Development of an ML-based Verification Tool for Timed CSP Processes

Takeshi Yamakawa, Tsuneki Ohashi
and Chikara Fukunaga[Presenter]
Tokyo Metropolitan University

Contents

- CSP and Timed CSP
- New Operators for Timed CSP
- Timed CSP Explorer
  - Implementation of (untimed) CSP processes with ML functions
  - Extension for Timed CSP
  - Execution of Timed CSP processes
- Refinement with Timed CSP explorer
  - Refinement of Fischer's algorithm
    (Exclusive control for shared memory)
- Summary and Outlook

Activity of Our group (TMU)

- Development of TPCORE (IP core of general purpose processor) (CPA2004)
  - compatible with transputer assembly language
  - Programs written in (Inmos) occam can be executed
  - 50MHz (FPGA) and 130MHz (ASIC)
- TPCORE2 (one of 4 link I/Fs→DS-link for virtual link and IEEE1355) , 1355 router (PDNS2010) and FPUs
- Application of these processors in embedded system
  - Real time programming
  - Need a Timed CSP verification tool ← Motivation for this study

CSP and Timed CSP

- CSP: Process transition with events only (event transition)

$$P \xrightarrow{\mu} P'$$

- Timed CSP: Consideration of process transition with time (evolution transition)

$$Q \overset{t}{\rightsquigarrow} Q'$$

New Operators for Timed CSP(1)

- Timed Event Prefix $a@u \longrightarrow P$

  Occurrence time of event *a* after start of the process is recorded in timed variable *u*. Once *a* is occurred, *P* can refer the value recorded in *u*.

$$(a@u \rightarrow P) \xrightarrow{a} P[0/u]$$

Reset *u* at event *a*

$$(a@u \rightarrow P) \overset{d}{\rightsquigarrow} (a@u \rightarrow P[u+d/u])$$

a is not occurred after time *d* has passed

$$(a@u \rightarrow out!u \rightarrow SKIP) \overset{5}{\rightsquigarrow}$$

$$(a@u \rightarrow out!(u+5) \rightarrow SKIP) \xrightarrow{a}$$

$$(out!(0+5) \rightarrow SKIP) \xrightarrow{out.5} SKIP$$

New Operators for Timed CSP(2)

- Timeout $P \overset{d}{\triangleright} Q$

  Choice with an event in *P* and the elapsed time from start of the process; for example,

  $$(a \rightarrow P) \overset{d}{\triangleright} Q$$

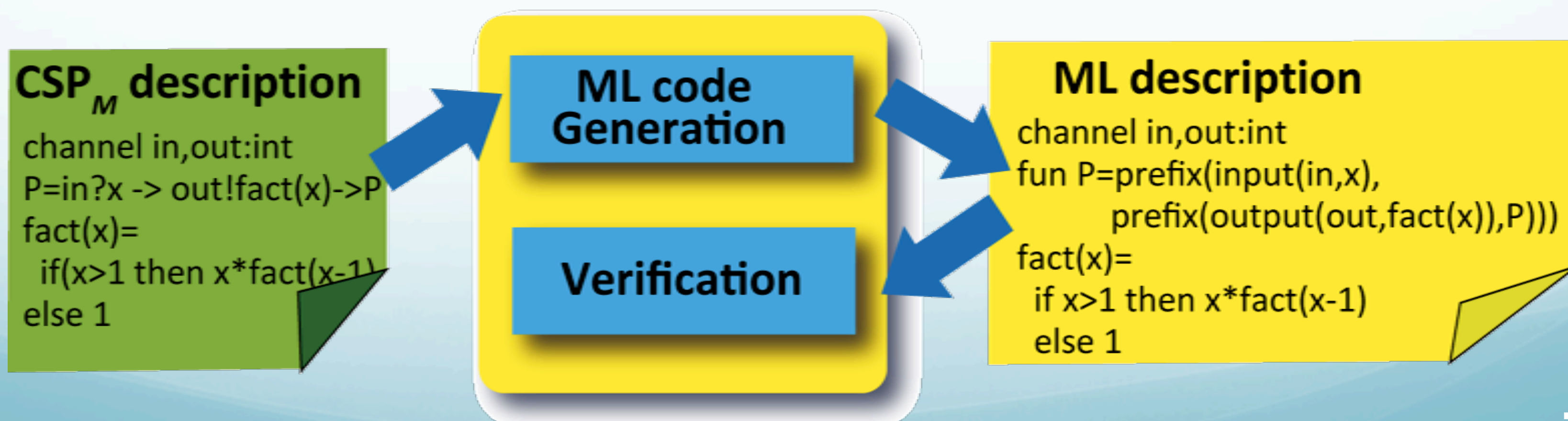
  the process *P* will follow after the event *a* if *a* takes place in *d* time unit otherwise the process *Q* will start.

