THE EXISTENCE PROPERTY IN THE PRESENCE OF FUNCTION SYMBOLS

Michiel Doorman
Department of Philosophy, Rijksuniversiteit te Utrecht

Logic Group
Preprint Series
No. 41
October 1988

Department of Philosophy
University of Utrecht
Heidelberglaan 2
3584 CS Utrecht
The Netherlands

CONTENTS

0. Introduction

We are going to reconsider the existence property in intuitionistic first order logic (IQC) with function symbols, as presented in Dag Prawitz': "Natural Deduction, A Proof Theoretical Study." That is, we will examine its formulation, and try to give a constructive proof of the theorem. I mention in particular the function symbols in IQC, because their presence give rise to a new view on this property. Especially a tool from resolution in automated theorem proving (Gallier [2]) will be necessary for the tightening up of the property.

In the first section I give a short resumé of the theory we need throughout this paper. The second section contains the disjunction property and the existence property, formulated according to Prawitz. Here it is explained why this last property needs a reconsideration. It will appear to be necessary to use an algorithm which computes a most simple term. This is the unification algorithm which is important in resolution.
Section three deals with this algorithm.
In section four the final theorem is presented and proved, the existence property for IQC with function symbols.
The conclusion of this paper follows in section five.

1. Preliminaries

1.0 For the proof of the existence property we need a few definitions and theorems. I will give a short summary of the results which Prawitz proved before arriving at this property. I do advise the reader to take notice of these theorems in Prawitz [4].
We only consider normal derivations, for this purpose I define the following.

1.1.1 *Definition.* In a derivation a segment is a sequence $\phi_1,\ldots,\phi_n$ of isomorfic formula occurences, such that:
(i) ϕ_1 is not the conclusion of a v-E or $\exists$-E, and
(ii) ϕ_i ($i<n$) is the minor premiss of v-E or $\exists$-E, and
(iii) ϕ_n is not the minor premiss of v-E or $\exists$-E.

1.1.2 *Definition.* In a normal derivation in IQC all the segments,which are the conclusion of an introduction rule and the major premiss of an elimination rule, are eliminated.

1.1.3 *The normalisation theorem.*
Every derivation D in IQC can be transformed in a normal derivation D' in IQC.

We also use a theorem which gives us information about the structure of normal proofs. Two more definitions are needed.

1.1.4 *Definition.* In a derivation a path is a sequence of formula occurences $\phi_1,\ldots,\phi_n$, such that:
(i) ϕ_1 is a hypothesis which is not cancelled by a v-E or $\exists$-E, and
(ii) ϕ_i ($i<n$) is not a minor premiss of $\to$-E, and either:
a) ϕ_i is not a major premiss of v-E or $\exists$-E and ϕ_{i+1} immediately below ϕ_i, or
b) ϕ_i is a major premiss of v-E or $\to$-E and ϕ_{i+1} is cancelled at that application,
and
(iii) ϕ_n is either:
a) a minor premiss of $\to$-E, or
b) the conclusion of the derivation, or
c) a major premiss of v-E or $\exists$-E when nothing is cancelled.

1.1.5 The definition of a strictly positive (sp) subformula is inductively given by the following clauses:
(i) ϕ is a sp subformula of ϕ ;
(ii) if $\phi_1 \vee \phi_2$ is a sp subformula of ϕ, then ϕ_1 and ϕ_2 are sp subformula of ϕ ;
(iii) if $\phi_1 \wedge \phi_2$ is a sp subformula of ϕ, then ϕ_1 and ϕ_2 are sp subformula of ϕ ;
(iv) if $Qx\phi_1(x)$ ($Q=\exists,\forall$) is a sp subformula of ϕ,then $\phi_1(t)$ is a sp subformula of ϕ ;
(v) if $\phi_1\to\phi_2$ is a sp subformula of ϕ, then ϕ_2 is a sp subformula of ϕ.

1.1.6 *Theorem.*

Let D be a normal derivation in IQC and $\pi = s_1,\ldots,s_n$ a path of segments in D. There is a minimum segment s_i which divides π in an introduction part and an elimination part, furthermore:

(i) every formula in a segment in the ∃-part of π is a strictly positive subformula of s_1 (ie. a hypothesis), and

(ii) the formula in the minimum segment of π is a strictly positive subformula of s_1 and, if not $\bot$, then a strictly positive subformula of s_n, and

(iii) every formula in a segment in the I-part of π is a strictly positive subformula of s_n.

1.1.7 The last definition in this section concerns terms, because they are of major importance throughout this paper. The definition is according to Prawitz.

Definition. t is a <u>term</u> if and only if

(i) t is a parameter or a constant (not a variable), or

(ii) $t = f(t_1,\ldots,t_n)$, where $t_1,\ldots,t_n$ are terms and f is an n-place function symbol.

2 <u>The disjunction and existence property</u>

2.0 First I give a short description of the disjunction property and its proof. Actually only the part we need in the proof of the existence property. Second I quote the existence property from Prawitz, and try to make clear which problems appear when we observe this property in IQC with function symbols.

2.1 *The disjunction property* (DP).

When no formula in Γ has a strictly positive subformula with $\vee$ as principle sign and $\Gamma\vdash\phi\vee\psi$, then $\Gamma\vdash\phi$ or $\Gamma\vdash\psi$.

The most important feature in the proof of this theorem is the fact that there can be only one endsegment (ie. a segment which contains the endformula $\phi\vee\psi$). Otherwise theorem 1.1.6 tells us that $\vee$ is a principle sign in Γ.

example.

Suppose there are two endsegments, these endsegments have to go through a $\vee$-E:

$$\begin{array}{ccc} & [\phi] & [\psi] \\ D' & D_1 & D_2 \\ \phi\vee\psi & \upsilon & \upsilon \\ \hline & \upsilon & \\ & \vdots & \\ & \upsilon & \end{array} \quad \vee\text{-E}$$

The path along $\phi\vee\psi$ with topformula τ does not go through a $\to$-E. So τ is not cancelled and according to theorem 1.1.6 $\phi\vee\psi$ is a strictly positive subformula of Γ, this contradicts the assumption.

The interested reader should consult Prawitz for the details of the proof.

2.2 The following theorem holds for a language without function symbols.

The existence property (EP).

