数理解析研究所講究録
第236巻 1975年 21-30

Dynamics of System Program
-- Progressive Induction --

Katsuhiko KAKEHI (University of Tokyo)*
Takakazu SIMAUTI (Rikkyo University)

* from 1974, Rikkyo University

0. Introduction

Theoretical tools have been developed for the correctness proofs of programs, which are of 'single control'. To prove correctness of 'cooperating programs', those tools are, of course, essential. But, adding to them, there should be some other tools to cope with the cooperating mechanism.

As to the single control program, the termination of it is a key point to prove its correctness. On the other hand, the correctness of the cooperating programs includes termination-free, or 'dead-lock-free', property.

The correctness of a set of programs which, as a whole, take care of PTR (Paper Tape Reader) input will be proved with a mechanism called 'progressive induction'.

1. Programs to be proved

Fig.1 shows the PTR-input routines in the system called SPS, which are resident in a conventional mini-computer HITAC-10. On the machine, they are coded in an assembly language. But in Fig.1 they are written in a higher-level language for the sake of readability.

They perform their own jobs with the interruption mechanism of the machine. Program A is a interruption handling routine and program B a input routine. When a character from PTR is requested, the character is passed to a user program by a call of B. Program C is an initiation routine and called by a user program on its first step.

```
A :
    A1: begin
          A2: T[t] ← read ptr ;
          A3: t ← t+1 mod m ;
          A4: e ← false ;
          A5: f ← (h=t) ;
          A6: if ¬f then start ptr
    A7: end
    A8:

B :
    B1: if e then go to B1 ;
    B2: ch ← T[h] ;
    B3; begin
          B4: if f then start ptr ;
          B5: h ← h+1 mod m ;
          B6: f ← false ;
          B7: e ← (t=h)
    B8: end
    B9:

C :
    C1: t ← 0 ; h ← 0 ;
    C2: f ← false ; e ← true ;
    C3: start ptr
    C4:
```

Fig. 1

PTR is driven by an istruction start ptr. When the next character on the paper tape is read into the hardware buffer, PTR will stop and an interruption will be caused. By the interruption, the control is transferred to A at once, except when the control is in an interruption-inhibited part of programs, in which case the transfer of control will be postponed until it goes out of the part.

After the completion of A, the control will be returned to the position where the interruption was caused. In A the character read is taken out by an instruction read ptr from the buffer.

In Fig.1, interruption-inhibited parts are enclosed with begin and end. To make the matter clear, we assume that the statement in the language consists with an inseparable action and any interruption may occur only at the end of statement.

Programs A,B and C have their own variables:

T : a round-robin buffer of characters, whose size is m,

h : a pointer to the first character in T.

t : a pointer to the first empty cell in T.

f : a flag indicating whether T is full or not.

e : a flag indicating whether T is empty or not.

User programs cannot access to those variables. For communication between the routines and user programs, a variable ch is used which will contain the resulting character.

2. Proof of the termination-free property

At first, we put the definition of the correctness of the routines.

Assumption:

(a1) A user program calls C at its first step.

(a2) On PTR, a paper tape is set on which a sequence of characters $c_0, c_1, c_2, \ldots$ is punched.

Correctness: (with (a1) and (a2))

On the i-th call($i \geq 0$) of B,

(c1) B completes its job, and

(c2) ch contains the character c_i.

Now, let's begin to prove (c1). It is easy to see that A and C are loop-free and B has only one loop, which is controled by the variable e. To show (c1), therefore, it is necessary to handle with the properties of the variables, which may well be a part of (c2). (c2), at the same time, is depend on the property (c1).

Let's change the view point.

Hypothesis:

(h1) After the competion of B, B will be called again in a finite period of time.

Let t_i and t_i' be the time of the i-th entry to and the i-th exit from A. Then we will prove:

(p1) With (h1), if t_i exists, there exists t_{i+1}.

proof

(1) A has no loops, the existence of t_i' is evident when t_i exists. On t_i', e is set to false and if f is set to false, PTR is driven. So if f=false, by the hardware mechanism, t_{i+1} exists.

(2) Now, consider the case f=true on t_i'. We will complete the proof by case analysis where the control is returned to on t_i'.

(case 1) to B1:

e=false and the loop is by-passed into (case 2).

(case 2) to B2 or B3:

The begin -- end part is performed and f=true makes PTR driven. Thus the existence of t_{i+1} is proved.

(case 3) to B9 or to a point in the user program:

With (h1), this case is treated as if (case 1). ■

The existence of t_0 is evident from (a1) because PTR is driven by C. Thus we have:

(p2) Under the assumptions (a1) and (h1), there exists the time t_i for every $i \geq 0$, or equivalently, the routines are termination-free.

The proof given above is based on an induction which we call 'progressive induction'. The idea is summerized as follows:

Progressive Induction

Let P_k be a program point with the assertion A_k on it.

(1) Assume that at the time t, the control is on P_k and A_k holds. Then, the existence of t' $(t'>t)$ when the control is again on P_k is proved and A_k holds on t'.

(2) The existence of such t with A_k is derived from the initial conditions of the programs.

With (1) and (2), we claim that the programs are termination-free.

3. Proof of the correctness

Having been proved the termination-free property, let's prove (c2) with an application of the assertion method to it.

Before going to prove it, we slightly modify the programs into their equivalent form, which is shown in Fig.2. In Fig.2, it is easy to see that on the i-th entry to B, h has the value i, and that on the i-th entry to A, or on the interruption for the i-th character, t has the value i.

We put assertions to the programs as seen in Fig.2. The proof should be completed with:

(1) to verify that in each program, the assertions attached satisfy the verification conditions;

(2) to show that the assertion on C4 implies one on B1, one on B9 implies one on B1, and that each one on B1,B2,B3 and B9 implies one on A1, and that one on A8 implies each one on B1,B2,B3 and B9.

Those are easily verified and we can now deduce from them that $ch=c_i$ holds at the i-th exit from B.

A :

```
A1: {P∨Q, G} begin
                A2: T[t mod m] ← read_ptr ;
                A3: t ← t+1 ;
                A4: e ← false ;
                A5: f ← ( h+m=t ) ;
                A6: if ¬f then start_ptr ;
A7: {P∨R, G} end
A8: {P∨R, G}
```

B :

```
B1: {P∨Q∨R,G} if e then go_to B1 ;
B2: {P∨R, G} ch ← T[h mod m] ;
B3: {P∨R, G} begin
                B4: if f then start_ptr ;
                B5: h ← h+1 ;
                B6: f ← false ;
                B7: e ← ( t=h ) ;
B8: {P∨Q, G} end
B9: {P∨Q∨R, G}
```

C :

```
C1: {true}   t ← 0 ;  h ← 0 ;
C2:          f ← false ;  e ← true ;
C3:          start_ptr
C4: {Q, G}
```

where

```
P :   h<t<h+m, ¬f, ¬e
Q :   h=t,     ¬f,  e
R :     t=h+m,  f, ¬e
```

G : $\bigwedge_{i=h}^{t-1} T[i \bmod m] = c_i$

Fig. 2

4. Conclusion

We have introduced a new tool 'progressive induction', and correctness of the example programs, which is a part of an actual system, is proved with it.

Application to various examples and construction of a formal system based on the idea would follow. Anyhow, the progressive induction, we believe, will perform the essential role on proofs about cooperating programs.

ACKNOWLEDGEMENT: We wish to thank Professor S. Igarashi for his kind advices.