

Synchronous Data-flow Modeling of Shared Resources

Erwan Jahier & Nicolas Halbwachs & Pascal Raymond
(Verimag)

November 2007

Bamberg

Context (cf the previous presentation of N. Halbwachs)

- ADL = joint description of software and hardware
 - software (process, thread, program, data)
 - hardware (processor, memory, bus)
- Modeling an ADL into a Synchronous language
 - Give a precise temporal semantics to the ADL
 - Take advantage of the validation (formal verif, test) of the host language
 - ⇒ accurate validation w.r.t. time

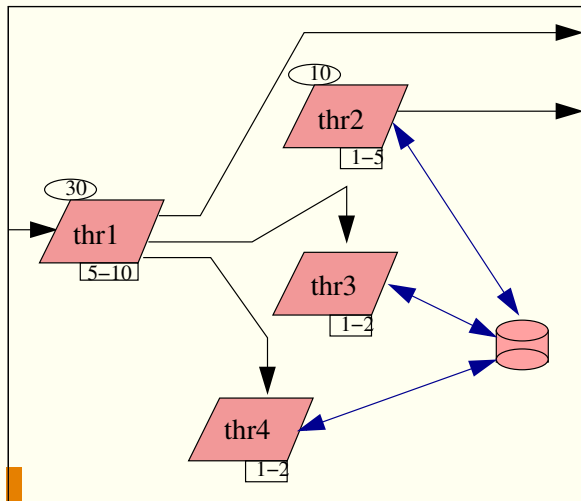
Motivations

Modeling Shared resources (bus, memory)

- Cannot be overlooked
- Fine-grained protocol analysis (compared to usual protocol criterion that ignore functional aspects)

$$\frac{WCET_1}{P_1} + \dots + \frac{WCET_n}{P_n} \leq n(2^{\frac{1}{n}} - 1)$$

Example



Synchronous modelling of Asynchrony and shared resources

- Execution time (for time-lasting tasks)
- Clock drifts (for tasks running on several processors)
- **Multitasking** (for tasks running on the same processor)

Shared Resources access protocol

Rate monotonic scheduling

- No lock
- Lock
- Basic Inheritance (BIP)
- Priority ceiling Protocol (PCP)

Scheduling n tasks and m resources

● Scheduler inputs:

$dispatched_k$ = Task t_k asks for the cpu

$ask_cs_{r_l}^{t_k}$ = Task t_k asks for the resource r_l

● Scheduler outputs:

cpu_k = Task t_k has the cpu ■

● convention: t_i has priority over t_j iff $i < j$

No Lock

In Lustre for 3 tasks

```
node cpu_from_dispatched(  
    dispatched1,  dispatched2,  dispatched3: bool  
    t1_ask_cs_r1, t1_ask_cs_r1, ... :bool)  
returns ( cpu1, cpu2, cpu3 : bool);  
let  
    cpu1 =  dispatched1;  
    cpu2 =  dispatched2 and not  cpu1;  
    cpu3 =  dispatched3 and not  cpu1 and not  cpu2;  
tel
```


No lock

For n tasks

$$\forall k \in [1, n] : \textcolor{red}{cpu}_k = \textcolor{green}{dispatched}_k \wedge \bigwedge_{0 < i < k} \overline{\textcolor{red}{cpu}_i}$$

Lock (1)

For n tasks and m resources

First, we compute a utility relation indicating which task is in critical section

$$\forall k \in [1, n] \quad \forall l \in [1, m] :$$

$$have_cs_{r_l}^{t_k} = ask_cs_{r_l}^{t_k} \wedge (\bullet have_cs_{r_l}^{t_k} \vee cpu_k)$$

$\bullet \equiv$ previous value

Lock (2)

Then, we compute a utility relation indicating if tasks t_i inhibits task t_k via a resource r_l

$$\forall k, i \in [1, n], i \neq k, \forall l \in [1, m] :$$

$$t_i \text{ inhibits }_{r_l}^{t_k} = \text{ask_cs}_{r_l}^{t_k} \wedge \text{have_cs}_{r_l}^{t_i}$$

