$ will never longer than the prefix of u. Since σ_J preserves the length of prefixes the second admissibility condition for (σ_Q, σ_J) holds as well.

We define σ_Q to be *orthogonally admissible* if $v \le u$ or $v \le \bar{u}$ holds for all $v \in \Delta$ occurring in $\sigma_Q(u)$, where $\bar{u}$ is the unique position in F that is orthogonal to u. This requirement is trivially fulfilled in most inductive proofs, as σ_Q often replaces a γ-variable in the conclusion by a function over the corresponding variable in the hypothesis or vice versa.

Because of Theorem 11, Lemma 12, and the above observations, its classical proof with an orthogonally admissible substitution implies the constructive validity of an orthogonal matrix.

COROLLARY 13. *An orthogonal formula F is constructively valid if all orthogonal connections in F are σ_Q-complementary under some orthogonally admissible term substitution σ_Q.*

Checking the complementarity of orthogonal connections has some resemblance to Bibel's linear connection method (Bibel, 1986) but is used for a different purpose. It is sufficient but not necessary for the validity of an inductive step case formula. If a proof attempt with orthogonal connections fails, the formula may be valid for reasons that either require more complex inductive reasoning of have nothing to do with induction at all. In these cases a conventional constructive matrix proof has to be attempted.

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 before the quantifier is instantiated. Each case is then proven independently using decision procedures and rippling as additional methods for closing leaf sequents. A justification of this case distinction is the observation that a formula F can be proven from a set of assumptions if and only if there is a complete set $\{c_1,..,c_n\}$ of (decidable) logical constraints, such that each $F_i \ \equiv \ c_i \Rightarrow F$ is provable.

But constraints are usually not introduced at the top level of an inductive proof. In Example 2 the constraint on the substitution for Y_c, the existentially quantified variable in the goal, depends on the existentially quantified variable y_h in the hypothesis and is introduced into a branch of the proof that deals only with a sub-formula of F_0.

While in a sequent proof a localized case analysis can be expressed quite naturally, the global nature of matrix proofs prevents us from using constraints in their instantiated form, because in general they cannot be expressed as part of the original formula. However, a case distinction $c_1 \vee .. \vee c_n$ may be viewed as an instance of a more general lemma. In Example 2 the case distinction $\neg(x{+}1{<}(y_h{+}1)^2) \vee (x{+}1{<}(y_h{+}1)^2)$ is a special case of the lemma $\forall u.\ \neg(x{+}1{<}(u{+}1)^2) \vee (x{+}1{<}(u{+}1)^2)$ with u being instantiated by y_h. Thus, instead of generating an instantiated constraint, the matrix-based proof method will *synthesize* both the lemma and a simple substitution for its universally quantified variables. In addition, it will increase the multiplicity of the affected γ-branch to allow for different substitutions of the γ-variable in each case.

The following theorem shows that we may check the complementarity of F under each constraint independently, where F is called *σ-complementary under the constraint c^j* if every path through F and c^j is σ-complementary. This enables us to generate constraints step by step in an inductive proof.

THEOREM 14. *A formula F is valid iff there is a set $\{c_1,..,c_n\}$ of constraints with free variables $x_1..x_m$ such that $\forall x_1..x_m, c_1 \vee .. \vee c_n$ is valid, a substitution σ_c on $x_1..x_m$, and for all $i \leq n$ a multiplicity μ_i, and a substitution σ_i such that F^{μ_i} is $\sigma_c \circ \sigma_i$-complementary under c_i.*

Proof. Assume $\{c_1,..,c_n\}$, σ_c, μ_i, and σ_i are given as described above. Consider the matrix for the formula $F_* \equiv (\forall x_1..x_m, c_1 \vee .. \vee c_n) \Rightarrow (F \vee .. \vee F)$, where F occurs n times in F_*. Define $\mu(u) = 1$ and $\sigma(u) = \sigma_c(u)$ for a γ-position u of the x_j, $\mu(u) = \mu_i(u)$ and $\sigma(u) = \sigma_i(u)$ for a γ-position u in the i-th copy of F. σ is admissible, because σ_c and the σ_i are.

Let p be an arbitrary path through F_*^μ. Since p must contain one of the c_i while going through all the copies of F, a sub-path of p must have a $\sigma_c \circ \sigma_i$-complementary connection. Thus F_* is valid according to Theorem 3. Because $\forall x_1..x_m, c_1 \vee .. \vee c_n$ is valid, F must be valid as well.
The other direction of the bi-implication follows trivially from Theorem 3. □

The induction proof method in (Pientka and Kreitz, 1999) is designed to synthesize its constraints c_i as a conjunction $c_i^1 \wedge .. \wedge c_i^k$ of (decidable) *atomic constraints*, i.e. atomic goals or their negation, and c_n as logical complement of the other constraints. Therefore the lemma $\forall x_1..x_m, c_1 \vee .. \vee c_n$ will be constructively valid by construction.

Furthermore, the method constructs constraints whenever an atomic subgoal cannot be proven, i.e. whenever the corresponding orthogonal connection cannot be proven complementary. Thus eventually, all paths through the formula are closed either by an orthogonal connection or by 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^j* if either (u,v) or c^j (to be precise, the atomic position corresponding to c^j) and one of u and v form a complementary connection. These concepts allow us to extend Theorem 11 to orthogonal formulae with constraints.

