数理解析研究所講究録
第470巻 1982年 97-103

モニタの検証

数理解析研究所　柴山　悦哉

John H. Howard. "Proving Monitors". CACM vol. 19 pp 273-279 の内容を以下に紹介する。

1.

モニタとは、コンカレント・プログラムのプロセス間で共有される データ、プロシジャの集まりである。

モニタで定義されたデータは、同じモニタで定義されたプロシジャ及び、モニタの初期化を行う部分からしか、参照、変更できないものとする。したがって、モニタは1つの抽象データ型とみなせる。

モニタ・コールは、同時に1つしか実行できないものとする。これによって、モニタで定義されたデータを排他的に使用することができる。

モニタ・コールを行っているプロセスの実行を、一時中断する命令（cond.wait）と、中断されていたモニタ・コールの実行を再開する命令（cond.signal）が用意されている。ここで condは、プロセスの待ち行列である。

モニタに関しては、Hoare [1]に詳しく述べられている。

2.

図1は、モニタを用いて記述した bounded buffer である。
$\langle x_1, x_2, \cdots, x_n \rangle$ は $x_1, \cdots, x_n$ よりなる列をあらわし、A@Bは2つの列 A, B を結合したものを表す。A.first は列の先頭の元。A.rest は、AからA.firstを除いたものである。

Sが長さ高々Nの bufferで、notfull, notempty はコンディション変数 [1] である。

Hoare 式のルールを用いて検証を行う場合、検証したい性質を、Hoare 式の論理式を用いて記述する必要がある。ところが、図1の列において

"bufferからremoveされた要素は、bufferにappendされた要素の前の方で、残りの要素の列がSである。"

という性質の正当性を検証しようとすると、たちまち行きづまってしまう。すなわち、図1にあらわれるプログラム・ステートメントやプログラム変数を、どう組み合わせてみても、上記の性質を記述するHoare式の論理式が書けないのである。

これは、モニタが1つの独立したプログラムではなく、他のプログラムから呼ばれるプロシジャの集まりにすぎないことに起因する。履歴に関する情報を、モニタは完全に記憶し

ているわけではない。

そこで、図2のように A,Rという2つの履歴に関する補助変数（history variable）を導入する。すると、前述の性質は "A=R@$" と記述することができる。

尚、このA,Rは、検証にのみ用いる変数である。history variableについては、[3]

3.

モニタのプログラム中に、cond.wait, cond.signal が出現する時、Hoare式の証明方法をどのように拡張したらよいであろうか。

Hoareは次のような公理を提唱している。[1]

$$J\ \{\text{cond.wait}\}\ J\&B$$

$$J\&B\ \{\text{cond.signal}\}\ J$$

ここで、Jはコントロールの流れが変わる時（モニタ・コールの前後，cond.wait,cond.signal の前後）に成りたつべき不変式である。

cond.wait は、モニタの状態がある条件を満たすようになるまで、待っている命令であり、Bはそのための条件と考えられる。したがって、Bはコンディション変数の数だけ存在する。

この新しい公理を使って、証明すべきは、以下のものである。

true {モニタの初期化部分} J

J {モニタの各プロシジャ} J

このHoareの提唱した公理は有用ではあるが、問題点がないわけではない。たとえば、J{cond.wait}J∧Bという論理式は、cond.wait 命令の実行後にBが満たされることは述べているが、cond.wait 命令の実行前にBが満たされていたかどうかという点については何も述べていない。

そこで、次のような公理が考えられる。

J∧E{cond.wait}J∧B

J∧B{cond.signal}J∧E

ここで、Eはモニタ・ユールの終了後に成りたつ条件で、しかも J∧E⊃TBが成りたつようなものとする。

この公理を用いると、待つ必要のある時(すなわちBが成りたたない時)にのみ cond.wait が実行されるという性質を記述できる。

この時、証明すべきは以下のものである。

true {モニタの初期化部分} J∧E

J∧E{モニタの各プロシジャ}J∧E

I=J∧E, C=J∧Bとすると、公理と証明すべきこと

は次のようになる。

$$I\{cond.wait\}C$$
$$C\{cond.signal\}I$$

$$true\ \{モニタの初期化部分\}\ I$$
$$I\ \{モニタの各プロシジャ\}\ I$$

4.

今までは、論理式の中にコンディション変数の状態を表わすような式が出現しないと暗に仮定してきた。もし、そのような式が現れた場合、cond.wait を3つの部分に分解して考える。すなわち、

① 待ち行列 cond にプロセスを入れる。

② cond.signal が実行されるまで待つ。

③ 待ち行列 cond からプロセスを取り出す。

したがって、この場合の証明ルールは次の通り。

$$\frac{P\{①\}I\quad I\{②\}C\quad C\{③\}R}{P\{cond.wait\}R}$$

ここで、①、③については、コンディション変数の状態は変わるが、他の変数の値は変わらないことに注意する。また②については、すでに議論した。

5. 問題点

この論文("Proving Monitors")は、モニタ証明のための2つの技法を与えている。1つはhistory variableの使用で、もう1つはwait, signal命令にたいする、公理的な定義である。これらの方法が、いくつかの問題に対して、有用であることは確かであるが、モニタの証明体系として完全であるわけではない。またデッド・ロックの問題も残されたままである。

References

1. Hoare,C.A.R. Monitors:an operating system structuring concept. CACM 17,,10 (Oct. 1974), pp.549-557. Corrigendum, CACM 18, 2(Feb. 1975), p 95.
2. Hoare,C.A.R. An axiomatic basis for computer programming. CACM 12, 10(Oct. 1969), pp.576-583.
3. Clint,M. Program proving:coroutines. Acta Informatica 2 (1973), pp.50-63.

```
bounded buffer:monitor

begin   S:sequence of message;
        notfull,otempty:condition;
        comment N is the maximum length of S;

procedure append(x:message);
begin   if S.length=N then notfull.wait;
        S:=S@⟨x⟩;
        if notempty.queue then notempty.signal;
end;

procedure remove(result x:message);
begin   if S.length=0 then notempty.wait;
        x:=S.first;
        S:=S.rest;
        if notfull.queue then notfull.signal;
end;

        S:=⟨ ⟩;
end
```

図1

```
bounded buffer:monitor

begin   A,R,S:sequence of message;
        notfull,notempty:condition;
        comment N is the maximum length of S;

procedure append(x:message);
begin   if S.length=N then notfull.wait;
        A:=A@⟨x⟩;
        S:=S@⟨x⟩;
        if notempty.queue then notempty.signal;
end;

procedure remove(result x:message);
begin   if S.length=0 then notempty.wait;
        x:=S.first;
        S:=S.rest;
        R:=R@⟨x⟩;
        if notfull.queue then notfull.signal;
end;

        A:=R:=S:=⟨ ⟩;
end
```

図2