"Let $t_1, \ldots, t_n$ ($n \geq 0$) be all the terms that occur in $\exists xA$ for some formula of Γ and let there be no formula of Γ that has a strictly positive subformula containing $\exists$ as principle sign. We then have:

(i) For $n>0$: If $\Gamma \vdash \exists xA$, then $\Gamma \vdash A^x{}_{t1} \vee \ldots \vee A^x{}_{tn}$.

(ii) For $n>0$ and provided that no formula of Γ has a strictly positive subformula that contains $\vee$ as principle sign : If $\Gamma \vdash \exists xA$, then $\Gamma \vdash A^x{}_{ti}$, for some $i \leq n$.

(iii) For $n=0$: If $\Gamma \vdash \exists xA$, then $\Gamma \vdash \forall xA$."

This property can be proved, rather easily, by means of Kripke semantics [1], or in categorical logic [2]. We want a constructive proof in the Gentzen system. The requested term t in the theorem (ii) really is computed.
The difference with Prawitz' EP is that we allow function symbols.
The next example shows that a more precise formulation of the theorem is needed in that case.

example.

$$\dfrac{\dfrac{\dfrac{\forall x\phi(x)}{\phi(ff(a))} \qquad \dfrac{\forall y\psi(f(y))}{\psi(ff(a))}}{\phi(ff(a)) \wedge \psi(ff(a))}}{\exists x(\phi(x) \wedge \psi(x))}$$

There are no terms in Γ, so according to Prawitz' EP we should conclude $\Gamma \vdash \forall z(\phi(z) \wedge \psi(z))$. This is not the case, for it would imply $\forall z(\psi(z))$.

In this example we could have chosen a more simple term in the $\forall$-E, which is $f(a)$. For the computation of the most simple term we need a so called unification algorithm. This concept will be defined in the next section.

1) D.v.Dalen. *Logic and structure* . Springer Verlag, Berlin, (second edition) 1983.

2) J.Lambek, P.J.Scott. *Introduction to higher order categorical logic* . Cambridge university press, Cambridge 1986.

2.3. The origin of the induced term

Terms can have their origin in formulas in G, but in a derivation new terms can be introduced, as in the example above. For the introduction of new terms we have to consider the following cases:

(i) $$\dfrac{\begin{matrix}[\phi(t)]\\ D\\ \psi\end{matrix}}{\phi(t)\to\psi}\ \to\text{-I}$$

(ii) $$\dfrac{\begin{matrix}D\\ \psi\end{matrix}}{\psi\vee\phi(t)}\ \vee\text{-I}$$

(iii) $$\dfrac{\begin{matrix}D\\ \perp\end{matrix}}{\phi(t)}\ \perp_i$$

(iv) $$\dfrac{\begin{matrix}D\\ \forall x\phi(x)\end{matrix}}{\phi(t)}\ \forall\text{-E}$$

In a derivation D we can have an $\wedge$-I as last application:

$$\dfrac{\begin{matrix}D_1\\ \phi(f(a))\end{matrix}\qquad\qquad \begin{matrix}D_2\\ \psi(b)\end{matrix}}{\phi(f(a))\ \wedge\ \psi(b))}\ \wedge\text{-I}$$

Suppose we want a proof of $\exists x(\phi(x) \wedge \psi(x))$ (eg. example). In that case f(a) and b have to be unified (if possible) at their introduction.
The derivation D is converted to:

$$\dfrac{\dfrac{\begin{matrix}D_1\\ \phi(f(a))\end{matrix}\qquad\qquad \begin{matrix}D_2'\\ \psi(f(a))\end{matrix}}{\phi(f(a))\ \wedge\ \psi(f(a))}\ \wedge\text{-I}}{\exists x(\phi(x)\ \wedge\ \psi(x))}\ \exists\text{-I}$$

We call f(a) the <u>original term</u> of ϕ, and b the <u>original term</u> of ψ, and the place in the derivation tree where the original term t is introduced the <u>origin</u> of t.
Furthermore we define:

2.3.1 *Definition.* Given a derivation $$\dfrac{\begin{matrix}D\\ \phi(t)\end{matrix}}{\exists x\phi(x)}\ \exists\text{-I},$$

we call t the <u>induced term</u> (i-term) of this inference.

We create a recursive algorithm SEARCH to find all the origins of an i-term in a derivation tree D. The input of the algorithm consists of an i-term t, a derivation D and an empty list U. The output is the list U which contains all the origins of t in D. In the appendix I include a program of this algorithm written in PROLOG to illustrate the procedure SEARCH.

2.4

The procedure SEARCH is defined by recursion in the last application of the derivation D (D may be empty, i.e. we arrived at a hypothesis). SEARCH has three arguments, the derivation tree D, the i-term t and the list U of origins which have to be found (at the initial call U is the empty list). I describe SEARCH in a procedural, PASCAL-like, language.

procedure SEARCH (*var* D : tree ; *var* t, t' : term ;
var U : list) ;

begin *case*

. $$\dfrac{\overline{\overline{\phi}}}{\psi(t)}$$:push t' on U ,
/* t' is the original term of ϕ, or */
/* of ψ when t not in ϕ. */

. $\wedge$-E $$\dfrac{\overset{D}{\phi \wedge \psi(t)}}{\psi(t)}$$:search (D, t, U),

. $\vee$-E $$\dfrac{\overset{D_1}{\phi \vee \tau} \quad \overset{\overset{[\phi]}{D_2}}{\psi(t)} \quad \overset{\overset{[\tau]}{D_3}}{\psi(t)}}{\psi(t)}$$:(search (D_3, t, U) ;
search (D_2, t, U) ;
if t in ϕ or τ
then search (D_1, t, U)
),

. $\rightarrow$-E $$\dfrac{\overset{D_1}{\phi \rightarrow \psi(t)} \qquad \overset{D_2}{\phi}}{\psi(t)}$$:(search (D_1, t, U) ;
if t in ϕ
then search (D_2, t, U)
),

. $\forall$-E $$\dfrac{\overset{D}{\forall x \psi(x)}}{\psi(t)}$$:(push t' on U ;
/* t' is the original term of ψ */
if t in $\forall x \psi(x)$
then search (D, t, U)
),

. $\exists$-E $$\dfrac{\overset{D}{\exists y \psi(t,y)}}{\psi(t,s)}$$:search (D, t, U),

. $\perp_i$ $$\dfrac{\overset{D}{\perp}}{\psi(t)}$$:push t' on U
/* t' is the original term of ψ */,

