October 2002

Proving program termination in higher order logic

Sava Krstic

John Matthews

Proving Program Termination in Higher Order Logic*

Sava Krstić John Matthews

OGI School of Science & Engineering
Oregon Health & Science University

October 17, 2002

Abstract

We suggest two simple additions to packages that use wellfounded recursion to justify termination of recursive programs:

- The *contraction condition*, to be proved in cases when termination conditions are difficult or impossible to extract automatically;
- user-supplied *inductive invariants* in cases of nested recursion.

We have implemented these additions in *Isabelle/HOL* and demonstrated their usefulness on a large example of the recursive BDD algorithm Apply.

The paper uses a simple predicate on functionals in higher order logic as an approximate model of program termination. We explore this model and prove that for every functional there exists a largest "domain of termination".

*The research reported in this paper was supported by the National Science Foundation Grants EIA-0072761 and CDA-9703218, Compaq Computer Corporation, and Intel Corporation.

1 Introduction

Proving termination of a recursively defined program f by wellfounded recursion means finding a wellfounded ordering (*termination relation*) on the program's input type and checking that for any argument x the recursive calls needed to compute $f(x)$ use only arguments smaller than x. When the pattern of recursive calls is simple enough, we can derive from the program text a finite set of inequalities (*termination conditions*) sufficient to guarantee termination. The whole process is also amenable to mechanization with various possible degrees of automation. The termination relation is guessed or supplied by the user, the termination conditions are generated from the program text, then either proved automatically, or given to user as proof obligations.

Some termination proofs are inherently difficult, but there are also annoying cases where an obvious termination is difficult to capture by automated tools. Two simple examples are given by recurrences

$$\begin{array}{rcl} f(0) & = & 0 \\ f(n+1) & = & f(0)+f(1)+\cdots+f(n) \end{array} \qquad \begin{array}{rcl} g(0) & = & 0 \\ g(n+1) & = & g(g(n)) \end{array}$$

that readily translate into valid and obviously terminating programs of any given language supporting recursion.

The problem with f is that it requires checking n termination conditions for any non-zero input n, whereas commonly used tools see only a fixed finite number of recursive calls in the program text and derive that many termination conditions.

The problem with g is *nested recursion*—occurrence of recursive calls within recursive calls. The termination condition $g(n) < n+1$ appears to be in a vicious circle: we need it in order to define g, but to prove it, we need to know something about g.

For programs like f, where standard algorithms for extraction of termination conditions do not work[1], we describe in Section 2 an alternative and more direct method that instead of termination conditions uses a simple *contraction property* as a proof obligation that guarantees termination.

The circularity in attempts to prove termination of nested recursive programs is a more serious problem, with which we deal in Section 3. We will show that circularity can be avoided if what is needed to know about the program in order to prove termination conditions (or contraction in the alternative approach) can be expressed as a *inductive invariant* property. Inductive invariants are certain input-output relations defined in Section 3. Using them we obtain a simple framework that generates clear and uniform proof obligations from a

[1] More interesting cases of the same kind are given by recurrences

$$B_{n+1} = \sum_{k=0}^{n} B_k \left(\begin{array}{c} n \\ k \end{array} \right) \qquad C_{n+1} = \sum_{k=0}^{n} C_k C_{n-k}$$

for Bell and Catalan numbers.

given nested recursive definition and (user-supplied or automatically generated) termination relation and an inductive invariant. Virtually all previously known termination proofs of nested cases of recursion fall within our framework.

In Section 5 we describe the use of contraction and inductive invariants on a large and difficult example of the imperative program APPLY that is a key component of any efficient BDD (boolean decision diagram) package.

Contraction and inductive invariants can be used in pencil-and-paper proofs and in many formalizations. We choose higher order logic as our formal mathematical environment in what follows because it is the most natural and popular for logical modeling of programs. Our implementations are written in the logic of *Isabelle/HOL* theorem prover (in the sequel, *HOL*) [11]. Among *Isabelle/HOL*'s many provided facilities is *recdef*, a powerful library for generating and verifying terminating conditions via wellfounded induction [13]. Our techniques are meant to be used when *recdef* is difficult to apply.

Simple modeling of programs as HOL functions cannot, of course, capture the notion of termination accurately. The practice has been that to prove termination of a program it suffices to prove by wellfounded recursion the existence and uniqueness of a fixpoint of the HOL functional associated with the original program. In order to put our results in a precise context, we adopt in Section 2 a stronger definition of "HOL termination" that is a better approximation of the concept.

In Section 4 we extend our definition and results to accomodate modeling of programs that terminate only on a subset of their input type. The intended domain of termination is, as in *Isabelle*'s *recdef*, given as an additional parameter together with the termination relation. This raises the question of the relationship between fixpoints of the same functional obtained by various choices for the termination relation and for the domain of definition. We answer this question in Section 6. Theorem 7 shows the existence of the largest domain of termination for every functional and also uniqueness of its fixpoint. Theorem 9 demonstrates an analogous result for a more realistic notion of termination that includes the requirement of finite height of all nodes in the calling tree.

2 Termination and Contraction

Declarations of recursive programs are of the form

$$f\,x = M, \tag{1}$$

where M is some term that contains no free variables other than f and x. Modeling the declaration in *HOL*, we assume M is a *HOL* term of type B, where f and x have types $A \Rightarrow B$ and A respectively. The equation (1) is not a valid *HOL* definition, so what a *HOL* recursive definition method needs to do is produce by some other means a definition of f and prove (1). It is convenient to abstract the variables in M and use the associated functional $F = \lambda f\,x.\ M\colon (A \Rightarrow B) \Rightarrow (A \Rightarrow B)$ as the input to the method. Thus (1)

translates into the *fixpoint equation* (or *recursion equation*)

$$\forall x.\ f\,x = F\,f\,x. \tag{2}$$

The existence and uniqueness of f satisfying (2) is a necessary condition for termination of the original program, but it is not sufficient, as seen in the example program $f\,x = 2*(f\,x)$. Since termination of programs depends crucially on the semantics of the programming language used, there is no hope of covering this notion exactly by some simple definition of *HOL* "terminating functionals". A definition that reasonably approximates the notion of termination is still better than no definition, so we adopt the following, postponing the discussion of the alternatives given to the end of the section.

Definition 1 *The functional* F terminates at *the function* ϕ *iff there exists a wellfounded relation* ρ *such that*

$$\forall f\,x.\ f =_{\rho^{-1}x} \phi \longrightarrow F\,f\,x = \phi\,x. \tag{3}$$

Here and in the sequel we use the notation $\rho^{-1}x \equiv \{y \mid y\,\rho\,x\}$ and $f =_D g \equiv \forall y \in D.\ f\,y = g\,y$. We will also use the related notation $f \restriction D$ for the restriction of f on D; this partial function is represented in *HOL* as a total function, and defined by

$$f \restriction D = \mathsf{if}\ x \in D\ \mathsf{then}\ f\,x\ \mathsf{else}\ \mathsf{Arb},$$

where $\mathsf{Arb} \equiv \varepsilon z.\ \mathsf{True}$ is an arbitrary element of the domain of f.

