Modeling scheduling policy with Constraint Satisfaction Problem approach

Hyuk Lee¹, Jin-Young Choi²
¹Department of Computer Science, Korea University, Seoul, Korea
²Graduate School of Information Security, Korea University, Seoul, Korea

Abstract—Many safety critical systems are real-time systems and a failure on the real-time property may cause serious damage on people and environment. Schedulability analysis is a good way of predicting system behavior and it is important for hard real-time tasks in which keeping deadline is critical. In this paper, we have proposed schedulability analysis with constraint solving approach. The behaviors of the scheduler are analyzed and decomposed into state-based behaviors in order to encode them into constraints. Then, constraints for task model are solved by SMT solver for satisfying interpretations. Treating schedulability analysis as a constraint satisfaction problem has some benefits. Constraints are focused on problems itself and it is based on logic formulas which are simple and clear. More importantly, modern SMT solvers can solve constraints in no time.

Keywords: Schedulability analysis, Constraint Satisfaction Problem, Satisfiability modulo theories

1. Introduction

Real-time systems are used in various places with different purposes. They are used from general purpose multimedia system to safety critical systems such as nuclear, automotive, missile and air traffic control system. Among safety critical systems, that require real-time property are mainly hard real-time systems which requires absolute guarantee to meet the deadline. Fail to meet the deadline in the real-time system means failure of the system, and failure of safety critical system can cause damage to people. In order to keep the real-time property well, the prediction of the behavior of a system is very important and it can be achieved through the schedulability analysis. Since the pioneering work by Liu and Layland[1], many kinds of research on schedulability analysis have been conducted with various methods[2], [3], [4], [5], [6].

There also have been many approaches to schedulability analysis with formal techniques. Some of the research have been done with process algebra[7], [8], [9], [10], state machine[11], [12], constraint solving approach[13], [14], [15]. Cheng and Zhang[14], [15] have proposed scheduling overloaded tasks with an SMT-based approach. However, in a hard real-time system which is our target, tasks are not supposed to be overloaded. Our proposed approach focuses on satisfiability of task model on certain scheduling discipline.

Various SMT solvers are available, CVC4[16], MathSAT5[17], raSAT[18], SMTInterpol[19], Yices[20], Z3[21]. We used Z3py for scheduling analysis which is Python front-end of SMT solver Z3. Python offers a simple and easy environment for beginners to use.

In this paper, we have proposed schedulability analysis with constraint solving approach. Once the scheduling algorithm is encoded as constraints, SMT solver can solve satisfiability of task model. The remainder of this paper is organized as follows. Additional information on schedulability analysis and constraints satisfaction problem for our approach is explained in Section 2. In Section 3, scheduler behavior is analyzed and decomposed and encoded as constraints. Section 4 demonstrates schedulability analysis result of example task models. The paper is concluded with Section 5.

2. System Model and Background

Most real-time systems employ preemptive priority-based scheduling algorithms in which task with a higher priority can preempt one with a lower priority task. Priority-based scheduling algorithms are further categorized as fixed priority and dynamic priority algorithm depends on the flexibility of the priority assignment. Priority in dynamic priority scheduling may vary at run-time, whereas fixed priority scheduling doesn’t. Rate Monotonic[1] and Deadline Monotonic[22] algorithm are well known fixed priority scheduling algorithm. As for the dynamic priority scheduling algorithm, Earliest Deadline First[1] and Least Laxity First[23] algorithm are widely used. Among those various scheduling algorithms, EDF and LLF are considered as optimal algorithms on preemptive uniprocessor environment since the utilization is maximal.

Constraint solving approach, used in this paper, is methods of solving problems in Constraint Satisfaction Problem. CSP is a mathematical problem that is defined as a collection of objects with states that must satisfy various constraints[24]. Properties of the problem are expressed as similar constraints, as a set of finite constraints on the variables, and solve them in the way of constraint problem. The constraints specify the allowable values for the variables
in that domain, and the solution to the problem is to assign a value satisfying all constraints to each variable. Constraints of the problem are then passed into the SMT solver which designed to solve SMT problems.

Satisfiability Modulo Theories (SMT) problem is: If a formula to be satisfiable, then there exist an interpretation that makes formula true. SAT or boolean satisfiability problem is a type of satisfiability problem expressed in boolean formulas. Since SAT is based on propositional logic, the expressiveness of logic has limitations. SMT is based on first-order logic with extended theories such as real numbers, the theory of integers, and the theories of various data structures such as lists, arrays, bit vectors. SMT can be thought of as a form of the constraint satisfaction problem and thus a certain formalized approach to constraint programming[25]. Once our schedulability problem is encoded as constraint satisfaction problem, SMT solver can find a satisfying interpretation of those constrained variables.