THEOREM 15. *Let F be an orthogonal formula and $c^1,..,c^k$ be atomic constraints. If all orthogonal connections in F are σ-complementary under one of the c^j then F is σ-complementary under $(c^1 \wedge .. \wedge c^k)$.*

Proof. According to the definition of paths, each path p through F and $(c^1 \wedge .. \wedge c^k)$ consists of all the c^j and a sub-path p_F through F. Because of Theorem 11 p_F must contain an orthogonal connection (u,v). Since either (u,v) or some c^j and one of u and v form a σ-complementary connection, p must be σ-complementary. □

Again, a classical proof is sufficient to prove the σ-complementarity under constraints for orthogonal formulae. The unifiability of the corresponding prefixes does not have to be checked, because they always unify. The formulation of this insight is somewhat complex, as we have to express that every orthogonal connection is considered exactly once and no occurrence of a constraint is used twice in a connection.

LEMMA 16. *Let F be an orthogonal formula with orthogonal connections $(u_1,v_1),..,(u_n,v_n)$, $c^1,..,c^k$ be atomic constraints. Then the prefixes of any disjoint set of connections $(w_1,z_1),..,(w_n,z_n)$, where (w_l,z_l) is either (u_l,v_l) or a connection between some c^j and either u_l or v_l, unify simultaneously.*

Proof. According to Lemma 12 the prefixes of all orthogonal connections in F unify simultaneously. Let (w_l,z_l) be a connection between some c^j and either u_l or v_l. As an atomic constraint c^j is either an atomic formula or its negation with free variables bound by a universal quantifier.

In the first case c^j (that is, the position corresponding to c^j) has polarity T, intuitionistic type ϕ, and thus a prefix of the form $a_0b_1..b_mb$, where the the $b_1..b_m$ are the variable positions of the universal quantifiers and b is a variable that is distinct from all other variables. As the prefix of either u_l or v_l always begins with the root a_0, the prefixes of (w_l,z_l) are unifiable.

In the other case, the position corresponding to c^j has polarity F, intuitionistic type ψ, and thus a prefix of the form $a_0b_1..b_mba_2$, where b is a distinct variable that corresponds to the negation symbol in c^j. As the polarity of the connected position must be T, its prefix ends with a distinct variable that unifies with a_2, while the rest of its prefix string unifies with $a_0b_1..b_mb$.

As all the variables in the prefixes of all connected positions are distinct, the prefixes of set of connections $(w_1,z_1),..,(w_n,z_n)$ unify simultaneously.

Again, the admissibility of the combined substitution (σ_Q,σ_J) is ensured by this construction if σ_Q is orthogonally admissible. The induced relation $\sqsubset_Q$ can only insert arrows from F to the univerals quantifiers in the constraint lemma, while $\sqsubset_J$ either inserts arrows from F into the constraints or from constraint atoms to atoms in F, which guarantees the irreflexivity of $\lhd$. □

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 (Theorem 9), consider classical proofs with orthogonal connections first (Theorem 11, Lemma 12), generate conditions if necessary (Theorem 15, Lemma 16), and then solve the problem for the complement of these conditions (Theorem 14). The following example shows the advantages of such a proof method.

EXAMPLE 17. *Consider the formula $F_0 \equiv \exists y\,\neg(x{<}y^2)\wedge x{<}(y{+}1)^2 \Rightarrow \exists y\,\neg(x{+}1{<}y^2)\wedge x{+}1{<}(y{+}1)^2$ and its matrix-representation, which was given in Figure 3. The matrix contains two orthogonal connections, $(x{<}^{\mathsf{T}}(y_h{+}1)^2,\ x{+}1{<}^{\mathsf{F}}(Y_c{+}1)^2)$ and $(Y_c^2{<}^{\mathsf{T}}x{+}1,\ y_h^2{<}^{\mathsf{F}}x)$. Applying extended matching to the first leads to the (orthogonally admissible)* *substitution* $\sigma_1{=}\{Y_c\backslash y_h{+}1\}$*, as described in Example 2. Because the instantiated second connection* $((y_h{+}1)^2{<}^{\mathsf{T}}x{+}1,\ y_h^2{<}^{\mathsf{F}}x)$ *does not describe an arithmetically valid implication, which can easily be checked by arithmetical decision procedures, it is not complementary. Therefore we add* $x{+}1{<}(U{+}1)^2$ *as constraint* c_1 *(i.e. with opposite polarity* F*) to the matrix, and extend* σ_1 *by* $\sigma_c{=}\{U\backslash y_h\}$*. According to Theorem 15 we have thus established the complementarity of* F_0 *under* c_1*, which completes the first subproof (marked by* $\boxed{1}$*)*

$$\left[\ \begin{bmatrix} x{<}^{\mathsf{F}}y_h^2 & x{<}^{\mathsf{T}}(y_h{+}1)^2 \end{bmatrix} \begin{bmatrix} x{+}1{<}^{\mathsf{T}}Y_c^2 \\ \\ x{+}1{<}^{\mathsf{F}}(Y_c{+}1)^2 \end{bmatrix} \quad \begin{matrix} \boxed{x{+}1{<}^{\mathsf{F}}(U{+}1)^2} \\ \\ \boxed{x{+}1{<}^{\mathsf{T}}(U{+}1)^2} \end{matrix}\ \right]$$

$$\sigma = \{Y_{c1}\backslash y_h{+}1, U\backslash y_h, Y_{c2}\backslash y_h\}$$

To complete the proof we now try to establish the complementarity of F_0 *under the negated constraint* $c_2 = x{+}1{<}^{\mathsf{T}}(U{+}1)^2$ *and the substitution* σ_c*. By connecting* c_2 *it to* $x{+}1{<}^{\mathsf{F}}(Y_c{+}1)^2$ *we get* $\sigma_2{=}\{Y_c\backslash y_h\}$ *through conventional unification. As the instantiated second connection* $(x{+}1{<}^{\mathsf{T}}y_h^2,\ x{<}^{\mathsf{F}}y_h^2)$ *can now be proven complementary within arithmetic and Lemma 16 applies to the two connections, we have completed the second subproof (marked by* $\boxed{2}$*).* F_0 *is valid because of Theorem 14.*