To explain Definition 1, note that a recursive program together with the operational semantics of its programming language define a *calling relation*: $y \prec x$ iff the evaluation of the program with a given argument x makes a recursive call to itself with argument y. (See [12] for some examples where the calling relation, called there the *recursion relation* can be effectively used.) The program terminates if and only if the calling relation is wellfounded. Thus, the condition expressed by Definition 1 is necessary for termination.

The main theorem behind wellfounded recursive definitions ([16], Theorem 10.19), as proved in *HOL* by Nipkow, reads as follows:

Theorem 1 ([13]) *Let* $F\colon (A \Rightarrow B) \Rightarrow (A \Rightarrow B)$ *and let* ρ *be a wellfounded relation on* A. *Let* $\psi = \mathsf{wfrec}\,\rho\,F$. *Then*

$$\psi\,x = F\,(\psi \restriction \rho^{-1}x)\,x, \tag{4}$$

for every $x \in A$. □

We do not need to know the exact definition of the function wfrec. Its existence is the point of the theorem; the uniqueness with respect to satisfying (4) follows immediately by wellfounded induction.

Lemma 1 *If* F *terminates at* ϕ, *then*

(a) ϕ *is the unique fixpoint of* F;

(b) $\phi = \mathsf{wfrec}\,\rho\,F$ *for some wellfounded relation* ρ.

Proof. *(a)* Wellfounded induction.
(b) Suppose ρ is such that (3) holds and let $\psi = \mathsf{wfrec}\,\rho\,F$. Instantiating f with $\psi \restriction \rho^{-1}x$ in (3), we get

$$\forall x.\ (\psi \restriction \rho^{-1}x) =_{\rho^{-1}x} \phi \longrightarrow F\,\psi \restriction \rho^{-1}x\,x = \phi\,x.$$

The antecedent here is clearly equvalent to $\psi =_{\rho^{-1}x} \phi$, while the consequent is, in view of Theorem 1, equivalent to $\psi\,x = \phi\,x$. Now $\psi = \phi$ follows by wellfounded induction. □

Now we state and prove the first sufficient condition for termination.

Definition 2 *The* contraction condition *for the functional* F *with respect to the wellfounded relation* ρ *is*

$$\forall f\,g\,x.\ f =_{\rho^{-1}x} g \longrightarrow F\ f\ x = F\ g\ x. \tag{5}$$

Theorem 2 *If the contraction condition (5) is satisfied, then* F *terminates at* $\mathsf{wfrec}\,\rho\,F$.

Proof. We need to check the condition

$$\forall f\,x.\ f =_{\rho^{-1}x} \psi \longrightarrow F\,f\,x = \psi\,x,$$

where $\psi = \mathsf{wfrec}\,\rho\,F$. The antecedent part is clearly equivalent to $f =_{\rho^{-1}x} (\psi \restriction \rho^{-1}x)$, while the consequent part is, by Theorem 1, equivalent to $F\,f\,x = F\,(\psi \restriction \rho^{-1}x)\,x$, making our goal an instance of the assumed contraction condition. □

2.1 *Recdef*

Recdef is a *HOL* definition package designed by Slind [11, 13, 15, 14], and its main purpose is the derivation of the fixpoint theorem

$$\forall x.\ \psi\,x = F\,\psi\,x$$

where $\psi = \mathsf{wfrec}\,\rho\,F$, for some wellfounded relation ρ and a given functional F. After a brief survey of *recdef*, we will show that when it succeeds in proving the fixpoint theorem, it could also derive a stronger conclusion that F satisfies the contraction condition.

Recdef begins with Theorem 1 in the form

$$WF\,\rho \longrightarrow \psi\,x = F\,(\psi \restriction \rho^{-1}x)\,x, \tag{6}$$

where WF denotes the wellfoundedness predicate and ρ, x are arbitrary. The procedure then analyzes the structure of the term M (recall that $F = \lambda f\,x.\ M$) in order to find occurrences $f\,t_1, \ldots, f\,t_m$ of recursive calls as subterms of M. For each calling site it also gathers the contextual information Γ_i and extracts

the *termination condition* $TC_i \equiv \Gamma_i \longrightarrow t_i\,\rho\,x$, where ρ is at the moment an unspecified wellfounded relation. Since, by definition of function restriction,

$$t_i\,\rho\,x \longrightarrow (\psi \restriction \rho^{-1}x)\,t_i = \psi\,t_i,$$

it is possible through a sequence of steps following the bottom-up traversal of the term M to transform the theorem (6) into one where all occurrences of $\psi \restriction \rho^{-1}x$ are replaced with ψ, at the price of adding termination conditions as assumptions:

$$WF\,\rho \wedge TC_1 \wedge \cdots \wedge TC_m \longrightarrow \psi\,x = F\,\psi\,x. \tag{7}$$

Proving the termination conditions TC_i for a suitably instantiated wellfounded relation ρ is the last step, the result of which is the desired recursion theorem for $\psi = \mathsf{wfrec}\,\rho\,F$.

Note now that if we started with the theorem $F\,(f \restriction \rho^{-1}x)\,x = F\,(f \restriction \rho^{-1}x)\,x$ instead of (6), then the same procedure (transforming the right-had side progressively, while keeping the left-hand side fixed) would yield the theorem

$$TC_1 \wedge \cdots \wedge TC_m \longrightarrow F\,(f \restriction \rho^{-1}x)\,x = F\,f\,x. \tag{8}$$

in which ρ and f are free variables. Thus, if the termination conditions TC_i are provable for some wellfounded relation ρ, then

$$\forall f\,x.\ F\,(f \restriction \rho^{-1}x)\,x = F\,f\,x$$

is provable as well. It is easy to see that this equation is equivalent to saying that F satisfies the contraction condition with respect to ρ, so we have the following meta-result.

Lemma 2 *The termination conditions generated by the* recdef *procedure imply the corresponding contraction condition.* □

2.2 Summary

Each of the following statements, in which ρ is assumed to be wellfounded, is a possible notion of "termination" in *HOL*.

T_1:	*recdef*'s termination conditions for F are true for some ρ
T_2:	F satisfies the contraction condition with respect to some ρ
T_3:	F terminates (Definition 1)
T_4:	for some ρ, $\mathsf{wfrec}\,\rho\,F$ is a unique fixpoint of F
T_5:	for some ρ, $\mathsf{wfrec}\,\rho\,F$ is a fixpoint of F

By the results given above (or for simpler reasons), each of the statements T_i implies one below it. The following examples show that none of these implications is reversible. Programs f and g given in the Introduction are counterexamples for $T_2 \not\Rightarrow T_1$ and $T_3 \not\Rightarrow T_2$ respectively. For $T_4 \not\Rightarrow T_3$, a counterexample is given by $F\,f\,x \equiv \mathsf{if}\ x = 0\ \mathsf{then}\ f(1) - f(0)\ \mathsf{else}\ f(x-1)$; its only fixpoint is

the constant zero function, which is also equal to $\mathsf{wfrec}\,(<)\,F$, but F does not terminate. Finally, for $T_5 \not\Rightarrow T_4$ take $F\,f\,x \equiv f\,(x-1)$; every constant function is is its fixpoint, and $\mathsf{wfrec}\ \ (<)\,F$ (which happens to be the function constantly equal to Arb) is just one of them.

