Vol. 34 No. 3 Transactions of Information Processing Society of Japan Mar. 1993

Short Note

Another Representation of Integers in Logic

MASAHITO KURIHARA[†] and AZUMA OHUCHI[†]

In first-order logic, natural numbers are usually represented by the terms constructed from the constant 0 and the successor function $s(\cdot)$. Addition is defined recursively by two program clauses. In this paper we present another representation based on difference, and show that addition is defined by a unit clause, thus without recursion.

1. Introduction

In first-order logic, natural numbers (including zero) are usually represented by the terms constructed from the constant 0 and the successor function $s(\cdot)$[7)–9)]. For example, the term $s(s(0))$ represents the natural number 2. In general, $s^n(0) \equiv \underbrace{s(s(\cdots s(0)\cdots))}_{n \text{ times}}$ represents n.

We will omit the parentheses. For example, $s(X)$ is written as sX.

The following logical formulas, written in Prolog, define the predicate **plus** for addition. The intended meaning of **plus** (X, Y, Z) is that $X+Y=Z$, where X, Y and Z are natural numbers represented by X, Y and Z, respectively.

$$\textbf{plus}\,(0,\ Y,\ Y). \qquad (1)$$

$$\textbf{plus}\,(sX,\ Y,\ sZ) :\text{-}\,\textbf{plus}\,(X,\ Y,\ Z). \qquad (2)$$

Since most of the logic programming languages implement integers as built-in objects, representation by terms has practically no importance, but at least theoretically and for educational purposes such representation is useful, because it demonstrates the capability of logic for (arithmetic) computation. In this paper we show another representation of integers which demonstrates the power of incomplete data structure in logic programming. It also suggests interesting combination of logic programming and term rewriting.

2. Another Representation

Recently, we noticed that another representation of integers is possible. The idea is similar to that of "difference list," which is a well-known representation of a list based on incomplete data structure in Prolog programming[3)]. For example, a list of integers 1, 2 and 3 may be represented by a pair of incomplete structures $[1, 2, 3|X]$ and X. A technical merit of this representation is that it is easy and efficient to append two lists. The list given above and the list represented by the pair of $[4|Y]$ and Y may be appended simply by unifying X with $[4|Y]$ and constructing the pair of $[1, 2, 3|X]$ and Y as the result, which would yield the pair of $[1, 2, 3, 4|Y]$ and Y if the most general unifier is applied to.

Now, back to the representation of integers, we introduce a two-place function symbol $d(\cdot, \cdot)$, which means difference of two integers. Integers are represented by the terms of the form $d(s^mX, s^nX)$, whose intended meaning is $m-n$, because s^mX and s^nX mean $m+X$ and $n+X$, respectively, so the difference is $(m+X)-(n+X)=m-n$. When $m\cdot n=0$, the term $d(s^mX, s^nX)$ is called the *normal form* representing $m-n$. For example, $d(X, X)$, $d(ssX, X)$ and $d(X, ssX)$ are normal forms representing 0, 2 and -2, respectively. For convenience, we use the notation $s^mX - s^nX$ for $d(s^mX, s^nX)$. Using this representation, addition is defined by the following unit clause.

$$\textbf{plus}\,(X-Y,\ Y-Z,\ X-Z). \qquad (3)$$

For example, the sum of 2 and 3 can be computed by the following query.

$$?\!-\textbf{plus}\,(ssX-X,\ sssY-Y,\ Z)$$

Verify that the answer substitution gives $Z=s^5Y-Y$, which means 5. Contrast this computation with the traditional way using (1) and (2), in which addition is defined recursively and computing $m+n$ requires $m+1$ steps of logical inference. On the other hand, our addition is defined non-recursively, and a single

[†] Faculty of Engineering, Hokkaido University

addition requires only a single step of logical inference.

We can also compute difference. For example, the difference of 3 and 1 is obtained by

$?-\mathbf{plus}\,(\boldsymbol{sX-X, Z, sssY-Y})$

The answer substitution gives $\boldsymbol{Z=ssY-Y}$. Another way of computing $3-1$ might be by the following query:

$?-\mathbf{puls}\,(\boldsymbol{Z, sX-X, sssY-Y})$

In this case, however, the answer substitution gives $\boldsymbol{Z=sssY-sY}$, which is not a normal form, although it is a correct answer. The following proposition summarizes the general situation. The trivial proofs are omitted.

Proposition 2. 1 *Let m, n, p and q be non-negative integers. We define $u(x)=x$ if $x\geq 0$; and $u(x)=0$ if $x<0$. Then:*

(1) *The answer substitution for the query* $?-\mathbf{plus}\,(\boldsymbol{s}^m\boldsymbol{X}-\boldsymbol{s}^n\boldsymbol{X}, \boldsymbol{s}^p\boldsymbol{Y}-\boldsymbol{s}^q\boldsymbol{Y}, \boldsymbol{Z})$. *gives* $\boldsymbol{Z}=\boldsymbol{s}^{m+u(p-n)}\boldsymbol{X}-\boldsymbol{s}^{q+u(n-p)}\boldsymbol{X}$.

(2) *The answer substitution for the query* $?-\mathbf{plus}\,(\boldsymbol{s}^m\boldsymbol{X}-\boldsymbol{s}^n\boldsymbol{X}, \boldsymbol{Z}, \boldsymbol{s}^p\boldsymbol{Y}-\boldsymbol{s}^q\boldsymbol{Y})$. *gives* $\boldsymbol{Z}=\boldsymbol{s}^{n+u(p-m)}\boldsymbol{X}-\boldsymbol{s}^{q+u(m-p)}\boldsymbol{X}$.

