数理解析研究所講究録
236巻 1975年 1-7

On Applying Scott's Logic to Termination Problems

S. ANDO
T. ITO
(Mitsubishi Electric Corp.)

§ 1. Introduction

Scott's logic incorporates a very powerful induction rule called fixed-point induction which realizes various types of structural induction in a uniform manner. Termination problems are also nicely treated in Scott's system as is demonstrated by the typical examples given in §2, §3, where termination means terimination under any computation rule which is a fixed-point one (see Manna 1972).

We propose the following method of presentation of termination problems. Let the domain of computation be such a primitive data type D as $D = S \cup \{UU\}$ $UU \notin S$

$\forall x, y \in S.\ x \le y \to x = y$

$\forall x \in S.\ UU \le x$

where 'S' is conceived to be the set of well-defined elements and 'UU' the undefinedness. Then, we introduce a recursively defined function ter from D to the data type of truth value TV such as

ter (UU) = UU

ter (x) = true (x ≠ UU)

which is similar to the '∂' of Milner 1972 but the recursive definition allows us to exploit the well-founded structure of D which, as is shown by Kanayama 1972, is essential in termination proof. Under these circumstances the termination of the computation which is represented by the recursive definition of the function f is expressed as

$\lambda x.\ g(x) \sqsubseteq \lambda x.\ ter(f(x))$

where g is also a recursively-defined function which never takes value 'false', which reads

"Whenever condition g is satisfied, f terminates."

§ 2. Natural number

The domain of natural number is categorically specified by the following axious $\mathcal{N}$ (Scott 1969)

1. $\lambda x. ((x+1) - 1) \equiv \lambda x. x$
2. $zero(UU) \equiv UU$
3. $0 - 1 \equiv UU$
4. $\alpha f. \lambda x.$ if zero (x) then 0 wlse $f(x-1)+1 \equiv \lambda x. x$

where '+1' '-1', means the successor and the prodecessor function respectively, and the predicate 'zero' means that the argument is zero.

Under these axious, 'ter' is given as follows.

ter $\equiv \alpha f. \lambda x.$ if zero (x) then true else f(x-1)

Example 1. Addition

$+ \equiv \alpha f. \lambda xy.$ if zero(y) then x else f(x, y-1)+1

Our goal is

$\mathcal{N} \vdash ter(x)\ \&\ ter(y) \sqsubseteq ter(x+y)$

(where $\& \equiv \lambda xy.$ if x then y else false.

it is easily obtaind that

$x \sqsubseteq true,\ y \sqsubseteq true \vdash x\&y \sqsubseteq y \wedge x\&y \sqsubseteq x$

$x \sqsubseteq y,\ x \sqsubseteq z \vdash x \sqsubseteq y\&z$)

to which the induction rule is applied with respect to the 'ter' of ter(y) generating the following two sub-goals.

1. $\mathcal{N} \vdash ter(x)\ \&\ UU \quad ter(x+y)$
2. $\mathcal{N}, \lambda ab.\ ter(a)\ \&\ tl(b) \sqsubseteq \lambda ab.\ ter(a+b) \vdash ter(x)\ \&$ (if zero(y) then true else $tl(y-1)) \sqsubseteq ter(x+y)$

1. is immediately verified. 2. is decomposed by the cases rule with respect to zero(y), into the following.

2.1 $\mathcal{N}, \lambda ab.\ ter(a)\ \&\ tl(b) \sqsubseteq \lambda ab. ter(a+b),\ zero(y) \equiv true$

$\vdash ter(x)\ \&\ true \sqsubseteq ter(x)$

which is immediately verified

2.2 $N, \lambda ab.ter(a) \& t1(b) \sqsubseteq \lambda ab.ter(a+b)$, $zero(y) \equiv false$

$\vdash ter(x) \& t1(y-1) \sqsubseteq ter(x+y)$

which we can prove by letting a and b of the assumption x and y-1 respectively and using modus-ponens rule and transitivity axiom involving

$$zero(y) \equiv false \vdash ter(x+y) \equiv ter\ ((x+(y-1))+1) \equiv ter\ (x+(y-1))$$

Example 2. Fibonacci function

$$Fib(x) \equiv \text{if } zero(x) \text{ then } 1 \text{ else if } x=1 \text{ then } 1 \text{ else } Fib\ (x-1) + Fib\ (x-2)$$

(where x=1 is an abbreviation of zero (x-1) and x-2,(x-1)-1.)

