Automated Termination Proofs with Measure Functions*

Jürgen Giesl
FB Informatik, Technische Hochschule Darmstadt,
Alexanderstr. 10, 64283 Darmstadt, Germany
Email: giesl@inferenzsysteme.informatik.th-darmstadt.de

Abstract. This paper deals with the automation of termination proofs for *recursively defined* algorithms (i.e. algorithms in a pure functional language). Previously developed methods for their termination proofs either had a low degree of automation or they were restricted to one single fixed *measure function* to compare data objects. To overcome these drawbacks we introduce a calculus for automated termination proofs which is able to handle arbitrary measure functions based on polynomial norms.

1 Introduction

Termination of algorithms is a central problem in software development. For an automation of program verification, termination proofs have to be performed mechanically, i.e. without human support. Of course, as the halting problem is undecidable, there is no procedure to prove or disprove termination of *all* algorithms.

In this paper we focus on *recursively defined* algorithms, i.e. algorithms in a pure (eager) functional language without iterative loops. An algorithm $f(x)$ terminates, if there is a *well-founded*[1] relation such that in each recursive call $f(t)$, the argument t is smaller than the corresponding input x.

While most work on the automation of termination proofs has been done in the areas of *term rewriting systems* (e.g. [Lan79], [Der87], [BL87], [Ste92]) and *logic programs* (e.g. [Plü90], [DSF93]), a well-known method for automated termination proofs of LISP functions has been implemented in the NQTHM system of *R. S. Boyer* and *J S. Moore* [BM79]. For such an algorithm $f(x)$ one has to prove that in each recursive call $f(t)$ a given *measure* is decreased. More precisely, Boyer and Moore use a *measure function* m which maps data objects to natural numbers. Then their theorem prover has to verify that the number $m(t)$ is smaller than the number $m(x)$.

For this proof the user has to supply so-called *induction lemmata*. An induction lemma points out that under a certain hypothesis Δ some operation drives some measure down. For the termination proof of an algorithm $f(x)$ with a recursive call $f(t)$, Boyer and Moore's theorem prover looks for an induction

* Appeared in *Proceedings of the 19th Annual German Conference on Artificial Intelligence*, Bielefeld, Germany, Springer–Verlag, LNAI 981, 1995.

[1] A relation $\prec$ is *well-founded* if there exists no infinite descending chain $\ldots \prec x_2 \prec x_1$.

lemma with a conclusion saying that the measure $m(t)$ is smaller than $m(x)$, i.e. an induction lemma of the form

$$\Delta \rightarrow m(t) < m(x).$$

The prover now only has to verify the hypothesis Δ of the induction lemma under the assumptions of f's case analysis. This is usually quite simple, i.e. the termination proof for an algorithm f often succeeds using case analysis and propositional reasoning only.

Boyer and Moore's technique for proving termination is very powerful, as the methods works for *arbitrary measure functions* m. It has also been adapted for termination proofs in other areas, e.g. an adaption of this technique for termination proofs of general[2] conditional term rewriting systems can be found in [BL93].

But a disadvantage of Boyer and Moore's induction lemma technique is a low degree of automation. Induction lemmata which specify that certain operations decrease measures have to be *formulated by the user* of the system. So the human has to find the idea why an algorithm terminates. Moreover, to ensure the soundness of the user supplied induction lemmata, these lemmata have to be verified by the system. This verification may be hard, as in general an *inductive proof* is needed.

Therefore an alternative method for automated termination proofs of algorithms has been developed by *C. Walther* [Wal88], [Wal94]. With this method a certain class of induction lemmata can be synthesized *automatically* and the soundness of these induction lemmata is *guaranteed by construction*. So compared to the technique of Boyer and Moore the advantage of Walther's method is a much higher degree of automation.

But a drawback of Walther's method is that it is restricted to *one single fixed* measure function, viz. the so-called *size measure function*, while the system of Boyer and Moore can use arbitrary measure functions m for termination proofs. Using the size measure function, data objects x are compared by their *size* $|x|_{\#}$, i.e. lists are compared by their length, trees are compared by the number of their nodes etc. Although Walther's method is successful for many algorithms, there are numerous interesting algorithms for whose termination proof a measure function different from *size* is needed and for these algorithms Walther's approach always fails.

