

CONCURRENCY IN PROOF NORMALIZATION AND LOGIC PROGRAMMING

Shigeki Goto

Computer Science Department, Stanford University
Stanford, California 94305 USA

and

NTT Musashino Electrical Communication Laboratory
3-9-11 Midori-cho, Musashino-shi, Tokyo 180 Japan [from Sept. 85]

ABSTRACT

Proof normalization manipulates formal proofs. It also provides a computation mechanism which belongs to the logic programming family.

Although proof normalization can treat full predicate calculus, it is less practical than the well-known programming language, Prolog.

In this paper, we propose a new technique of attaching proofs to Skolem functions. This technique enables one to normalize a proof eagerly; that is, one can get a partial answer before the proof is totally normalized. This improves the usability of proof normalization. Partial answers are also useful in normalizing proofs concurrently. We compare our method with computation in Concurrent Prolog.

1. Introduction

Proof normalization has a long history in mathematical logic [Prawitz 1965]. The significant result for computer science is as follows: *If there is a proof of the formula $\exists z(A(z))$, one can get an answer t , which satisfies $A(t)$, after normalizing the proof.*

This realizes a computation, which is appropriately called logic programming. However, it is less efficient than Prolog.

[Goad 1980] proposes an extended λ -calculus, named p-calculus, to represent proofs. P-calculus terms are executed efficiently. [Hagiya 1982] noticed that most part of a proof is irrelevant for the computation. He introduces new notions to eliminate the unnecessary normalization steps.

This paper describes a new approach to improve the usefulness of the proof normalization. Our method is to attach proofs to Skolem functions, which enables us to normalize eagerly. Eager normalization produces a partial answer in advance. Using this capability, proof normalization can be performed concurrently rather than one proof at a time. Concurrency here means the same as in Concurrent Prolog [Shapiro 1983]. We apply our method to an example from Concurrent Prolog [Takeuchi 1984] and compare this to computation in Concurrent Prolog.

2. Logical Framework

Proof normalization is elegantly explained using a natural deduction system. Figure 1 summarizes the inference rules for natural deduction, which are used in this paper.

A typical proof in natural deduction is illustrated in

Figure 2. This proof includes a rule of induction (IND). The proof concludes $\exists z(t + y = z)$. The assumptions are: (1) $0 + y = y$ and (2) $\forall xw(x + w = z \supset s(x) + w = s(z))$. Two other formulas enclosed in brackets, $[a + y = c]$ and $[\exists z(a + y = z)]$, are not considered to be assumptions. The formal definition is given below.

Definition 1 *At applications of $\supset I$, $\exists E$, and IND rules, certain assumptions (indicated by brackets in figure 1) are said to be "discharged" by the rule. The conclusion of a proof is said to depend on the assumptions that have not been discharged.*

$\frac{A \quad B}{A \wedge B} \wedge I$	$\frac{A \wedge B}{A} \wedge E$	$\frac{A \wedge B}{B} \wedge E$	$\frac{[A]}{B} \supset I$	$\frac{A \quad A \supset B}{B} \supset E$
$\frac{A(a)}{\forall x(A(x))} \forall I$	$\frac{\forall x(A(x))}{A(t)} \forall E$	$\frac{A(t)}{\exists x(A(x))} \exists I$	$\frac{[A(a)]}{C} \exists E$	
	$\frac{[A(a)]}{A(0) \quad A(s(a))} \text{IND}$			

Function symbol "s" in IND means the successor function: $s(x) = x + 1$

Figure 1: Inference rules in natural deduction

Sometimes, it is important to distinguish between variables. In Figure 2, there are five variables: x, y, z, w, a and c . Variables x, z and w are called *bound* variables. Variable y is a *free* variable. Two variables a and c are both called *proper* parameters, which are defined below.

Definition 2 *In applications of the rule $\forall I$, $\exists E$ and IND, the variable "a" (see Figure 1) is called the proper parameter of the inference rule.*

The proper parameters look like free variables, however, they are bound not by the quantifiers ($\forall$ and $\exists$), but by the inference rules. The precise definition of the bound variable is given by the formulae.as.types approach [Goad 1980].

3. Computation by normalization

In short, proof normalization reduces the redundant part

$$\begin{array}{c}
\frac{\frac{\frac{0+y=y}{\exists x(0+y=z)} \exists I \quad \frac{[a+y=c] \quad a+y=c \supset s(a)+y=s(c)}{\exists z(s(a)+y=z)} \supset E}{\exists z(s(a)+y=z)} \supset E \\
\frac{\exists z(s(a)+y=z)}{\exists z(t+y=z)} \text{IND}
\end{array}$$

Figure 2: Proof using a rule of induction in natural deduction proofs ([Prawitz 1965] and [Troelstra 1973]). Here, we explain only two reduction rules which are relevant to this paper.