. $\rightarrow$-I $$\dfrac{\overset{\overset{[\phi]}{D}}{\psi}}{\phi \rightarrow \psi}$$:(search (D, t, U) ;
if t in ϕ
then push t' on U
/* t' is the original term of ϕ */
),

. $\wedge$-I $$\dfrac{\overset{D_1}{\psi(t)} \qquad \overset{D_2}{\phi}}{\psi(t) \wedge \phi}$$:(search (D_1, t, U) ;
if t in ϕ
then search (D_2, t, U)
),

. $\vee$-I $\dfrac{\overset{D}{\phi}}{\phi \vee \psi(t)}$:(push t' on U ;
/* t' is the original term of ψ */
<u>if</u> t in ϕ
<u>then</u> search (D, t, U)),

. $\forall$-I $\dfrac{\overset{D}{\psi(t,a)}}{\forall x\psi(t,x)}$:search (D, t, U),

. $\exists$-I $\dfrac{\overset{D}{\psi(t,s)}}{\exists x\psi(t,x)}$:search (D, t, U),

<u>out</u> push t on U
<u>end</u> search.

According to the structure of the derivation, U always precisely contains the origins of the i-term.
Whenever one of the terms in U occurs in a hypothesis, we say that the origin of the i-term is in Γ.
For the computation of the most simple term t for which Γ proofs $\phi(t)$ in the EP, the terms in U have to be unified. The unification algorithm produces the most general unifier of the terms in U, and this most general unifier determines the most simple term t.

3. <u>The unification algorithm</u>

3.1 We represent terms as trees.

(i) When t is a parameter or a constant, then t has no descendants;

(ii) When $t = f(t_1,\ldots,t_n)$, with $t_1,\ldots,t_n$ terms, then f is a node with n descendants (ie. outdegree n), one for each t_i $(1 \le i \le n)$.

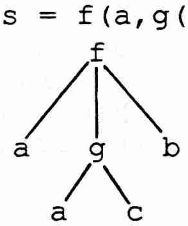
We use the following notation for tree t:

(i) t(e) is the node in tree t with adress e, which is the root.

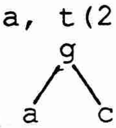
(ii) Let u be a sequence of numbers (ie. an adress in t), then $\{t(ui) : i \in N\}$ are the nodes direct under t(u), and t(uj) is immediately to the left of t(u(j+1)).

(iii) t/u is the subtree of t rooted at u.

example. Term $s = f(a,g(a,c),b)$ is represented as tree t:



t(e) = f and
t(1) = a, t(2) = g, t(3) = b,
t(2 1) = a, t(2 2) = c and
t/2 =



3.1.1 *definitions.*

1) The <u>complexity</u> of a term t with tree T(t) is the depth of T(t).

2) Term t with tree T(t) is <u>simpler</u> than term s with tree T(s), when depth(T(t)) < depth(T(s)).

3) (i) A <u>substitution</u> is a function σ.
Notation: $\sigma(t) = t(s/a)$, σ is the substitution that substitutes term s for a in term t.

(ii) <u>Composition</u> of substitutions is defined as composition of functions, $\sigma \circ \theta(t) = \sigma(\theta(t))$.

(iii) σ is a <u>unifier</u> of s and t when $\sigma(s) = \sigma(t)$.
σ is a <u>most general unifier</u> of s and t, when σ is a unifier of s and t, and for every other unifier θ of s and t, there exists a substitution ρ such that $\theta = \rho \circ \sigma$. We say that $u = \sigma(t) = \sigma(s)$ is the <u>most common instance</u> of t and s.

The unification algorithm computes, given two terms t_1 and t_2, represented in trees, the most general unifier s of t_1 and t_2, if t_1 and t_2 have a unifier at all.

3.1.2 *lemma.* The most common instance u of t and s has the minimal complexity of all possible instances of t and s.
proof.
$u = \sigma(t) = \sigma(s)$ where σ is a most general unifier of t and s. Suppose u' is another common instance of t and s, i.e. $u' = \theta(t) = \theta(s)$, where θ is a unifier of t and s. There is a substitution ρ, such that $\theta = \rho \circ \sigma$, which means that $u' = \rho(u)$, but then we have $\text{depth}(u) \leq \text{depth}(u')$.

$\otimes$

The list U that is produced by the algorithm search may contain more than just two terms. The following lemma says that unification of n terms can be reduced to unification of two terms.

3.1.3 *lemma.* Let \$ be a new function symbol of rank n. σ is a most general unifier of $\$(t_1,\ldots,t_n)$ and $\$(t_1,\ldots,t_1)$ iff σ is a most general unifier of $t_1,t_2,\ldots,t_n$ simultaneously.
proof.
σ is a homomorfism, so $\sigma(\$(t_1,\ldots,t_n)) = \$(\sigma(t_1),\ldots,\sigma(t_n))$ and $\sigma(\$(t_1,\ldots,t_1)) = \$(\sigma(t_1),\ldots,\sigma(t_1))$.
Hence $\sigma(\$(t_1,\ldots,t_n)) = \sigma(\$(t_1,\ldots,t_1))$ iff $\sigma(t_1) = \sigma(t_1)$, $\sigma(t_1) = \sigma(t_2),\ldots,\sigma(t_1) = \sigma(t_n)$ or equivalently $\sigma(t_1) = \sigma(t_2) = \ldots = \sigma(t_n)$.
$\Rightarrow$
When σ is a most general unifier of $\$(t_1,\ldots,t_n)$ and $\$(t_1,\ldots,t_1)$, σ is a unifier of $t_1,t_2,\ldots,t_n$.
Let θ be an arbitrary unifier of $t_1,\ldots,t_n$,
then θ is a unifier of $\$(t_1,\ldots,t_n)$ and $\$(t_1,\ldots,t_1)$, so there exists a substitution ρ with $\theta = \rho \circ \sigma$. We conclude that σ is a most general unifier of $t_1,t_2,\ldots,t_n$.
$\Leftarrow$ This is proved similarly.

3.2 The algorithm is mainly inspired by Gallier [2]. The unification procedure computes whether two terms are unifiable, and if they are unifiable, it computes their most general unifier.
The input consists of two rectified trees (ie. the trees are not allowed to contain identical parameters), which represent

not allowed to contain identical parameters), which represent the terms. The algorithm compairs them by means of a depth-first tree traversal, and determines where the two trees disagree. In our case we know that the two trees are unifiable and we are only trying to find the most common instance of the original terms. This puts us in a rather comfortable position, as there can not be non-repairable (fatal) disagreements (this would imply that the terms aren't unifiable). The only thing we have to do when we discover a disagreement is repairing it.
The output of the unification procedure is the most common instance of the two trees, and the most general unifier.