In this paper we therefore present an extension of Walther's method to *arbitrary measure functions* (satisfying certain requirements). Walther's technique consists of a calculus (the so-called *estimation calculus*) to prove that the size of

[2] *Recursively defined* algorithms can be regarded as a special type of (conditional) term rewriting systems with an *innermost evaluation* strategy. But due to their special form and due to this evaluation strategy it is possible to use a different approach for termination proofs of algorithms than it is necessary for term rewriting systems [Gie95b]. For instance, for algorithms it is sufficient to compare the input *arguments* with the *arguments* in the recursive calls, while for term rewriting systems left and right hand sides of *all rules* have to be compared.

an object x is smaller than the size of another object y (i.e. the calculus proves statements of the form $|x|_{\#} < |y|_{\#}$). The central idea of our extension is to adapt a technique from the area of term rewriting systems [Lan79] and to replace the estimation calculus by a technique for proving *polynomial inequalities* automatically. While the estimation calculus only works for the *size measure function*, our approach allows the use of any measure function based on so-called *polynomial norms. For any given measure function* our method synthesizes sound induction lemmata and proves termination of algorithms *fully automatically.* So the presented technique combines the high degree of automation of Walther's technique with the powerful generality of Boyer and Moore's method.

The remainder of the paper is organized as follows: In Section 2 we introduce our approach with two examples. Instead of Walther's estimation calculus we define a calculus for arbitrary polynomial norms in Section 3. Section 4 shows how to derive sound induction lemmata. In Section 5 we comment on some refinements of the proposed method and Section 6 ends with a conclusion and an outlook on future work.

2 Termination Proofs with Arbitrary Measure Functions

In this section we illustrate how techniques for proving polynomial inequalities can be used for termination proofs of algorithms. As an example regard the following normalization algorithm for conditional expressions which constitutes one of the hardest termination problems in [BM79]. To represent conditional expressions we use a data structure cexpr whose objects are built with the *constructors* atomic and cond. The nullary function atomic represents an atomic expression and the function cond : cexpr $\times$ cexpr $\times$ cexpr $\rightarrow$ cexpr is used to represent a conditional. Here, $\mathsf{cond}(x, y, z)$ stands for "*if* x *then* y *else* z".

The aim of the algorithm normalize is to transform a conditional expression into an equivalent expression whose conditions are atomic. The algorithm is based on the fact that the expression "*if (if u then v else w) then y else z*" is equivalent to "*if u then (if v then y else z) else (if w then y else z)*".

function normalize $(x$: cexpr$)$: cexpr $\Leftarrow$
 if $x = \mathsf{atomic}$ *then* x
 if $x = \mathsf{cond}(\mathsf{atomic}, y, z)$ *then* $\mathsf{cond}(\mathsf{atomic}, \mathsf{normalize}(y), \mathsf{normalize}(z))$
 if $x = \mathsf{cond}(\mathsf{cond}(u, v, w), y, z)$ *then* $\mathsf{normalize}(\mathsf{cond}(u, \mathsf{cond}(v, y, z), \mathsf{cond}(w, y, z)))$

To prove the termination of this algorithm[3], there has to be a well-founded relation such that the arguments in the recursive calls are smaller than the corresponding inputs. Using a measure function m one therefore has to prove

$$m(y) < m(\mathsf{cond}(\mathsf{atomic}, y, z)), \tag{1}$$

$$m(z) < m(\mathsf{cond}(\mathsf{atomic}, y, z)), \tag{2}$$

$$m(\mathsf{cond}(u, \mathsf{cond}(v, y, z), \mathsf{cond}(w, y, z))) < m(\mathsf{cond}(\mathsf{cond}(u, v, w), y, z)). \tag{3}$$

[3] To ease readability we have used a formulation of the algorithm with *pattern matching* instead of *selectors* (or *destructors*). For the handling of selectors the reader is referred to Section 3.

The method of Walther compares objects by their *size*, where the size of a conditional expression is the number of occurring conditionals. So the size measure function maps atomic expressions to the number 0 (i.e. $|\mathsf{atomic}|_{\#} = 0$) and for conditionals we have $|\mathsf{cond}(x, y, z)|_{\#} = 1 + |x|_{\#} + |y|_{\#} + |z|_{\#}$. Using the size measure function the inequalities (1) and (2) can be verified, as $\mathsf{cond}(\mathsf{atomic}, y, z)$ contains one more occurrence of cond than y or z. But the third inequality (3) is not true for the size measure, as we have

$$|\mathsf{cond}(u, \mathsf{cond}(v, y, z), \mathsf{cond}(w, y, z))|_{\#} = 3 + |u|_{\#} + |v|_{\#} + |w|_{\#} + 2|y|_{\#} + 2|z|_{\#}$$

which is *greater* than

$$|\mathsf{cond}(\mathsf{cond}(u, v, w), y, z)|_{\#} = 2 + |u|_{\#} + |v|_{\#} + |w|_{\#} + |y|_{\#} + |z|_{\#}.$$

So the size measure function cannot be used for the termination proof of normalize and therefore Walther's method fails.