(3) *The answer substitution for the query* $?-\mathbf{plus}\,(\boldsymbol{Z}, \boldsymbol{s}^m\boldsymbol{X}-\boldsymbol{s}^n\boldsymbol{X}, \boldsymbol{s}^p\boldsymbol{Y}-\boldsymbol{s}^q\boldsymbol{Y})$. *gives* $\boldsymbol{Z}=\boldsymbol{s}^{p+u(n-q)}\boldsymbol{X}-\boldsymbol{s}^{m+u(q-n)}\boldsymbol{X}$.

Let us restrict ourselves to the computation on natural numbers represented by normal forms, and assume that $n=q=0$. We also assume that $m\leq p$ to ensure that the result of the subtraction is a natural number $Z=p-m\geq 0$. Then by Proposition 2. 1, queries of the form given in (1) and (2) always yield a normalized answer for $\boldsymbol{Z}$. Queries of the form in (3) do not necessarily yield a normalized answer. However, if the answer is to be passed to a continuation which does not require the value to be in normal form, we need not normalize the answer. In particular, our program for addition does not require the normalization of the arguments. If you want to normalize the result, you can use the following equation as a rewrite rule[4] which rewrites instances of the left-hand side to the corresponding instances of the right-hand side. For example, $\boldsymbol{ssssZ-ssZ}$ would be normalized to $\boldsymbol{ssZ-Z}$.

$$\boldsymbol{sX-sY=X-Y} \qquad (4)$$

3. Conclusion

We have presented a representation of integers based on difference. We believe that this is useful at least for educational purposes, because in spite of its simplicity it demonstrates a unique feature of logic programming such as

- the computational capability of logic even for arithmetic,
- relational aspects of predicates (i. e., the definition of addition allows subtraction), and
- use of incomplete data structure to make efficient programs.

In particular, the third point is not achievable by the traditional representation.

From the viewpoint of research activity, we feel that the combination of Prolog-like logic programming and term rewriting suggested in the previous section is interesting. In the literature, a lot of approaches to such combination are proposed[1),2),5),6),10)]. Although it is out of the scope of this paper, it might be interesting as a future work to see how the proposed approaches perform computation for the example in this paper, and how they give mathematical semantics to the computation.

References

1) Ait-Kaci, H. and Nasr, R.: Integrating Logic and Functional Programming, *Lisp and Symbolic Computation*, Vol. 2, pp. 51-89 (1989).

2) Fribourg, L.: SLOG: A Logic Programming Languages Interpreter Based on Clausal Superposition and Rewriting, *Proc. Symp. Logic Programming,* pp. 172-184 (1985).

3) Furukawa, K.: Overview of Prolog, *J. IPS Japan,* Vol. 25, pp. 1313-1318 (1984), in Japanese.

4) Futatsugi, K. and Toyama, Y.: Term Rewriting Systems and Their Applications: A Survey, *J. IPS Japan,* Vol. 24, pp. 133-146 (1983), in Japanese.

5) Giovannetti, E. et al.: Kernel-LEAF: A Logic Plus Functional Language, *J. Comput. Syst. Sci.,* Vol. 42, pp. 139-185 (1991).

6) Goguen, J. A. and Meseguer, J.: EQLOG: Equality, Types, and Generic Modules for Logic Programming, DeGroot, D. and Lindstrom, G. ed., *Logic Programming: Functions, Relations, and Equations,* pp. 295-364, Prentice Hall (1986).

7) Goto, S.: Prolog Programming, *J. IPS Japan,* Vol. 25, pp. 1319-1328 (1984), in Japanese.

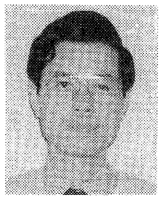
8) Manna, Z. and Waldinger, R.: *The Logical*

Basis for Computer Programming, Vol. 1 : Deductive Reasoning, Addison-Wesley (1985).

9) Nakashima, H. : The Language Prolog and Its Interpreters, *J. IPS Japan,* Vol. 23, pp. 1049-1062 (1982), in Japanese.

10) Okui, S. and Ida, T. : Lazy Narrowing Calculi, Report, University of Tsukuba, ISE-TR-92-97, pp. 1-33 (1992).

(Received May 18, 1992)
(Accepted January 19, 1993)



Masahito Kurihara was born in Japan in 1955. He graduated from Hokkaido University in 1980 and received the Ph.D. degree in engineering from Hokkaido University in 1986. Since 1990, he has been an associate professor of Information Engineering at Hokkaido University. His current research includes term rewriting systems, automated reasoning and formal methods for software engineering. In 1990, he got the IPSJ Best Paper Award Commemorating the 30th Anniversary of IPS Japan. He is a member of the Information Processing Society of Japan, the Institute of Electronics, Information and Communication Engineers of Japan, the Japan Society for Software Science and Technology, the Japanese Society for Artificial Intelligence and the IEEE Computer Society

Azuma Ohuchi was born in Japan in 1945. He graduated from Hokkaido University and received the Ph.D. degree in engineering from Hokkaido University in 1974. Since 1989, he has been a professor of Information Engineering at Hokkaido University. His current research includes systems and information engineering, artificial intelligence and medical systems. In 1990, he got the IPSJ Best Paper Award Commemorating the 30th Anniversary of JPS Japan. He is a member of the Information Processing Society of Japan, the Institute of Electronics, Information and Communication Engineers of Japan, the Institute of Electrical Engineers of Japan, the Japanese Society for Artificial Intelligence, the Society of Instrument and Control Engineers of Japan, the Operations Research Society of Japan, the Society of Medical Information of Japan, the Society of Hospital Management of Japan and the IEEE Systems, Man and Cybernetics Society.