Under the substitution $\sigma = \{Y_{c1}\backslash y_h{+}1, U\backslash y_h, Y_{c2}\backslash y_h\}$ *the four connections are complementary w.r.t the theory of arithmetic and span the indexed matrix for* $\forall U.\,(x{+}1{<}(U{+}1)^2) \vee \neg(x{+}1{<}(U{+}1)^2) \Rightarrow F_0$ *in the conventional way. The matrix proof of this extension of* F_0 *can then be converted into a sequent proof for* F_0*, which introduces the decidable instantiated case analysis* $(x{+}1{<}(y_h{+}1)^2) \vee \neg(x{+}1{<}(y_h{+}1)^2)$ *and then instantiates the existential quantifier in each case differently. The proof and its extracted algorithm are the same as in Example 2.*

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. The method, which is summarized in Figure 4, uses the same basic steps as the sequent-based method described in Figure 1 (Section 2) but in a more refined and goal-directed way due to the use of a matrix-representation and an emphasis on connections.

1. *Decompose* 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 F is valid.
3. For each connection that fails the complementarity test add the current active subgoal as *constraint* c^j for the complementarity of F.
   F is now proven to be complementary under the constraint $(c^1 \wedge .. \wedge c^k)$.
4. *Prove the complementarity of* F *under the constraint* $\neg c^j$ for each j≤k with the standard connection method extended by standard rippling and decision procedures.
   If the proof fails generate more constraints by continuing with step 2.
5. Generate a sequent proof from the matrix proof rippling sequences and the set of conditional subtitutions.

Figure 4. Matrix-based proof method for an inductive step case formula F.

In the first step *we decompose the step case formula* F by constructing its matrix-representation. This means building the syntax tree of F and annotating it with position names, polarities, and types. In the process δ-quantified variables will become constants and γ-variables will become (meta-)variables. Because dependencies between quantifiers are dealt with by the admissibility condition on substitutions, the matrix-based proof method can be applied to inductive proof problems with arbitrarily nested quantifiers. In contrast to that the sequent-based method is limited to (top-level) existential quantifiers in the conclusion and universal quantifiers in the induction hypothesis.

In the second step we attempt an inductive proof of the orthogonal matrix. Following Theorems 11 and 9 we *check the complementarity of all orthogonal connections* and generate a substitution σ for all γ-variables in the process. As in the usual connection method (cf. Theorems 5 and 6) we investigate these connections in an order given by *active subgoals*, i.e. sets of mutually β-related atoms that must be covered by complementary connections, and construct σ incrementally.

Given a partial substitution σ' and an orthogonal connection $(u^{\mathsf{T}}, v^{\mathsf{F}})$ with $A{=}label(u)$ and $\bar{A}{=}label(v)$ we (1) try to unify the terms $\sigma'(A)$ and $\sigma'(\bar{A})$, (2) apply a decision procedure to prove $\sigma'(A) \Rightarrow_{\mathcal{T}} \sigma'(\bar{A})$, or (3) apply the rippling / reverse rippling heuristic to find a substitution σ'' and a rippling sequence $\sigma''(\sigma'(\bar{A})) \stackrel{R}{\longmapsto} \ldots \stackrel{R}{\longmapsto} \sigma''(\sigma'(A))$. Because of Lemma 12 we do not have to check the unifiability of the corresponding prefixes. If all orthogonal connections are complementary and σ is orthogonally admissible, the formula F is valid according to Theorem 11.

If the test for complementarity fails for some connection (u,v) we *synthesize a constraint* c for the complementarity of F: we select the current active subgoal from the two connected atoms and add its instantiation c^j as constraint to F. This makes (u,v) σ-complementary under c^j. Technically, we take inverse of c^j, replace each δ-variable δ_i by a *new* γ-variable x_i, add the implicitly quantified result (which is also called c^j) as new α-related atom to the matrix, and extend σ by $\sigma_c = \{..\delta_i \backslash x_i..\}$. We then continue by investigating the remaining orthogonal connections.[3]

As a result of this step we get a constraint $c \equiv c^1 \wedge .. \wedge c^k$, a substitution σ_c for the free variables in c, a substitution σ_1 for γ-variables in F, and a proof for the $\sigma_c \circ \sigma_1$-complementarity of F and c according to Theorems 15 and 9. Note that c is usually decidable, because it is composed of atomic literals.

Using Theorem 14 we now have to *prove the complementarity of F under the constraint* $\neg c^j$ for all $j \leq k$ to complete the proof. For this purpose, our procedure adds the constraints $\neg c^j$ as β-related atoms to c, which creates a representation of $c \vee \neg c^1 \vee .. \vee \neg c^k$, and increases the multiplicity of F accordingly. The latter allows us to generate different substitutions for each constraint while reusing the structure that was built for constructing the matrix-proof for F under the constraint c.