When examining several methods to prove termination of normalize, *L. Paulson* suggested to use the following mapping $|.|$ from objects of the data structure cexpr to natural numbers [Pau86]. Atomic expressions should be mapped to the number 1 (i.e. $|\mathsf{atomic}| = 1$) and the mapping for conditionals is $|\mathsf{cond}(x, y, z)| = |x|(1 + |y| + |z|)$. So for example $|\mathsf{cond}(\mathsf{atomic}, \mathsf{atomic}, \mathsf{atomic})| = 1(1 + 1 + 1) = 3$. Using $|.|$ as a measure function, the inequalities (1) - (3) become

$$\begin{aligned} |y| &< 1 + |y| + |z|, \\ |z| &< 1 + |y| + |z|, \\ |u|(1 + (|v| + |w|)(1 + |y| + |z|)) &< |u|(1 + |v| + |w|)(1 + |y| + |z|). \end{aligned}$$

Application of simple arithmetical laws transforms the above inequalities into

$$0 < 1 + |z|, \tag{4}$$

$$0 < 1 + |y|, \tag{5}$$

$$0 < |u|(|y| + |z|). \tag{6}$$

These new inequalities are *polynomial inequalities* as on their right sides there are polynomials over the "variables" $|u|, |y|, |z|$. To prove termination of normalize we have to verify that these polynomial inequalities are true for all instantiations of u, y, z with conditional expressions. As objects of the data structure cexpr are only mapped to numbers greater or equal than 1, it suffices if (4) - (6) hold for all $|u|, |y|, |z| \geq 1$. So instead of the estimation calculus (which only works for the special measure function $|.|_{\#}$) we now need a method to prove polynomial inequalities. Note that in general it is undecidable whether a polynomial inequality is true for all instantiations of its variables with naturals [Lan79].

The use of polynomial mappings $|.|$ for termination proofs is a technique often used for termination proofs of *term rewriting systems* [Lan79], [Der87]. Therefore in this area several methods have been developed to automatically prove that given polynomial inequalities hold for all instantiations of the variables with natural numbers greater than some minimal value. One of the best known approaches has been developed by *A. Ben Cherifa* and *P. Lescanne* [BL87] and has been refined by *J. Steinbach* [Ste92]. Recently, we have presented a simpler and

slightly more powerful alternative approach for proving polynomial inequalities which is based on Lankford's *partial derivative technique* [Gie95a].

The main idea of this approach is that instead of proving inequality (6) for all natural numbers $|u|, |y|, |z| \geq 1$ it is sufficient if this inequality holds for $|u| = 1$ and if $|u|(|y| + |z|)$ is not decreasing when $|u|$ is increasing. In other words the partial derivative of $|u|(|y| + |z|)$ with respect to $|u|$ should be non-negative. Therefore we can replace (6) by

$$0 < |y| + |z| \quad \text{(resulting from } |u| = 1) \quad \text{and} \tag{7}$$

$$0 \leq |y| + |z| \quad \text{(resulting from partial derivation).} \tag{8}$$

By further application of this technique (i.e. demanding that (7) and (8) hold for $|y| = 1$ and that the partial derivatives with respect to $|y|$ are non-negative) (7) is transformed into $0 < 1 + |z|$ and $0 \leq 1$ and (8) is transformed into $0 \leq 1 + |z|$ and $0 \leq 1$. Finally the variable $|z|$ is eliminated in the same way. This yields the inequalities $0 < 2$ and $0 \leq 1$ resp. $0 \leq 2$ and $0 \leq 1$. As these resulting inequalities (between *numbers*) are true, the original inequality (6) also holds for all $|u|, |y|, |z| \geq 1$. The validity of the other two inequalities (4) and (5) can be verified in the same way. Hence, the termination of normalize is proved.

So we replace Walther's estimation calculus by a (simple, efficient and easy to implement) method for proving polynomial inequalities. This results in a more powerful technique, because instead of the size measure function our method can use arbitrary *polynomial norms*.

Formally, a *polynomial norm* $|.|$ is defined by associating an n-ary polynomial $POL(c)$ with each n-ary constructor c (e.g. in our example the nullary polynomial 1 is associated with the constructor atomic and the polynomial $x(1 + yz)$ is associated with cond). In this way each object $c(x_1, \ldots, x_n)$ of a data structure is mapped to a natural number $POL(c)\,(|x_1|, \ldots, |x_n|)$, i.e. a polynomial norm is a homomorphism from the data structure to the naturals.

To conclude this section let us extend our approach to algorithms with more than one formal parameter. As an example consider the following multiplication algorithm by *T. Kolbe* which uses a data structure nat to represent natural numbers. Objects of this data structure are built with the constructors zero and succ : nat $\rightarrow$ nat (for the successor function). The algorithm times has three arguments, where the third argument is used as an accumulator. times(x, y, z) computes $xy + z$ (i.e. if z is zero, then times computes the multiplication of x and y).

function times $(x, y, z :$ nat$)$: nat $\Leftarrow$
if $x =$ zero $\wedge\, z =$ zero *then* zero
if $x =$ succ$(u) \wedge z =$ zero *then* times(u, y, y)
if $z =$ succ(w) *then* succ(times(x, y, w))