We first define a few functions:

3.2.1 *definition.*

```
leaf(u) = true  iff u is a leaf (has no descendants);
parameter(u) = true  iff t(u) is a parameter ;
left(u) = if leaf(u) then nil else u 1 ;
right(u) = if u(i+1) adress in t  then  u(i+1)
                                        else  nil  .
```

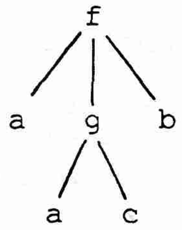
Now we are able to formulate the algorithm.
(The appendix also includes a PROLOG-program of a unification procedure.)

3.3

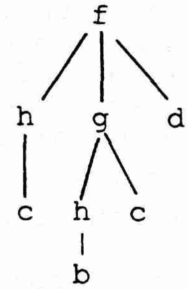
```
procedure  unification  ( var t1, t2: tree ;
                          var mgu : substitution );

   procedure  unify  ( var node : treereference ;
                       var mgu  : substitution );

   var  newnode : treereference;
   var σ: substitution;
begin  if  t1(node)<>t2(node)  then  case
         . parameter (t1(node)) :
           (σ := (t2/(node))/(t1(node)) ; mgu := σ ∘ mgu;
            t1 := σ(t1) ; t2 := σ(t2)  ),

         . parameter (t2(node)) :
           (σ := (t1/(node))/(t2(node)) ; mgu := σ ∘ mgu;
            t1 := σ(t1) ; t2 := σ(t2)  ),

       endif

       if  left(node)<>nil  then  newnode := left(node) ;
       while  newnode<>nil  do
                 unify (newnode,unifier) ;
                 newnode := right (newnode)
       endwhile
       endif
   end unify ;

 /* main program */

 begin  mgu := nil ;
        node := e  ;
        unify (node,mgu)
 end unification.
```

example. $t_1 = f(a,g(a,c),b)$ and $t_2 = f(h(c),g(h(b),c),d)$.

t_1:

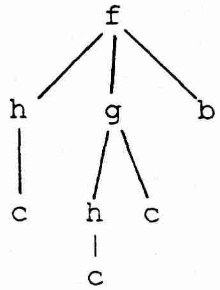


t_2:

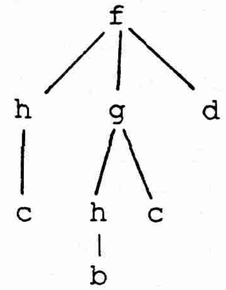


The first disagreement is found at node 1.
parameter $(t_1(1))$, so $\sigma := h(c)/a$ and mgu $:= h(c)/a$,

t_1:

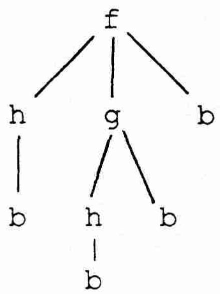


t_2:

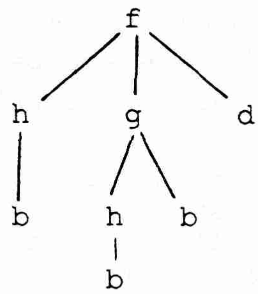


The second disagreement is found at node 211.
parameter $(t_1(211))$, so $\sigma := b/c$ and mgu $:= b/c \circ h(c)/a$,

t_1:



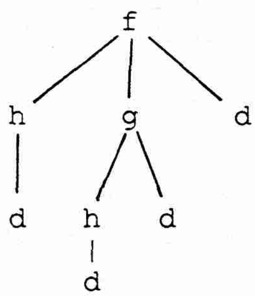
t_2:



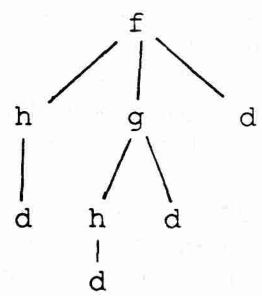
The last disagreement at node 3 follows.
parameter $(t_1(3))$, so $\sigma := d/b$ and mgu $:= d/b \circ b/c \circ h(c)/a$,

finally we have:

t_1:



t_2:



We also need the crucial theorem, which is proved in Gallier [2] p.390

Theorem. When there is a unifier of two terms, then there exists a most general unifier of these terms, produced by the algorithm.

3.4 The next question is:
Is it possible to give a maximum of the complexity of the most common instance of the list U ?
Suppose we arrive during the procedure UNIFY at node u, and $t_1(u)$ is a parameter a, so $\sigma := ((t_2/u)/a)$ (when $t_1(u)<>t_2(u)$).
Let $depth(t_1) = m_1$ and $depth(t_2) = m_2$ at that moment. We distinguish two cases:

1) a does not occur somewhere in t_1 or t_2 :
$depth(\sigma(t_1)/u) = depth(\sigma(t_2)/u)$, hence
$max(depth(t_1),depth(t_2)) = max(depth(\sigma(t_1)),depth(\sigma(t_2)))$.

2) a does occur somewhere else in t_1 or t_2 :
$depth(t_2/u) \leq max(depth(t_1),depth(t_2))$, hence
$max(depth(\sigma(t_1)),depth(\sigma(t_2))) \leq 2 * max(depth(t_1),depth(t_2))$.

Furthermore, we know that after every substitution during the unification algorithm the number of distinct parameters decreases with one.
These results give rise to the next lemma on the maximal complexity of the most common instance of two given terms.

3.4.1 *Lemma*. Given the most common instance t of two terms t_1 and t_2.
Let n be the number of distinct parameters in t_1 and t_2:
(i) If no parameter in t_1 or t_2 occurs more than once, then $depth(t) \leq max(depth(t_1),depth(t_2))$.
(ii) If there are parameters in t_1 or t_2, which occur twice or more, then $depth(t) \leq 2^{n-1} * max(depth(t_1),depth(t_2))$.

proof.
(i) From the preceding results we know that after every substitution, the complexity of the resulting term is smaller than or equal to the maximal complexity of the former terms.
(ii) At most n parameters occur more than once. We have at most n-1 substitutions which can double the maximal complexity.