In practice, the weakest condition T_5 seems to be the most often used criterion for termination. However, most of the existing termination proofs proceed by checking satisfaction of termination conditions (as *recdef* does), so they actually demonstrate that T_1 holds.

3 Inductive Invariants

Consider again the conditional recursion equation (3)

$$WF\,\rho \wedge TC_1 \wedge \cdots \wedge TC_m \longrightarrow \psi\,x = F\,\psi\,x,$$

where $\psi = \mathsf{wfrec}\,\rho\,F$. If we are dealing with nested recursion, then ψ appears in the termination conditions TC_i, and it may be impossible to prove these conditions without knowing something about ψ. However, these conditions can be (and often are) properties of ψ that one can directly prove; that is, without assuming the truth of the recursion equation $\psi\,x = F\,\psi\,x$. Here we show that a whole class of properties, called *inductive invariants*, can be proved without using the recurrence equation, and therefore can be used in the proof of that equation. In fact, as our survey of the literature shows, most of the existing proofs of termination of functions defined by nested recursion are based on some inductive invariant property.

Predicates of the form $S\colon A \Rightarrow B \Rightarrow \mathsf{bool}$ express input-output relations for functions $f\colon A \Rightarrow B$. We will say that f *satisfies* S if $S\ x\ (f\ x) = \mathsf{True}$ holds for all $x \in A$.

Definition 3 *A predicate* $S : A \Rightarrow B \Rightarrow \mathsf{bool}$ *is a* inductive invariant *of the functional* $F\colon (A \Rightarrow B) \Rightarrow (A \Rightarrow B)$ *associated with the wellfounded relation* ρ *iff the following condition is satisfied:*

$$\forall f\,x.\ (\forall y.\ y\,\rho\,x \longrightarrow S\,y\,(f\,y)) \longrightarrow S\,x\,(F\,f\,x) \tag{9}$$

If f satisfies an inductive invariant S, it follows from (9) that $F\,f$ satisfies S as well. Thus inductive invariants are invariants.

Checking inductive invariance is an inductive method of proving properties of $\mathsf{wfrec}_{\prec}\,F$, as the following key result shows.

Lemma 3 *If* ρ *is wellfounded then* $\mathsf{wfrec}\,\rho\,F$ *satisfies all inductive invariants of* F *associated with* ρ*.*

Proof. Denote $\psi = \mathsf{wfrec}\,\rho\,F$. Assuming S is an inductive invariant and instantiating f in (9) with $\psi \restriction \rho^{-1}x$, we get

$$\forall x.\ (\forall y.\ y\,\rho\,x \longrightarrow S\,y\ \big((\psi \restriction \rho^{-1}x)\,y\big) \longrightarrow S\,x\,(F\,(\psi \restriction \rho^{-1}x)\,x).$$

This is equivalent to

$$\forall x.\ (\forall y.\ y\,\rho\,x \longrightarrow S\,y\,(\psi\,y)) \longrightarrow S\,x\,(\psi\,x) \tag{10}$$

because $\psi \restriction \rho^{-1}x\,y = \psi\,y$ is clearly true when $y\,\rho\,x$, and $F\,(\psi \restriction \rho^{-1}x)\,x = \psi\,x$ is Theorem 1. The formula (10) is exactly what is needed for the inductive proof of the formula $\forall x.\ S\,x\,(\psi\,x)$ saying that ψ satisfies S. □

We will now weaken the contraction condition by adding a premise involving an input-output relation; then we will prove that the weakened condition is still sufficient to guarantee termination, at the price of establishing first that the input-output relation used is an inductive invariant.

Definition 4 *The* restricted contraction condition *for the functional F with respect to a wellfounded relation ρ and an inductive invariant S is given by the formula:*

$$\forall f\,g\,x.\ f =_{\rho^{-1}x} g \wedge (\forall y.\ S\,y\,(g\,y)) \longrightarrow F\ f\ x = F\ g\ x. \tag{11}$$

Thus, in (11) the restriction is that g satisfies S. The old contraction condition (5) is a special case of (11), corresponding to the trivial (constantly true) inductive invariant S.

Theorem 3 *Suppose the restricted contraction condition (11) is satisfied and S is an inductive invariant of F associated with ρ. Then F terminates at* $\mathsf{wfrec}\,\rho\,F$.

Proof. Let $\psi = \mathsf{wfrec}\,\rho\,F$. Instantiate (11) with $g = \psi$. The second conjunct is true by Lemma 3, so we obtain

$$\forall f\,x.\ f =_{\rho^{-1}x} \psi \longrightarrow F\,f\,x = F\,\psi\,x. \tag{12}$$

It only remains to prove $\forall x.\ F\ \psi\ x = \psi\ x$. For this, jsut instantiate $f = \psi \restriction \rho^{-1}x$ in (12) and use Theorem 1. □

Theorem 3 gives a method for proving termination: find a inductive invariant and prove the corresponding restricted contraction condition. The method is conceptually simple and easily implementable. Given definitions of a wellfounded relation and an input-output relation, it generates two goals—inductive invariance of the relation and restricted contraction condition—that can be either passed to the user, or (sometimes) disposed of automatically.

3.1 Combining Inductive Invariants with *recdef*

Inductive invariants can also be used in conjunction with *recdef* (Section 2.1), so that *recdef* is used in place of checking the restricted contraction condition. The conclusion is weaker, though: instead of termination, we only get the fixpoint equation.

Recall that the procedure for extracting the termination conditions associated with the functional F produces a predicate TC such that

$$TC\,\rho\,f \longrightarrow F\,(f \restriction \rho^{-1}x)\,x = F\,f\,x, \tag{13}$$

This is just formula (7) of Section 2 with $TC \equiv \lambda\rho\,f.\ (TC_1 \wedge \cdots \wedge TC_m)$.

Theorem 4 *If there exists a wellfounded relation ρ and an inductive invariant S associated with ρ such that*

$$(\forall x.\ S\,x\,(f\,x)) \longrightarrow TC\,\rho\,f,$$

then $\mathsf{wfrec}\,\rho\,F$ *is a fixpoint of* F.

Proof. Instantiate $f = \mathsf{wfrec}\,\rho\,F$ in (13), combine with Theorem 1, and use Lemma 3 to eliminate the assumption TC. □

Theorem 4 suggests the possibility of treating nested recursive definitions fully automatically with an extension of *recdef*: Use *recdef* to generate termination conditions, then check if they can be expressed as input-output relations, and if so then prove their inductive invariance. That would result in a method similar to the one developed by Giesl [5].

3.2 The Method of Alternative Specifications

A well-known method for proving termination of nested recursive equations was proposed by Moore [8]. Given a functional F as before, it asks the user to provide a concrete function h satisfying two conditions:

(M_1)	$\forall x.\ F\ h\ x = h\ x$ ("partial correctness");
(M_2)	$TC\,\rho\,h$, for some wellfounded relation ρ.

Theorem 5 *If h satisfies the conditions (M_1) and (M_2) then h is a fixpoint of F and $h = \mathsf{wfrec}\,\rho\,F$.*

Proof. The conditions (M_1) and (M_2) together with (13) imply

$$F\,(h \restriction \rho^{-1}x)\,x = h\,x$$

which is the equation (4) whose unique solution is $\mathsf{wfrec}\,\rho\,F$. □

Note that every function h defines an input-output predicate S_h that only it satisfies: $S_h\,x\,y$ iff $y = h\,x$. It is immediate to check that substituting S_h for S in the formulas (9) and (11) results in both cases in

$$\forall f\,x.\ f =_{\rho^{-1}x} h \longrightarrow F\,f\,x = h\,x.$$

Thus, inductive invariance of S_h and the restricted contraction condition associated with it are both equivalent to terminating at h.

