The Verification of Real Time Systems

Sérgio Campos

Real Time Systems

Real Time systems are systems in which a delayed response has severe consequences.

- Used in several life critical applications:
  - Aircraft and air traffic controllers.
  - Power plant controllers.
  - Medical systems.
  - ...
- Correctness is even more important!

Rate Monotonic Scheduling Theory

Given a set of tasks:

$$\{(C_i, T_i) \mid C_i \text{ is execution time, } T_i \text{ is period of task } i\}$$

define: $U_i = C_i/T_i$ — CPU Utilization by task i
The task set is schedulable if:

$$\sum_{i=1..n} U_i \leq U_n^* = n(2^{1/n} - 1)$$

Example:

$$T = \{(1, 0.25), (1.25, .1), (1.5, .3), (1.75, .07), (2, .1)\}$$
$$\sum U_i = .62 \leq U_{rm(5)} = .743$$

Rate Monotonic Scheduling Theory

RMS Assumes:

- Tasks are periodic
- Deadline is equal to the period
- No synchronization

Many extensions have been developed:

- Aperiodic tasks,
- synchronization
- Exact schedulability analysis

But severe limitations still exist...

Verus — A Model Checking Approach

The Verus Language

- Real-time systems description is easy and efficient.
- Imperative language, syntax similar to C.
- Special constructs for timing aspects such as deadlines, priorities and time delays.

A Verus Example

```
 1 motor_control()
 2 {
 3   boolean start, finish;
 4
 5   periodic(0, 40, 40) {
 6      start = 1;
 7      priority(10){
 8         data_in = dev_rdy & !abort;
 9         wait(1);
10      };
11      priority(7){
12         wait(5);
13         data_out = data_in & !abort;
14      };
15     finish = 1;
16   };
17 }
```


A Verus Example

```
 1 motor_control()
 2 {
 3   boolean start, finish;
 4
 5   deadline(40) {
 6      start = 1;
 7      priority(10){
 8         data_in = dev_rdy & !abort;
 9         wait(1);
10      };
11      priority(7){
12         wait(5);
13         data_out = data_in & !abort;
14      };
15      finish = select{0,1};
16   };
17 }
```


A Verus Example

```
 1 main()
 2 {
 3   int timer; /* 8 bits */
 4   int small : 4; /* 4 bits */
 5
 6   process ms1 motor_control(comm),
 7           ms2 motor_sensor(pos);
 8   small = 10;
 9
10   compute min[ms1.start,ms1.end];
11   compute max[ms1.start,ms1.end];
12 }
```

How Does it Work ? Prioritized Execution

```
 1  ...
 2  /* Beginning of execution */
 3  req1 = true;
 4  while (granted != 1) wait(1);
 5  wait(1);
 6  ...
 7  while (granted != 1) wait(1);
 8  wait(1);
 9  /* End of execution */
10  req1 = false;
11  wait(1);
12  ...
```


Prioritized Execution

```
 1 scheduler(req1, req2, req3, granted)
 2 int req1, req2, req3, granted;
 3 {
 4   while (true) {
 5     if (req1) granted = 1; else
 6     if (req2) granted = 2; else
 7     if (req3) granted = 3; else
 8     granted = 0;
 9
10     wait(1);
11
12     time = time + 1;
13   }
14 }
```


Deadline Statement

```
deadline(n1) {
  S;
  wait(1);
};
```

corresponds to:

```
timer = 0;
S;
timer = timer + 1;
if (timer >= n1) error_code();
wait(1);
```

- Transformation must be done *to all* wait *statements* inside S.
- In practice, sharing timers is more efficient.

Verification Algorithms for Real Time — RTCTL

In *CTL* it is possible to express properties such as:

- p will eventually be asserted.
- p will never occur.

But not:

- Requests are acknowledged in less than 10ms.

The bounded until operator overcomes this problem:

$$\mathbf{AG}(req \rightarrow \mathbf{AF}_{[2,4]}\ ack)$$

This can be translated to:

$$\mathbf{AG}(req \rightarrow \mathbf{AX}\,\mathbf{AX}(ack \vee \mathbf{AX}\,ack))$$

Special algorithms perform this task without the need for expanding the formula.