Lock (3)

Computing the elected thread

$$\forall k \in [1, n] :$$

$$cpu_k = dispatched_k \wedge \bigwedge_{0 < i < k} \overline{cpu_i} \wedge \bigwedge_{i \neq k} \overline{t_i_inhibits_{r_l}^{t_k}}$$

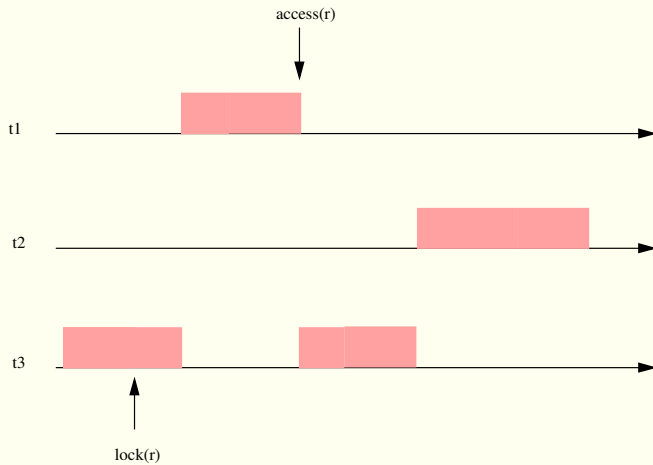
Lock (2-bis)

Computing the inhibits relation (bis)

$$\begin{aligned}
 have_cs_{r_l}^{t_k} &= ask_cs_{r_l}^{t_k} \wedge (\textcolor{red}{cpu}_k \vee \bullet have_cs_{r_l}^{t_k}) \\
 \textcolor{red}{cpu}_k &= \textcolor{green}{dispatched}_k \wedge \bigwedge_{0 < i < k} \overline{\textcolor{red}{cpu}_i} \wedge \bigwedge_{i \neq k} \overline{t_i_inhibits_{r_l}^{t_k}} \\
 t_i_inhibits_{r_l}^{t_k} &= \textcolor{green}{ask_cs}_{r_l}^{t_k} \wedge \overline{have_cs_{r_l}^{t_i}} \\
 t_i_inhibits_{r_l}^{t_k} &= \textcolor{green}{ask_cs}_{r_l}^{t_k} \wedge \bullet have_cs_{r_l}^{t_i} \wedge \textcolor{green}{ask_cs}_{r_l}^{t_i}
 \end{aligned}$$

cycle: $t_inhibits_r^t \rightarrow have_cs_r^t \rightarrow \textcolor{red}{cpu} \rightarrow t_inhibits_r^t.$

Priority Inversion



Basic Inheritance Protocol

Fixing the priority inversion problem

● idea: a task that have locked a resource r inherits of the priority of tasks that want to access to r ■

⇒ t_i *inherits* t_k^* relation

Computing the $t_i_inhibits_{t_k}^*$ relation

● $path(i, k) = \{ \text{cycle-free inhibit. paths from } t_i \text{ to } t_k \}$

● $\forall (i, k), i \neq k : t_i_inhibits_{t_k}^* =$

$$\bigvee_{p=(i_1, \dots, i_s) \in path(i, k)} t_i_inhibits_{r_1}^{t_{i_1}} \wedge t_{i_1}_inhibits_{r_2}^{t_{i_2}}$$

$$\wedge \dots \wedge t_{i_s}_inhibits_{r_k}^{t_{i_s}} \\ \wedge \overline{t_i_is_inhibited}$$