$\otimes$

But the implementation we choose in lemma 3.1.3 for unifying n terms leads to a crude maximum of the complexity.
(When n>2, there will always occur a parameter twice or more in the tree $\$(t_1,....,t_1)$.)
An implementation which doesn't suffer from this disadvantage is illustrated by the following example:

Example. Suppose four rectified (this is important for the implementation !) terms have to be unified. We compute:
(i) The most general unifier σ_1 of t_1 and t_2;
(ii) The most general unifier σ_2 of t_3 and t_4;
(iii) The most general unifier σ_3 of $\sigma_1(t_1)$ and $\sigma_2(t_3)$.

Claim. $\sigma_3 \circ \sigma_2 \circ \sigma_1$ is a most general unifier of t_1, t_2, t_3 and t_4.
The proof is in two steps:

1) $\sigma_3 \circ \sigma_2 \circ \sigma_1$ is a unifier of t_1, t_2, t_3 and t_4.
   - $\sigma_3 \circ \sigma_2 \circ \sigma_1 (t_1) = \sigma_3 \circ \sigma_2 \circ \sigma_1 (t_2)$.
   - $\sigma_3 \circ \sigma_2 \circ \sigma_1 (t_1) = \sigma_3 \circ \sigma_1 (t_1) = \sigma_3 \circ \sigma_2 (t_3) = \sigma_3 \circ \sigma_2 \circ \sigma_1 (t_3)$.
   - $\sigma_3 \circ \sigma_2 \circ \sigma_1 (t_1) = \sigma_3 \circ \sigma_2 (t_3) = \sigma_3 \circ \sigma_2 (t_4) = \sigma_3 \circ \sigma_2 \circ \sigma_1 (t_4)$.

2) $\sigma_3 \circ \sigma_2 \circ \sigma_1$ is a most general unifier of t_1, t_2, t_3 and t_4.
   If θ is a unifier of t_1, t_2, t_3 and t_4, then:
   a) θ is a unifier of t_1 and t_2 : $\theta = \rho_1 \circ \sigma_1$,
   b) θ is a unifier of t_3 and t_4 : $\theta = \rho_2 \circ \sigma_2$,
   c) $\theta(\sigma_1(t_1)) = \rho_1 \circ \sigma_1(\sigma_1(t_1)) = \rho_1(\sigma_1(t_1)) = \theta(t_1) = \theta(t_3) = \rho_2(\sigma_2(t_3)) = \rho_2 \circ \sigma_2(\sigma_2(t_3)) = \theta(\sigma_2(t_3))$. Hence θ is a unifier of $\sigma_1(t_1)$ and $\sigma_2(t_3)$: $\theta = \rho_3 \circ \sigma_3$.
      Let $\rho = \rho_1 \circ \rho_2 \circ \rho_3$, we have $\rho \circ \sigma_3 \circ \sigma_2 \circ \sigma_1 =$
      $\rho_1 \circ \rho_2 \circ \rho_3 \circ \sigma_3 \circ \sigma_2 \circ \sigma_1 = \rho_1 \circ \rho_2 \circ \theta \circ \sigma_2 \circ \sigma_1 = \rho_1 \circ \rho_2 \circ \rho_2 \circ \sigma_2 \circ \sigma_2 \circ \sigma_1 =$
      $\rho_1 \circ \rho_2 \circ \sigma_2 \circ \sigma_1 = \rho_1 \circ \theta \circ \sigma_1 = \ldots = \theta$.

Conclusion: Given a substitution θ, which is a unifier of t_1, t_2, t_3 and t_4, there exists a substitution ρ such that $\theta = \rho \circ \sigma_3 \circ \sigma_2 \circ \sigma_1$, and $\sigma_3 \circ \sigma_2 \circ \sigma_1$ is a most general unifier of t_1, t_2, t_3 and t_4.

⊗

Different most general unifiers can, by definition, only be alphabetic variants of each other. Hence, the resulting most common instances do have the same complexity.
The lemma on the maximal complexity for the most common instance of n terms becomes:

3.4.2 *Lemma.* Given n terms $t_1, \ldots, t_n$ with most common instance t. Let k be the number of distinct parameters in $t_1, \ldots, t_n$:

(i) If no parameter occurs twice a in term, then $\mathrm{depth}(t) \leq \max(\mathrm{depth}(t_1), \ldots, \mathrm{depth}(t_n))$.

(ii) If a parameter occurs twice or more, then $\mathrm{depth}(t) \leq 2^k {}_\star \max(\mathrm{depth}(t_1), \ldots, \mathrm{depth}(t_n))$.

4. The final theorem.

4.0 The two special tools introduced for our existence property are the procedures SEARCH and UNIFICATION. They are crucial for the formulation and proof of the theorem.

4.1 *lemma*: When $\Gamma \vdash \exists x\phi(x)$ and no formula in Γ has a strictly positive subformula containing $\exists$ as principal sign, then every endsegment σ is the conclusion of a $\exists$-I or $\bot_i$.

proof: Let $\psi = \exists x\phi(x)$. The proof consists of three steps.

(i) σ contains no minor premiss of $\exists$-E.
Suppose we have the derivation with endformula ψ:

$$\begin{array}{c}\dfrac{\begin{array}{c}[\upsilon(u)]\\ D_1\\ \exists y\upsilon(y)\end{array}\qquad\qquad \begin{array}{c}D_2\\ \psi\end{array}}{\psi}\ \exists\text{-E}\\ \vdots\\ \psi\end{array}$$

The path through $\exists y\upsilon(y)$ with topformula τ contains no $\rightarrow$-I (by definition). So τ is not cancelled and so is (1.1.5) $\exists$ the principal sign of a strict positive subformula in Γ. Contradiction with the assumptions on Γ.

(ii) σ is not the consequence of an elimination rule. When σ was the consequence of an elimination rule, σ had to be the minimum segment of the paths where it belongs to. The topformulae of these paths belong to Γ and have $\exists x\phi(x)$ as strictly positive subformula. Again a contradiction on the assumptions on Γ.

(iii) σ is the conclusion of a $\exists$-I or $\bot_i$.
σ has to bo the conclusion of an introduction rule, or $\bot_i$.

$\otimes$

The derivation of $\exists x\phi(x)$ from Γ has the following form. We have the conclusion $\exists x\phi(x)$ and possibly more endsegments σ, because there can be $\vee$-eliminations in σ (2.1). Every endsegment is the conclusion of an $\exists$-I or $\bot_i$. The induced term at that $\exists$-I is called the i-term of that endsegment.