Next, the paths through the $\neg c^j$ will be tested for complementarity. Since it is very likely that the constraints $\neg c^j$ characterize special cases of the proof that do not rely on induction at all, we attempt to derive a standard matrix proof and proceed with the standard connection method (Kreitz and Otten, 1999) with unification enhanced by decision procedures and standard rippling. Following Theorem 9 this will also include attempts to prove unary connections with decision procedures. No constraints will be introduced in this step.

If the conventional matrix proof for F under $\neg c^j$ 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 an additional case split in the program. The examples we tested so far required at most one such case split.

In the final step we convert the resulting matrix proof into a sequent proof, which enables us to extract a verifiably correct algorithm if the proven formula represents a program specification. Using the transformation algorithm presented in (Schmitt and Kreitz, 1995; Kreitz and

[3] The rationale behind the construction 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.

Schmitt, 2000) we generate a sequent proof skeleton. The leaves of this skeleton will then be proven by the axiom rule if unification was used in the corresponding connection, by a decision procedure, or by a series of reverse implications that correspond to a rippling sequence (see (Kurucz, 1997; Bibel et al., 1998) for details).

The extraction of a program from the sequent proof can be done in systems like NuPRL (Constable et al., 1986; Allen et al., 2000) by calling a built-in extraction mechanism. In the resulting algorithm the constraints generated during the proof will be reflected by a case distinction. Substitutions that resulted from orthogonal connections will describe a recursive case, while others will characterize the result when the recursion terminates.

EXAMPLE 18. *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. 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 step case formula* F_1*:*

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

This induction expresses that for every search for an integer square root n *of* x *that begins at* y *will succeed. By using the auxiliary variable* k *we can go backwards in the induction: a search starting at* y *can use results of a search starting at* $y{+}1$*. Thus a search beginning at* $y{=}0$ *will proceed by incrementing* y *until a solution has been found. This induction is more demanding for the prover but will result in a significantly more efficient program.*

After unfolding $\leq$ *we generate the matrix-representation of* F_1 *and the six orthogonal connections, as depicted in Figure 5. 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* $\boxed{1a}$ $\ldots$ $\boxed{1f}$ *using a decision procedure* Arith *for arithmetic (*Chan, 1982*), 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* $\sigma = \{X\backslash x,\ Y\backslash y{+}1\}$*.*

$\boxed{1b}$ *As* $x{<}(y{+}1)^2 \Rightarrow x{<}y^2$ *is not valid in arithmetic, we add the inverse of the instantiated active subgoal as constraint to the matrix: we extend the matrix by* $c_1 \equiv U <^{\mathsf{F}} (V{+}1)^2$ *and* σ *by* $\sigma_c{=}\{U\backslash x, V\backslash y\}$*.*

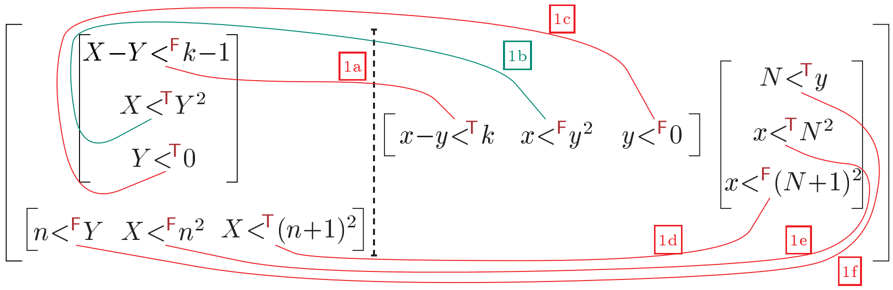
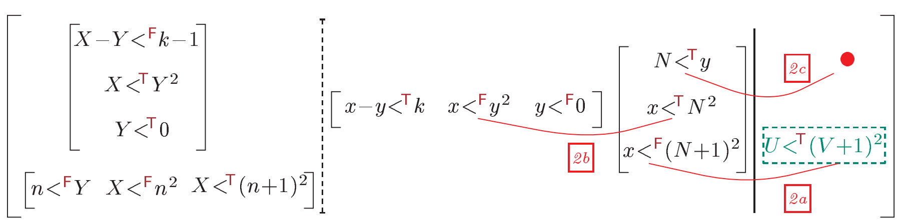


Figure 5. Matrix of F_1 with orthogonal connections

- **1c** *The instantiated third connection requires us to prove the implication $y+1<0 \Rightarrow_{\mathcal{A}} y<0$, which is shown by* Arith.
- **1d** *The fourth connection can be solved by unifying the terms $x<(n+1)^2$ and $x<(N+1)^2$. This extends σ to $\{X\backslash x, Y\backslash y+1, N\backslash n, U\backslash x, V\backslash y\}$.*
- **1e** *The terms in the instantiated fifth connection are now equal.*
- **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 complementarity of F_1 under the negation of c_1, i.e. $c_2 \equiv U <^{\mathsf{T}} (V+1)^2$, and the substitution σ_c. We add c_2 to the matrix, increase the multiplicity to 2, and select this literal as starting point for the second sub-proof, which will now be constructed with the standard connection method.