To prove termination of times we must now use a measure function m on *tuples* of data objects such that

$$m(u, y, y) < m(\text{succ}(u), y, \text{zero}), \tag{9}$$

$$m(x, y, w) < m(x, y, \text{succ}(w)). \tag{10}$$

In addition to a polynomial norm $|.|$ which maps objects of the data structure nat to natural numbers, we therefore also have to use a *polynomial mapping* $M : \mathbb{N}^3 \to \mathbb{N}$ to map triples of natural numbers to one natural number. So for algorithms with several formal parameters we use measure functions m which measure *data objects* by a *polynomial norm* $|.|$ and which measure *tuples* of naturals by a *polynomial mapping* M. Then $m(t_1, \ldots, t_n)$ is defined as $M(|t_1|, \ldots, |t_n|)$.

For instance, termination of times can be proved with the norm $|\mathsf{zero}| = 0$, $|\mathsf{succ}(x)| = 1 + |x|$ and the mapping $M(x, y, z) = xy + x + y + z$. Then we have

$$m(u, y, y) = M(|u|, |y|, |y|) = |u||y| + |u| + 2|y| \quad \text{and}$$
$$m(\mathsf{succ}(u), y, \mathsf{zero}) = M(|\mathsf{succ}(u)|, |y|, |\mathsf{zero}|) = |u||y| + |u| + 2|y| + 1.$$

By simple arithmetical laws (9) (and also (10)) are both transformed into the obviously valid inequality $0 < 1$. Therefore the termination of times is verified.

3 A Calculus for Termination Proofs

The simple termination proof method sketched in Section 2 only works if the examined algorithm does not call other algorithms in its recursive calls. We illustrate this problem with an example by *J. McCarthy*, viz. the algorithm samefringe which calls the algorithm gopher. For gopher's termination proof a measure function different from *size* must be used. Therefore the method of Walther cannot prove termination of gopher and consequently it cannot prove termination of samefringe either.

The algorithm samefringe works on *s-expressions*, the data structure used in the programming language LISP. The data structure sexpr has the nullary constructor nil, the unary constructor $\mathsf{atom} : \mathsf{nat} \to \mathsf{sexpr}$ and the binary constructor $\mathsf{cons} : \mathsf{sexpr} \times \mathsf{sexpr} \to \mathsf{sexpr}$ to build pairs of s-expressions. To ease readability we write $\mathsf{cons}(0, 0)$ instead of $\mathsf{cons}(\mathsf{atom}(\mathsf{zero}), \mathsf{atom}(\mathsf{zero}))$ etc.

The "fringe" of an s-expression is the sequence of atoms obtained by reading the s-expression from left to right. Therefore the s-expressions $\mathsf{cons}(\mathsf{cons}(1, 2), 3)$ and $\mathsf{cons}(1, \mathsf{cons}(2, 3))$ have the same fringe $[1, 2, 3]$.

The algorithm samefringe examines whether two s-expressions have the same fringe. It uses the algorithm gopher which "rotates" an s-expression until the first argument of the outermost cons is an atom, i.e. $\mathsf{gopher}(\mathsf{cons}(\mathsf{cons}(1, 2), 3)) = \mathsf{cons}(1, \mathsf{cons}(2, 3))$.

function gopher $(x : \mathsf{sexpr}) : \mathsf{sexpr} \Leftarrow$
if $x = \mathsf{cons}(\mathsf{cons}(u, v), w)$ *then* $\mathsf{gopher}(\mathsf{cons}(u, \mathsf{cons}(v, w)))$
otherwise x

Termination of gopher can easily be proved using the polynomial norm $|\mathsf{nil}| = 0$, $|\mathsf{atom}(n)| = 0$, $|\mathsf{cons}(u, v)| = 1 + |u|$.

For each n-ary constructor c of a data structure we need n *selector algorithms* (or *destructors*) $d_1, \ldots, d_n$ which, given an object $c(x_1, \ldots, x_n)$, return the corresponding arguments, i.e. $d_i(c(x_1, \ldots, x_n)) = x_i$. For the data structure sexpr

we therefore use the selectors car and cdr, where car returns the first element of a cons-pair and cdr returns the second element (i.e. $\mathsf{car}(\mathsf{cons}(u,v)) = u$ and $\mathsf{cdr}(\mathsf{cons}(u,v)) = v$). If applied to nil or atom(...), car and cdr yield the result nil.

function samefringe $(x, y : \mathsf{sexpr}) : \mathsf{bool} \Leftarrow$
if $x = y$ *then* true
if $x \neq y \wedge x = \mathsf{cons}(\ldots) \wedge y = \mathsf{cons}(\ldots)$
$\wedge\, \mathsf{car}(\mathsf{gopher}(x)) = \mathsf{car}(\mathsf{gopher}(y))$ *then* $\mathsf{samefringe}(\mathsf{cdr}(\mathsf{gopher}(x)), \mathsf{cdr}(\mathsf{gopher}(y)))$
otherwise false