3.3 Examples

Nested Zero The functional

$$G\,g\,n \equiv \mathsf{if}\ n = 0\ \mathsf{then}\ 0\ \mathsf{else}\ g\,(g\,(n-1))$$

corresponds to the second example in the Introduction. The termination condition $g\,n < n+1$ is an inductive invariant. (More precisely, the inductive invariant is defined by: $S\,n\,m$ iff $m \leq n+1$.) Indeed, one needs to check

$$\forall g\,n.\ (\forall m < n.\ g\,m < m+1) \longrightarrow G\,g\,n < n+1.$$

This is clearly true for $n = 0$, while for $n > 0$ it reads as

$$\forall g.\ (\forall m < n.\ g\,m < m+1) \longrightarrow g\,(g\,(n-1)) < n+1$$

and the conclusion of this formula follows by using the assumption twice.

The restricted contraction condition also holds: if $f\,i = g\,i$ holds for all $i < n$ and if $g\,i < i+1$ (the inductive invariant) holds for all i, then it follows immediately that $G\,f\,n = G\,g\,n$. But note again that plain contraction (without the additional assumption about g) is unprovable.

A direct proof by induction that G terminates at the constant zero function is also possible.

McCarthy's Ninety-One Function This classical example is a recursive definition of a function of type $\mathsf{nat} \Rightarrow \mathsf{nat}$, given by the functional

$$F\,f\,x \equiv \mathsf{if}\ x > 100\ \mathsf{then}\ x - 10\ \mathsf{else}\ f(f(x+11))$$

The termination conditions for F are

$$x \leq 100 \longrightarrow x + 11 \prec x \quad \text{and} \quad x \leq 100 \longrightarrow f(x+11) \prec x,$$

for a suitable wellfounded relation $\prec$. A relation that works is the ordering defined by $1 \succ 2 \succ 3 \succ \cdots \succ 99 \succ 100$ and $100 \succ 100 + i$ for all $i \geq 1$. This makes the first termination condition satisfied, while the second can be rewritten as

$$11 \leq x \leq 111 \longrightarrow x < f(x) + 11.$$

This expresses an obvious input-output relation and one can check that the relation is in fact an inductive invariant. We will check that an even stronger (but simpler) relation, namely

$$S\,x\,y \equiv x < y + 11 \tag{14}$$

is an inductive invariant for F. Thus, we need to prove

$$z < (F\,f\,z) + 11 \tag{15}$$

assuming

$$x < f(x) + 11 \tag{16}$$

holds for all $x \prec z$. For $z > 100$, the relation (15) reduces to $z < (z-10) + 11$, which is true. In the remaining case $z \leq 100$, the relation (15) rewrites as

$$z < f\,(f\,(z+11)) + 11. \tag{17}$$

Now we can assume that (16) holds for all $x > z$, since $x \prec z$ and $x > z$ are equivalent when $z \leq 100$. In particular, $z + 11 < f\,(z+11) + 11$ must hold, giving us $z < f\,(z+11)$. Instantiaing x with $f\,(z+11)$ in (16) is now legitimate and gives us $f\,(z+11) < f\,(f\,(z+11)) + 11$, so (17) follows by transitivity of $<$.

Takeuchi's Tarai Function This is the function of type $\mathsf{int} \times \mathsf{int} \times \mathsf{int} \Rightarrow \mathsf{int}$ defined by the functional

$$F\, f\, (x, y, z) \equiv \mathsf{if}\ x \leq y\ \mathsf{then}\ y\ \mathsf{else}\ f(f(x-1, y, z), f(y-1, z, x), f(z-1, x, y)).$$

Moore [8] proved the termination of F by showing that the function h defined by

$$h(x, y, z) = \mathsf{if}\ x \leq y\ \mathsf{then}\ y\ \mathsf{else}\ (\mathsf{if}\ y \leq z\ \mathsf{then}\ z\ \mathsf{else}\ x)$$

satisfies the conditions (M_1) and (M_2) above with respect to the wellfounded relation induced by a certain measure ρ. It turns out that predicates

$$\begin{aligned} S_1\,(x, y, z)\, u &\equiv x \leq y \longrightarrow u = y \\ S_2\,(x, y, z)\, u &\equiv y < x \wedge y \leq z \longrightarrow u = z \\ S_3\,(x, y, z)\, u &\equiv y < x \wedge z \leq y \longrightarrow u = x \end{aligned}$$

are all inductive invariants with respect to the relation ρ, so their conjunction is an inductive invariant as well. But this conjunction is clearly the predicate S_h satisfied by h only, so F terminates at h.

***HOL* Version of the While Combinator** In *Isabelle/HOL* [11], for any predicate $b\colon A \Rightarrow \mathsf{bool}$ and function $c\colon A \Rightarrow A$, the function $\mathsf{while}\ b\ c\colon A \Rightarrow A$ is defined as $\mathsf{wfrec}\, \rho\, W$, where

$$W\, f\, x \equiv \mathsf{if}\ b\, x\ \mathsf{then}\ f\,(c\, x)\ \mathsf{else}\ x$$

and ρ is defined by

$$x\, \rho\, y \equiv (\exists n.\ b\,(c^n\, x)) \wedge x = c\, y.$$

Given any two predicates $P\colon A \Rightarrow \mathsf{bool}$ and $Q\colon A \Rightarrow \mathsf{bool}$, one can check that the following relation S is an inductive invariant for W:

$$\begin{aligned} S\, x\, y \quad &= \quad P\, x \\ &\wedge \quad (\forall z.\ P\, x \wedge b\, z \longrightarrow P\,(c\, z) \wedge c\, z \prec z) \\ &\wedge \quad (\forall z.\ P\, x \wedge \neg(b\, z) \longrightarrow Q\, z) \\ &\longrightarrow \quad Q\, y \end{aligned}$$

The theorem `while_rule` supplied with *Isabelle/HOL* says precisely that $\mathsf{while}\, b\, c$ satisfies every relation S of this form. Thus, `while_rule` is essentially an instance of Lemma 3.

4 Partial Termination

Many interesting recursive programs terminate only on some subset of the input type, so we need to generalize the notion of termination to *termination on a given subset*. Here are the important definitions of the last two sections in their generalized form.