- Timed Interrupt $P \triangle_d Q$

  Unless the process *P* is finished until *d*, it is terminated and the process *Q* will follow, but *P* is finished within *d*, *Q* will not be invoked to start

Timed CSP Explorer

- A program described in ML (Meta Language)
- Lexical analysis, parsing of Timed CSP Processes described in CSP_M and generation of ML functions
- Execution of the ML functions
- Refinement analysis (timed trace/time-wise trace) for safety (trace based) verification



Implementation of CSP processes with ML (1)

- Definition of additional ML datatypes
  - `datatype process = Proc of (event -> process) | Stop | Skip | Bleep ;`
    - `process` has 4 constructors
    - `Proc` is a function to take an `event` as argument and return a `process`
  - `datatype event = Event of string*chanType ;`
    - `event` as a tuple of (string, chanType)
    - Time information for Timed CSP processes will be added afterwards
  - `datatype chanType = Int of int | Seq of int list | String of string | Any | None ;`

Implementation of CSP processes with ML (2)

- Example of a CSP Operator ; Event Prefix $a \longrightarrow P$

```
prefix(Event(ch,v),P:process)=
let
  fun temp(Event(ch',v')) if ch=ch' andalso v=v' then P else Bleep
in
  Proc temp
end
```

- Execution of processes ; $P = b \longrightarrow (a \longrightarrow SKIP)$
  - execution of the process with single step

    ```
    fun run(Proc temp)
    = temp;
    ```

```
-
- val P= prefix(Event("b",None),prefix(Event("a",None),Skip)) ;
val P = Proc (fn,[Event (#,#)]) : process
- val P1=run(P) ;
b
val P1 = fn : event -> process
- val P2=run(P1(Event("b",None))) ;
a
val P2 = fn : event -> process
- val P3=run(P2(Event("a",None))) ;
tick.
val P3 = fn : event -> process
-
- 
```

Extension for Timed CSP

- Addition of Time constructor to datatype event

```
datatype event
= Event of string*chanType | Time of int ;
```

- Modification of ML functions for CSP operators extended by time concept

- Implementation of Timed CSP operators
  - fun tprefix(Event(ch,v), Time d, P:process)
  - fun timeout(P:process, Time d, Q:process)
  - fun tinterrupt(P:process, Time d, Q:process)

TOKYO METROPOLITAN UNIVERSITY 首都大学東京

Execution of Timed CSP processes (1)

- Process with timed interrupt

$$test = out!10 \rightarrow$$
$$(\ (a \rightarrow SKIP\ {}_{\{a\}}\|_{\{b\}}\ b \rightarrow SKIP)\ \triangle_{10}\ c \xrightarrow{5} SKIP\ )$$

- ML expression & execution

```
fun test() =
 prefix(Event("ou
       concurren
             p a
       ),
       Time 10,
       tprefix(E
 ) )
```

```
prefix(Event("b",None),Skip)
```

```
- exec(test()) ;
event: out.10
out.10
event: a b 10time
a
event: b 10time
b
tick
event: 10time
10time
event: c
c
event: 5time
5time
tick
*
*
*
```

```
event: b 10time
b
event: a 10time
a
tick
event: 10time
10time
event: c
c
event: 5time
5time
tick
*
*
*
```

```
event: 10time
10time
event: c
c
event: 5time
5time
tick
*
*
*

finished.
val it = () : unit
- 
```

Execution of Timed CSP processes (2)

TOKYO METROPOLITAN UNIVERSITY
首都大学東京

- Process with Timed event prefix and timeout

$$HELEN = (\ meet \xrightarrow{60} work \rightarrow SKIP) \overset{30}{\triangleright}\ work \rightarrow SKIP$$

$$CARL(d) = WAIT\ d \,;\, (\ (meet \xrightarrow{60} home \rightarrow SKIP) \overset{45}{\triangleright} home \rightarrow SKIP)$$