We attempt to prove termination of samefringe[4] with the polynomial norm $|\mathsf{nil}| = 1$, $|\mathsf{atom}(n)| = 1$, $|\mathsf{cons}(u,v)| = |u| + |v|$. For termination we only consider the first argument of samefringe, i.e. we use the polynomial mapping $M(x,y) = x$. So our measure function is $m(x,y) = M(|x|,|y|) = |x|$. To prove samefringe's termination we have to show that under the condition $x \neq y \wedge x = \mathsf{cons}(\ldots) \wedge \ldots$ the following inequality holds.

$$|\mathsf{cdr}(\mathsf{gopher}(x))| < |x| \tag{11}$$

For the termination proofs of algorithms like normalize, times and gopher we can directly apply the definition of the polynomial norm and obtain a set of polynomial inequalities whose validity implies termination of the algorithm. For these termination proofs two data objects built only with *constructors* have to be compared. But for an algorithm like samefringe which calls two other *algorithms* gopher and cdr, this simple transformation into polynomial inequalities is not possible. The reason is that the polynomial norm is defined in terms of the constructors nil, atom and cons. Therefore we cannot directly say which number corresponds to the data object resulting from $\mathsf{cdr}(\mathsf{gopher}(x))$ by evaluation of the algorithms gopher and cdr.

To solve this problem we need *induction lemmata* on the operations gopher and cdr. The following two induction lemmata state that both gopher and cdr are *argument-bounded* operations, i.e. under the given norm $|.|$ the results of $\mathsf{cdr}(x)$ and $\mathsf{gopher}(x)$ are always smaller or equal than the argument x.

$$|\mathsf{gopher}(x)| \leq |x|, \tag{12}$$
$$|\mathsf{cdr}(x)| \leq |x|. \tag{13}$$

In Section 4 we will illustrate how to synthesize such induction lemmata automatically. Induction lemma (12) states that $|\mathsf{gopher}(x)|$ is smaller or equal than $|x|$ and by induction lemma (13), $|\mathsf{cdr}(\mathsf{gopher}(x))|$ is smaller or equal than $|\mathsf{gopher}(x)|$. Therefore we can conclude

$$|\mathsf{cdr}(\mathsf{gopher}(x))| \leq |\mathsf{gopher}(x)| \leq |x|. \tag{14}$$

[4] In the above formulation of the algorithm samefringe, the condition "$x = \mathsf{cons}(\ldots)$" is an abbreviation for "$x = \mathsf{cons}(u,v)$" (or "$x = \mathsf{cons}(\mathsf{car}(x), \mathsf{cdr}(x))$"), i.e. it ensures that x is built with the constructor cons.

Hence by the transitivity of "$\leq$" the *non-strict* version of (11) is proved, i.e. $|\mathsf{cdr}(\mathsf{gopher}(x))| \leq |x|$.

But for the termination of samefringe we have to prove that $|\mathsf{cdr}(\mathsf{gopher}(x))|$ is *strictly* smaller than $|x|$ under the condition of samefringe's recursive case. So under this condition one of the two inequalities in (14) has to be strict.

In the next section we will also demonstrate how to generate so-called *difference algorithms* Δ_{gopher} and Δ_{cdr} to indicate if $|\mathsf{gopher}(x)|$ resp. $|\mathsf{cdr}(x)|$ are *strictly* smaller than x. For instance, we obtain the following *difference algorithm* Δ_{cdr} (for the above norm $|.|$).

function Δ_{cdr} $(x : \mathsf{sexpr}) : \mathsf{bool} \Leftarrow$
if $x = \mathsf{nil} \vee x = \mathsf{atom}(n)$ *then* false
if $x = \mathsf{cons}(u, v)$ *then* true

Then the following induction lemmata hold as well.

$$\Delta_{\mathsf{gopher}}(x) \rightarrow |\mathsf{gopher}(x)| < |x|, \tag{15}$$

$$\Delta_{\mathsf{cdr}}(x) \rightarrow |\mathsf{cdr}(x)| < |x|. \tag{16}$$

Let $\langle p \leq q,\ \Delta \rangle$ denote that the *non-strict* inequality $p \leq q$ is true and that the *strict* inequality $p < q$ holds if the *difference formula* Δ is true. By the induction lemmata (12) and (15) on the argument-boundedness of gopher we can therefore derive

$$\langle |\mathsf{gopher}(x)| \leq |x|,\ \Delta_{\mathsf{gopher}}(x) \rangle. \tag{17}$$

Induction lemma (13) on the argument-boundedness of cdr implies that $|\mathsf{cdr}(\mathsf{gopher}(x))|$ is smaller or equal than $|\mathsf{gopher}(x)|$ and by the induction lemma (16) we know that $\Delta_{\mathsf{cdr}}(\mathsf{gopher}(x))$ indicates if $|\mathsf{cdr}(\mathsf{gopher}(x))|$ is strictly smaller than $|\mathsf{gopher}(x)|$. So using the induction lemmata (13) and (16) we can derive the following fact from (17).