We presuppose the termination of + i.e. $ter(x)\ \&\ ter(y) \sqsubseteq ter\ (x+y)$

Let $P \equiv \lambda x.\ ter\ (Fib\ (x))$ then,

$$\text{if } zero(x) \text{ then } true \text{ else if } (x=1) \text{ then } true \text{ else } p(x-1) \& p(x-2) \sqsubseteq p(x) \quad \cdots\cdots 1)$$

Our goal being $ter(x) \sqsubseteq p(x)$

subgoal $ter(x) \sqsubseteq p(x)\ \&\ p(x+1)$ is chosen for the reason of induction.

The induction subgoals are

1. $\vdash UU(x) \sqsubseteq p(x)\ \&\ p(x+1)$ which is immediate.
2. $t1 \sqsubseteq \lambda a.p(a)\ \&\ p(a+1)$

   $\vdash$ if zero(x) then true else $t1(x-1) \sqsubseteq p(x)\ \&\ p(x+1)$

Subgoal 2 is decomposed by the cases rule into the following.

2.1 $\vdash true \sqsubseteq p(0)\ \&\ p(1)$ (case $zero(x) \equiv true$)

2.2 $\vdash t1(0) \sqsubseteq p(1)\ \&\ p(2)$ (case $x=1 \equiv true$)

2.3 $zero(x) \equiv (x=1) \equiv false,\ t1 \sqsubseteq \lambda a.\ p(a)\ \&\ p(a+1)$

$\vdash t1(x-1) \sqsubseteq p(x)\ \&\ p(x+1)$

2.1 and 2.2 is verified by proving

$$p(0) \equiv p(1) \equiv p(2) \equiv true$$

To prove 2.3, first we obtain

$$t1(x-1) \sqsubseteq p(x-1)\ \&\ p(x)$$

by applying x-1 to the assumption,

$$t1(x-1) \sqsubseteq p(x+1)$$

is obtaind from 1) and $zero(x) \equiv (x = 1) \equiv false$, therefore we obtain, with definition of &,

$t1(x-1) \sqsubseteq p(x-1)\&p(x)\&p(x+1) \sqsubseteq p(x)\&p(x+1)$ Q.E.D

Example 3. Ackermans function

Ack $\equiv$ αf. λx. λy. if zero(x) then y+1 else if zero(y) then f(x-1, 1) else f(x-1, f(x, y-1))

Let p(x, y) $\equiv$ ter (Ack (x, y)) then

p(x, y) $\equiv$ if zero(x) then ter(y) else if zero(y) then p(x-1, 1) else p(x-1, Ack(x, y-1))

Our goal is

$\vdash$ ter(x) & ter(y) $\sqsubseteq$ p(x, y)

First we resort to the induction rule with respect to the 'ter' of ter(x) which generates the following two subgoals

1. $\vdash$ UU(x) & ter(y) $\sqsubseteq$ p(x, y)

2. λab. tl(a) & ter(b) $\sqsubseteq$ p

$\vdash$ (if zero(x) then true else tl (x-1)) & ter(y) $\sqsubseteq$ p(x y)

1. is verified immediately

2. is decomposed by the cases rule with respect to zero(x) into

2.1. zero(x) $\equiv$ true $\vdash$ true & ter(y) $\sqsubseteq$ p(0 y) $\equiv$ ter(y)

which is immediately verified, and

2.2 tl(a) & ter(b) $\sqsubseteq$ p(a b); zero(x) $\equiv$ false $\vdash$ tl(x-1) & ter(y) $\sqsubseteq$ p(x y)

From 2.2 the induction rule with respect to the 'ter' of ter(y) generates the following.

2.2.1 tl(x-1) & UU(y) $\sqsubseteq$ p(x, y)

2.2.2 λab. tl(a) & ter(b) $\sqsubseteq$ p, zero(x) $\equiv$ false, λcd. tl(C-1) & t2(d) $\sqsubseteq$ p

$\vdash$ tl(x-1) & (if zero(y) then true else tl(y-1) $\sqsubseteq$ p(x y)

2.2.1 is immediate, 2.2.2 is decomposed by the cases rule with respect to zero(y) into the following 2.2.2.1 and 2.2.2.2.