Basic Inheritance Protocol

$$ii_0 = 0$$

$$\forall k \in [1, n] : (cpu_k, ii_k) =$$

$$\overline{dispatched_k} \rightarrow (False, ii_{k-1}) \quad (1)$$

$$(cpu_1 \vee \dots \vee cpu_{k-1}) \rightarrow (False, -1) \quad (2)$$

$$\{ t_j-inhibits_{t_k}^* \rightarrow (False, j) \}_{j \in [1, n], j \neq k} \quad (3)$$

$$ii_{k-1} = k \rightarrow (True, -1) \quad (4)$$

$$ii_{k-1} > 0 \rightarrow (False, ii_{k-1}) \quad (5)$$

$$True \rightarrow (\overline{t_k-is-inhibited}, 0) \quad (6)$$

$dispatched_1, dispatched_2, dispatched_3, dispatched_4$

$$ii_0 = 0$$

$$\forall k \in [1, n] : (cpu_k, ii_k) =$$

$$\overline{dispatched_k} \rightarrow (False, ii_{k-1}) \quad (1)$$

$$(cpu_1 \vee \dots \vee cpu_{k-1}) \rightarrow (False, -1) \quad (2)$$

$$\{ t_j_inhibits_{t_k}^* \rightarrow (False, j) \}_{j \in [1, n], j \neq k} \quad (3)$$

$$ii_{k-1} = k \rightarrow (True, -1) \quad (4)$$

$$ii_{k-1} > 0 \rightarrow (False, ii_{k-1}) \quad (5)$$

$$True \rightarrow (\overline{t_k_is_inhibited}, 0) \quad (6)$$

$\overline{dispatched_1}, dispatched_2, dispatched_3, dispatched_4$

$ii_0 = 0$

$(cpu_1, ii_1) =$

$$\overline{dispatched_1} \rightarrow (False, ii_0) \quad (1)$$

$$(False) \rightarrow (False, -1) \quad (2)$$

$$\{ t_j_inhibits_{t_1}^* \rightarrow (False, j) \}_{j \in [1, n], j \neq 1} \quad (3)$$

$$ii_0 = 1 \rightarrow (True, -1) \quad (4)$$

$$ii_0 > 0 \rightarrow (False, ii_0) \quad (5)$$

$$True \rightarrow (\overline{t_1_is_inhibited}, 0) \quad (6)$$

$\overline{dispatched_1}, dispatched_2, dispatched_3, dispatched_4$

$ii_0 = 0$

$(cpu_1, ii_1) =$

$$\overline{dispatched_1} \rightarrow (False, ii_0) \quad (1)$$

$$(False) \rightarrow (False, -1) \quad (2)$$

$$\{ t_j_inhibits_{t_1}^* \rightarrow (False, j) \}_{j \in [1, n], j \neq 1} \quad (3)$$

$$ii_0 = 1 \rightarrow (True, -1) \quad (4)$$

$$ii_0 > 0 \rightarrow (False, ii_0) \quad (5)$$

$$True \rightarrow (\overline{t_1_is_inhibited}, 0) \quad (6)$$

$\overline{dispatched_1}, dispatched_2, dispatched_3, dispatched_4$

$ii_1 = 0$

$(cpu_2, ii_2) =$

$$\overline{dispatched_2} \rightarrow (False, ii_1) \quad (1)$$

$$(cpu_1) \rightarrow (False, -1) \quad (2)$$

$$\{ t_j_inhibits_{t_2}^* \rightarrow (False, j) \}_{j \in [1, n], j \neq 2} \quad (3)$$

$$ii_1 = 2 \rightarrow (True, -1) \quad (4)$$

$$ii_1 > 0 \rightarrow (False, ii_1) \quad (5)$$

$$True \rightarrow (\overline{t_2_is_inhibited}, 0) \quad (6)$$

$\overline{dispatched_1}, dispatched_2, dispatched_3, dispatched_4$

$ii_1 = 0$

$(cpu_2, ii_2) =$

$$\overline{dispatched_2} \rightarrow (False, ii_1) \quad (1)$$

$$(cpu_1) \rightarrow (False, -1) \quad (2)$$

$$\{ t_j_inhibits_{t_2}^* \rightarrow (False, j) \}_{j \in [1, n], j \neq 2} \quad (3)$$