$$\langle |\mathsf{cdr}(\mathsf{gopher}(x))| \leq |x|,\ \Delta_{\mathsf{cdr}}(\mathsf{gopher}(x)) \vee \Delta_{\mathsf{gopher}}(x) \rangle \tag{18}$$

To finish the termination proof of samefringe we have to prove that the *difference formula* $\Delta_{\mathsf{cdr}}(\mathsf{gopher}(x)) \vee \Delta_{\mathsf{gopher}}(x)$ is true under the condition of samefringe's recursive case, i.e.

$$x \neq y \wedge x = \mathsf{cons}(\ldots) \wedge \ldots \ \rightarrow \ \Delta_{\mathsf{cdr}}(\mathsf{gopher}(x)) \vee \Delta_{\mathsf{gopher}}(x).$$

This theorem can easily be verified by a theorem prover. The reason is that if x is a cons-pair then $\mathsf{gopher}(x)$ also returns a data object built with cons and therefore $\Delta_{\mathsf{cdr}}(\mathsf{gopher}(x))$ returns true under the above conditions.

Analyzing the termination proof for samefringe we obtain the following *calculus* for termination proofs with arbitrary measure functions. For each recursive call $f(t_1, \ldots, t_n)$ of an algorithm $f(x_1, \ldots, x_n)$ we apply the calculus to derive $\langle M(|t_1|, \ldots, |t_n|) \leq M(|x_1|, \ldots, |x_n|),\ \Delta \rangle$ for some formula Δ. Then we use a theorem prover to verify that the difference formula Δ holds under the condition of the recursive case. Our calculus consists of the following three inference rules to derive formulas of the form $\langle p \leq q,\ \Delta \rangle$.

Strict Inequality Rule

If p and q are *polynomials* then we can use the technique sketched in Section 2 to prove the polynomial inequality $p < q$. Validity of $p < q$ implies the non-strict inequality $p \leq q$ and the corresponding difference formula is always true. This results in the following rule.

$$\frac{}{\langle p \leq q,\, \mathsf{true} \rangle}\,, \quad \text{if } p \text{ and } q \text{ are polynomials and } p < q \text{ holds.}$$

Non-Strict Inequality Rule

If p and q are *polynomials* and we can prove $p \leq q$ but we cannot prove $p < q$ then we can at least derive $\langle p \leq q,\, \mathsf{false} \rangle$.

$$\frac{}{\langle p \leq q,\, \mathsf{false} \rangle}\,, \quad \text{if } p \text{ and } q \text{ are polynomials and } p \leq q \text{ holds.}$$

Estimation Rule

This rule formalizes the *estimation* of an argument-bounded operation by its argument. Let p contain[5] a call of an algorithm g, i.e. $p = \ldots + |g(r)| + \ldots$ If we know an induction lemma stating that g is argument-bounded (i.e. $|g(x)| \leq |x|$) and if we know g's difference algorithm Δ_g and the induction lemma $\Delta_g(x) \rightarrow |g(x)| < |x|$, then we can conclude

$$\langle |g(r)| \leq |r|,\, \Delta_g(r) \rangle \tag{19}$$

for all terms r. Assume that $\langle |r| \leq q,\, \Delta \rangle$ holds. Then from (19) we can derive $\langle |g(r)| \leq q,\, \Delta_g(r) \vee \Delta \rangle$. We formalize this reasoning step by the following inference rule

$$\frac{\langle \ldots + |r| + \ldots \leq q,\, \Delta \rangle}{\langle \ldots + |g(r)| + \ldots \leq q,\, \Delta_g(r) \vee \Delta \rangle}\,, \quad \text{if } g \text{ is an argument-bounded operation.}$$

With this calculus we can derive $\langle |x| \leq |x|, \mathsf{false} \rangle$ (by the *non-strict inequality rule*) and by two applications of the *estimation rule* we obtain the formula (18) needed for the termination proof of samefringe. A proof procedure for this calculus is obtained by using the inference rules in *reverse* direction, cf. [Wal94].

4 Synthesis of Induction Lemmata

Our termination proof calculus is based on induction lemmata stating that certain operations are argument-bounded under the given norm $|.|$. Unlike the method of Boyer and Moore, our method is able to synthesize such induction lemmata automatically. The obtained induction lemmata are sound by construction, i.e. they do not have to be verified by a theorem prover. For the recognition of argument-bounded operations we again adapt the approach of Walther and extend it to arbitrary polynomial norms.

[5] In general p can have the form $p = \ldots + m\,|g(r)|^n + \ldots$, where n is a natural number and m is a monomial. The above formulation of the estimation rule is only used to ease readability.