### 3. Modeling constraint based scheduler

#### 3.1 System and Definitions

A real-time system can be defined by the characteristics of tasks. There is a system with a set of tasks \( \mathcal{T} \), and worst case computation time of task \( \tau_i \) is \( c_i \). The system is a real-time system if there exists at least one task \( \tau_i \) which belongs to following categories[26].

- **Hard real-time task.** Task \( \tau_i \) must complete its computation \( c_i \) before deadline \( d_i \).
- **Soft real-time task.** Task \( \tau_i \) completes its computation \( c_i \) later than deadline \( d_i \). Task \( \tau_i \) gets penalty
- **Firm real-time task.** Task \( \tau_i \) completes its computation \( c_i \) before deadline \( d_i \). Task \( \tau_i \) gets reward

#### Assumptions on tasks are as follows:

- All tasks are periodic tasks
- All tasks have the worst computation time
- All tasks arrive at the same time
- All tasks are independent tasks, no shared resources
- All overloads such as context switch are ignored

The simplified behavior of scheduling process by a scheduler is described in Algorithm 1. The scheduler takes input task model and proceeds to schedule. It is divided into two cases within the period.

- Computation time is completed
  - Time left until next period \( \rightarrow \) IDLE
  - Time reached next period \( \rightarrow \) RESET

- Computation time is remaining
  - Time left until next period
    * Priority is the highest \( \rightarrow \) EXECUTE
    * Priority is not the highest \( \rightarrow \) IDLE
  - Time reached next period \( \rightarrow \) DEADLINE MISS

The behavior of the scheduler can be further subdivided into categories of constraints.

- **Scheduler behavior Constraints**
  - Only one task can be executed at one time instant
    * if \( \tau_i \) is executed at time instant \( t \), then \( \tau_j \) cannot be executed, where \( i \neq j \)
  - A task is executed between its offset and deadline.
    * if \( \tau_i \) to be executed, time \( t \) must be \( o_i \leq t \leq d_i \)

- **Task behavior Constraints**
  - When a task is executed, computation time is updated
    * if \( \tau_i \) is executed at time \( t \), then \( s_{i,t+1} = s_{i,t} + 1 \)
    - If a task is being executed, other tasks should idle for next turn
      * if \( \tau_j \) is idle at time \( t \), then \( s_{j,t+1} = s_{j,t} \)
    - When the period of a task is reached, the execution time is initialized
      * if time \( t = p_i \), then \( s_{i,t} = 0 \)

- **Scheduling policy Constraints**
  - The task with the highest priority is executed
    * if \( \tau_i \) to be executed at time instant \( t \), then \( \pi_{i,t} \geq \pi_{j,t} \), where \( i \neq j \)
  - Assign priority according to the policy
    * if EDF is used, calculate priority
      \[ \pi_{i,t} = d_{\text{max}} - (d_i - t) \]
    * if LLF is used, calculate priority
      \[ \pi_{i,t} = d_{\text{max}} - (d_i - t) - (c_i - s_{i,t}) \]
      \[ d_{\text{max}} = 1 + \max(d_i), \text{where } i = (1, \ldots, n) \]

- **Schedulability Constraints**
  - Computation time must be executed before deadline
    * if time \( t = d_i \), then \( s_{i,t} = c_i \)
In the program, a few uninterpreted functions and variables possibly offsets. All the task information from the task information about periods, computation time, deadlines and constraints into the solver.

to import, and solving constraints by simply adding encoded input task model. SMT solver Z3 provides API for Python and rest of the program is for processing and parsing the part of the program is mostly done with constraints encoding which checks schedulability of the task model. The main following pieces of Python code is parts of the program the behavior constraints defined in the previous chapter.

3.2 Constraints encoding

In order to encode the behavior of the scheduler as constraints of state relations, the scheduler is modeled using the behavior constraints defined in the previous chapter. Following pieces of Python code is parts of the program which checks schedulability of the task model. The main part of the program is mostly done with constraints encoding and rest of the program is for processing and parsing the input task model. SMT solver Z3 provides API for Python to import, and solving constraints by simply adding encoded constraints into the solver.