Definition 5 *Given a functional* $F\colon (A \Rightarrow B) \Rightarrow (A \Rightarrow B)$*, a wellfounded relation* ρ *on* A*, a subset* D *of* A*, and a relation* $S\colon A \Rightarrow B \Rightarrow \mathsf{bool}$*, we say that*

- F terminates on D at ϕ by means of ρ *iff*

$$\forall f\, x.\ x \in D \wedge f =_{D \cap (\rho^{-1}\, x)} \phi \longrightarrow F\, f\, x = \phi\, x; \tag{18}$$

- *a function* $f\colon A \Rightarrow B$ satisfies S on D *iff*

$$\forall x.\ x \in D \longrightarrow S\, x\, (f\, x); \tag{19}$$

- S *is an* inductive invariant of F on D by menans of ρ *iff*

$$\forall f\, x.\ x \in D \wedge (\forall y \in D.\ y\, \rho\, x \longrightarrow S\, y\, (f\, y)) \longrightarrow S\, x\, (F\, f\, x); \tag{20}$$

- *the* restricted contraction condition for F with respect to ρ, S, and D *is*

$$\forall f\, g\, x.\ x \in D \wedge f =_{D \cap (\rho^{-1}\, x)} g \wedge (\forall y \in D.\ S\, y\, (g\, y)) \longrightarrow F\ f\ x = F\ g\ x. \tag{21}$$

Now we restate Lemma 1, Lemma 3 and Theorem 3 in this general setting. Generalizations are direct, but note a twist in the uniqueness statement that now relates termination at two subsets. The proofs, being rather straightforward extensions of those given in previous sections are omitted.

Lemma 4 *(a) If* F *terminates at* ϕ *on* D *then* ϕ *satisfies the restricted fixpoint equation* $\phi =_D F\, \phi$*. Moreover, if the termination is by means of* ρ*, then* $\phi =_D \mathsf{wfrec}\, \rho\, F$*.*

(b) If $D_1 \subseteq D_2$ *and* F *terminates at* ϕ_i *on* D_i *(*$i = 1, 2$*), then the restriction of* ϕ_1 *terminates on* D_1 *if and only if* $\phi_1 =_{D_1} \phi_2$*.*

□

Lemma 5 $\mathsf{wfrec}\, \rho\, F$ *satisfies on* D *all inductive invariants of* F *associated with* D *and* ρ*.* □

Theorem 6 *Suppose the restricted contraction condition (21) is satisfied and* S *is an inductive invariant of* F *associated with* D *and* ρ*. Then* F *terminates on* D *at* $\mathsf{wfrec}\, \rho\, F$*.* □

5 Example: The BDD Apply function

In this section we survey the proof of termination of the imperative BDD program Apply. This proof has been verified in *HOL* and reported on in [6]. Our *HOL* proof does not use the *recdef* mechanism, but is instead based on the *HOL* version of Theorem 6.

Binary decision diagrams (BDDs) are a widely used representation of boolean functions. Intuitively, a BDD is a finite rooted directed acyclic graph in which

every node except special nodes $TrueNode$ and $FalseNode$ is labeled by a *variable* and has two children: *low* and *high*. Special nodes represent the constant boolean functions, and the function f_u represented by any other node u is defined recursively by $f_u = \mathsf{if}\ x\ \mathsf{then}\ f_l\ \mathsf{else}\ f_h$, where x is the variable associated with u and l, h are its left and right children respectively. Bryant [4] originally proved that every function is represented by a unique reduced ordered BDD, where *reduced* means that no two nodes represent the same function, and *ordered* means that variable names are totally ordered and that every node's variable name precedes the variable names of its children. Efficient BDD packages implement reduced ordered BDDs. An abstract, but detailed presentation of such a package of programs is given in [1]. Our work [6] contains a *HOL* model of a significant part of the package. Referring to these papers for more detail, we will now describe just the minimum required to define the Apply program.

The global state used by any BDD package contains a pool of BDD nodes. We assume there is an abstract type Node representing node addresses, and a type Var of variables. A primitive function $active\colon \mathsf{Node} \Rightarrow \mathsf{bool}$ indicates the presence of a node in the current state, and the accessor functions $var, low, high$ take a node as an argument and return the associated variable and children. What, if anything, these functions return if the argument is not an active node, is left unspecified. For simplicity we will assume that Var is the type of natural numbers whose natural ordering corresponds to the ordering of variables needed to implement the concept of ordered BDDs.

The BDD routine Mk takes a variable x and two nodes l, h as inputs and returns a node u such that $var(u) = x$, $low(u) = l$, and $high(u) = h$. If a node with these three attributes already exists in the state, the state is left unchanged; otherwise Mk adds u to the state. We gloss over the details how Mk tests whether it needs to add a node to the existing state and the possibility that Mk can raise an out-of-memory exception.

The crucial routine Apply takes a binary boolean operation *op* and two nodes u and v, and returns a node w which represents the boolean function f_w specified by

$$\forall x.\ f_w\, x = op(f_u\, x, f_v\, x). \tag{22}$$

A recursive declaration of Apply is given in pseudocode in Figure 1.

Clearly, the variable *op* is of little significance for termination of Apply. Assuming *op* is constant, we can think of Apply as being a function of type $\mathsf{Node} \times \mathsf{Node} \times \mathsf{State} \Rightarrow \mathsf{State} \times \mathsf{Node}$.[2] Recursion makes Apply one of the most complicated programs in the package. Pondering the algorithm in Figure 1, one realizes that even a hand proof of termination of Apply requires effort. The ultimate reason for termination is clear: in an ordered BDD (and only those we would like to consider), the level (that is, the *var* value) goes down when passing to children nodes, so in all recursive calls of Apply the level decreases either for both node arguments, or decreases for the "higher", while the other stays the same. Thus, in order to prove that the arguments decrease in recursive calls, it is

[2] In [6], we show how to use monadic interpretation to hide "state threading" and translate imperative programs to visibly equivalent *HOL* counterparts.

```
1    APPPLY[T](op, u, v) =
2    if u, v ∈ {TrueNode, FalseNode} then op(u, v)
3    else if var(u) = var(v) then
4        w ⟵ MK(var(u), APPLY(op, low(u), low(v)), APPLY(op, high(u), high(v)))
5    else if var(u) < var(v) then
6        w ⟵ MK(var(u), APPLY(op, low(u), v), APPLY(op, high(u), v))
7    else
8        w ⟵ MK(var(v), APPLY(op, u, low(v)), APPLY(op, u, high(v)))
9    return w
```

Figure 1: The program APPLY, as in [1], omitting the memoization part, inessential for the purpose of proving termination. The global variable T is the table of BDD nodes.

necessary to work with a restricted set of states, described by a predicate goodSt that needs to be preserved by APPLY. A workable invariant goodSt asserts that the associated BDD to each active node is ordered and reduced. Clearly, we cannot expect termination for all input-"good state" pairs. We obviously need to add at least the restriction that the two input nodes be present in the input state. These restrictions define a subset D of $\mathsf{Node} \times \mathsf{Node} \times \mathsf{State}$ on which we can reasonably expect termination by means of the wellfounded relation defined by the measure that associates to an input-state triple (u, v, T) the maximum of the values $var(u)$, $var(v)$ in T.