The main idea is to construct a *meta-induction proof* for the argument-boundedness of an algorithm g. This meta-induction is based on the recursions in the algorithm g. In parallel to the proof steps of the meta-induction, the cases of the difference algorithm Δ_g are generated. We illustrate this approach with the algorithm gopher. For each case of this algorithm we prove that $|\mathsf{gopher}(x)| \leq |x|$ holds and we derive a difference formula which implies the strict inequality $|\mathsf{gopher}(x)| < |x|$. For that purpose we use the calculus defined in Section 3.

In the non-recursive second case of the algorithm gopher the result of $\mathsf{gopher}(x)$ is x. We can immediately derive $\langle |x| \leq |x|, \mathsf{false}\rangle$. So in this case gopher is argument-bounded and as the difference formula is false, in this case the difference algorithm Δ_{gopher} should also return false.

The step case of our meta-induction proof corresponds to the recursive first case of gopher. In this case x is of the form $\mathsf{cons}(\mathsf{cons}(u,v),w)$ and the result of $\mathsf{gopher}(x)$ is $\mathsf{gopher}(\mathsf{cons}(u,\mathsf{cons}(v,w)))$. Hence we have to verify the inequality

$$|\mathsf{gopher}(\mathsf{cons}(u,\mathsf{cons}(v,w)))| \leq |\mathsf{cons}(\mathsf{cons}(u,v),w)|.$$

By the *non-strict inequality rule* we can derive

$$\langle |\mathsf{cons}(u,\mathsf{cons}(v,w))| \leq |\mathsf{cons}(\mathsf{cons}(u,v),w)|, \mathsf{false}\rangle. \tag{20}$$

As *induction hypothesis* we can now assume that $\langle |\mathsf{gopher}(x)| \leq |x|, \Delta_{\mathsf{gopher}}(x)\rangle$ holds for the argument of gopher's recursive call, i.e. for $x = \mathsf{cons}(u,\mathsf{cons}(v,w))$. Then by the induction hypothesis and (20) we can derive

$$\langle |\mathsf{gopher}(\mathsf{cons}(u,\mathsf{cons}(v,w)))| \leq |\mathsf{cons}(\mathsf{cons}(u,v),w)|, \Delta_{\mathsf{gopher}}(\mathsf{cons}(u,\mathsf{cons}(v,w))) \vee \mathsf{false}\rangle. \tag{21}$$

The inference of (21) from (20) is an *instance* of the estimation rule in Section 3. So although the requirements for the application of the estimation rule are not satisfied (i.e. the argument-boundedness of gopher is not yet verified), we can nevertheless use this rule for the estimation of gopher due to an inductive argument. Summing up, gopher is recognized as an argument-bounded operation with the difference algorithm

function Δ_{gopher} $(x : \mathsf{sexpr}) : \mathsf{bool} \Leftarrow$
if $x = \mathsf{cons}(\mathsf{cons}(u,v),w)$ *then* $\Delta_{\mathsf{gopher}}(\mathsf{cons}(u,\mathsf{cons}(v,w))) \vee \mathsf{false}$
otherwise false.

Of course the disjunction with false in the result of the first case can be omitted and Δ_{gopher} can be transformed into an algorithm without recursive calls (which always returns false).

In general we use the following procedure to recognize argument-bounded operations g and to synthesize their difference algorithm Δ_g:

if for each case "*if* φ *then* r" of the algorithm $g(x)$ we can derive

$$\langle |r| \leq |x|, \Delta\rangle \text{ for some formula } \Delta,$$

then g is argument-bounded and Δ_g contains the case "*if* φ *then* Δ".

By an inductive argument, in the derivation of $\langle |r| \leq |x|,\ \Delta \rangle$ the estimation rule can also be used for g. In this way induction lemmata of the form "$|g(x)| \leq |x|$" and "$\Delta_g(x) \rightarrow |g(x)| < |x|$" can be synthesized automatically.

5 Refinements of the Approach

In the preceding sections we only regarded *unary* argument-bounded operations. Of course the method can be extended to operations with *several* arguments in a straightforward way. Such operations can be argument-bounded by several of their arguments. For the operation $\mathsf{min}(x, y)$ which computes the minimum of two numbers we could for instance synthesize the induction lemmata $|\mathsf{min}(x, y)| \leq |x|$ and $|\mathsf{min}(x, y)| \leq |y|$. Then we would also generate two difference algorithms Δ^1_{min} and Δ^2_{min} which indicate whether $|\mathsf{min}(x, y)|$ is strictly smaller than its first resp. than its second argument.

There are several improvements Walther suggested for his method which can be directly transferred to our approach. For instance, our calculus could be extended by a fourth *minimum rule* which is only applicable if data objects built with the constructor c_1 are the smallest ones under the current norm $|.|$ and if all objects built with the remaining constructors $c_2, \ldots, c_m$ are mapped to greater numbers (i.e. polynomial inequalities ensuring this condition have to be proved). Then for any term r this rule would allow to derive

$$\langle |c_1(t_1, \ldots, t_n)| \leq |r|,\ r = c_2(\ldots) \vee \ldots \vee r = c_m(\ldots) \rangle.$$

If our calculus is extended by this inference rule then Walther's technique can be obtained as a special case of our method by using the *size* measure function.

