IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received August 13, 2020, accepted August 29, 2020, date of publication September 9, 2020, date of current version September 22, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3022847

Constraint-Based Schedulability Analysis in Multiprocessor Real-Time Systems

HYUK LEE AND JIN-YOUNG CHOI, (Member, IEEE)
School of Cybersecurity, Korea University, Seoul 02841, South Korea

Corresponding author: Jin-Young Choi (narnia@korea.ac.kr)

This work was supported by the Next-Generation Information Computing Development Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science and ICT under Grant 2017M3C4A7083676.

ABSTRACT As the demand for more functions and capabilities in the system increases, the application of multiprocessors has brought advantages in many ways. Many systems now have multiprocessors, and safety-critical systems with real-time properties are no exception. In these systems where the satisfaction of real-time properties is directly linked to the safety of life, the predictability of the behavior is very important, and the behavior of the system can be predicted using the schedulability analysis. In this paper, we propose the schedulability analysis of a real-time system in a homogeneous multiprocessor environment through constraint solving approach. First, the constraints that must be satisfied in order for the system to function properly were derived. These include the constraints of the task behavior, the scheduling behavior, and the operating conditions of a homogeneous multiprocessor environment. Once all the constraints were identified, they were encoded in the form of first-order logic expressions. The encoded constraints are then entered into a constraint solver along with a set of tasks. Finally, the solver provides a schedulable answer if the set of tasks satisfies all the constraints.

INDEX TERMS Constraint satisfaction problem, satisfiability modulo theories, real-time schedulability analysis, multiprocessor schedulability analysis.

I. INTRODUCTION

Multiprocessors systems are used in a variety of applications, including safety-critical systems with real-time properties, such as control systems used in automotive and avionics [1]. The complexity of the system has increased and the functionality that was previously implemented in hardware has been replaced by software. Thus, the multiprocessor paradigm has brought efficient and better computing power to improve the functionality and capability of the software [2]. However, in order to ensure real-time properties in safety-critical system, the behavior of the system must be predictable, which can be achieved through a schedulability analysis [3].

There are two approaches to schedulability analysis in multiprocessor systems: utilization bound tests and simulation [3]–[13]. Schedulability analysis using utilization bound tests provides safe results but lacks flexibility, whereas simulation-based analysis is flexible but not safe because it cannot be applied to all possible state traces. There are various studies using model checking techniques to overcome the shortcomings of simulation by identifying all possible status traces. However, there is still the problem of state explosion, which is the limitation of the model checking techniques. In this paper, the framework we proposed uses a constraint solving approach that does not have a state explosion problem to solve scheduling problems.

Our approach considers the scheduling problem as a constraint satisfaction problem and focuses on whether a given set of tasks satisfies the constraints. The motivation of this paper is to present how a real-time system in a homogeneous multiprocessor environment can be encoded into a set of constraints, and that a real-time schedulability can be analyzed by showing satisfiability of such a set of constraints.

We define the system model which we used in schedulability analysis of real-time systems in a homogeneous multiprocessor environment. Also, we identify constraints for schedulability analysis in the system model. Once the system model definition and constraints identification is done, we encode these into the form of simple, clear, and strict logical expressions using first-order logic. In doing so, we create

The associate editor coordinating the review of this manuscript and approving it for publication was Yang Liu.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

a set of constraints that let us know whether a given set of tasks can be scheduled according to a specific scheduling policy. In this paper, we have employed a preemptive dynamic-priority scheduling algorithm. The constraint set is used as an input to the satisfiability modulo theories (SMT) solver along with the set of tasks for which we want to analyze the scheduleability. SMT solver returns a model of interpretation if all of the constraints are satisfied, otherwise, it returns *Unsatisfiable*. This indicates whether a given set of tasks can be scheduled or not.

In this paper, we propose extension of our past method [14] which can be applied to a real-time system in a homogeneous multiprocessor environment. The contributions of this paper are 1) A method for transforming a real-time system in a homogeneous multiprocessor environment into a set of constraints of simple, clear, and strict logical expressions is proposed, 2) A way to prove that a real-time schedulability in a homogeneous multiprocessor environment can be analyzed by showing satisfiability of such a set of constraints is presented.

The remainder of this paper is structured as follows. We discuss different methods for schedulability analysis in a multiprocessor environment based on formal techniques in Section II. In order to represent real-time systems in a multiprocessor environment as a set of constraints, we define system model in Section III. Section IV identifies the constraints of the system model and represents them in the form of logical expressions using first-order logic. The implementation and examples of the proposed method are described in Section V. In Section VI, we present the conclusions of this paper and discuss future research.

II. RELATED WORK

Analytical approach, such as utilization bound test, is difficult to apply to complex systems with multiprocessors or hierarchies. In these system environments, a simulation-based approach is more appropriate, but the aforementioned disadvantages must be overcome. For this reason, many studies of simulation-based schedulability analysis have considered various formal modeling languages [15]–[24]. Especially, timed automata (TA) have been used widely to specify system models [20], [24]–[29]. Krcal *et al*. [25] propose schedulability analysis using task automata (TA extended with tasks), and focus on partitioned scheduling (i.e., Each task runs on a fixed processor and cannot be migrated to another processor). Guan *et al*. [20] also use TA to specify system models. However they use UPPAAL as a modeling tool and a model checker of a given system model. In [26], Guan *et al*. use NuSMV as a modeling tool in a similar way. Also, Gu *et al*. [28] use MPETA which is an extension of TA for multiprocessor scheduling. Boudjadar *et al*. [24] propose Job automata with hierarchical scheduling framework, which separate task instantiation from task automata in [30]. Other than techniques based on TA, Oğuz *et al*. [31] use timed CSP as a modeling language and PAT model checking for non-preemptive fixed-priority multiprocessor scheduling.

Model checking tools such as UPPAAL, NuSMV, and PAT are very useful tools and they provide a fast, automated verification technique. However, the model checking technique itself has a *state-explosion problem* in which the number of states increases exponentially as the number of variables increases [32]–[34]. Unlike model checking, our approach, constraint solving with boolean satisfiability problem (SAT) which is based on boolean expressions does not suffer from the state-explosion problem.

At a very abstract level, the difference between our approach and model checking is as follows. In model checking, the system behavior model is built using modelling language, and verification properties (e.g., schedulability) are translated in temporal logic. The model checker then finds out whether the property is true in the system behavior model. On the other hand, in our approach, the constraints of system behavior and schedulability condition (i.e., what must be satisfied at each behavioral step) are modelled in a first-order logic based language. Then, the constraint solver finds an interpretation that satisfies all of the constraints. Our approach uses SMT, an extension of the SAT, which uses first-order logic with some useful theories such as linear arithmetic theory, bit vectors, arrays, and strings for problem solving [35].