Next we need to deal with nesting. Nesting is not immediately seen in Figure 1 becasue the state is not explicitly mentioned in the program text, being thus an extra hidden argument. Consider the line 6; in expanded form, this piece of code could read like this:

```
6₁    l_u ⟵ low(u)
6₂    h_u ⟵ high(u)
6₃    x ⟵ var(u)
6₄    l ⟵ APPLY(op, l_u, v)
6₅    h ⟵ APPLY(op, h_u, v)
6₆    w ⟵ MK(x, l, h)
```

If we made the state explicit, these lines would look as follows, with primes denoting the appropriate modifications of functions representing programs:

```
6₁    (l_u, T₁) ⟵ low'(u, T)
6₂    (h_u, T₂) ⟵ high'(u, T₁)
6₃    (x, T₃) ⟵ var'(u, T₂)
6₄    (l, T₄) ⟵ APPLY'(op, l_u, v, T₃)
6₅    (h, T₅) ⟵ APPLY'(op, h_u, v, T₄)
6₆    (w, T₆) ⟵ MK'(x, l, h, T₅)
```

exposing nesting in the definition of h:

$$h = \mathsf{fst}(\textsc{Apply}'(op, h_u, v, \mathsf{snd}(\textsc{Apply}'(op, l_u, v, T_3)))).$$

To prove termination, we need to find input-output properties of Apply that are sufficient to show that the measure goes down in each recursive call, and then to prove that these properties are inductive invariants of the functional defining Apply. It turns out that the input-output predicate S defined by

$$\begin{aligned} S\,(u, v, T)\,(w, T') \quad &\equiv \quad T' \text{ contains } T \\ &\wedge \quad w \text{ is active in } T' \\ &\wedge \quad var'(w, T') \leq var'(u, T), var'(v, T) \end{aligned} \tag{23}$$

is such an inductive invariant. We have carried out the proof of inductive invariance and the proof of the corresponding restricted contraction condition in *Isabelle/HOL* [6].

6 Fixpoint Operators in HOL

Is there a canonical way to associate in *HOL* a partial function with every functional $F\colon (A \Rightarrow B) \Rightarrow (A \Rightarrow B)$? Given F, we would like to define a natural *domain* D (subset of A) associated with it, and a function $\phi\colon A \Rightarrow B$ which satisfies the restricted fixpoint equation $\phi =_D F\,\phi$. We will give two answers to this question. The first is proveded by Theorem 7 below saying that there exists a maximal subset of A on which F terminates in the sense of Definition 5. The second answer comes from modeling the standard iterative fixpoint construction in *HOL*. Finally, we will show that the two fixpoints are the same in the case when (vaguely speaking) the calling relation associated with F is finitary in the sense that for every $x \in A$ there are only finitely many y such that $y \prec x$.

We should note that proofs of results of this section have not been formally verified yet.

6.1 Largest Domain and Natural Fixpoint

Definition 6 *We say that a subset D of A is a* domain for F *if F terminates on D. When F terminates on D by means of ρ, we will say that the pair (D, ρ) is a* germ *of F.*

Lemma 6 *Suppose (D, ρ) is a germ of F and D' is a downward closed subset of D with respect to ρ. Then (D', ρ') is a germ of F, where ρ' is the restricton of ρ on D'.*

Proof. Straightforward. □

Theorem 7 *There exists a largest domain for F.*

Proof. Consider the ordering on the set of germs given by: $(D, \rho) \preceq (D', \rho')$ iff $D \subseteq D'$ and the restriction of ρ' on D is equal to ρ. For every chain $\mathcal{C}$ in this ordering, consider the pair $(\cup D, \cup \rho)$, where the unions are taken over all elements of the chain $\mathcal{C}$. It is easy to see that this pair is a germ and an upper bound for $\mathcal{C}$. By Zorn's Lemma, there exists a maximal germ, say (D, ρ). We claim that D is the largest domain for F.

Assuming the contrary of the claim, suppose (D', ρ') is germ of F such that D' is not contained in D. Now, there exists $z \in D' \setminus D$ such that all smaller elements (with respect to ρ') than z in D' belong to D as well. In view of Lemma 6, it is no loss of generality to assume that $D' = E \cup \{z\}$, where $E \subseteq D$ and z is the greatest element in D'.

Let $\psi = \mathsf{wfrec}\, \rho\, F$ and $\psi' = \mathsf{wfrec}\, \rho'\, F$. By Lemma 4(a), F terminates at ψ on D and terminates at ψ' on D'. By Lemma 6, E is a domain for F, as downward closed in D'. Now E is a subdomain of both D and D', so by Lemma 4(b), we obtain that the restrictions of ψ and ψ' on E coincide.

Define the function ϕ by

$$\phi\, x \equiv \mathsf{if}\ x \in D\ \mathsf{then}\ \psi\, x\ \mathsf{else}\ \psi'\, x.$$

Since ψ and ψ' have equal restrictions on $E = D \cap D'$, we have that $\phi =_D \psi$ and $\phi =_{D'} \psi'$. By Lemma 4(b), F terminates at ϕ on both D and D'.

We want to prove that F terminates at ϕ on $D \cup D' = D \cup \{z\}$ by means of σ, where σ is the (obviously wellfounded) relation $\rho \cup \rho'$. This amounts to checking that

$$f =_{(D \cup \{z\}) \cap (\sigma^{-1} x)} \phi \longrightarrow F\, f\, x = \phi\, x \tag{24}$$

holds for every f and every $x \in D \cup \{z\}$. Consider first the case when $x \in D$. Then $(D \cup \{z\}) \cap (\sigma^{-1} x) = D \cap \rho^{-1} x$ and (24) follows since F terminates at ϕ on D. In the remaining case when $x = z$, we have $(D \cup \{z\}) \cap (\sigma^{-1} x) = E$ and (24) follows since F terminates at ϕ on D'. □

Definition 7 *The largest domain for F will be denoted D_F and called the* termination domain *of F. The* termination fixpoint νF *is defined by*

$$\nu F\, x \equiv \mathsf{if}\ x \in D_F\ \mathsf{then}\ \mathsf{wfrec}\, \rho\, F\ \mathsf{else}\ \mathsf{Arb},$$

where ρ is any wellfounded relation such that F terminates on D by means of ρ.

6.2 Finite Termination and Iterative Fixpoint

Standard operational semantics suggests that the function recursively defined by the functional $F\colon (A \Rightarrow B) \Rightarrow (A \Rightarrow B)$ is the limit of the sequence

$$\phi_0 = \mathsf{Arb}, \phi_1, \phi_2, \ldots$$

where $\phi_{n+1}\, x = F\, \phi_n\, x$. But what is the limit partial function ϕ? An obvious candidate would have the domain consisting of all x for which the sequence

$$\phi_0\, x, \phi_1\, x, \phi_2\, x, \ldots$$

stabilizes, and for such x define $\phi\, x$ as the limit value of the sequence. However, the fixpoint equation does not have to hold for this (too large) domain, as demonstrated by the example

$$F\, f\, x \equiv \mathsf{if}\ \mathit{constant}\, f\ \mathsf{then}\ x\ \mathsf{else}\ x - 1.$$

The domain on which the stabilization limit function satisfies the fixpoint equation is best defined as the union of an increasing sequence of subsets. The subsets Δ_n of A defined inductively by

$$\Delta_0 = \emptyset \qquad \Delta_{n+1} = \{x \,|\, \forall f.\ f =_{\Delta_n} \phi_n \longrightarrow F\, f\, x = \phi_{n+1}\, x\}. \tag{25}$$

Lemma 7 *$\Delta_n \subseteq \Delta_{n+1}$ and $\phi_{n+1} =_{\Delta_n} \phi_n$, for every n.*

Proof. Both statements are trivially true for $n = 0$. Proceeding by induction, for $\Delta_n \subseteq \Delta_{n+1}$ we need to prove $F\, f\, x = F\, \phi_n\, x$ assuming $f =_{\Delta_n} \phi_n$ and $x \in \Delta_n$. We will prove $F\, f\, x = F\, \phi_{n-1}\, x$ and $F\, \phi_n\, x = F\, \phi_{n-1}\, x$. The second equation follows from the definition of Δ_n and the induction hypothesis $\phi_n =_{\Delta_{n-1}} \phi_{n-1}$. The first equation similarly follows from the definition of Δ_n and $f =_{\Delta_{n-1}} \phi_{n-1}$. It remains just to prove this last relation. It follows by transitivity of $=_{\Delta_{n-1}}$ from the relations $f =_{\Delta_{n-1}} \phi_n$ and $\phi_n =_{\Delta_{n-1}} \phi_{n-1}$. The second of these two formulas is an already used induction hypothesis, while the first is a consequence of the induction hypothesis $\Delta_{n-1} \subseteq \Delta_n$.