4.2 **EP**: Let $\Gamma \vdash \exists x\phi(x)$ and let $t_1,\ldots,t_n$ be all the terms in Γ. Assume that no formula in Γ has a strictly positive subformula containing $\exists$ as principle sign. Then

(i) If there are q endsegments, then $\Gamma \vdash \phi(s_1) \vee \ldots \vee \phi(s_q)$, where terms $s_1,\ldots,s_q$ are obtained by the unification process from $t_1,\ldots,t_n$ and terms which are introduced in the derivation. We can compute a maximal complexity for the terms $s_1,\ldots,s_q$.

(ii) If Γ has no formula with a strictly positive subformula containing $\vee$ as principle sign, then $\Gamma \vdash \phi(s)$, where s is obtained by unification from $t_1,\ldots,t_n$ and terms in the derivation. We can give a maximal complexity of s. (If s only is obtained from terms introduced in the derivation we can conclude $\Gamma \vdash \forall x\phi(s(x))$.)

proof:

(i) Let σ_1 be an endsegment (for example the most left one in the derivation). From lemma 4.1 we conclude that σ_1 is the conclusion of an $\exists$-I or $\perp_i$. When the premiss of that application is $\phi^x{}_u$ we determine with the procedure SEARCH the origin of u. If u doesn't have its origin in Γ, we compute the most common instance s_1 of list U (produced by SEARCH) with the procedure UNIFICATION. We repeat this for every endsegment in the derivation. Next we convert the derivation into a derivation of $\Gamma \vdash \phi(s_1) \vee \ldots \vee \phi(s_q)$. Every endsegment in the original derivation was a conclusion of $\exists$-I or $\perp_i$. In case of $\exists$-I we delete this $\exists$-I and insert q $\vee$-introductions with conclusion $\phi(s_1) \vee \ldots \vee \phi(s_q)$. In case of $\perp_i$ we don't conclude $\exists x\phi(x)$, but directly $\phi(s_1) \vee \ldots \vee \phi(s_q)$. At this moment every endsegment σ_i' is the conclusion of the last $\vee$-I or $\perp_i$ and the endsegments σ_i' have $\exists x\phi(x)$ substituted by this disjunction. We obtain the required proof. The maximal complexity of $s_1,\ldots,s_q$ follows from 3.4.1

(ii) This is an immediate conclusion from the remark in 2.1 at the disjunctive property and (i) above.
If the term s does not occur in any hypothesis on which this endformula depends (n=0), we can apply a $\forall$-introduction and obtain the desired proof.

$\otimes$

5. Conclusion.

The main difference between the existence property we have formulated and the EP from Prawitz [4] is that we have to examine the derivation. We must discover the origin of the i-term, because terms which are introduced during the derivation can contain function symbols and are not just parameters. The presence of function symbols gives rise to a unification procedure. It is this procedure which made it possible to handle the function symbols in a proper way, though it has become quite a complicated way to come to the essential conclusions of the theorem.
Our result is a complete and constructive proof of the EP for intuitionistic logic with function symbols. We really compute the required term(s) and can therefore determine a boundary of its complexity.
(However, in my opinion there is some future work in making this boundary smaller.)

6 References

[1] D. v. Dalen. *Logic and structure*. Springer Verlag, Berlin, second edition 1983.

[2] J.H. Gallier. *Logic for computer science: Foundations of automated theorem proving*. Harper & Row, 1986.

[3] S.G. v/d Meulen. *Kunstmatige intelligentie*. Unpublished, 1987.