2.2.2.1

zero(y) $\equiv$ true, zero(x) $\equiv$ false λab. tl(a) & ter(b) $\sqsubseteq$ p
tl(x-1) & treu $\sqsubseteq$ p(x y)

PROOF λaλb. tl(a) & ter(b) $\sqsubseteq$ p $\vdash$ tl(x-1) & ter(1) $\sqsubseteq$ p(x-1, 1)
zero(x) $\equiv$ false, zero(y) $\equiv$ true $\vdash$ p(x-1, 1) $\equiv$ p(x, y)

By applying modus-ponens and transitivity several times to these, we can obtain 2.2.2.1

2.2.2.2

zero(y) $\equiv$ zero(x) $\equiv$ false, λab. tl(a) & ter(b) $\sqsubseteq$ p, λcd. tl(c-1) & t2(d) $\sqsubseteq$ p

$\vdash$ tl(x-1) & t2(y-1) $\sqsubseteq$ p(x y)

PROOF

From

λab, tl(a) & ter(b) $\sqsubseteq$ p $\vdash$ tl(x-1) & p(x, y-1) $\sqsubseteq$ p(x-1, ack(x, y-1)) and zero(x) $\equiv$ zero(y) $\equiv$ false $\vdash$ p(x-1, ack(x, y-1)) $\equiv$ p(x y) several applications of modus ponens and transitivity gives

assumptions $\vdash$ tl(x-1) & p(x, y-1) $\sqsubseteq$ p(x y)

And several application of modus ponens and transitivity to this and

λcd. tl(c-1) & t2(d) $\sqsubseteq$ p $\vdash$ t1(x-1) & t2(y-1) $\sqsubseteq$ p(x , y-1)

and the definition of '&' gives

assumptions $\vdash$ tl(x-1) & t2(y-1) $\sqsubseteq$ p(x y) Q. E. D

§ 3. List

The axioms of list is similar to that of natural number

1. λxλy. car(cons (x y)) $\equiv$ λxλy. x
2. λxλy. cdr(cons (x y)) $\equiv$ λxλy. y
3. if atom(x) then car(x) else UU $\equiv$ UU
4. if atom(x) then cdr(x) else UU $\equiv$ UU
5. atom (NIL) $\equiv$ true
6. αf. λx. (if null(x) then NIL
   else if atom(x) then x
   else cons (f(car(x))f(cdr(x))) $\equiv$ λx. x

In this case the definition of ter is

ter $\equiv$ αf. λ x . if atom(x) then true
    else f(car(x)) & f(cdr(x))

Example   Primitive Recursive Function of lists

f1(x, y1, . . . , yk)
  ≡ if atom(x) then g2(x, y1, . . . , yk)
    else h1(x, f1(car(x), y1, . . . , yk), f2(cdr(x), y1, . . . , yk),
        y1, . . . , yk)

f2(x, y1, . . . , yk)
  ≡ if atom(x) then g2(x, y1, . . . , yk)
    else h2(x, f1(car(x), y1, . . . , yk), f2(cdr(x), y1, . . . , yk),
        y1, . . . , yk)

From the above definition we can prove the termination of f1 and f2 under the assumption that g1, g2, h1, and h2 terminates.

§ 4. Conclusion

Termination of a computation is always equivalent to downward monotonicity of that computation process with respect to some well-founded relation (Y. Kanayama 1972). That is why structural induction is essential in termination problems. This feature is also explicit in Floyd-Manna's first-order method because it always resorts to some axiom scheme of structural induction, say, mathematical induction in the domain of natural number. The success of the proof depends totally upon the choice of the well-founded relation, which is not at all uniform and relies heavily upon intuition. On the contrary, in our method we may not be concious of the well-founded structure explicitly. It is inplicitly incorporated in the recursive definitions of conditions on the arguments. Fixed-point induction with respect to that condition function automatically generates subgoals which reflects the well-founded structure that is needed to establish the termination. Of course, in case very complicated well-founded relation is involved, some ingenuity is necessary in designing the condition function. But in any case, our formal frame-work is simple enough to make the whole reasoning transparent, and helps to guide our intuition. It is a good candidate for the formalism to be adopted when trying to mechanize the termination problem.

References

Kleene 1952 Introduction to Metamathematics. North Holland

Scott 1969 A Type-theoretical alternative to Unpublished

Milner 1972 "Implimentation and -" proc. of A.C.M conf. New Mexico

Manna 1972 "Inductive methods - " the same as above

Kanayama 1972 "Algebraic Properties of Program" proc of 1st Japan U.S.A computer conf.