In addition to the techniques mentioned above, there are studies that analyze the schedulability as a constraint problem. Kwak *et al*. [16] present schedulability analysis framework based on Algebra of Communicating Shared Resources with Value Passing (ACSR-VP). The specification in ACSR-VP is analyzed by means of a symbolic algorithm to produce a set of the predicate equations that can be solved later by a constraint solver. However, they do not attempt to solve these constraints in their work. Also, a process algebraic approach uses a parallel composition method to simulate the behavior of the tasks together, which can cause scalability limitations. Hong *et al*. [36] propose a SAT-based scheduling method to solve the dynamic offline scheduling problem in big data processing system. They focus on finding schedulable answers through the SAT solver, not schedulability analysis in a given scheduling policy. Zhang *et al*. [37], [38] propose a framework of transforming the scheduling problem of Clock Constraint Specification Language (CCSL) into SMT-based decision procedure. Pedro *et al*. [39] also propose a transformation of a scheduling problem in Restricted Metric Temporal Logic with durations (RMTL-$\int$) into the Satisfiability Modulo Theories Library (SMT-LIB) language. Cheng *et al*. [22], [23] propose SMT-based scheduling method for overloaded real-time systems in a uniprocessor and multiprocessor environment. In this case, they focus on maximizing the number of schedulable tasks without a specific scheduling algorithm. However, our work is aimed at scheduling hard real-time tasks using conventional scheduling disciplines. Our previous work [14] presents constraint solving approach to schedulability analysis in a uniprocessor environment. The main differences from our previous work [14] are 1) Task behavior model is

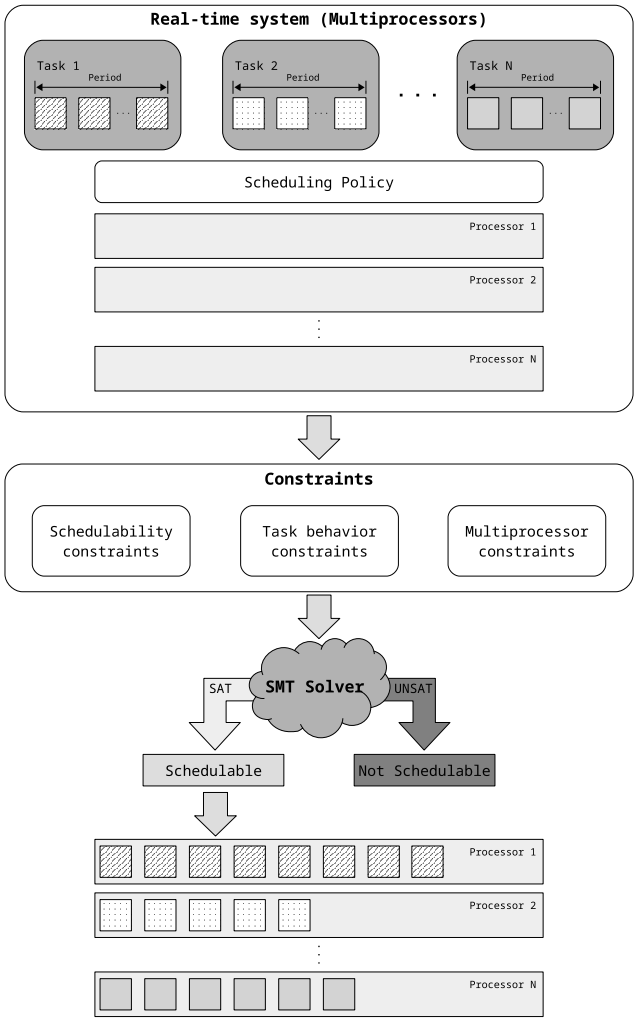

FIGURE 1. Overview of the proposed approach.

generalized for better expression and 2) The framework is extended to represent the task behavior in a multiprocessor environment, as shown in Fig. 1.

III. SYSTEM AND TASK BEHAVIOR MODEL

A. SYSTEM MODEL

A real-time system consists of a set of real-time tasks $\Gamma = (\tau_1, \tau_2, \cdots, \tau_n)$ and a real-time task τ_i, where $\tau_i \in \Gamma$, is characterized by $\tau_i = (C_i, D_i, T_i)$ in which C_i is a worst-case computation time, D_i is a deadline, and T_i is a period for task τ_i respectively. For a periodic task τ_i, T_i represents the period of the task such that the time interval between the release of jobs separated by T_i. If a task τ_i has an implicit deadline, that is, the deadline of τ_i is equal to the period (i.e., $D_i = T_i$), then the deadline can be omitted as $\tau_i = (C_i, T_i)$. In a homogeneous multiprocessor environment, scheduling a set of tasks is done on m identical processors, $P = \{p_1, \cdots, p_m\}$. We assume that the time domain uses discrete time in our approach. Further assumptions for the tasks are as follows:

- All tasks are periodic
- All tasks have known worst-case computation time

TABLE 1. Symbols and Definitions.

Symbol	Definition
Γ	set of real-time tasks, $\Gamma = \{\tau_1, \cdots, \tau_n\}$
P	set of processors, $P = \{p_1, \cdots, p_m\}$
τ_i	real-time task, $\tau_i \in \Gamma$
p_j	processor, $p_j \in P$
C_i	computation time of τ_i
D_i	deadline of τ_i
T_i	period of τ_i
T_Γ	hyper-period of set Γ (i.e., $T_\Gamma = lcm(T_1, \cdots, T_n)$)
$es_{i,j}^k$	execution status of τ_i in processor p_j at kth step
ct_i^k	computed time of τ_i at kth step
tp_i^k	priority of τ_i at kth step

- All tasks have implicit deadline
- All tasks arrive and are released at the same time
- All tasks are independent, with no shared resources
- All costs for context switching and migration overhead are ignored (assuming they are included in worst-case computation time).

1) SCHEDULABILITY

For a given set of tasks, if all tasks can complete the execution time for their periods, the task set is said to be schedulable.

A periodic task is that the task is regularly invoked with the fixed time interval. When scheduling periodic tasks, there exists a certain behavioral pattern that repeats infinitely. It is called a hyper-period.

2) HYPER-PERIOD

The hyper-period of a task set is the minimum time interval in which the periodic pattern of all tasks is repeated. It is defined as the least common multiple (LCM) of all task periods ($T_\Gamma = lcm(T_1, \cdots, T_n)$).