[4] D. Prawitz. *Natural deduction, a proof theoretical study*. Almqvist & Wiksell, Stockholm 1965.

```
/*                          APPENDIX                                      */

/* This program searches for original terms in a proof, and unifies them. */
/* It uses a predicate 'modify', which translates the input in a list.    */
/* This list is searched by the predicate 'search', 'search' computes a   */
/* list U of original terms. U is unified by the predicate 'unificate'.   */

/* These operators make it easier to define input.                         */

:- op(600,xfx,&).
:- op(600,xfx,v).
:- op(600,xfx,->).
:- op(550,fy,~).

/* The procedure modify has two arguments.                                */
/* The first argument contains the input specification.                   */
/* The second argument specifies the output for that application, when   */
/* the condition is satisfied.                                            */

modify(and_intro(A,B,C) , [A1,B1,C1]) :- modify(A,A1),
                                          modify(B,B1),
                                          modify(C,C1).

modify(or_intro(A,B) , [A1,B1]) :- modify(A,A1),
                                   modify(B,B1).

modify(imp_intro(A,B) , [A1,B1]) :- modify(A,A1),
                                    modify(B,B1).

modify(for_all_intro(A,B) , [A1,B1]) :- modify(A,A1),
                                        modify(B,B1).

modify(there_is_intro(A,B) , [A1,B1]) :- modify(A,A1),
                                         modify(B,B1).

modify(falsum(A,falsum,B) , [A1,[falsum|B1]]) :- modify(A,A1),
                                                 modify(B,B1).

modify(and_el(A,B) , [A1,B1]) :- modify(A,A1),
                                 modify(B,B1).

modify(or_el(A,B,C,D) , [A1,B1,C1,D1]) :-
                               modify(A,A1),
                               modify(B,B1),
                               modify(C,C1),
                               modify(D,D1).

modify(imp_el(A,B,C) , [A1,B1,C1]) :- modify(A,A1),
                                      modify(B,B1),
                                      modify(C,C1).

modify(for_all_el(A,B) , [A1,B1]) :- modify(A,A1),
                                     modify(B,B1).

modify(A & B , [and,A1,B1]) :- modify(A,A1),
                               modify(B,B1).

modify(A v B , [or,A1,B1]) :- modify(A,A1),
                              modify(B,B1).

modify(A -> B , [imp,A1,B1]) :- modify(A,A1),
                                modify(B,B1).
```

```
modify(~A , A1) :- modify(A,A1).

modify(for_all(X,B) , [for_all,B1]) :- modify(B,B1).

modify(there_is(X,B) , [there_is,B1]) :- modify(B,B1).

modify(A,A).

/* The procedure search computes the list U of original terms, given    */
/* the modified list of the proof and the i-term.                        */

search([ A | [] ], T , [Ot]) :-
                flatten([A],B),
                member(T,B),
                original(Ot,A).

search([ A | [] ], T , []).

search([ A  , [[and,A,B]|L] ], T , U) :-
                search(A , T , U).

search([ A , [[or,B,C]|L1],[A|L2],[A|L3] ], T , U) :-
                search([ [or,B,C]|L1 ], T , U1),
                search([ A|L2 ], T , U2),
                search([ A|L3 ], T , U3),
                append(U1,U2,V),
                append(U3,V ,U).

search([ A , [[imp,B,A]|L1],[B|L2] ], T , U) :-
                search([ [imp,B,A]|L1 ], T , U1),
                search([ B|L2 ], T , U2),
                append(U1,U2,U).

search([ A , [[for_all|B]|L] ], T , U) :-
                flatten(B,C),
                member(T,C),
                search([ [for_all|B]|L ], T , U).

search([ A , [[for_all|B]|L] ], T , [Ot]) :-
                original(Ot,A).

search([ A , [[there_is|B]|L] ], T , U) :-
                search([ [there_is|B]|L ], T , U).

search([ A , [falsum|L] ], T , [Ot]) :-
                original(Ot,A).

search([ A , [falsum|L] ], T , U) :-
                search( L , T , U).

search([ [imp,A,B] , [B|L] ], T , U) :-
                search([ B|L ], T , U).

search([ [and,A,B] , [A|L1] , [B|L2] ], T , U) :-
                search([ A|L1 ], T , U1),
                search([ B|L2 ], T , U2),
                append(U1,U2,U).

search([ [or,A,B] , [A|L] ], T , [Ot|U]) :-
                flatten(B,C),
                member(T,C),
                original(Ot,B),
                search([ A|L ], T , U).
```

```
search([ [for_all|B] , [A|L] ], T , U) :-
                    search([ A|L ], T , U).

search([ [there_is|B] , [A|L] ], T , U) :-
                    search([ A|L ], T , U).

/* 'Flatten' flattens a formula, ie. creates a list of all the symbols,      */
/* which occur in the formula, in order to decide whether T is a 'member'    */
/* of the formula.                                                           */

flatten([],[]).

flatten([X|Xs],Ys) :- flatten(X,Ys1),
                      flatten(Xs,Ys2),
                      append(Ys1,Ys2,Ys).

flatten(X,[X]).

/* The procedure unificate unifies the list U.                            */
/* The procedure unify unifies two terms. When a disagreement is found    */
/* during the searching through the 'term tree', we have to make a        */
/* substitution in the terms. These terms are the third and fourth        */
/* argument of the predicate. The results after the substitutions at a    */
/* disagreement are put in the fifth and sixth argument of 'unify'.       */
/* The first two arguments contain the current node of the trees, and     */
/* specify the substitution when a disagreement is found.                 */

unificate([U1,[]],U1).

unificate([U1|[U2|[]]],X) :- unify(U1,U2,U1,U2,X,Y).

unificate([U1|L],Y) :- unify(U1,X,U1,X,SX,Y),
                       unificate(L,X).

unify(X,Y,Xor,Yor,SX,SY) :- atom(X),
                            substitute(X,Y,Xor,SX),
                            substitute(X,Y,Yor,SY).

unify(X,Y,Xor,Yor,SX,SY) :- atom(Y),
                            substitute(Y,X,Xor,SX),
                            substitute(Y,X,Yor,SY).

unify(X,Y,Xor,Yor,SX,SY) :- functor(X,F,N),
                            functor(Y,F,N),
                            unify_args(N,X,Y,Xor,Yor,SX,SY).

unify_args(N,X,Y,Xor,Yor,SSX,SSY) :- N > 0,
                                     unify_arg(N,X,Y,Xor,Yor,SX,SY),
                                     M is N-1,
                                     unify_args(M,X,Y,SX,SY,SSX,SSY).

unify_args(0,X,Y,Xor,Yor,Xor,Yor).

unify_arg(N,X,Y,Xor,Yor,SX,SY) :- arg(N,X,Xn),
                                  arg(N,Y,Yn),
                                  unify(Xn,Yn,Xor,Yor,SX,SY).

/* 'Substitute' replaces all the occurences of the first argument for   */
/* the second argument in the third argument. The resulting term is     */
/* the fourth argument of this procedure.                               */

substitute(X,Y,X,Y).
```

```
substitute(X,Y,Xor,Xor) :- atom(Xor).

substitute(X,Y,Xor,SX ) :- functor(Xor,F,N),
                           functor(SX ,F,N),
                           substitute(N,X,Y,Xor,SX).

substitute(N,X,Y,Xor,SX):- N > 0,
                           arg(N,Xor,Xor_n),
                           substitute(X,Y,Xor_n,SX_n),
                           arg(N,SX,SX_n),
                           M is N-1,
                           substitute(M,X,Y,Xor,SX).

substitute(0,X,Y,Xor,SX).

/* 'Solve' is first called by the user, with in the first argument the  */
/* proof and in the second argument the i-term t.                        */
/* The proof is written:                                                 */
/* <application_rule>(<conclusion>,<premiss_1>,...,<premiss_i>),         */
/* where i = 1,2,3.                                                      */
/* A premiss can be a proof itself, or a hypothesis, or a formula which */
/* is introduced at that application.                                    */
/* Formulas which are a hypothesis, or introduced at an application     */
/* are already written as a list: [<operator>,<operand_1>,<operand_2>], */
/* of course the negation sign has only one operand.                     */
/* An atom a(t) becomes [a,t].                                            */

solve(X,T,Mgu) :- modify(X,Y),
                  search(Y,T,U),
                  unificate(U,Mgu).

/* Three examples are included in this appendix :                        */
```

I asserted the following facts to the data-base:

```
proof2(imp_intro(([a,t] & [b,t]) -> ~([a,t] ->[b,t]),falsum(~([a,t]
-> [b,t]),falsum,and_intro([b,t] & [~b,t],imp_el([b,t],[[imp, [a,t]
,[b,t]]],and_el([a,t],[[and,[a,t],[~b,t]]])),and_el([~b,t] ,[[and,
[~b,t],[a,t]]])))))).

proof3(and_intro([a,f(f(t))] & [b,f(f(t))],for_all_el([a,f(f(t))],
for_all(x,a),for_all_el([b,f(f(t))],for_all(x,b))))).

original(t,[imp,[a,t],[b,t]]).
original(u,[and,[~b,t],[a,t]]).
original(s,[and,[a,t],[~b,t]]).
original(f(t),[a,f(f(t))]).
original(s,[b,f(f(t))]).
```