$$ii_1 = 2 \rightarrow (True, -1) \quad (4)$$

$$ii_1 > 0 \rightarrow (False, ii_1) \quad (5)$$

$$True \rightarrow (\overline{t_2_is_inhibited}, 0) \quad (6)$$

$\overline{dispatched_1, dispatched_2, dispatched_3, dispatched_4}$

$ii_2 = 0$

$(cpu_3, ii_3) =$

$$\overline{dispatched_3} \rightarrow (False, ii_2) \quad (1)$$

$$(cpu_1 \vee cpu_2) \rightarrow (False, -1) \quad (2)$$

$$\{ t_j_inhibits_{t_3}^* \rightarrow (False, j) \}_{j \in [1, n], j \neq 3} \quad (3)$$

$$ii_2 = 3 \rightarrow (True, -1) \quad (4)$$

$$ii_2 > 0 \rightarrow (False, ii_2) \quad (5)$$

$$True \rightarrow (\overline{t_3_is_inhibited}, 0) \quad (6)$$

$\overline{dispatched_1, dispatched_2, dispatched_3, dispatched_4}$

$ii_2 = 0$
 $(cpu_3, ii_3) =$

$$\overline{dispatched_3} \rightarrow (False, ii_2) \quad (1)$$


$$(cpu_1 \vee cpu_2) \rightarrow (False, -1) \quad (2)$$

$$\{ t_j_inhibits_{t_3}^* \rightarrow (False, j) \}_{j \in [1, n], j \neq 3} \quad (3)$$

$$ii_2 = 3 \rightarrow (True, -1) \quad (4)$$

$$ii_2 > 0 \rightarrow (False, ii_2) \quad (5)$$

$$True \rightarrow (\overline{t_3_is_inhibited}, 0) \quad (6)$$

$dispatched_1, dispatched_2, dispatched_3, dispatched_4,$
 $t_3_inhibits_{t_1}^*$ 

$ii_0 = 0$
 $(cpu_1, ii_1) =$

$$\overline{dispatched_1} \rightarrow (False, ii_0) \quad (1)$$

$$(False) \rightarrow (False, -1) \quad (2)$$

$$\{ t_j_inhibits_{t_1}^* \rightarrow (False, j) \}_{j \in [1, n], j \neq 1} \quad (3)$$

$$ii_0 = 1 \rightarrow (True, -1) \quad (4)$$

$$ii_0 > 0 \rightarrow (False, ii_0) \quad (5)$$

$$True \rightarrow (\overline{t_1_is_inhibited}, 0) \quad (6)$$

$dispatched_1, dispatched_2, dispatched_3, dispatched_4,$
 $t_3_inhibits_{t_1}^*$

$ii_0 = 0$
 $(cpu_1, ii_1) =$

$$\overline{dispatched_1} \rightarrow (False, ii_0) \quad (1)$$

$$(False) \rightarrow (False, -1) \quad (2)$$

$$t_3_inhibits_{t_1}^* \rightarrow (False, 3) \quad (3)$$

$$ii_0 = 1 \rightarrow (True, -1) \quad (4)$$

$$ii_0 > 0 \rightarrow (False, ii_0) \quad (5)$$

$$True \rightarrow \overline{(t_1_is_inhibited, 0)} \quad (6)$$

$dispatched_1, dispatched_2, dispatched_3, dispatched_4,$
 $t_3_inhibits_{t_1}^*$

$ii_1 = 3$
 $(cpu_2, ii_2) =$

$$\overline{dispatched_2} \rightarrow (False, ii_1) \quad (1)$$

$$(cpu_1) \rightarrow (False, -1) \quad (2)$$

$$\{ t_j_inhibits_{t_2}^* \rightarrow (False, j) \}_{j \in [1, n], j \neq 2} \quad (3)$$

$$ii_1 = 2 \rightarrow (True, -1) \quad (4)$$

$$ii_1 > 0 \rightarrow (False, ii_1) \quad (5)$$

$$True \rightarrow (\overline{t_2_is_inhibited}, 0) \quad (6)$$

$dispatched_1, dispatched_2, dispatched_3, dispatched_4,$
 $t_3_inhibits_{t_1}^*$

