A Tempura Interpreter

3Q-4

Masahiro Fujita Shinji Kono Hidehiko Tanaka
Tohru Moto-oka
(Faculty of Engineering University of Tokyo)

I Introduction

Tempura [1] is not a *food* but a powerful *logic programming language* based upon Interval Temporal Logic (ITL), which is originally developed by Moszkowski and Manna [1]. ITL is considered to be an extension of Linear Time Temporal Logic [2] which is used in our logic design assistance system [3]. ITL has several kinds of time-related predicates which are useful for describing and simulating both software and hardware. We implement an interpreter for Tempura on C-Prolog (C-Prolog has been developed at Edinburgh University). Tempura includes pure Prolog and has a very similar syntax to Prolog. First we briefly introduce ITL and Tempura, and second we present two examples of Tempura: one is for software and the other is for hardware.

II Tempura

Interval Temporal Logic is one of modal logics, which has a discrete time concept. We briefly explain its syntax and semantics; for execution of Tempura can be regarded as a reasoning process of ITL. Truth values of ITL formulae cannot be determined in a state but determined in an interval that is successive states of finite length.

We introduce three time-related predicates: *next*, *chop* and *always*. '@' is a *next* operator.

@p (p is ITL formula)

means that p is true at next clock or the present time is the end of the interval (no next time exists). *Chop* '&&' is an operator that separates an interval into two sub-intervals.

p && q

is true if p is true in the former sub-interval and q is true in the remaining sub-interval. Partition point of an interval is arbitrary. The last temporal operator we introduce here is *always* operator '#'.

#p

is true in any arbitrary sub-intervals of that interval.

Tempura also has variables, functions and logical operators. Identifiers beginning with a capital letter are regarded as variables. We can use global variable that is an atom prefixed by '*'. Values of logical and global variables may change as time advances.

Tempura formula is asserted in the same way as C-Prolog except using ':=' instead of ':-'. For example,

head := body.

We can use a logical variable contained in a body that is not included in its head, as an existential quantified variable.

Here are useful macros of Tempura and some special operators.

'if a then b else c': if-then-else structure

'A<==B, B==>A': temporal assignment

Temporal assignment enables to transfer the value of B at the beginning of the interval to A at the end of that interval.

```
philosopher(Lfork,Rfork,Sta,Id):=Sta=think,
        @Sta=pick_l.
philosopher(Lfork,Rfork,Sta,Id):=Sta=pick_l,
        ((Lfork=Id,@Lfork=Id,@Sta=pick_r)
        ; @Sta=pick_l).
philosopher(Lfork,Rfork,Sta,Id):=Sta=pick_r,@Lfork=Lfork,
        ((Rfork=Id,@Rfork=Id,@Sta=eating)
        ; @Sta=pick_r).
philosopher(Lfork,Rfork,Sta,Id):=Sta=eating,@Rfork=Rfork,
        @Sta=free_l.
philosopher(Lfork,Rfork,Sta,Id):=Sta=free_l,
        @Sta=free_r.
philosopher(Lfork,Rfork,Sta,Id):=Sta=free_r,
        @Sta=think.

dining_philosopher(F,S):=
        [Fa,Fb,Fc,Fd,Fe]=F,[Sta,Stb,Stc,Std,Ste]=S,
       #philosopher(Fa,Fb,Sta,a),
       #philosopher(Fb,Fc,Stb,b),
       #philosopher(Fc,Fd,Stc,c),
       #philosopher(Fd,Fe,Std,e),
       #philosopher(Fe,Fa,Ste,f).
```

Figure 1 Dining philosopher program

III Examples

Dining Philosophers

Figure 1 is a sample program of Dining Philosophers. 'Philosopher' predicates has three arities. First and second arguments mean his forks. Next argument is a state of a philosopher that cycles six states: thinking, having the left fork, having both forks, eating and having right fork. Here we write six clauses for a philosopher that corresponds to his states. We may, however, write it with only a single clause using 'if statement'. Last argument is used for an identifier to distinguish fork users. Each predicate describes state transition of a philosopher. 'Dining_philosopher' predicate combines five philosophers. Variable F is a set of fork's state, and S is a set of philosopher's state. Fork state is shared in two philosophers.

Hardware description

Prolog is known to a good language for hardware logic design description. Because of the lack of a concept of time, however, it expresses logic circuit delay in rather rough way. As compared to this, Tempura basically contains clock, so we can write it in more sophisticated way.

Figure 2 (a) is an algorithm that calculates approximation of the square root of A^2+B^2, and figure 2 (b) is a corresponding Tempura program. Figure 3 shows an implementation of the hardware for it. Figure 4 is a Tempura program for figure 3. The execution of 'magasync' will show the pipeline execution is correctly done.

V Conclusions

Since Tempura is founded on formal logic, it is a promising tool for formal verification and synthesis, which is partially reported in [4]. Also, Tempura is a delicious extension of Prolog for iteration and state transition.

References

[1] Stanford Univ. Rep. No.STAN-CS-83-969.

[2] Stanford Univ. Rep. No.STAN-CS-81-877.

[3] M. Fujita, Doctoral Dis., Information Eng., Univ. of Tokyo, 1984.

[4] IECEJ, EC84, Feb. 1985.

$$\text{let } G = \max \left| \frac{A}{B} \right|$$

$$L = \min \left| \frac{A}{B} \right|$$

$$\sqrt{A^2+B^2} \approx \max \left| \frac{G}{\frac{7}{8}G + \frac{1}{2}L} \right|$$

(a) Algorithm for calculating the magnitude of vector A and B

```
magasync_algorithm(A,B,Res):=
        length(1),
        (if A < 0 then Aab <= - A
                  else Aab <= A ),
        (if B < 0 then Bab <= - B
                  else Bab <= B )
    &&  length(1)
        if Aab > Bab
           then (G <= Aab, L <= Bab)
           else (G <= Bab, L <= Aab)
    &&  length(1)
        Sqs <= G * 7 / 8 + L / 2 , G <= G
    &&  length(1)
        if G > Sqs
           then Res <= G
           else Res <= Sqs,length(1).
```

(b) Tempura program for (a)

Figure 2 Algorithm for calculating the magnitude of vector A and B

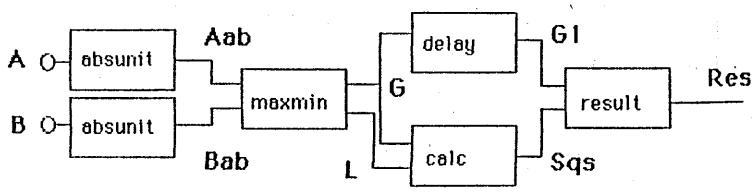

Figure 3 Data path for figure 2

```
magasync(A,B,Res):=
        #abs_unit(A,Aab),
        #abs_unit(B,Bab),
        #maxmin(Aab,Bab,G,L),
        #calc(G,L,Sqs),
        #delay(G,Gl),
        #result(Gl,Sqs,Res).
delay(G,Gl):=
        length(1),
        Gl <= G.
abs_unit(I,O):=
        length(1),
        if I < 0 then O <= -I else O <= I.
maxmin(I1,I2,O1,O2):=
        length(1),
        if I1 > I2 then (O1 <= I1, O2 <= I2)
                   else (O1 <= I2, O2 <= I1).
calc(I1,I2,O):=
        length(1),
        O <= I1 * 7 / 8 + I2 / 2.
result(I1,I2,O):=
        length(1),
        if I1 > I2 then O <= I1 else O <= I2.
```

Figure 4 Tempura program for figure 3 (a)