Further improvements include a better exploitation of the information contained in the condition of a recursive case. In the recursive case of samefringe we know that x is of the form $\mathsf{cons}(u, v)$ and therefore instead of proving that $|\mathsf{cdr}(\mathsf{gopher}(x))|$ is smaller than $|x|$ it suffices to prove $|\mathsf{cdr}(\mathsf{gopher}(\mathsf{cons}(u, v)))| < |\mathsf{cons}(u, v)|$.

Another refinement concerns the optimization of difference algorithms. Several techniques for their simplification have been presented in [Wal94]. These optimizations considerably ease the proof that the difference formula Δ is valid under the condition of the algorithm's recursive case.

It is also possible to use measure functions based on *lexicographic* combinations of polynomial norms. This would for instance enable a termination proof for *Ackermann*'s well-known function. Nevertheless we found no frequent need for such an extension of our approach. (Note that to prove termination of times in Section 2 with the *size* norm $|.|_{\#}$ a lexicographic ordering would be necessary.)

6 Conclusion and Outlook

We have presented a technique for automated termination proofs of algorithms in a pure functional language. The main idea of this technique is to extend the approach of Walther [Wal94] by a procedure for proving polynomial inequalities. In this way our technique combines the advantages of the methods of Walther and

of Boyer and Moore [BM79], i.e. it is a fully automated procedure which works for arbitrary measure functions based on polynomial norms. The technique has been implemented within the induction theorem prover INKA and proved successful, i.e. termination of all 82 algorithms in the data base of [BM79] could be proved automatically[6].

While in the system of Boyer and Moore any measure function defined by an algorithm can be used, up to now our approach can only deal with measure functions which are based on *polynomial norms*. Future work will include an examination on how our method could be extended to other measure functions.

For Boyer and Moore's method the user has to supply both the measure function and the induction lemmata. In this paper we introduced a technique to synthesize induction lemmata for *given* measure functions automatically. To increase the level of automation we are also working on a method to generate suited measure functions by machine [Gie95b].

Acknowledgements
I would like to thank Jürgen Brauburger, Stefan Gerberding, Thomas Kolbe, Martin Protzen, Christoph Walther and the referees for helpful comments.

References

[BL87] A. Ben Cherifa & P. Lescanne. Termination of Rewriting Systems by Polynomial Interpretations and its Implementation. *Science of Computer Programming*, 9(2):137-159, 1987.

[BL93] E. Bevers & J. Lewi. Proving Termination of (Conditional) Rewrite Systems. *Acta Informatica*, 30:537-568, 1993.

[BM79] R. S. Boyer & J S. Moore. *A Computational Logic.* Academic Press, 1979.

[DSF93] S. Decorte, D. De Schreye & M. Fabris. Automatic Inference of Norms: A Missing Link in Automatic Termination Analysis. In *Proc. Int. Logic Programming Symp.*, Vancouver, Canada, 1993.

[Der87] N. Dershowitz. Termination of Rewriting. *Journal of Symbolic Computation*, 3(1, 2):69-115, 1987.

[Gie95a] J. Giesl. Generating Polynomial Orderings for Termination Proofs. *Proc. 6th Int. Conf. Rewriting Tech. & Applications*, Kaiserslautern, Germany, 1995.

[Gie95b] J. Giesl. Termination Analysis for Functional Programs using Term Orderings. *Proc. 2nd Int. Static Analysis Symposium*, Glasgow, Scotland, 1995.

[Lan79] D. S. Lankford. On Proving Term Rewriting Systems are Noetherian. Technical Report Memo MTP-3, Louisiana Tech. Univ., Ruston, LA, 1979.

[Pau86] L. C. Paulson. Proving Termination of Normalization Functions for Conditional Expressions. *Journal of Automated Reasoning*, 2(1):63-74, 1986.

[Plü90] L. Plümer. *Termination Proofs for Logic Programs.* Springer, 1990.

[Ste92] J. Steinbach. Proving Polynomials Positive. In *Proc. 12th Conf. Foundations Software Technology & Theoretical Comp. Sc.*, New Delhi, India, 1992.

[Wal88] C. Walther. Argument-Bounded Algorithms as as Basis for Automated Termination Proofs. *Proc. 9th Int. Conf. Aut. Deduction*, Argonne, Illinois, 1988.

[Wal94] C. Walther. On Proving the Termination of Algorithms by Machine. *Artificial Intelligence*, 71(1):101-157, 1994.

[6] As mentioned in [Wal94] one algorithm (greatest.factor) must be slightly modified.