Input file of the program is a task model which has tasks information about periods, computation time, deadlines and possibly offsets. All the task information from the task model is stored as constant variables within the program. In the program, a few uninterpreted functions and variables are declared for constraints encoding. Each declaration of uninterpreted function \( h, x, dp \) is for execution status, execution time, priority status respectively. Function \( h \) takes two arguments of type Proc and Int, returns Exe type value. Function \( x \) and \( dp \) also take two arguments of type Proc and Int, return Int type value. Variables \( i, a, b \) are Z3 generated variables with Int, Proc type and they works inside of the solver. Rest of variables \( j, k, \text{maxt}, TN \) are Python variables. Variables \( j, k \) are used in manipulating index for loops and lists and \( TN \) represents the total number of tasks in the input task model and \( \text{maxt} \) represents the least common multiple of the period of tasks in the input task model. In the Listing 2 and 6, Python list variable \( p, d \) are used to represent period, deadline of the input task model, while \( t \) is Proc type Z3 variable for the task.

To clarify the code below, some of the symbols defined previously used differently here, because of some redundancy issues. Some of the symbols from Table 1 maps as follows. Symbol for task \( \tau \) used as \( t[j] \), time instant \( t \) as \( i \), priority of task \( \pi \) as \( dp(i) \), executed time \( s \) as \( x() \), computation time \( c \) as \( e[i] \).

a) Scheduler behavior constraints: The scheduler selects a task to be executed according to the scheduling policy. In the case of EDF and LLF policy, a task with the highest priority is selected. Once the task is selected, only the selected task must be executed and other tasks idle for their next turn. Atomic execution of a task can be encoded as Listing 1 below. \( h(a, i) \) will return the execution status of \( a \) at time instant \( i \). Proc type variable \( a \) represents a selected task and non-selected tasks are represented as \( b \).

Listing 1: Constraint for atomic execution

\[
\text{ForAll}([i], \text{Implies(And}(i > 0, i <= \text{maxt}), \text{ForAll}([a], \text{Implies}(h(a, i) == \text{TRUE}, \text{ForAll}([b], \text{Implies}(b != a, h(b, i) == \text{FALSE}))))))
\]

b) Task behavior constraints: At the beginning of the task period, the executed time is the same as computation time that the task must complete within each period. When a task is selected and executed by the scheduler, execution time must be updated. That is, the execution time of a task increases or decreases depending on the implementation. Execution time assignment of a task can be encoded as Listing 2 below. When the task is executed at time instant \( i \), its accumulated execution time at \( i + 1 \) must increase by 1. Otherwise, the accumulated execution time must stay the same. It applies to all the tasks and at all legitimate time instant \( i \).

Listing 2: Constraint for execution time assignment

\[
\text{ForAll}([i], \text{Implies(And}(i > 0, i <= \text{maxt}), \text{If}(h([j], i) == \text{TRUE}, \text{If}(\%p[j] == p[j] - 1, x([j], i+1) == 0, x([j], i+1) == x([j], i) + 1)), \text{If}(\%p[j] == p[j] - 1, x([j], i+1) == 0, x([j], i+1) == x([j], i))))))
\]

for \( j \) in range(TN)

c) Scheduling policy constraints: Scheduling policy is about determining which task is to be executed, and scheduler prioritizes the tasks that are ready to be executed. When selecting a task to execute, the task with the highest priority is selected. Selecting the highest priority task can be encoded as Listing 3 below. A selected task \( a \) to be executed, its priority at time instant \( i \) must be higher than other task’s.
priority. $dp(a, i)$ will return the priority status of $a$ at time instant $i$.

**Listing 3: Constraint for task selection**

```plaintext
ForAll([a],
  Implies(h(a, i) == TRUE,
    ForAll([b],
      Implies(b != a, dp(a, i) >= dp(b, i))))
)
```

EDF scheduling policy assigns tasks with early deadlines to a higher priority than tasks with late deadlines. Constraint for EDF priority assignment can be encoded as Listing 4 below. $dp(t[j], k)$ will have constrained value that the task with earliest deadline gets higher priority.

**Listing 4: Constraint for priority assignment (EDF)**

```plaintext
for j in range(TN):
  for k in range(maxt):
    s.add(If(x(t[j], k) < e[j],
      dp(t[j], k)==maxd−(d[j]−k%p[j]),
      dp(t[j], k)==0))
```

Similarly, LLF scheduling policy assigns the highest priority to the task with the least laxity. The remaining computation time is taken into account in the priority calculation. Constraint for LLF priority assignment can be encoded as Listing 5 below. $dp(t[j], k)$ will have constrained value that the task with least laxity gets higher priority.