$ii_2 = 3$
 $(cpu_3, ii_3) =$

$$\overline{dispatched_3} \rightarrow (False, ii_2) \quad (1)$$

$$(cpu_1 \vee cpu_2) \rightarrow (False, -1) \quad (2)$$

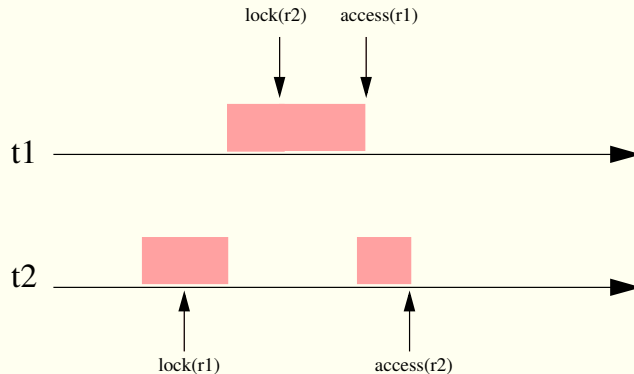
$$\{ t_j_inhibits_{t_3}^* \rightarrow (False, j) \}_{j \in [1, n], j \neq 3} \quad (3)$$

$$ii_2 = 3 \rightarrow (True, -1) \quad (4)$$

$$ii_2 > 0 \rightarrow (False, ii_2) \quad (5)$$

$$True \rightarrow \overline{(t_3_is_inhibited, 0)} \quad (6)$$

problem: the BIP can deadlock



→ (statically) forbid such intertwined use of locks

→ Priority Ceiling Protocol

Priority ceiling Protocol

- Priority ceiling of a resource r $PC(r)$ is the priority of the more priority task that might access to r (static) ■
- The priority ceiling of a task t_k PC_k is the maximum of the priority ceilings of the resources locked by other tasks than t_k (dynamic) ■

$$\forall k : PC_k = \text{Min} \left\{ \begin{array}{l} \text{if } \bigvee_{i \in [1, n], i \neq k} (\text{ask_cs}_{r_i}^{t_i} \wedge \bullet \text{have_cs}_{r_i}^{t_i}) \\ \text{then } PC(l) \text{ else } n \end{array} \right\}_{l \in [1, m]}$$

Priority ceiling Protocol

- BIP + t_k can lock a resource r only if its priority is higher than its priority ceiling ($k < PC_k$).

Priority ceiling Protocol

$$ii_0 = 0$$

$$\forall k \in [1, n] : (cpu_k, ii_k) = \text{■}$$

$$\overline{dispatched_k} \rightarrow (False, ii_{k-1}) \quad (1)$$

$$(cpu_1 \vee \dots \vee cpu_{k-1}) \rightarrow (False, -1) \quad (2)$$

$$\{ t_j_inhibits_{t_k}^* \rightarrow (False, \textcolor{red}{pc}(j)) \}_{j \in [1, n], j \neq k} \quad (3)$$

$$ii_{k-1} = k \rightarrow (True, -1) \quad (4)$$

$$ii_{k-1} > 0 \rightarrow (False, ii_{k-1}) \quad (5)$$

$$\begin{aligned} True &\rightarrow (\overline{t_k_is_inhibited} \\ &\quad \wedge (ask_cs^{t_k} \Rightarrow PC_k > k), \\ &\quad 0) \end{aligned} \quad (6)$$

where $\textcolor{red}{pc}(j) = \textit{if } PC_j > j \textit{ then } j \textit{ else } 0$

Demo

● `wc *.ml`

→ `“cpu.mli”`

→ `“cpu_test.ml”`

● `lesar` finds the deadlock in BIP

● `lesar` proves the absence of deadlock in PCP