Lemma 1: If a periodic task set S can be scheduled up to the hyper-period, then S is schedulable [40].

Our task model is based on preemptive dynamic-priority scheduling. In our task model, the general behavior of task execution can be described in two perspectives (i.e., task perspective and scheduler perspective). From a task perspective, the task execution within a period can be briefly described as follows. If the task consumed all of its computation time, then the task remains idle until the next period. Otherwise, the task is assigned a priority according to the scheduling policy and waits for execution. From the viewpoint of the scheduler, the task execution in the period can be described as follows. If there is an available processor, the processor executes one task. Or, if more than one processor is available, the processor executes the same number of tasks as the available processor, in the highest priority order. Otherwise, no tasks are executed. For a given set of tasks, if all tasks are appropriately allocated and schedulable across multiple processors, then the set of tasks is schedulable in a given multiprocessor environment.

The timing graph in Fig. 2 shows the typical scheduling behavior of task set Γ_1 under global rate monotonic (G-RM) scheduling algorithm in a multiprocessor environment with

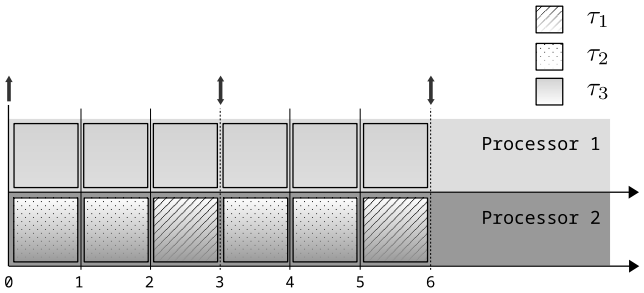

FIGURE 2. Multiprocessor scheduling task set Γ_1 under G-RM.

two processors, where $\Gamma_1 = \{\tau_1, \tau_2, \tau_3\}$, $P = \{p_1, p_2\}$, $\tau_1 = (1, 3)$, $\tau_2 = (2, 3)$, and $\tau_3 = (3, 3)$. The priority is $(\tau_3 > \tau_2 > \tau_1)$, and the up and down arrows indicate the beginning and end of the each task period, respectively. In Fig. 2, processors p_1 and p_2 execute both τ_3 and τ_2, concurrently. For each processor, task is selected for execution according to the scheduling policy. However, subtask-level parallelism is not allowed. That is, at time instant zero, τ_3 which is the highest priority, can not be executed in parallel on both processors p_1 and p_2.

3) MULTIPROCESSOR SCHEDULABILITY

In a multiprocessor environment, a real-time system is said to be schedulable with a certain scheduling algorithm if, it is possible to find a mapping from tasks to processors such that all the tasks are schedulable with the certain scheduling algorithm.

In a multiprocessor environment, scheduling methods can be split in two major ways. One is global scheduling with a common ready queue for all processors, and the other is a partitioned scheduling with a ready queue dedicated to each processor. There is also a semi-partitioned scheduling that combines two methods that are not yet popular. Each scheduling method has advantages and disadvantages. In partition scheduling [41]–[45], each processor has a dedicated ready queue, which has the advantage of being able to apply a uniprocessor scheduling algorithm. However, proper distribution of tasks is another issue that needs to be addressed and is known to be solvable by the bin packing algorithm [46]. There are many studies on constraint-based approaches in the bin packing algorithm [47]–[51]. In global scheduling [44], [45], all processors use a common ready queue, so tasks are not dependent on any processor. In other words, task migration can make processor operations more efficient. However, the optimal scheduling algorithm used in a uniprocessor environment can no longer guarantee optimal results.

Scheduling disciplines are algorithms that determine how resources are allocated [4]. Well-known scheduling algorithms in a uniprocessor are Rate Monotonic (RM), Earliest Deadline First (EDF), and Least Laxity First (LLF). Their extensions are G-RM, G-EDF, and G-LLF, which can also be used for global scheduling in a multiprocessor environment. While these scheduling algorithms no longer guarantee optimal scheduling results, the proposed approach focuses on constraint-based methodology rather than finding the optimal solution. The following formulas are used to calculate task priority at a kth step:

- G-RM: $D_{max} - T_i$
- G-EDF: $D_{max} - (D_i - (k \bmod T_i))$
- G-LLF: $D_{max} - (D_i - (k \bmod T_i) - (C_i - ct_i^k))$
  where $D_{max} = 1 + \max(D_i)$

B. COMPARING TASK BEHAVIOR MODEL

In our previous work [14], we considered scheduling in a uniprocessor environment and used a state-based task behavior model. The state-based task behavior model and the execution-based task behavior model are briefly described as follows.

In the state-based task behavior model, the state of a task is determined based on the current status of the state variables (i.e., accumulated execution time and the elapsed time of a task). The task is in one of the following four states: *Start, Wait, Ready*, and *Deadlock*, depending on whether the computation time is left and the time to deadline is left. *Start* state indicates that the task has reached its deadline and that the computation time is complete. If the task has not reached its deadline, it is in *Wait* state when the computation time is complete, and in *Ready* state if the computation time is not completed. *Deadlock* state indicates that the task has reached its deadline but has not completed its calculation time. The term *Deadlock* was used to indicate that the task was in a dead-end state where it could no longer proceed. The state transition occurs according to the priority of the current state and determines the next state which changes the state variables of all tasks.

The main advantage of using the state-based task behavior model is that the constraint encoding for state transitions is straightforward. However, expanding the task model into a multiprocessor environment complicates the state transition rules and reduces the flexibility of constraint encoding.

In this paper, we use a basic behavior model, which is an execution-based behavior model for scheduling in a multiprocessor environment. In the execution-based behavior model, tasks consume computation time independently of each other. Therefore, constraint expressions for task execution behavior can be easily extended. The basic task execution constraints are as follows. A task must complete their computation time before its deadline. The number of tasks that can be executed simultaneously cannot be greater than the number of processors.

IV. ENCODING CONSTRAINTS

In this section, constraints for the schedulability analysis are described, and the constraints encoded in first-order logic are presented. The constraint encoding of schedulability analysis is roughly divided into three categories. The first is constraints on the general behavior that a task must satisfy. The second is the constraints for a task to be executed in a

multiprocessor environment. The third is the constraints that must be satisfied in order for the task to be schedulable.