$$test = HELEN \parallel (CARL(15) \sqcap CARL(40))$$

- ML

```
val
fun
    end
(Event("home"
val test =
    share(HELEN,[Event("meet",None)], internal(CARL(15),CARL(40)))
```

```
- use "para.ml" ;
[opening para.ml]
val HELEN = Proc (fn,[Event (#,#),Time 30]) : process
val CARL = fn : int -> process
val test = Proc (fn,[Event (#,#),Event (#,#),Time 30]) : process
val spec = fn : chanType -> process
val Meet = fn : chanType -> process
val it = () : unit
- exec(test);
event: L R 30time
L
event: 15time
15time
event: meet 15time
meet
event: 60time
60time
event: work home
work
event: home
home
tick
*
event: home
home
event: work
work
tick
*
*
*
```

```
event: 15time
15time
event: work 30time
work
event: 30time
30time
event: home
home
tick
*
*
event: 30time
30time
event: work home
work
event: home
home
tick
*
event: home
home
event: work
work
tick
*
*
*
*
*
```

```
event: R 30time
R
event: 30time
30time
event: work 10time
work
event: 10time
10time
event: 45time
45time
event: home
home
tick
*
*
*
event: 10time
10time
event: work 45time
work
event: 45time
45time
event: home
home
tick
*
*
```

```
event: 45time
45time
event: work home
work
event: home
home
tick
*
event: home
home
event: work
work
tick
*
*
*
*
*
```

```
event: 40time
40time
event: work 45time
work
event: 45time
45time
event: home
home
tick
*
*
event: 45time
45time
event: work home
work
event: home
home
tick
*
event: home
home
event: work
work
tick
*
*
*
*
*

finished.
val it = () : unit
- 
```

Refinement with Timed CSP Explorer

- Trace refinement (Safety verification with Timed CSP traces)

$$SPEC \sqsubseteq_T IMP \iff traces(SPEC) \supseteq traces(IMP)$$

- Trace timewise refinement

$$SPEC \;{}_T\!\sqsubseteq_{TF} IMP$$
$$\iff \forall (s, X) \in TF[IMP] \cdot \sharp s < \infty \Rightarrow strip(s) \in traces(SPEC)$$

  - $strip(<(15, meet), (45, work), (45, home)>) = <meet, work, home>$

- refine(*Spec*,*Imp*)
  - All the traces generated by Imp process are followed with Spec
  - If Spec follows them all w/o Bleep, refinement is established

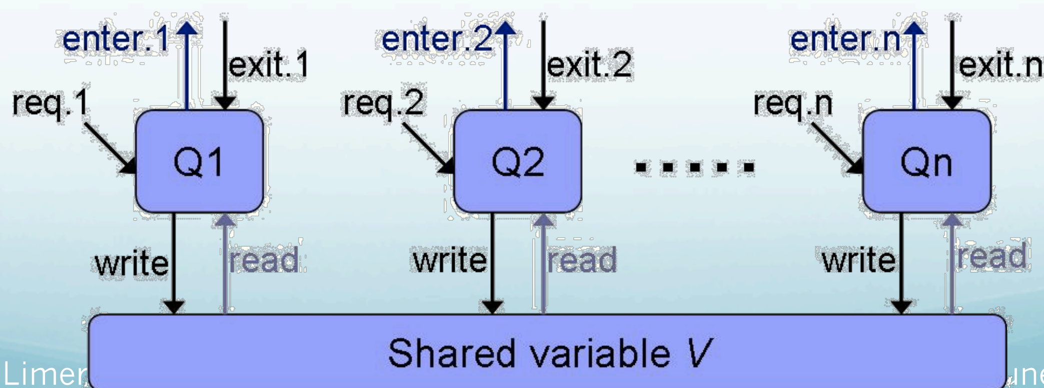
Exclusive control of shared resources

- Exclusive access control for the critical region in shared resources
  - The critical region can be accessed by any number of processes but only one can access at one time
  - If a process accesses the region the other process should wait this access is over
  - We need an efficient and safe mechanism to control the processes in as each process accesses the critical region as if only one process occupies it.
- As one of candidates to be used for the exclusive control is called Fischer's algorithm
  - Idea of this example comes from S. Schneider, "Concurrent and Real-time systems The CSP Approach" (2000)

Fischer's algorithm with CSP

- $Q(i) = req.i \rightarrow read?x \rightarrow$
  $\quad$ if $x \neq 0$ then $SKIP$
  $\quad$ else $write!i \rightarrow enter.i \rightarrow exit.i \rightarrow STOP$

  $QS = Q(1) \mid\mid\mid Q(2) \mid\mid\mid \cdots \mid\mid\mid Q(n)$
  - read shared memory and if it is 0, write I (process ID) and enter (occupy)
  - Independent operation
- $V(value) = (write?x \rightarrow V(x)) \,\square\, (read!value \rightarrow V(value))$
- $FIS = QS \,[|\, read, write \,|]\, V(0)$
  - V value must be initially zero
- Specification: *enter.i* must be followed with *exit.i* before *enter.i'*



Refinement of Fischer's algorithm

- Trace verification with Timed CSP explorer for this algorithm with n=2
  - as a Specification for this algorithm,

$$SPEC = (enter.1 \rightarrow exit.1 \rightarrow SPEC \, \Box \, enter.2 \rightarrow exit.2 \rightarrow SPEC)$$