There are several $<^{\mathsf{F}}$-literals in the matrix that could be connected to the already instantiated atom c_2. Among those, all but $x <^{\mathsf{T}} (N+1)^2$ will immediately fail a complementarity proof because of a constant mismatch. Thus c_2 will be connected to the term $x <^{\mathsf{F}} (N+1)^2$ **2a** *and unification results in the substitution $\{N\backslash y\}$.*

This leaves the β-related atoms $x<^{\mathsf{T}}y^2$ and $y<^{\mathsf{T}}y$ as active subgoals. To solve the first, we connect to $x<^{\mathsf{F}}y^2$ **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***.*

As the prefixes of the connections are unifiable as well, the second subproof is now complete. The formula F_1 has been extended to $\forall U, V.\ (U{<}(V{+}1)^2 \lor \neg(U{<}(V{+}1)^2)) \Rightarrow F_1$, which is valid under the indexed substitution $\sigma = \{X_1\backslash x,\ Y_1\backslash y{+}1,\ N_1\backslash n,\ U\backslash x, V\backslash y,\ N_2\backslash y\}$.

After converting the above proof into a sequent proof, an algorithm for computing an integer square root n for each input x can be extracted. After stripping away the proof content and the auxiliary variable k we arrive at the following inductive program.

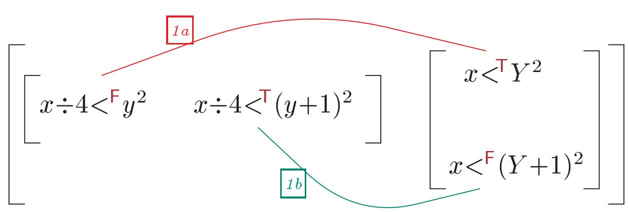
```
fun sqrt x = let fun aux x y =
                   if x < (y+1)^2  then y
                   else aux x (y+1)
             in
               aux x 0
             end
```

The example shows that depending on the chosen induction and the matrix proof one will get different algorithms for the same specification. The matrix proof for the integer square root specification in Example 17 results in an algorithm that is linear in the input x, while the algorithm in Example 18 operates in $\mathcal{O}(\sqrt{x})$. The algorithm that we will develop in the following example is even more efficient, but requires a non-standard induction that puts higher demands on the rippling / reverse rippling heuristic.

EXAMPLE 19. *An efficient form of computation on numbers is to operate on their binary representation. The corresponding induction scheme requires proving a conclusion $P(x)$ from an induction hypothesis $P(x{\div}2)$, where $\div$ denotes integer division. This particular induction scheme is not well suited for the integer square root specification, but its variation,* 4-adic induction, *is. Applied to $\forall x\, \exists y\ y^2{\leq}x \land x{<}(y{+}1)^2$ it yields (after unfolding $\leq$) the following step case formula*

$$F_2 \ \equiv\ \exists y\, \neg(x{\div}4{<}y^2) \land x{\div}4{<}(y{+}1)^2 \ \Rightarrow\ \exists y\, \neg(x{<}y^2) \land x{<}(y{+}1)^2$$

To prove this formula, we generate its matrix-representation and the orthogonal connections, depicted below. We select the induction hypothesis as starting point and try to prove the complementarity of the connections $\boxed{1a}$ and $\boxed{1b}$ using the same mechanisms as in Example 18.



$\boxed{\text{1a}}$ *To prove the connection complementary we have to find a substitution* σ *and a rippling sequence from* $\sigma(x{\div}4{<}y^2)$ *to* $\sigma(x{<}Y^2)$. *Using wave rules derived from the definitions of* $\div$ *and* y^2 *as well as the corresponding monotonicity laws and laws about constants, reverse rippling succeeds and generates the substitution* $\sigma = \{Y\backslash 2y\}$.

$\boxed{\text{1b}}$ *As* $x{\div}4{<}(y{+}1)^2 \Rightarrow_{\mathcal{A}} x{<}(2y{+}1)^2$ *is not valid, we add the inverse of the active subgoal, i.e.* $c_1 \equiv x{<}^{\mathsf{T}}(2U{+}1)^2$ *as constraint to the matrix and extend* σ *by* $\sigma_c{=}\{U\backslash y\}$, *which concludes the first subproof.*

We now have to prove the complementarity of F_2 *under the negation of* c_1 *and the substitution* σ_c. *We add* $c_2 \equiv x{<}^{\mathsf{F}}(2U{+}1)^2$ *to the matrix, increase the multiplicity, and select* c_2 *as starting point for the second sub-proof.*

$$\left[\;\left[\; x{\div}4{<}^{\mathsf{F}}y^2 \qquad x{\div}4{<}^{\mathsf{T}}(y{+}1)^2 \;\right] \quad \left[\begin{array}{c} x{<}^{\mathsf{T}}Y^2 \\[2em] x{<}^{\mathsf{F}}(Y{+}1)^2 \end{array}\right] \quad \boxed{x{<}^{\mathsf{F}}(2U{+}1)^2}\;\right]$$

Two $<^{\mathsf{T}}$*-literals can be connected to* c_2. *The complementarity test for* $x{\div}4{<}^{\mathsf{T}}(y{+}1)^2$ *fails immediately as in* $\boxed{\text{1b}}$, *while unifying* c_2 *with* $x{<}^{\mathsf{T}}Y^2$ *(connection* $\boxed{2a}$*) results in the substitution* $\{Y\backslash 2y{+}1\}$. *This leaves the* β*-related atom* $x{<}^{\mathsf{F}}(2y{+}2)^2$ *as active subgoal. We choose the connection* $\boxed{2b}$ *to the atom* $x{\div}4{<}^{\mathsf{T}}(y{+}1)^2$, *and prove it complementary by* Arith.