To prove $\phi_{n+1} =_{\Delta_n} \phi_n$, assume $x \in \Delta_n$; the goal is $\phi_{n+1}\, x = \phi_n\, x$. By definition of ϕ_n, we have $\phi_n\, x = F\, \phi_{n-1}\, x$. We have proved $\Delta_n \subseteq \Delta_{n+1}$; thus, $x \in \Delta_{n+1}$ and so, by definition of ϕ_{n+1}, we have $\phi_{n+1}\, x = F\, \phi_n\, x$. Our goal can now be rewritten as $F\, \phi_n\, x = F\, \phi_{n-1}\, x$. In follows from the definition of Δ_n and the induction hypothesis $\phi_n =_{\Delta_{n-1}} \phi_{n-1}$. □

Definition 8 *For a given functional F, define its* iterative domain Δ_F *as the union of the subsets Δ_n above. Define the* iterative fixpoint μF *of F by*

$$\mu F\, x = \begin{cases} \phi_n\, x & \text{if } x \in \Delta_n \\ \mathsf{Arb} & \text{if } x \notin \Delta_F \end{cases}$$

Theorem 8 *F terminates at μF on Δ_F. In particular, μF satisfies the restricted recursion equation $\mu F =_{\Delta_F} F\,(\mu F)$.*

Proof. Take the wellfounded relation ρ such that $x\, \rho\, y$ holds if and only if $x \in \Delta_n$ and $y \in \Delta_{n+1}$ for some n. Thus, to prove that F terminates at μF on Δ by means of ρ, we need to check that

$$f =_{\Delta_n} \mu F \longrightarrow F\, f\, x = \mu F\, x \tag{26}$$

holds for all f and $x \in \Delta_{n+1}$. We have $\mu F\, x = \phi_{n+1}\, x$ (since $x \in \Delta_{n+1}$) and also $\mu F =_{\Delta_n} \phi_n$. Thus, (26) can be written as

$$f =_{\Delta_n} \phi_n \longrightarrow F\, f\, x = \phi_{n+1}\, x,$$

which is true by definition of Δ_{n+1}. □

6.3 Relationship Between νF and μF

First, we have an immediate consequence of Definition 7 and Theorem 8.

Corollary 1 *$\Delta_F \subseteq D_F$ and $\mu F =_{\Delta_F} \nu F$.* □

In many cases the domains D_F and Δ_F are actually equal. To explain such situations, we need a more restrictive notion of termination than given in Definition 5.

Definition 9 *We say that F is* finitely terminates *at ϕ on D by means of a wellfounded relation ρ if the following condition is satisfied: for every $x \in D$ there exists a finite subset $D(x)$ of $D \cap (\rho^{-1}x)$ such that*

$$\forall f\, x.\ x \in D \wedge f =_{D(x)} \phi \longrightarrow F\, f\, x = \phi\, x. \tag{27}$$

Theorem 9 *If F finitely terminates on D by means of ρ, then $D \subseteq \Delta_F$ and $\mu F =_D \mathsf{wfrec}\, \rho\, F$.*

Proof. Denote $\psi = \mathsf{wfrec}\, \rho\, F$ and $\phi = \mu F$. Let ρ' be the relation defined by

$$x\, \rho'\, y \equiv x\, \rho\, y \wedge x \in D(y).$$

Clearly, ρ' is wellfounded and F terminates at ψ by means of ρ'. We claim that every element of D has finite height, where the height of x is defined as the supremum of lengths of downward chains (under ρ') starting at x. Indeed, since there are only finitely many elements smaller (under ρ') than x, the existence of arbitrarily long downward chains would by König's Lemma imply the existence of an infinite downward chain, which would contradict wellfoundedness of ρ'.

Let D_n denote the subset of D consisting of all its elements of height n or less. (Thus, $D_0 = \emptyset$ and D_1 is the set of minimal elements under ρ'.) It will suffice to prove

$$D_n \subseteq \Delta_n \quad \text{and} \quad \phi =_{D_n} \psi, \tag{28}$$

which we do by induction on n. The case $n = 0$ is trivial.

With (28) as the induction hypothesis, we need to prove that for every $x \in D_{n+1}$

$$x \in \Delta_{n+1} \quad \text{and} \quad \phi\, x = \psi\, x. \tag{29}$$

Using the induction hypothesis, we can strengthen our assumption to that the height of x is exactly $n + 1$. Note that now we have $D(x) \subseteq D_n$ because every $y \in D(x)$ has height at most n. By definitions of Δ_{n+1} and ϕ_{n+1}, for the first goal in (29) it suffices to prove $F\, f\, x = F\, \phi_n\, x$, with an additional assumption $f =_{\Delta_n} \phi_n$. This additional assumption implies $f =_{D(x)} \phi$ because $\phi =_{\Delta_n} \phi_n$ and $D(x) \subseteq D_n \subseteq \Delta_n$. Using the induction hypothesis $\phi =_{D_n} \psi$, we then obtain $f =_{D(x)} \psi$ and $\phi_n =_{D(x)} \psi$, and using (27) twice deduce the desired equality $F\, f\, x = F\, \phi_n\, x$. Now we know $x \in \Delta_{n+1}$, so $x \in \Delta_F$, and so $\phi\, x = F\, \phi\, x$, by Theorem 8. On the other hand, $\phi =_{D(x)} \psi$ implies by (27) $F\, \phi\, x = \psi\, x$, finishing the proof of $\phi\, x = \psi\, x$. □

Since μF clearly finitely terminates on Δ_F, Theorems 8 and 9 have the following consequence.

Corollary 2 *$D_F = \Delta_F$ if and only if F finitely terminates on D_F.* □

Note that $D_F = \Delta_F$ in Corollary 2 can be replaced with $\nu F = \mu F$.