The variables used for the constraints are *es*, *ct*, *tp*, and k, representing execution status, computed time, task priority, and step index, respectively. Execution status describes the task execution (i.e., `TRUE` or `FALSE`), and $es_{i,j}^k$ represents the execution status of task τ_i in processor p_j at a kth step. Computed time describes the number of jobs that the task executed, and ct_i^k represents the computed time of task τ_i at a kth step. Similarly, tp_i^k represents the priority of task τ_i at a kth step.

A. TASK BEHAVIOR CONSTRAINTS

The following constraints must be satisfied in order for the task to be schedulable. The schedulability of periodic task sets can be verified by schedulability analysis up to hyper-period by Lemma 1. Therefore, all tasks in Γ must satisfy the following constraints for any step index k within T_Γ.

1) EXECUTION RANGE

The computed time cannot exceed the computation time or be negative. This constraint indicates that the task can execute only as much as the computation time. Based on these attributes, the constraints on the computed time can be defined as (1):

$$\mathit{ExecRange} \stackrel{def}{=} \forall \tau_i \in \Gamma\ \forall k \in \{0, \cdots, T_\Gamma\} \left(ct_i^k \geq 0 \wedge ct_i^k \leq C_i\right) \quad (1)$$

2) INSTANT EXECUTION

In order to prevent unnecessary resource waste, the ready tasks are executed immediately when resources are available. Based on these attributes, the constraints for instant execution can be defined as (2):

$$\begin{aligned}\mathit{InstExec} \stackrel{def}{=} &\forall \tau_i \in \Gamma\ \forall k \in \{0, \cdots, T_\Gamma\} \\ &\Big((tp_i^k > 0) \rightarrow \left(\exists p_j \in P(es_{i,j}^k == \texttt{TRUE})\right) \vee \\ &\left(\forall p_j \in P\ \exists \tau_{i'} \in \Gamma\ (i \neq i') \wedge\right. \\ &\left.(tp_{i'}^k \geq tp_i^k) \wedge (es_{i',j}^k == \texttt{TRUE})\right)\Big) \end{aligned} \quad (2)$$

3) ASSIGN PRIORITY

This constraint guarantees that the appropriate priority is assigned to the tasks which have not yet completed execution, according to the scheduling policy. The tasks that have completed their execution time receives the lowest priority. The calculation of priorities for scheduling policies (e.g., G-RM, G-EDF, and G-LLF) can be done using the formulas provided in the previous section. Based on these attributes, the constraints for assign priority can be defined as (3):

$$\begin{aligned}\mathit{AssignPriority} \stackrel{def}{=} &\forall \tau_i \in \Gamma\ \forall k \in \{0, \cdots, T_\Gamma\} \\ &\Big(\left((ct_i^k < C_i) \rightarrow (tp_i^k == \texttt{Priority})\right) \\ &\wedge\left((ct_i^k \geq C_i) \rightarrow (tp_i^k == 0)\right)\Big) \end{aligned} \quad (3)$$

4) TASK IDLE

Tasks that have completed their computation time remain idle. *TaskIdle* is closely related to the *ExecRange* and *AssignPriority*. Tasks can only be executed as much as their computation time, and once the computation time is complete during that period, they receive the lowest priority. This constraint indicates that the task with the lowest priority must not be executed. Based on these attributes, the constraints for task idle can be defined as (4):

$$\begin{aligned}\mathit{TaskIdle} \stackrel{def}{=} &\forall \tau_i \in \Gamma\ \forall k \in \{0, \cdots, T_\Gamma\} \\ &\Big((tp_i^k == 0) \rightarrow \forall p_j \in P\ (es_{i,j}^k == \texttt{FALSE})\Big) \end{aligned} \quad (4)$$

5) EXECUTION COUNT

When a task is executed, the computed time of the task must be increased by one. Otherwise, it remains the same. This constraint indicates that the pre- and post-condition of a task execution, expressed in the execution count, should be satisfied for all tasks and their steps of execution. Based on these attributes, the constraints for task idle can be defined as (5):

$$\begin{aligned}&\mathit{ExecCount} \\ &\stackrel{def}{=} \forall \tau_i \in \Gamma\ \forall p_j \in P\ \forall k \in \{0, \cdots, T_\Gamma\} \\ &\Big(\left(((k \bmod T_i) == (T_i - 1)) \rightarrow (ct_i^{k+1} == 0)\right) \wedge \\ &\left((es_{i,j}^k == \texttt{TRUE}) \rightarrow\right. \\ &\left.\left(((k \bmod T_i) \neq (T_i - 1)) \rightarrow (ct_i^{k+1} == ct_i^k + 1)\right)\right) \wedge \\ &\left((es_{i,j}^k == \texttt{FALSE}) \rightarrow\right. \\ &\left.\left(((k \bmod T_i) \neq (T_i - 1)) \rightarrow (ct_i^{k+1} == ct_i^k)\right)\right)\Big) \end{aligned} \quad (5)$$

6) RESTART PERIOD

Periodic tasks have computation times to execute in each period. The computed time is set to the default value at the beginning of every period. This constraint indicates that the computed time of a task should be zero for every first step of the period. Based on this attribute, the constraint for restart period can be defined as (6):

$$\begin{aligned}&\mathit{RestartPeriod} \\ &\stackrel{def}{=} \forall \tau_i \in \Gamma\ \forall k \in \{0, \cdots, T_\Gamma\} \\ &\Big(((k \bmod T_i) == 0) \rightarrow (ct_i^k == 0)\Big) \end{aligned} \quad (6)$$

B. MULTIPROCESSOR ENVIRONMENT CONSTRAINTS

The following constraints must be satisfied in order for the task to behave correctly in a multiprocessor environment.

1) ATOMIC EXECUTION

A processor can execute only one task at a time. This constraint indicates that if a task is executed at a particular time in a particular processor, then another task should not be executed at that time in that processor. This property can be regarded as a conservative extension of the atomic execution

of a uniprocessor. Based on these attributes, the constraints for atomic execution can be defined as (7):

$$\begin{aligned} &AtomExec \\ &\overset{def}{=} \forall \tau_i \in \Gamma\ \forall p_j \in P\ \forall k \in \{0, \cdots, T_\Gamma\} \\ &\quad \Big((es^k_{i,j} == \texttt{TRUE}) \rightarrow \\ &\quad \forall \tau_{i'} \in \Gamma\ \big((i \neq i') \wedge (es^k_{i',j} == \texttt{FALSE}) \big) \Big) \end{aligned} \tag{7}$$

2) PARALLEL EXECUTION

At any point in time, as many tasks as the number of processors can be executed simultaneously. However, a task cannot be executed on more than one processor. This constraint indicates that if a task is executed at a specific time in a particular processor, then the task cannot be executed at the same time in another processors. Based on these attributes, the constraints for job-level parallelism prohibition can be defined as (8):

$$\begin{aligned} &NoParallel \\ &\overset{def}{=} \forall \tau_i \in \Gamma\ \forall p_j, p_{j'} \in P\ \forall k \in \{0, \cdots, T_\Gamma\} \\ &\quad \Big((es^k_{i,j} == \texttt{TRUE}) \rightarrow \\ &\quad \big((j \neq j') \wedge (es^k_{i,j'} == \texttt{FALSE}) \big) \Big) \end{aligned} \tag{8}$$

C. SCHEDULABILITY CONSTRAINTS

To ensure that a set of tasks can be scheduled in a multiprocessor environment, the following constraints must be satisfied.