**Listing 5: Constraint for priority assignment (LLF)**

```plaintext
for j in range(TN):
  for k in range(maxt):
    s.add(If(x(t[j], k) < e[j],
      dp(t[j], k)==maxd−(d[j]−k%p[j]−e[j]−x(t[j], k)),
      dp(t[j], k)==0))
```

d) **Schedulability constraint**: A task model is said to be *schedulable* when all tasks in the task model are able to meet their deadline for a given time period. Schedulability constraint of a task can be encoded as Listing 6 below. Accumulated execution time of a task must be equal to its computation time before its deadline for every period. However, as the accumulation of execution time happens one time instant after its execution, both time instant at the deadline and one before the deadline is checked here. In order to the task model to be schedulable, this constraint must hold for all the tasks in the task model for all given time period.

**Listing 6: Constraint for schedulability**

```plaintext
for j in range(TN):
  for k in range(maxt):
    if k%p[j]==d[j]−1:
      s.add(Or(And(x(t[j], k)==e[j], h(t[j], k)==FALSE),
        And(x(t[j], k)==e[j]−1, h(t[j], k)==TRUE))
```

### 4. Evaluation

The evaluation of SMT scheduler model has done with two task models, model H and model L as shown in Table 2 and 3. In fact, model H is schedulable, but Model L is not schedulable because utilization exceeds 100 percent. In model H, tasks have computation time of 1, 2, 1 and period of 3, 4, 6 and deadline of 3, 3, 5 and offset of 1, 0, 2, respectively. Also, the least common multiple of all task’s period is 12, which indicates that if the tasks are schedulable up to this point of time instant, they are guaranteed to be schedulable. In Fig. 2 and Fig. 3, task execution is represented as shading boxes. Solid lines, dotted lines, and dashed area represent the period, deadline and offset respectively.

<table>
<thead>
<tr>
<th>Period</th>
<th>Computation</th>
<th>Deadline</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task 1</td>
<td>3</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>Task 2</td>
<td>4</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Task 3</td>
<td>6</td>
<td>1</td>
<td>5</td>
</tr>
</tbody>
</table>

**Table 2: Task model H**

<table>
<thead>
<tr>
<th>Period</th>
<th>Computation</th>
<th>Deadline</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task 1</td>
<td>3</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Task 2</td>
<td>4</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

**Table 3: Task model L**

The evaluation result of SMT scheduler model has done with two task models, model H and model L as shown in Table 2 and 3. In fact, model H is schedulable, but Model L is not schedulable because utilization exceeds 100 percent. In model H, tasks have computation time of 1, 2, 1 and period of 3, 4, 6 and deadline of 3, 3, 5 and offset of 1, 0, 2, respectively. Also, the least common multiple of all task’s period is 12, which indicates that if the tasks are schedulable up to this point of time instant, they are guaranteed to be schedulable. In Fig. 2 and Fig. 3, task execution is represented as shading boxes. Solid lines, dotted lines, and dashed area represent the period, deadline and offset respectively.

The evaluation of SMT scheduler model has done with two task models, model H and model L as shown in Table 2 and 3. In fact, model H is schedulable, but Model L is not schedulable because utilization exceeds 100 percent. In model H, tasks have computation time of 1, 2, 1 and period of 3, 4, 6 and deadline of 3, 3, 5 and offset of 1, 0, 2, respectively. Also, the least common multiple of all task’s period is 12, which indicates that if the tasks are schedulable up to this point of time instant, they are guaranteed to be schedulable. In Fig. 2 and Fig. 3, task execution is represented as shading boxes. Solid lines, dotted lines, and dashed area represent the period, deadline and offset respectively.

The evaluation result of SMT scheduler model has done with two task models, model H and model L as shown in Table 2 and 3. In fact, model H is schedulable, but Model L is not schedulable because utilization exceeds 100 percent. In model H, tasks have computation time of 1, 2, 1 and period of 3, 4, 6 and deadline of 3, 3, 5 and offset of 1, 0, 2, respectively. Also, the least common multiple of all task’s period is 12, which indicates that if the tasks are schedulable up to this point of time instant, they are guaranteed to be schedulable. In Fig. 2 and Fig. 3, task execution is represented as shading boxes. Solid lines, dotted lines, and dashed area represent the period, deadline and offset respectively.