Since Lemma 16 applies to the two connections, the second subproof is complete as well. Thus the formula F_2, *extended by the constraint lemma* $\forall U.\,(x{<}(2U{+}1)^2) \vee \neg(x{<}(2U{+}1)^2)$ *has been proven valid under the substitution* $\sigma = \{Y_1\backslash 2y,\ U\backslash y,\ Y_2\backslash 2y{+}1\}$ *and the above proof leads to the following algorithm.*

```
fun sqrt x = if x=0 then 0
             else let val y = sqrt (x/4)
                  in
                    if x < (2*y+1)^2  then 2*y
                    else 2*y+1
                  end
```

The algorithm runs in logarithmic time, since both division by 4 and multiplication by 2 can be implemented by bit shift operations, while squaring requires logarithmic time.

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 and for the automated synthesis of recursive programs. The search for an inductive proof 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. Case distinctions with constraints are introduced if there is no uniform substitution for the quantified variables. The resulting matrix-proof can be converted into a verifiably correct conditional algorithm for a given inductive program specification.

Bringing together techniques from two different streams of automated reasoning enables us to deal with more complicated proof problems that require both logical proof search and rewriting. Matrix methods provide a uniform foundation for instantiating existentially and universally quantified variables, as they abstract from the notational peculiarities and focus on their tableaux 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. The systematic generation of case distinctions over constraints for the validity of a subformula introduces additional flexibility to the proof search procedure. This allows us to deal with complex induction schemes like the one in Example 18, which cannot be handled by either first-order and inductive proof methods individually, and to synthesize efficient algorithms for inductive programming problems.

Constraints are also a means for integrating conditional rewriting into the matrix-based framework. Conditions resulting from the application of conditional wave-rules can be added to the matrix and then be tested for complementarity to some other atom in the matrix. If this test fails we can either reject the conditional rewrite rule or treat the condition like a constraint as described in step 3 of our proof method (c.f. Figure 4). The following example illustrates this idea.

EXAMPLE 20. *The formula* $\forall x, l_1, l_2\ \ \mathtt{mem}(x, l_2) \Rightarrow \mathtt{mem}(x, l_1 \mathtt{@} l_2)$ *describes a condition for membership in a concatenated list. To prove this condition sufficient, we apply list induction over* l_1*, which leads to the inductive step case formula*

$$F_3 \equiv (\mathtt{mem}(x, l_2) \Rightarrow \mathtt{mem}(x, l_1 \mathtt{@} l_2)) \Rightarrow (\mathtt{mem}(x, l_2) \Rightarrow \mathtt{mem}(x, (y{::}l_1) \mathtt{@} l_2))$$

whose matrix representation is given below.

$$\left[\begin{bmatrix} \mathtt{mem}(x,l_2)^{\mathsf{F}} \\ \\ \mathtt{mem}(x,l_1\mathtt{@}l_2)^{\mathsf{T}} \end{bmatrix} \begin{bmatrix} \mathtt{mem}(x,l_2)^{\mathsf{T}} & \mathtt{mem}(x,(y{::}l_1)\mathtt{@}l_2)^{\mathsf{F}} \end{bmatrix}\right]$$

The matrix contains the orthogonal connections $\mathtt{mem}(x,l_2)^{\mathsf{F}}$, $\mathtt{mem}(x,l_2)^{\mathsf{T}}$ *and* $\mathtt{mem}(x,l_1\mathtt{@}l_2)^{\mathsf{T}}$, $\mathtt{mem}(x,(y{::}l_1)\mathtt{@}l_2)^{\mathsf{F}}$*. Proving the complementarity of the first is trivial. To prove the complementarity of the second, we need to apply the following set of conditional rewrite rules for membership (*`mem`*) and list concatenation (*`@`*):*

$$\begin{array}{rrcll} & \mathtt{mem}(X,\mathtt{nil}) & \longmapsto & \mathtt{false} & (6) \\ X{=}Y \;\Rightarrow & \mathtt{mem}(X,Y::L) & \longmapsto & \mathtt{true} & (7) \\ X{\neq}Y \;\Rightarrow & \mathtt{mem}(X,Y::L) & \longmapsto & \mathtt{mem}(X,L) & (8) \\ & \mathtt{nil}\mathtt{@}L & \longmapsto & L & (9) \\ & (X{::}L_1)\mathtt{@}L & \longmapsto & X{::}(L_1\mathtt{@}L) & (10) \end{array}$$

Rewriting with rule (8) leads to a condition $x{\neq}y$*, which we add as constraint to the matrix. Following our proof method, we now have to prove the complementarity of* F_3 *under the negated constraint* $x{=}y$*. Applying rewrite rule (7) converts the second connection into* $\mathtt{mem}(x,l_1\mathtt{@}l_2)^{\mathsf{T}}$, $\mathtt{true}^{\mathsf{F}}$*, which is trivially complementary according to definition 7.*

In general, constraints introduced by the application of conditional rewrite rules can be dealt with in two ways. We may either add the constraint and its negation, increase the multiplicity of the formula, and proceed as described above; or we may attempt to connect the constraint to some other literal in the matrix, which means the condition was already satisfied. The latter could also be handled by the concept of *3-ary theory connections* (Bibel, 1987), i.e. connections between 3 α-related atoms. In the future we intend to elaborate a formal account of constraints and conditional rewriting.