Reduction rules

1. **E-reduction rule:** If a proof has an $\exists I$ -rule immediately followed by an $\exists E$ -rule, the proof is simplified by cancelling both applications of rules. Here, Π and $\Pi'(a)$ stand for subproofs. a in $\Pi'(a)$ indicates a proper parameter.

$$\begin{array}{c}
\Pi \\
A(t) \\
\hline
\exists x(A(x)) \quad C
\end{array}
\begin{array}{c}
[A(a)] \\
\Pi'(a) \\
\hline
C
\end{array}
\begin{array}{c}
\exists E
\end{array}
\text{ is reduced to } \Rightarrow
\begin{array}{c}
\Pi \\
[A(t)] \\
\Pi'(t) \\
\hline
C
\end{array}$$

2. **IND-reduction rule:** IND-reduction rule simplifies the term t if the term is of the form 0 or $s(t)$.

$$\begin{array}{c}
\Pi_0 \quad [A(a)] \\
A(0) \quad \Pi(a) \\
\hline
A(0) \quad A(s(a)) \quad \text{IND} \Rightarrow \Pi_0 \quad A(0)
\end{array}$$

or

$$\begin{array}{c}
\Pi_0 \quad [A(a)] \\
A(0) \quad \Pi(a) \\
\hline
A(s(t)) \quad A(s(a)) \quad \text{IND} \Rightarrow \Pi_0 \quad A(0) \quad A(s(a)) \quad \text{IND} \Rightarrow \Pi(t) \quad A(s(t))
\end{array}$$

Definition 3 A proof Π is said to be normal, if no reduction rule is applicable to Π .

Before stating the key property of a normal proof, the concept of the Harrop formula should be introduced.

Definition 4 The class of Harrop formulas is defined inductively as follows:

1. Every atomic formula is a Harrop formula.
2. If A and B are Harrop formulas, then $A \wedge B$ and $\forall x A(x)$ are Harrop formulas.
3. If B is a Harrop formula, then $A \supset B$ is a Harrop formula, regardless of the form of A .

The Harrop formula has no positive $\forall$ nor $\exists$, except in the left hand side of the implications ($\supset$).

Proposition 1 A normal proof of $\exists z A(z)$, where assumptions are only Harrop formulas, contains a subproof of $A(t)$ for a suitable term t .

We can normalize the proof in Figure 2 to calculate the sum of two numbers because the two assumptions are both Harrop formulas, and the proof satisfies the condition of Proposition 1.

4. Attaching proofs

Although proof normalization provides a theoretically justified computation, it does not always produce a satisfactory answer. Consider the example in Figure 3, in which $A(x, z)$ stands for $(x+y=z)$. The uppermost applications of $\forall E$ and $\supset E$ are abridged to $\pi(a)$. We want to add two terms $s(\alpha)$ and y , where α is some term. Normalization cannot give the answer because the term α does not take the form of $s(t)$ nor 0 . The IND-reduction rule cannot be applied further.

$$\begin{array}{c}
\text{STEP 1: original proof} \\
\text{substituted } t+s(\alpha)
\end{array}
\begin{array}{c}
[A(a, c)] \\
\pi(a) \\
A(s(a), s(c)) \\
\hline
\exists z(A(s(a), z)) \quad \exists I \\
\hline
\exists z(A(0, z)) \quad \exists E \\
\hline
\exists z(A(s(a), z)) \quad \exists E \\
\hline
\exists z(A(s(a), z)) \quad \text{IND}
\end{array}$$

$$\begin{array}{c}
\text{STEP 2: apply IND-reduction} \\
(t=s(\alpha))
\end{array}
\begin{array}{c}
[A(a, c)] \\
\pi(a) \\
A(s(a), s(c)) \\
\hline
\exists z(A(s(a), z)) \quad \exists I \\
\hline
\exists z(A(0, z)) \quad \exists E \\
\hline
\exists z(A(s(a), z)) \quad \exists E \\
\hline
\exists z(A(s(a), z)) \quad \text{IND}
\end{array}$$

↓ indicates the same formula

$$\begin{array}{c}
\exists z(A(s(a), z)) \\
\downarrow \\
\exists z(A(s(a), z)) \\
\downarrow \\
\exists z(A(s(a), z)) \\
\downarrow \\
\exists z(A(s(a), z))
\end{array}$$

$$\begin{array}{c}
[A(a, c)] \\
\pi(a) \\
A(s(a), s(c)) \\
\hline
\exists z(A(s(a), z)) \quad \exists I \\
\hline
\exists z(A(s(a), z)) \quad \exists E \\
\hline
\exists z(A(s(a), z)) \quad \exists E
\end{array}$$

STEP 3: cannot apply IND-reduction, because α is not 0 nor $s(t)$. GET STUCK!