An example when termination does not mean finite termination is given by this recursive definition of $f\colon \mathsf{nat} \times \mathsf{nat} \Rightarrow \mathsf{bool}$:

$$\begin{aligned} f(0,0) &= \forall i > 0.\ f(i,0) \\ f(0,j) &= \mathsf{True} \ \text{ if } j \neq 0 \\ f(i,j) &= f(i-1, j+1) \ \text{ if } i \neq 0 \end{aligned}$$

Take the wellfounded relation ρ on $\mathsf{nat} \times \mathsf{nat}$ determined by the call graph of this recursive definition. The minimal elements are $(0, j)$, the maximal element is $(0, 0)$, and the remaining elements are organized into chains of length n going from $(0, n)$ up to $(n, 0)$, for every $n > 0$. The contraction condition holds on the whole input type, so the natural domain is $D_F = \mathsf{nat} \times \mathsf{nat}$ (and we can prove that $\mu F(i, j) = \mathsf{True}$). On the other hand, we have $\Delta_n = \{(i, j) \mid i < n\} \setminus \{(0, 0)\}$ and so the iterative domain $\Delta_F = \mathsf{nat} \times \mathsf{nat} \setminus \{(0, 0)\}$ is strictly smaller than D_F.

7 Conclusion and Related Work

We have described and implemented techiques that can be used to justify recursive definitions in higher order logic when direct checking of termination conditions is not possible. For example, the *contraction condition* can be easily provable even in cases when termination conditions are impossible to extract directly from the program text. In the cases where termination conditions are difficult to prove due to nested recursive calls, we have shown how using suitable *inductive invariants* can make these conditions easier to establish. Furthermore, inductive invariants in combination with the corresponding *restricted contraction condition* can be applied when we have to deal with both nested recursion and problematic extraction of termination conditions. The BDD Apply algorithm is a substantial example that presented us with all these difficulties and that we were able to handle with our techniques.

Contraction conditions are a standard way of proving fixpoint theorems (*à la* Banach). In the context of higher order logic, this technique has been used by Matthews [7] to support recursive function definitions over types with coinductive structure.

The challenge of justifying nested recursive definitions has attracted a great deal of research, in various formal systems; [12, 8, 15, 5, 3] are but a few examples of interesting case studies and general methods. They are all related to our work for the simple reason that inductive invariants (even if not recognized as such) are at the core of most of the known termination proof of nested recursion. Our contribution is in presenting this common pattern of reasoning as a general method, where much can be proved once and for all, leaving the user only with clear proof obligations specific to the problem at hand. The obligations are: (1)

to find a suitable inductive invariant (in addition, and analogous to finding a suitable wellfounded relation) (2) to prove that it is an inductive invariant; and (3) to prove the termination conditions (with the additional assumption that the inductive invariant holds).

Often the nested termination conditions themselves are inductive invariants, so the first attempt would be to try them. The method described by Giesl [5] follows a similar pattern: heuristic generation of "induction lemmas" that correspond to nested termination conditions, then proving their "partial correctness", which amounts to inductive invariance. High level of automation, which is the main virtue of Giesl's method, is also its limitation: There are complicated cases, for example the APPLY function above, or the unification algorithm (as in [15]), that this approach would have difficulties with. Compared with Giesl's method, ours is more general and, as a consequence of that greater generality, our proofs are considerably simpler than those in [5].

Slind [15] has demonstrated that his *recdef* package can be used for nested recursive definitions, even as complex as the unification algorithm. The method here is to start with the conditional recursion theorem and the corresponding conditional induction theorem, both automatically derived by *recdef*, then combine them to eventually get rid of the termination conditions. The combination process, however, requires a substantial effort from the user's part that in the presented examples always includes proving some inductive invariant (derived from the nested termination conditions). The impression is that our conceptually simpler method is at least equally powerful and efficient.

Giesl and Slind [5, 15] make a point that, contrary to the previous understanding, it is generally possible to prove termination of functions defined by nested recursion without simultaneously proving their correctness/specification. Note, however, the vagueness in the notions of correctness and specification. The fact is that some form of specification is invariably being used, namely the inductive invariant! And this is all quite natural: the specification of our example algorithm APPLY[3] can hardly be used to prove its termination; what makes termination proof possible is inductive invariance of the much simpler predicate given in (23).

Modeling program termination in HOL can be done in several ways. Müller and Slind have surveyed the topic in [10]. The most accurate modeling can be done in HOLCF [9], the HOL version of domain theory. We have adopted the simplest, most often taken and often the most convenient approach where partial functions are modeled as total functions taking an arbitrary value on arguments outside the specified domain of definition. Modeling such programs as partial functions raises the question of defining a natural fixpoint operator on functionals, and we have shown that the "most general" fixpoint can be defined in HOL. We have also shown the existence of the "most general" fixpoint under the constraint corresponding to finite depth of recursive calling, (cf. [2]). Assessing usefulness of this concept requires further study. In particular, it would

[3]The correctness of APPLY is expressed by the equation (22), which involves the interpretation function that associates boolean functions to BDD nodes.

be interesting to find conditions under which the HOL fixpoint of a functional corresponds to the fixpoint of the corresponding functional in HOLCF.

References

[1] H. R. Andersen. An Introduction to Binary Decision Diagrams. Internet, September 1996.

[2] A. Balaa and Y. Bertot. Fonctions récursives générales par itération en théorie des types. In *Proceedings of Journées francophones des langages applicatifs, JFLA'02*. INRIA, January 2002.

[3] A. Bove and V. Capretta. Modelling general recursion in type theory, September 2002.

[4] R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.

[5] J. Giesl. Termination of nested and mutually recursive algorithms. *Journal of Automated Reasoning*, 19(1):1–29, 1997.

[6] S. Krstić and J. Matthews. Verifying BDD algorithms through monadic interpretation. In A. Cortesi, editor, *Verification, Model Checking, and Abstract Interpretation, Third International Workshop, VMCAI 2002.*, pages 182–195. Springer LNCS, Volume 2294, April 2002.

[7] J. Matthews. Recursive function definition over coinductive types. In Y. Bertot, G. Dowek, A. Hirschowitz, C. Paulin, and L. Théry, editors, *Proc. Intl. Conf. on Theorem Proving in Higher Order Logics (TPHOLS)*, Lecture Notes in Computer Science, pages 73–90. Springer LNCS 1690, September 1999.

[8] J. S. Moore. A mechanical proof of the termination of Takeuchi's function. *Information Processing Letters*, 9:176–181, 1979.

[9] O. Müller, T. Nipkow, D. von Oheimb, and O. Slotosch. HOLCF = HOL + LCF. *Journal of Functional Programming*, 9:191–223, 1999.

[10] O. Müller and K. Slind. Treating partiality in a logic of total functions. *The Computer Journal, 40(10)*, 1997.

[11] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.

[12] L. C. Paulson. Proving termination of normalization functions for conditional expressions. *Journal of Automated Reasoning*, 2(1):63–74, 1986.

[13] K. Slind. Function definition in higher order logic. In J. von Wright, J. Grundy, and J. Harrison, editors, *Proc. Intl. Conf. on Theorem Proving in Higher Order Logics (TPHOLS)*, Lecture Notes in Computer Science, pages 381–397. Springer LNCS 1125, August 1996.

[14] K. Slind. *Reasoning about Terminating Functional Programs*. PhD thesis, Institut für Informatik, Technische Universität München, 1999.

[15] K. Slind. Another look at nested recursion. In M. Aagaard and J. Harrison, editors, *Proc. Intl. Conf. on Theorem Proving in Higher Order Logics (TPHOLS)*, Lecture Notes in Computer Science, pages 498–518. Springer LNCS 1869, August 2000.

[16] G. Winskel. *The Formal Semantics of Programming Languages: an Introduction*. MIT Press, 1993.