There have been other approaches to automate the instantiation of quantified variables in inductive specification proofs. Biundo (Biundo, 1988) 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. (Kraan et al., 1993) refines this idea by using rippling and middle-out reasoning (Hesketh, 1991) to control the search space. Both approaches, however, are not integrated into a logical proof environment and cannot guarantee that the synthesized program is correct, which means that it needs to be verified afterwards.

Armando, Smaill, and Green (Armando et al., 1999) use proof planning based on middle-out reasoning to synthesize recursive programs. In their method, propositional reasoning is performed by rippling with domain-independent wave-rules. While this uniform approach can be successfully applied to a variety of interesting examples, the increased set of (domain-specific and domain independent) wave-rules also increases the search space for rippling. Furthermore, due to a guidance by annotations, it cannot be guaranteed that logical reasoning is complete. In contrast to that our hybrid approach separates domain-specific from domain-independent reasoning and has the potential of achieving the best of both worlds. It allows rippling to focus on rewriting atomic propositions with a smaller set of rules, which reduces the search space for rewriting and also makes it easier to analyze failure.

Hutter (Hutter, 1994) introduces a technique to synthesize nontrivial induction orderings for existence proofs that is related to our reverse rippling heuristic (Pientka and Kreitz, 1999). 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. To generate the induction scheme it then looks at the definition of the recursive argument. This localized approach, however, causes difficulties when function symbols are defined by two-step or even more complex recursions.

The focus on generating induction schemes makes Hutter's work orthogonal to our approach, which assumes the induction scheme to be provided by a user. In inductive specification proofs this has the advantage of giving a user control over the efficiency of the synthesized program. But in other cases inductive theorem proving may profit from the combination of Hutter's and our work. The example of the integer square root shows that inductive theorem proving is not trivial even if we provide the induction ordering. We have to find the right case split and different instantiations for quantified variables. On the other hand, generating an appropriate induction scheme restricts the search space and provides more information to work with.

The INKA (Hutter and Sengler, 1996; Autexier et al., 1999) and SPASS (Weidenbach, 2000) provers integrate induction techniques into resolution-based theorem proving to handle complex proof problems. However, resolution theorem proving is generally non-constructive and neither of the systems is designed to synthesize case distinctions. This limits the extent to which these tools can be applied to inductive program specifications.

A different approach to solving complex reasoning problems is based on an extension of proof planning (Hutter, 1998). It decomposes large proof obligations into simpler tasks, which are then solved by specialized methods selected by the proof planning heuristic. The overall proof is then synthesized from the arising partial solutions. In contrast to that our approach builds on a complete logical proof strategy instead of a planning heuristic as the fundamental paradigm of search.

Our approach is a first step towards a full integration of constructive first-order and inductive theorem proving. Most of the individual components used in our method, i.e. matrix-based proof search for constructive logic (Kreitz and Otten, 1999), the rippling / reverse rippling heuristic (Pientka and Kreitz, 1999), and decision procedures for arithmetic (Chan, 1982) have already been implemented as independent components of the NuPRL proof development system. We intend to combine these components as described in Section 5 while 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 (Schmitt and Kreitz, 1995; Kreitz and Schmitt, 2000; Kurucz, 1997; Bibel et al., 1998) and use them to guide the development of proofs in interactive proof assistants and for the synthesis of conditional recursive programs.

References

Allen, S., R. Constable, R. Eaton, C. Kreitz, and L. Lorigo: 2000, 'The Nuprl Open Logical Environment'. In: D. McAllester (ed.): *17th Conference on Automated Deduction*, Vol. 1831 of *Lecture Notes in Artificial Intelligence*. pp. 170–176.

Altenkirch, T., V. Gaspes, B. Nordström, and B. von Sydow: 1994, 'A user's guide to ALF'. University of Göteborg.

Andrews, P.: 1981, 'Theorem-Proving via General Matings'. *Journal of the Association for Computing Machinery* **28**(2), 193–214.

Armando, A., A. Smaill, and I. Green: 1999, 'Automatic synthesis of recursive programs: The proof-planning paradigm'. *Automated Software Engineering* **6**(4), 329–356.

Autexier, S., D. Hutter, H. Mantel, and A. Schairer: 1999, 'Inka 5.0 - a logical voyager'. In: H. Ganzinger (ed.): *16th Conference on Automated Deduction*, Vol. 1632 of *Lecture Notes in Artificial Intelligence*. pp. 207–211.

Basin, D. and T. Walsh: 1996, 'A Calculus for and Termination of Rippling'. *Journal of Automated Reasoning* **16**(2), 147–180.

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

Bibel, W.: 1981, 'On Matrices with Connections'. *Journal of the Association for Computing Machinery* **28**, 633–645.

Bibel, W.: 1986, 'A deductive solution for plan generation'. *New Generation Computing* **4**, 115–132.

Bibel, W.: 1987, *Automated Theorem Proving*. Vieweg Verlag, second edition.

Bibel, W., D. Korn, C. Kreitz, F. Kurucz, J. Otten, S. Schmitt, and G. Stolpmann: 1998, 'A multi-level approach to program synthesis'. In: N. E. Fuchs (ed.):

Seventh International Workshop on Logic Program Synthesis and Transformation (LOPSTR'97), Vol. 1463 of *Lecture Notes in Artificial Intelligence*. pp. 1–25.

Biundo, S.: 1988, 'Automated synthesis of recursive algorithms as a theorem proving tool'. In: Y. Kodratoff (ed.): *ECAI-88 – 8^{th} European Conference on Artificial Intelligence*. pp. 553–558.