1) SCHEDULABLE CONSTRAINTS

In order for a given set of tasks to be schedulable, all tasks must execute all their computation time within their respective periods before the deadline. This constraint indicates that the execution counters for all tasks must be equal to their computation time at the end of each period. Based on these attributes, the constraints for schedulable constraints can be defined as (9):

$$\begin{aligned} Schedulable \overset{def}{=} &\forall \tau_i \in \Gamma\ \forall k \in \{0, \cdots, T_\Gamma\} \\ &\Big(((k \bmod T_i) == D_i) \rightarrow (ct^k_i == C_i) \Big) \end{aligned} \tag{9}$$

D. PUTTING IT ALL TOGETHER

All encoding of constraints for schedulability analysis has been completed. The conjunction of these constraints is all the constraints that must be satisfied in order for a set of tasks to be schedulable. By putting all of these encoded constraints together, the constraints on the schedulability analysis of a given task set can be defined as (10):

$$\begin{aligned} MPSCHED \overset{def}{=} \big(&ExecRange \wedge InstExec \\ &\wedge AssignPriority \wedge TaskIdle \\ &\wedge ExecCount \wedge RestartPeriod \\ &\wedge AtomExec \wedge NoParallel \\ &\wedge Schedulable \big) \end{aligned} \tag{10}$$

Theorem 1: The real-time periodic task set S is schedulable in a multiprocessor environment iff S satisfies the set of constraints MPSCHED.

Proof: ($\Rightarrow$): Let S be a real-time periodic task set. To prove Theorem 1, assume S is a schedulable task set. Following the definitions of *Schedulability*, *Multiprocessor schedulability* and by Lemma 1, all tasks in task set S complete the computation time for all their periods up to the hyper-period with given processors. Because the constraints *Schedulable*, *AtomExec*, and *NoParallel* in MPSCHED, tasks with incomplete computation times within a hyper-period and multiple tasks running on a single processor and a single task on multiple processors are not allowed. Thus, if task set S is scheduleable, then task set S satisfies MPSCHED.

($\Leftarrow$): Assume task set S satisfy the set of constraints MPSCHED. Following the definition of MPSCHED, the behavior of task set S satisfies all of the constraints defined in this section. Especially, by constraints *Schedulable*, *AtomExec*, and *NoParallel*, all tasks in the task set S satisfy the following constraints up to the hyper-period; the completion of computation time for all the respective periods and the execution of atomic task per atomic processor for a given number of processors. Thus, by the definitions of *Schedulability*, *Multiprocessor schedulability* and Lemma 1, the given task set S is schedulable in a multiprocessor environment. □

Apart from the above constraints, additional constraint can be defined for the consecutive task execution in a multiprocessor environment. The addition of constraints is a time-consuming addendum from a verification perspective. However, this is a factor that can reduce unnecessary overhead when tasks are run in real-world environments.

Consecutive Execution. When a task is executed consecutively, it must be executed on the same processor. This constraint indicates that if a task is executed consecutively more than a single time unit, the processor on which it executes must be the same processor. Based on these attributes, the constraints for consecutive execution can be defined as (11):

$$\begin{aligned} &ConseExec \\ &\overset{def}{=} \forall \tau_i \in \Gamma\ \forall p_j, p_{j'} \in P\ \forall k \in \{0, \cdots, T_\Gamma\} \\ &\quad \Big(\big((es^k_{i,j} == \texttt{TRUE}) \wedge (es^{k+1}_{i,j'} == \texttt{TRUE}) \big) \rightarrow \\ &\quad (j == j') \Big) \end{aligned} \tag{11}$$

V. IMPLEMENTATION AND EXPERIMENTS

This section presents an implementation of the schedulability analysis based on the proposed approach. We used Python and Z3Py, a Python API for SMT solver Z3 [52], for implementation. The pseudo style code of the implementation is presented in Algorithm. 1. Z3py allows almost direct encoding of first-order logic, the constraints in the pseudo code may look very similar to what we have encoded in the previous section. For a given set of tasks, the solver verifies

Algorithm 1 A Pseudo Code for Schedulability Analysis

Input : A list of tasks τ_i, where $i > 1$, and a number of processors p_j, where $j \geq 1$.
Output : Schedulability Result. (Satisfiable (with model) / Unsatisfiable)

```
/* Task information is given as the
   form of a tuple of (C_i, D_i, T_i).      */
Period_HYPER = LCM(T_i)      // Least Common
Multiple of periods.
Con_TASK = ExecRange ∧ InstExec ∧ AssignPriority
∧ TaskIdle ∧ ExecCount ∧ RestartPeriod
Con_ENV = AtomExec ∧ NoParallel
Con_SCHED = Schedulable
MPSCHED = Con_TASK ∧ Con_ENV ∧ Con_SCHED
/* Encoded constraints for each
   peoperties to be satisfied.      */
solver = [ ]
solver.append(MPSCHED)
/* Add all the encoded constraints
   into solver.                     */
if solver.check() != sat then
    return Unsatisfiable
else
    return Satisfiable, solver.model()
```

that all the constraints defined in the previous section are satisfied. If all constraints are satisfied, that is, if the result is *Satisfiable*, the solver returns its interpretation, called the *Model*. Otherwise, the solver returns *Unsatisfiable*, meaning that the solver has not found an interpretation that satisfies all the constraints.

We performed experiments on schedulability analysis for several example task sets using the implementation of the proposed approach. The system environment for the experimentation was as follows: *Intel i7 CPU 3.40GHz Octacore CPU with 16 GB RAM, Linux kernel 4.4.0-21-generic x86_64, Z3 version 4.4.2, Python 2.7.12*. The experiment was carried out in two parts. One is to compare the performance of the proposed method with our previous study [14] and to show the performance difference of the schedulability analysis due to the task behavior model. The other was an experiment to investigate the performance of the schedulability analysis in a multiprocessor environment. In order to avoid confusion, we refer to the method used in previous study as the past method, and the method proposed in this paper as the present method or the proposed method.