The evaluation result of SMT scheduler model has done with two task models, model H and model L as shown in Table 2 and 3. In fact, model H is schedulable, but Model L is not schedulable because utilization exceeds 100 percent. In model H, tasks have computation time of 1, 2, 1 and period of 3, 4, 6 and deadline of 3, 3, 5 and offset of 1, 0, 2, respectively. Also, the least common multiple of all task’s period is 12, which indicates that if the tasks are schedulable up to this point of time instant, they are guaranteed to be schedulable. In Fig. 2 and Fig. 3, task execution is represented as shading boxes. Solid lines, dotted lines, and dashed area represent the period, deadline and offset respectively.

The evaluation result of SMT scheduler model has done with two task models, model H and model L as shown in Table 2 and 3. In fact, model H is schedulable, but Model L is not schedulable because utilization exceeds 100 percent. In model H, tasks have computation time of 1, 2, 1 and period of 3, 4, 6 and deadline of 3, 3, 5 and offset of 1, 0, 2, respectively. Also, the least common multiple of all task’s period is 12, which indicates that if the tasks are schedulable up to this point of time instant, they are guaranteed to be schedulable. In Fig. 2 and Fig. 3, task execution is represented as shading boxes. Solid lines, dotted lines, and dashed area represent the period, deadline and offset respectively.

The evaluation result of SMT scheduler model has done with two task models, model H and model L as shown in Table 2 and 3. In fact, model H is schedulable, but Model L is not schedulable because utilization exceeds 100 percent. In model H, tasks have computation time of 1, 2, 1 and period of 3, 4, 6 and deadline of 3, 3, 5 and offset of 1, 0, 2, respectively. Also, the least common multiple of all task’s period is 12, which indicates that if the tasks are schedulable up to this point of time instant, they are guaranteed to be schedulable. In Fig. 2 and Fig. 3, task execution is represented as shading boxes. Solid lines, dotted lines, and dashed area represent the period, deadline and offset respectively.
result of EDF scheduling of model L. Actual scheduling result by SMT scheduler model came as the timing graph (d). In model L, the deadlines and periods of all tasks are the same. At time instant $i = 0$ and $i = 1$, Task 1 is executed and completed its first period since the deadline of Task 1 is ahead of the deadline of Task 2. Task 2 is then executed at time instant $i = 2$ without competing with other tasks. At time instant $i = 3$, Task 2 with an earlier deadline is executed. Task 1 fails to complete the computation time and reaches the deadline (end of period) at time instant $i = 9$.

SMT scheduling result of task model L was unsatisfiable. This means that SMT scheduler was not able to find the solution that satisfies the constraints for a given time period ($i \leq 12$). The timing graph (d) only shows scheduled tasks of model L up to time instant $i = 8$, the point where constraints were satisfiable. Since one of our constraint for schedulability was that the computation time should be completed within the deadline, the constraints were only satisfied for a partial time period. However, it is insufficient bound in order to guarantee schedulability of task model L.

As for the LLF scheduling policy, it is similar to the EDF scheduling policy. LLF scheduling policy assigns the highest priority to the most urgent task based on deadline and execution time. As described in Listing 4 and Listing 5 in the previous section, LLF scheduling policy can be implemented by simply replacing constraints on scheduling policy. The timing graph (a) in Fig. 3 represents an anticipated result and timing graph (b) by SMT scheduler model of LLF scheduling of model H. Coincidentally, the result of the LLF scheduling happens to be the same as the result of EDF scheduling policy. However, actual scheduling result came out differently which caused by non-deterministic choice.
5. Conclusions and future work

In this paper, We have described the constraint solving approach of schedulability analysis. Scheduler behavior is decomposed into smaller state-based behaviors in order to encode them into constraints. Then, our constraint based schedulability analysis was evaluated with two task models.

Treating schedulability analysis as a constraint satisfaction problem has some benefits. Constraints are focused on the problem itself and it is based on logic formula which is simple and clear. More importantly, modern SMT solvers can solve constraints very quickly.

Constraint solving can be very powerful on describing problem. If a problem is encoded exactly as intended, it will give the exact result. However, few minor mistakes on constraints can give out an awful result which, however, perfectly satisfies encoded constraints.

For our future work, we intent to work on hierarchical system scheduling. It will be challenging since hierarchical system can be very complex in constraint point of view.

Acknowledgement

This research was supported by the MSIP (Ministry of Science, ICT and Future Planning), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2017-2015-0-00445) supervised by the IITP (Institute for Information & communications Technology Promotion)

References