The questions were:

```
?- proof2(P2),modify(P2,X),search(X,t,U),unificate(U,Mgu-u).
?- proof3(P3),modify(P3,Y),search(Y,f(f(t)),V),unificate(V,Mgu-v).
?- unificate([ f(g(s,t),s,t) , f(u,h(k),h(k)) ] , Mgu).
```

Printed are: X, U, Mgu-u,
Y, V, Mgu-v,
Mgu.

```
output          Wed Feb 24 15:48:46 1988          1

X = [[imp,[and,[a,t],[~b,t]],[imp,[a,t],[b,t]]],[[imp,[a,t],[b,t]],[falsum,
[and,[b,t],[~b,t]],[[b,t],[[imp,[a,t]],[b,t]]],[[a,t],[[and,[a,t],[~b,t]]]]]
,[[~b,t],[[and,[~b,t],[a,t]]]]]]].

U = [t,s,u]

Mgu-u = t


Y = [[and,[a,f(f(t))],[b,f(f(t))]],[[a,f(f(t))],[[for_all,a]]],[[b,f(f(t))],
[[for_all,b]]]].

V = [f(t),s]

Mgu-v = f(t)


Mgu = f(g(h(k)),h(k),h(k))
```

Logic Group Preprint Series
Department of Philosophy
University of Utrecht
Heidelberglaan 2
3584 CS Utrecht
The Netherlands

nr. 1 C.P.J. Koymans, J.L.M. Vrancken, *Extending Process Algebra with the empty process*, September 1985.
nr. 2 J.A. Bergstra, *A process creation mechanism in Process Algebra,* September 1985.
nr. 3 J.A. Bergstra, *Put and get, primitives for synchronous unreliable message passing,* October 1985.
nr. 4 A.Visser, *Evaluation, provably deductive equivalence in Heyting's arithmetic of substitution instances of propositional formulas*, November 1985.
nr. 5 G.R. Renardel de Lavalette, *Interpolation in a fragment of intuitionistic propositional logic,* January 1986.
nr. 6 C.P.J. Koymans, J.C. Mulder, *A modular approach to protocol verification using Process Algebra* , April 1986.
nr. 7 D. van Dalen, F.J. de Vries, *Intuitionistic free abelian groups,* April 1986.
nr. 8 F. Voorbraak, *A simplification of the completeness proofs for Guaspari and Solovay's R*, May 1986.
nr. 9 H.B.M. Jonkers, C.P.J. Koymans & G.R. Renardel de Lavalette, *A semantic framework for the COLD-family of languages,* May 1986.
nr. 10 G.R. Renardel de Lavalette, *Strictheidsanalyse,* May 1986.
nr. 11 A. Visser, *Kunnen wij elke machine verslaan? Beschouwingen rondom Lucas' argument*, July 1986.
nr. 12 E.C.W. Krabbe, *Naess's dichotomy of tenability and relevance,* June 1986.
nr. 13 Hans van Ditmarsch, *Abstractie in wiskunde, expertsystemen en argumentatie,* Augustus 1986
nr. 14 A.Visser, *Peano's Smart Children, a provability logical study of systems with built-in consistency* , October 1986.
nr. 15 G.R.Renardel de Lavalette, *Interpolation in natural fragments of intuitionistic propositional logic,* October 1986.
nr. 16 J.A. Bergstra, *Module Algebra for relational specifications,* November 1986.
nr. 17 F.P.J.M. Voorbraak, *Tensed Intuitionistic Logic,* January 1987.
nr. 18 J.A. Bergstra, J. Tiuryn, *Process Algebra semantics for queues,* January 1987.
nr. 19 F.J. de Vries, *A functional program for the fast Fourier transform,* March 1987.
nr. 20 A. Visser, *A course in bimodal provability logic*, May 1987.
nr. 21 F.P.J.M. Voorbraak, *The logic of actual obligation, an alternative approach to deontic logic,* May 1987.
nr. 22 E.C.W. Krabbe, *Creative reasoning in formal discussion,* June 1987.
nr. 23 F.J. de Vries, *A functional program for Gaussian elimination*, September 1987.
nr. 24 G.R. Renardel de Lavalette, *Interpolation in fragments of intuitionistic propositional logic*, October 1987.(revised version of no. 15)
nr. 25 F.J. de Vries, *Applications of constructive logic to sheaf constructions in toposes*, October 1987.
nr. 26 F.P.J.M. Voorbraak, *Redeneren met onzekerheid in expertsystemen*, November 1987.
nr. 27 P.H. Rodenburg, D.J. Hoekzema, *Specification of the fast Fourier transform algorithm as a term rewriting system,* December 1987.

nr. 28 D. van Dalen, *The war of the frogs and the mice, or the crisis of the Mathematische Annalen,* December 1987.
nr. 29 A. Visser, *Preliminary Notes on Interpretability Logic,* January 1988.
nr. 30 D.J. Hoekzema, P.H. Rodenburg, *Gauß elimination as a term rewriting system*, January 1988.
nr. 31 C. Smorynski, *Hilbert's Programme,* January 1988.
nr. 32 G.R. Renardel de Lavalette, *Modularisation, Parameterisation, Interpolation,* January 1988.
nr. 33 G.R. Renardel de Lavalette, *Strictness analysis for POLYREC, a language with polymorphic and recursive types*, March 1988.
nr. 34 A. Visser, *A Descending Hierarchy of Reflection Principles,* April 1988.
nr. 35 F.P.J.M. Voorbraak, *A computationally efficient approximation of Dempster-Shafer theory*, April 1988.
nr. 36 C. Smorynski, *Arithmetic Analogues of McAloon's Unique Rosser Sentences,* April 1988.
nr. 37 P.H. Rodenburg, F.J. van der Linden, *Manufacturing a cartesian closed category with exactly two objects*, May 1988.
nr. 38 P.H. Rodenburg, J. L.M.Vrancken, *Parallel object-oriented term rewriting : The Booleans*, July 1988.
nr. 39 D. de Jongh, L. Hendriks, G.R. Renardel de Lavalette, *Computations in fragments of intuitionistic propositional logic*, July 1988.
nr. 40 A. Visser, *Interpretability Logic,* September 1988.
nr. 41 M. Doorman, *The existence property in the presence of function symbols,* October 1988.