A. PERFORMANCE COMPARISON WITH PREVIOUS METHOD

The first part of the experiment compares the performance between the past and present methods in a uniprocessor environment and the results are shown in Table. 2. In this experiment, we experimented with example task sets that contained 2, 3, 4, and 8 tasks, respectively, to level with the previous method. Each example task set has a task utilization of 1, where the utilization is $\left(\sum_{\tau_i} C_i/T_i\right)$. Both the example task sets 8(a) and 8(b) have 8 tasks, but 8(b) has a longer hyper-period configuration, which in the end requires more execution. This configuration increases the solver's search space to find a satisfactory solution.

TABLE 2. Performance comparison of schedulability analysis for past and present methods.

	Time (s)		Memory (MB)	
No. of Task	Past	Present	Past	Present
2	0.054	0.017	1.93	1.55
3	0.072	0.024	2.83	1.76
4	0.163	0.044	4.11	2.22
8	5.115	0.114	42.52	4.67
8(a)	19.754	0.285	63.34	7.72
8(b)	68.049	0.362	76.06	8.10

The results show that for all cases the present method performs better in a uniprocessor environment, both in terms of time spent and the maximum amount of memory used. In particular, the difference is large for example task set with 8 tasks.

In our observation, the first noticeable difference is the way in which problems are represented. In other words, we have further simplified the way in which the actions of the task are represented in the present method, thereby reducing the state space for the representation of the overall task behavior. The second observation is obvious one that the complexity of a task set affects performance. There was also an increase in the number of possible state combinations, which was a major factor complicating the implementation of encoded constraints. The present method, the execution-based behavioral model, has overcome the scalability problem for task sets with many tasks, as shown in Table. 2.

B. PERFORMANCE EVALUATION OF PROPOSED METHOD

The second part of the experiment evaluates the performance of the proposed method in a multiprocessor environment. The experiment was performed on a task set of up to 32 tasks in 2, 3, 4, 8, and 16 multiprocessors environment, and the results are shown in Table. 3. Task sets with less than the number of processors and task sets that failed to output verification results within 72 hours (i.e., 259,200 seconds) were excluded. Also, task utilization of every task set in the example is 1, where $\left(\sum_{\tau_i} C_i/T_i/m\right) = 1$, and m is the number of processors.

We observed several points on following aspects through the experiment. The performance values, the time spent and the maximum memory used for schedulability analysis are proportional to the number of tasks and the number

TABLE 3. Performance evaluation of schedulability analysis with proposed method.

No. of Processors	2		3		4		8		16	
No. of Task	Time (s)	Memory (MB)	Time	Memory	Time	Memory	Time	Memory	Time	Memory
2	0.029	1.67	·	·	·	·	·	·	·	·
3	0.045	2.04	0.184	6.15	·	·	·	·	·	·
4	0.066	2.56	0.296	7.79	0.411	10.76	·	·	·	·
8	0.208	6.55	1.601	20.84	3.201	26.24	7.337	18.61	·	·
16	1.866	27.21	212.368	80.22	190.644	98.41	335.426	97.87	17391.952	414.08
24	1539.498	77.77	8489.691	210.21	6781.216	285.59	10094.168	323.51	-	-
32	253.627	170.39	176646.922	719.37	-	-	-	-	-	-

of processors. These factors ultimately determine the size of the space that the solver must seek to find the answer.

Since the verification time was limited to 72 hours, some experimental results were missing. In Table. 3, the dash(-) indicates that the result is omitted due to timeout and dot(·) indicates that the experiment didn't take place due to complexity reason (i.e., no. of tasks < no of processors). For example, the number of task sets that gave verification result in time decreased with the number of processors added. In the 4-processor and 8-processor environments, the verification result was given up to a set of tasks consisting of 24 tasks. In the 16 processor environment, only the task set consisting of only 16 tasks gave the verification result.

Although the proposed method has reduced the time and memory required for schedulability analysis compared to the past method, it still requires extensive resources. To reduce the verification time, we need to find a way to optimize the task behavior representation. For this reason, the proposed method is more suitable for systems where ensuring schedulability at the design phase is important, such as safety-critical systems.

VI. CONCLUSION AND FUTURE WORK

A schedulability analysis can be used to predict the behavior of a real-time system to ensure that the behavior satisfies real-time properties. In this paper, we have described the schedulability analysis of real-time system in multiprocessor environment using the constraint solving approach.

In order to use constraint approach on schedulability analysis, the behaviors of a real-time system in multiprocessor environment need to be identified and be expressed in the form of constraints. After that, the constraints are encoded as first-order logic expressions, along with a task set, use as input into SMT solver, and have let SMT solver find an answer which satisfies all the constraints.

Several experiments were performed to investigate the performance of the proposed method and the factors affecting performance were examined. We observed that, in the proposed method, the time spent and the memory used for the schedulability analysis of real-time system in a multiprocessor environment increased in proportion to the number of tasks and the number of processors. This is because these factors are closely related to the size of the space created when finding satisfactory answers to all the constraints.

The proposed method is one of many possible methods that can be applied for schedulability analysis based on constraint solving approach. It appears that there is room for improvement to achieve better performance through various task behavior models and constraint expression methods.

In future research, we plan to optimize the representation of task model and constraints to improve performance. We also plan to extend our research to scheduling for dependent tasks and schedulability analysis of hierarchical real-time systems.

Although we did not consider a scheduling of dependent tasks in this paper, it seems possible to apply the proposed method to a task model using shared resources. We can not discuss the idea in detail, but here are some thoughts.

Consider two tasks τ_i and τ_j, where $i \neq j$, sharing a resource CS_{ij}. In order for τ_i to be scheduled with CS_{ij} at a specific time, CS_{ij} should not be occupied by τ_j at that time. This attribute is one of the execution constraint of τ_i.

With respect to hierarchical scheduling, the constraints of the task model become more complex. For example, consider a following configuration:

- Three sets of tasks Γ_{L1}, Γ_{L2}, and Γ_{L3} in different hierarchy level (Γ_{L1} is the highest level)
- $\Gamma_{L1} = \{\tau_1, \tau_2\}$, $\Gamma_{L2} = \{\tau_3, \tau_4\}$ and $\Gamma_{L3} = \{\tau_5, \tau_6\}$
- τ_1 associated with τ_3, τ_4
- τ_3 associated with τ_5, τ_6

In order for task set Γ_{L1} to be schedulable, the execution constraints of τ_1 must include the execution constraints in which tasks τ_3, τ_4, τ_5, and τ_6 can be scheduled.