Figure 3: Addition of two terms, $s(\alpha)$ and y

We solve the problem by attaching proofs to Skolem functions. The new method is as follows:

Attaching proofs:

1. Eliminate the existential quantifiers in the conclusion of an IND rule, using Skolem functions.

axiom. It should be noted here that predicate "write(N)" is a built-in predicate in Concurrent Prolog, and it has a side effect to write the term N. How should it be handled in normalization? Again, the attaching technique solves the problem. This time a program is attached to a Skolem function. We do not specify any programming language here. To attach a program, there should be an existential quantifier. Thus, write(x) is modified to have an explicit output variable: write@ (x,z). The new predicate is considered "built-in"; that means that $\forall x \exists z (\text{write@}(x,z))$ is an axiom. Formally, write(a) in Example 3 is replaced by the subproof below:

$\frac{\frac{\frac{\text{write@}(a,p(a))}{\exists z(\text{write@}(a,z))} \text{EI}}{\text{write}(a)} \text{VE}}{\text{write}(a)} \text{VE}$	A certain program is attached to "p". $\frac{\frac{\frac{\forall x(\text{write@}(x,p(x)))}{\exists z(\text{write@}(a,z)) \supset \text{write}(a)} \text{EI}}{\text{write}(a)} \text{VE}}{\text{write}(a)} \text{VE}$
---	---

In the subproof, a Skolem function p(a) is used. A certain program is attached to function p(a) to print the term "a" when "a" is substituted by some term. This requires more explanation:

When a term is printed?

A term can be printed anytime unless it is bound.
That is, no bound variables in the scope of $\forall$ or $\exists$, or no proper parameters can be printed.

In the example, "a" is printed after it is substituted. If the proof is normalized after giving $t = [foo|bar]$, "foo" will be printed, because it is substituted for "a" in the proof.

At last, two proofs are combined to produce the same effect as clause (3) in Concurrent Prolog. Figure 5 illustrates only a few steps of the normalization. However, it is easy to see that computation is performed concurrently. The IND $\downarrow$ subproof produces the list, and IND[] subproof consumes the list. The condition on the print function protects the proper parameter from being printed before it is substituted. This realizes a kind of synchronization, which is attained by the read-only annotation in Concurrent Prolog [Shapiro 1983] [Takeuchi 1984].

6. Conclusion

In this paper we propose a new method of proof normalization, which utilizes Skolem functions to normalize a proof eagerly. Each Skolem function represents an existential variable ($\exists z$), and the relation is preserved by attaching proofs to Skolem functions.

Our method facilitates concurrent normalization. Concurrent Prolog is formally well-explained by our method.

It is easy to see that Prolog itself is closely related to proof normalization because: (1) Prolog always generates normal proofs, and (2) A Horn clause is necessarily a Har-

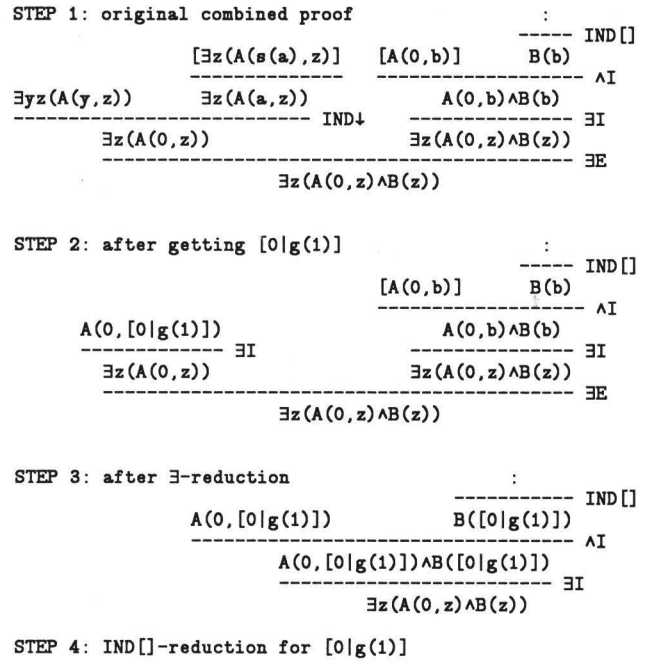


Figure 5: The combined proof

rop formula. Thus, Prolog computation can be considered as a special kind of proof normalization.

ACKNOWLEDGEMENT

I would like to thank Dr. Carolyn Talcott at Stanford University for many valuable comments.

REFERENCES

- [1] C.Goad, Computational Use of the Manipulation of Formal Proofs, *PhD Thesis, Department of Computer Science, Stanford University*, 1980.
- [2] M.Hagiya, A Proof Description Language and Its Reduction System, *Department of Information Science, Tech. Report 82-03, University of Tokyo*, Feb. 1982.
- [3] Z.Manna and R.J.Waldinger, Toward Automatic Program Synthesis, *Comm. ACM*, **14**, no.3, 151-165, 1971.
- [4] D.Prawitz, Natural Deduction, Almquist and Wksell, Stockholm, 1965.
- [5] E.Shapiro, A subset of Concurrent Prolog and Its Implementation, *Technical Report TR-003, ICOT*.
- [6] A.Takeuchi, Concurrent Prolog, *Computer Today*, No.1, pp.48-55, 1984. (in Japanese)
- [7] A.S.Troelstra, Metamathematical Investigation of Intuitionistic Arithmetic and Analysis, *Lecture Notes in Mathematics Vol.344*, Springer-Verlag 1973.
- [8] R.Weyhrauch, Prolegomena to a theory of mechanized Formal Reasoning, *Artificial Intelligence* **13**, North-Holland, 1980.