  - and hided the events *read, req, write* from *FIS*

$$FIS = FIS \setminus \{| \, read, write, req \, |\}$$

  - Refinement result: observation of *enter.1* & *enter.2* in a trace

```
event: L R
L
event: (read).0 N
(read).0
event: (write).2 N
(write).2
event: (write).1
(write).1
event: enter.1 enter.2
enter.1
event: exit.1 enter.2
exit.1
stop
event: enter.2
enter.2

not satisfy
(req).1 (req).2 (read).0 L (read).0 (write).2 (write).1 enter.1 enter.2
val it = () : unit
-
```

Extension of Fischer's algorithm with Timed CSP

- Redefinition of process Q(i) with Timed CSP

$$\begin{aligned} Q(i) = {} & req.i \rightarrow read?x \rightarrow \\ & \quad \text{if } x \neq 0 \text{ then } SKIP \\ & \quad \text{else } (\ write!i \xrightarrow{\epsilon} read?y \rightarrow \\ & \qquad \text{if } y \neq i \text{ then } SKIP \\ & \qquad \text{else } enter.i \rightarrow exit.i \rightarrow SKIP\ ) \overset{\delta}{\triangleright} STOP \end{aligned}$$

- Introduction of waiting time ε for writing *i* into the shared memory
- Introduction of maximum time limit for occupation of the shared memory δ
- $\varepsilon > \delta$ should be satisfied
  - In this analysis, we have modified also *SPEC,FIS* slightly

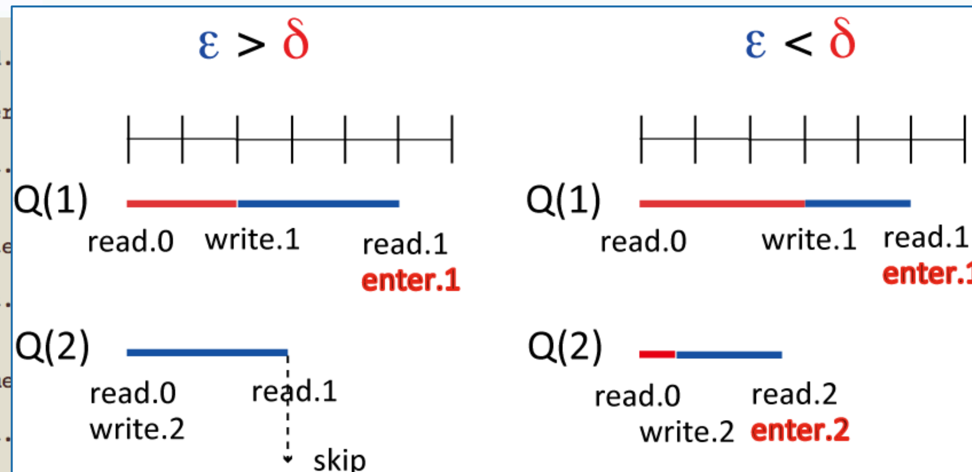
Refinement of the extended Fischer's algorithm

- Trace timewise refinement with Timed CSP Explorer
  - $\varepsilon = 4, \delta = 2$ $\varepsilon = 2, \delta = 4$

```
read.0
event: write.1 2time
write.1
event: 4time
4time
event: read.1
read.1
event: enter.1
enter.1
event: exit.1
exit.1
stop
*
*
*
*
event: 2time
2time
*
*
*
*
*
*
satisfy
val it = () : unit
- 
```

Succeeded

```
2time
event: read.
read.2
event: enter
enter.2
event: exit.
exit.2
stop
event: write
write.1
event: exit.
exit.2
stop
event: 2time
2time
event: exit.
exit.2
stop
event: read.1
read.1
event: enter.1 exit.2
enter.1
here2:
not satisfy
req.1 req.2 read.0 L read.0 write.2 2time read.2 enter.2 write.1 2time read.1 enter.1
val it = () : unit
- 
```



Failed

Summary and Outlook

- We have developed (are developing) a verification tool for Timed CSP processes, but it has just started a year ago out of our urgent necessity
- Timed CSP Explorer makes a lexical analysis and parsing of a machine readable (CSP_M) description of a Timed CSP process, and generates the corresponding ML expression
- run, exec command of the tool can generate trace sequences that the process will produce (step by step or continuously till end)
- refine command can verify the process with its specification in terms of trace and trace timewise refinement
- Development is still underway
  - We need more efficient, complete and robust parsing system
  - and must add failures based (failure timewise and timed failure) refinement facility