REFERENCES

[1] R. Mall, *Real-Time Systems: Theory and Practice*. Noida, India: Pearson, 2009.

[2] S. Baruah, M. Bertogna, and G. Buttazzo, *Multiprocessor Scheduling for Real-Time Systems*. Cham, Switzerland: Springer, 2015.

[3] N. C. Audsley, A. Burns, R. I. Davis, K. W. Tindell, and A. J. Wellings, "Fixed priority pre-emptive scheduling: An historical perspective," *Real-Time Syst.*, vol. 8, nos. 2–3, pp. 173–198, 1995.

[4] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a Hard-Real-Time environment," *J. ACM*, vol. 20, no. 1, pp. 46–61, Jan. 1973.

[5] K. Jeffay, D. F. Stanat, and C. U. Martel, "On non-preemptive scheduling of period and sporadic tasks," in *Proc. 12th Real-Time Syst. Symp.*, 1991, pp. 129–139.
[6] K. Tindell and J. Clark, "Holistic schedulability analysis for distributed hard real-time systems," *Microprocess. Microprogramm.*, vol. 40, nos. 2–3, pp. 117–134, Apr. 1994.
[7] M. Litoiu and R. Tadei, "Real time task scheduling allowing fuzzy due dates," *Eur. J. Oper. Res.*, vol. 100, no. 3, pp. 475–481, Aug. 1997.
[8] J. C. Palencia and M. Gonzalez Harbour, "Schedulability analysis for tasks with static and dynamic offsets," in *Proc. 19th IEEE Real-Time Syst. Symp.*, Dec. 1998, pp. 26–37.
[9] S. Manolache, P. Eles, and Z. Peng, "Schedulability analysis of applications with stochastic task execution times," *ACM Trans. Embedded Comput. Syst.*, vol. 3, no. 4, pp. 706–735, Nov. 2004.
[10] M. Bertogna, M. Cirinei, and G. Lipari, "Improved schedulability analysis of EDF on multiprocessor platforms," in *Proc. 17th Euromicro Conf. Real-Time Syst. (ECRTS)*, 2005, pp. 209–218.
[11] P. K. Muhuri and K. K. Shukla, "Real-time task scheduling with fuzzy uncertainty in processing times and deadlines," *Appl. Soft Comput.*, vol. 8, no. 1, pp. 1–13, Jan. 2008.
[12] F. Zhang and A. Burns, "Schedulability analysis for real-time systems with EDF scheduling," *IEEE Trans. Comput.*, vol. 58, no. 9, pp. 1250–1258, Sep. 2009.
[13] G. Zhang, H. Xu, H. Gao, and A. Liu, "Applying probability model to the genetic algorithm based cloud rendering task scheduling," in *Proc. 29th Int. Conf. Softw. Eng. Knowl. Eng.*, Jul. 2017, pp. 12–17.
[14] H. Lee and J.-Y. Choi, "Constraint solving approach to schedulability analysis in real-time systems," *IEEE Access*, vol. 6, pp. 58418–58426, 2018.
[15] J.-Y. Choi, I. Lee, and H.-L. Xie, "The specification and schedulability analysis of real-time systems using ACSR," in *Proc. 16th IEEE Real-Time Syst. Symp.*, Dec. 1995, pp. 266–275.
[16] H.-H. Kwak, I. Lee, A. Philippou, J.-Y. Choi, and O. Sokolsky, "Symbolic schedulability analysis of real-time systems," in *Proc. 19th IEEE Real-Time Syst. Symp.*, Dec. 1998, pp. 409–418.
[17] H. Ben-Abdallah, J.-Y. Choi, D. Clarke, Y. S. Kim, I. Lee, and H.-L. Xie, "A process algebraic approach to the schedulability analysis of real-time systems," *Real-Time Syst.*, vol. 15, no. 3, pp. 189–219, 1998.
[18] A. N. Fredette and R. Cleaveland, "RTSL: A language for real-time schedulability analysis," in *Proc. Real-Time Syst. Symp.*, 1993, pp. 274–283.
[19] S.-J. Kim and J.-Y. Choi, "Formal modeling for a real-time scheduler and schedulability analysis," in *Proc. Int. Conf. Parallel Comput. Technol.* Cham, Switzerland: Springer, 2003, pp. 253–258.
[20] N. Guan, Z. Gu, Q. Deng, S. Gao, and G. Yu, "Exact schedulability analysis for static-priority global multiprocessor scheduling using model-checking," in *Proc. IFIP Int. Workshop Softw. Technol. Embedded Ubiquitous Syst.* Cham, Switzerland: Springer, 2007, pp. 263–272.
[21] A. de Matos Pedro, D. Pereira, L. M. Pinho, and J. S. Pinto, "Logic-based schedulability analysis for compositional hard real-time embedded systems," *ACM SIGBED Rev.*, vol. 12, no. 1, pp. 56–64, Mar. 2015.
[22] Z. Cheng, H. Zhang, Y. Tan, and Y. Lim, "Scheduling overload for real-time systems using SMT solver," in *Proc. 17th IEEE/ACIS Int. Conf. Softw. Eng., Artif. Intell., Netw. Parallel/Distrib. Comput. (SNPD)*, May 2016, pp. 189–194.
[23] Z. Cheng, H. Zhang, Y. Tan, and Y. Lim, "SMT-based scheduling for multiprocessor real-time systems," in *Proc. IEEE/ACIS 15th Int. Conf. Comput. Inf. Sci. (ICIS)*, Jun. 2016, pp. 1–7.
[24] J. Boudjadar, J. H. Kim, L. T. X. Phan, I. Lee, K. G. Larsen, and U. Nyman, "Generic formal framework for compositional analysis of hierarchical scheduling systems," in *Proc. IEEE 21st Int. Symp. Real-Time Distrib. Comput. (ISORC)*, May 2018, pp. 51–58.
[25] P. Krcal, M. Stigge, and W. Yi, "Multi-processor schedulability analysis of preemptive real-time tasks with variable execution times," in *Proc. Int. Conf. Formal Modeling Anal. Timed Syst.* Berlin, Germany: Springer, 2007, pp. 274–289.
[26] N. Guan, Z. Gu, M. Lv, Q. Deng, and G. Yu, "Schedulability analysis of global fixed-priority or EDF multiprocessor scheduling with symbolic model-checking," in *Proc. 11th IEEE Int. Symp. Object Compon.-Oriented Real-Time Distrib. Comput. (ISORC)*, May 2008, pp. 556–560.
[27] F. Yu, G. Li, and N. Xiong, "Schedulability analysis of multi-processor real-time systems using uppaal," in *Proc. 2nd Int. Conf. Inf. Sci. Eng.*, Dec. 2010, pp. 1–6.