Quantitative Algorithms

Basic Idea:

- To compute quantitative timing information, such as minimum and maximum times between a request and its response

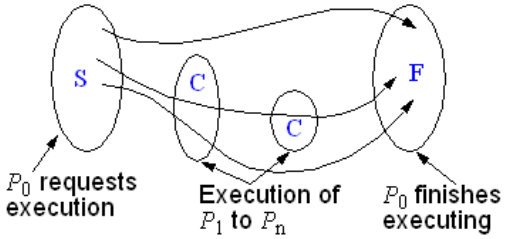
These algorithms:

- Provide insight into system behavior.
- Determine characteristics of the system.
- Determine effects of changes in the design.

Computing Quantitative Information

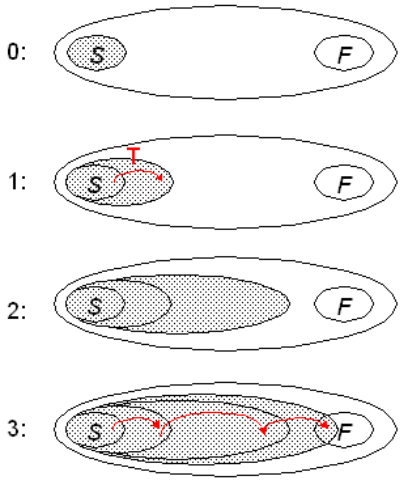
Given a set of all paths leading from a set of starting states to a set of final states, the algorithms determine:

- The minimum and maximum length of all paths:
  - Reaction time to events
- The minimum and maximum number of states along any such path satisfying a given condition:
  - Priority inversion: During the execution of a high priority process P0 how much time is spent executing lower priority processes ?



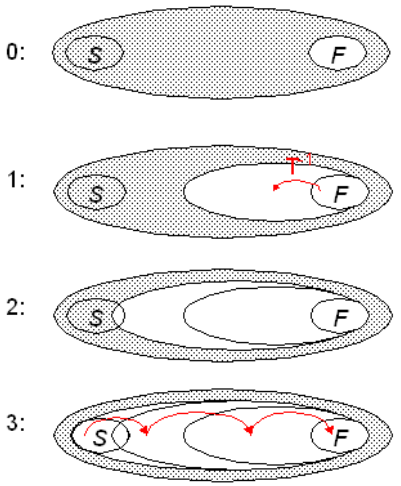
Minimum Algorithm

- Searches forward
- Stops when F is found



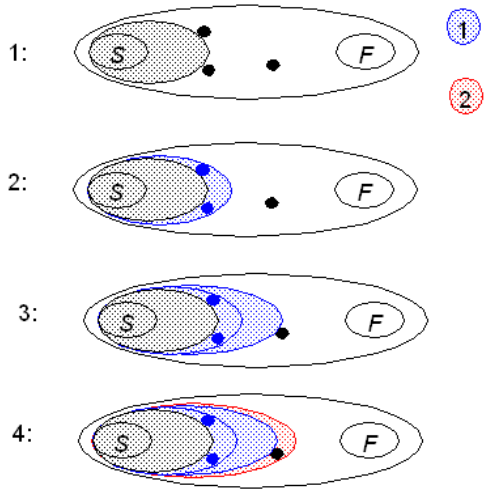
Maximum Algorithm

- Searches backwards from $\neg F$
- Stops when S is no longer reached.



Condition Counting Algorithms

- Visit states in *condition* reachable in one step.
- Then visit all states reachable from the current frontier that are not in *condition*.



Real Time Model Checking

With CTL model checking + quantitative algorithms we can:

- Model timing characteristics of time critical systems
- Verify correctness
- Verify timing properties, e.g., adherence to deadlines
- Determine quantitative properties such as reaction time to events
  - This gives an *insight* into the behavior of the system
  - Allows *fine tuning*