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

Bundy, A., F. van Harmelen, A. Smaill, and A. Ireland: 1990, 'Extensions to the Rippling-Out Tactic for Guiding Inductive Proofs'. In: M. E. Stickel (ed.): *10^{th} Conference on Automated Deduction*, Vol. 449 of *Lecture Notes in Computer Science*. pp. 132–146.

Chan, T.: 1982, 'A decision procedure for checking PL/CV arithmetic inferences'. In: *Introduction to the PL/CV2 Programming Logic*, Vol. 135 of *Lecture Notes in Computer Science*. Springer Verlag, pp. 227–264.

Constable, R. L., S. F. Allen, H. M. Bromley, W. R. Cleaveland, J. F. Cremer, R. W. Harper, D. J. Howe, T. B. Knoblock, N. P. Mendler, P. Panangaden, J. T. Sasaki, and S. F. Smith: 1986, *Implementing Mathematics with the NuPRL proof development system*. Prentice Hall.

Dowek, G. and et. al: 1991, 'The Coq proof assistant user's guide'. Institut National de Recherche en Informatique et en Automatique. Report RR 134.

Hesketh, J.: 1991, 'Using Middle-Out Reasoning to Guide Inductive Theorem Proving'. Ph.D. thesis, Dept. of Artificial Intelligence, University of Edinburgh.

Hutter, D.: 1994, 'Synthesis of Induction Orderings for Existence Proofs'. In: A. Bundy (ed.): *12^{th} Conference on Automated Deduction*, Vol. 814 of *Lecture Notes in Artificial Intelligence*.

Hutter, D.: 1998, 'Structuring deduction by using abstractions'. In: T. Ellman (ed.): *International Symposium on Abstraction, Reformulation, and Approximation (SARA-98)*.

Hutter, D. and C. Sengler: 1996, 'INKA, The Next Generation'. In: *13^{th} Conference on Automated Deduction*. pp. 288–292.

Kraan, I., D. Basin, and A. Bundy: 1993, 'Logic Program Synthesis via Proof Planning'. In: *4^{th} International Workshop on Logic Program Synthesis and Transformation*. pp. 1–14.

Kreitz, C., H. Mantel, J. Otten, and S. Schmitt: 1997, 'Connection-Based Proof Construction in Linear Logic'. In: W. McCune (ed.): *14^{th} Conference on Automated Deduction*, Vol. 1249 of *Lecture Notes in Artificial Intelligence*. pp. 207–221.

Kreitz, C. and J. Otten: 1999, 'Connection-based Theorem Proving in Classical and Non-classical Logics'. *Journal of Universal Computer Science* **5**(3), 88–112.

Kreitz, C., J. Otten, and S. Schmitt: 1996, 'Guiding Program Development Systems by a Connection Based Proof Strategy'. In: M. Proietti (ed.): *Fifth International Workshop on Logic Program Synthesis and Transformation*, Vol. 1048 of *Lecture Notes in Computer Science*. pp. 137–151.

Kreitz, C. and S. Schmitt: 2000, 'A Uniform Procedure for Converting Matrix Proofs into Sequent-Style Systems'. *Journal of Information and Computation* **162**(1–2), 226–254.

Kurucz, F.: 1997, 'Realisierung verschiedender Induktionsstrategien basierend auf dem Rippling-Kalkül'. Diplomarbeit, TU Darmstadt, FG Intellektik.

Mantel, H. and C. Kreitz: 1998, 'A Matrix Characterization for MELL'. In: U. F. J. Dix, F. L. Del Cerro (ed.): *6^{th} European Workshop on Logics in AI (JELIA-98)*, Vol. 1489 of *Lecture Notes in Artificial Intelligence*. pp. 169–183.

Otten, J. and C. Kreitz: 1995, 'A Connection Based Proof Method for Intuitionistic Logic'. In: P. Baumgartner, R. Hähnle, and J. Posegga (eds.): *4th Workshop on Theorem Proving with Analytic Tableaux and Related Methods*, Vol. 918 of *Lecture Notes in Artificial Intelligence*. pp. 122–137.

Paulson, L. C.: 1990, 'Isabelle: The next 700 theorem provers'. In: P. Odifreddi (ed.): *Logic and Computer Science*. Academic Press, pp. 361–386.

Pientka, B. and C. Kreitz: 1999, 'Automating inductive Specification Proofs'. *Fundamenta Informatica* **39**(1–2), 189–209.

Schmitt, S. and C. Kreitz: 1995, 'On Transforming Intuitionistic Matrix Proofs into Standard-Sequent Proofs'. In: P. Baumgartner, R. Hähnle, and J. Posegga (eds.): *4th Workshop on Theorem Proving with Analytic Tableaux and Related Methods*, Vol. 918 of *Lecture Notes in Artificial Intelligence*. pp. 106–121.

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

Tammet, T.: 1996, 'A resolution theorem prover for intuitionistic logic'. In: M. A. McRobbie and J. K. Slaney (eds.): *13th Conference on Automated Deduction*, Vol. 1104 of *Lecture Notes in Artificial Intelligence*. pp. 2–16.

Wallen, L.: 1990, *Automated deduction in nonclassical logics*. MIT Press.

Weidenbach, C.: 2000, 'SPASS: Combining Superposition, Sorts and Splitting'. In: A. Robinson and A. Voronkov (eds.): *Handbook of Automated Reasoning*. Elsevier Science Publishers B.V. (to appear).