[28] Z. Gu, Z. Wang, H. Chen, and H. Cai, "A model-checking approach to schedulability analysis of global multiprocessor scheduling with fixed offsets," *Int. J. Embedded Syst.*, vol. 6, nos. 2–3, pp. 176–187, 2014.
[29] W. Wang, "Schedulability analysis and symbolic verification method for heterogeneous multicore real-time systems," *Int. J. Performability Eng.*, vol. 13, no. 6, p. 785, 2017.
[30] T. Amnell, E. Fersman, L. Mokrushin, P. Pettersson, and W. Yi, "TIMES: A tool for schedulability analysis and code generation of real-time systems," in *Proc. Int. Conf. Formal Modeling Anal. Timed Syst.* Cham, Switzerland: Springer, 2003, pp. 60–72.
[31] O. Oğuz, J. F. Broenink, and A. Mader, "Schedulability analysis of timed CSP models using the pat model checker," *Communicating Process Archit.*, vol. 2012, pp. 65–88, Aug. 2012.
[32] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu, "Symbolic model checking without BDDs," in *Proc. Int. Conf. Tools Algorithms Construct. Anal. Syst.* Cham, Switzerland: Springer, 1999, pp. 193–207.
[33] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking*. Cambridge, MA, USA: MIT Press, 1999.
[34] E. M. Clarke, W. Klieber, M. Nováček, and P. Zuliani, "Model checking and the state explosion problem," in *Proc. LASER Summer School Softw. Eng.* Cham, Switzerland: Springer, 2011, pp. 1–30.
[35] M. Davis and H. Putnam, "A computing procedure for quantification theory," *J. ACM*, vol. 7, no. 3, pp. 201–215, Jul. 1960.
[36] H. Hong, L. Khan, A. Gbadebo, Z. Shaohua, and W. Yong, "A complex task scheduling scheme for big data platforms based on Boolean satisfiability problem," in *Proc. IEEE Int. Conf. Inf. Reuse Integr. (IRI)*, Jul. 2018, pp. 170–177.
[37] M. Zhang and Y. Ying, "Towards SMT-based LTL model checking of clock constraint specification language for real-time and embedded systems," *ACM SIGPLAN Notices*, vol. 52, no. 5, pp. 61–70, Sep. 2017.
[38] M. Zhang, F. Song, F. Mallet, and C. Xiaohong, "SMT-based bounded schedulability analysis of the clock constraint specification language," in *Proc. FASE-Fundam. Approaches Softw. Eng.*, 2019, pp. 61–78.
[39] A. de Matos Pedro, D. Pereira, L. M. Pinho, and J. S. Pinto, "SMT-based schedulability analysis using RMTL-$\int$," *ACM SIGBED Rev.*, vol. 14, no. 3, pp. 40–42, Nov. 2017.
[40] I. Ripoll and R. Ballester-Ripoll, "Period selection for minimal hyper-period in periodic task systems," *IEEE Trans. Comput.*, vol. 62, no. 9, pp. 1813–1822, Sep. 2013.
[41] J. M. Lopez, M. Garcia, J. L. Diaz, and D. F. Garcia, "Worst-case utilization bound for EDF scheduling on real-time multiprocessor systems," in *Proc. 12th Euromicro Conf. Real-Time Syst. Euromicro RTS*, 2000, p. 25.
[42] S. Baruah and N. Fisher, "The partitioned multiprocessor scheduling of sporadic task systems," in *Proc. 26th IEEE Int. Real-Time Syst. Symp. (RTSS)*, Miami, FL, USA, Dec. 2005, pp. 321–329.
[43] T. P. Baker, "Comparison of empirical success rates of global vs. partitioned fixed-priority and EDF scheduling for hard real time," Dept. Comput. Sci., Florida State Univ., Tallahassee, FL, USA, Tech. Rep. TR-050601, 2005.
[44] A. Bastoni, B. B. Brandenburg, and J. H. Anderson, "An empirical comparison of global, partitioned, and clustered multiprocessor EDF schedulers," in *Proc. 31st IEEE Real-Time Syst. Symp.*, Nov. 2010, pp. 14–24.
[45] R. I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," *ACM Comput. Surv.*, vol. 43, no. 4, p. 35, 2011.
[46] E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson, "An application of bin-packing to multiprocessor scheduling," *SIAM J. Comput.*, vol. 7, no. 1, pp. 1–17, Feb. 1978.
[47] P. Shaw, "A constraint for bin packing," in *Proc. Int. Conf. Princ. Pract. Constraint Program.* Cham, Switzerland: Springer, 2004, pp. 648–662.
[48] P. Schaus, "Solving balancing and bin-packing problems with constraint programming," Ph.D. dissertation, Université catholique de Louvain, Ottignies-Louvain-la-Neuve, Belgium, 2009.
[49] T. Soh, K. Inoue, N. Tamura, M. Banbara, and H. Nabeshima, "A SAT-based method for solving the two-dimensional strip packing problem," *Fundamenta Informaticae*, vol. 102, nos. 3–4, pp. 467–487, 2010.
[50] J.-C. Régin and M. Rezgui, "Discussion about constraint programming bin packing models.," *AI Data Center Manage. Cloud Comput.*, vol. 11, p. 8, Jan. 2011.

[51] M. Mistry, A. C. D'Iddio, M. Huth, and R. Misener, "Satisfiability modulo theories for process systems engineering," *Comput. Chem. Eng.*, vol. 113, pp. 98–114, May 2018.
[52] L. De Moura and N. Bjørner, "Z3: An efficient SMT solver," in *Proc. Int. Conf. Tools Algorithms Construct. Anal. Syst.*, Berlin, Germany: Springer, 2008, pp. 337–340.

HYUK LEE received the B.S. degree from the University of Technology Sydney, Sydney, Australia, in 2006, and the M.S. and Ph.D. degrees from Korea University, Seoul, South Korea, in 2009 and 2019, respectively. He is currently a Research Professor with the Graduate School of Information Security, Korea University, Seoul. His current research interests include formal methods, constraint problem solving, and secure software engineering.

JIN-YOUNG CHOI (Member, IEEE) received the B.S. degree from Seoul National University, Seoul, South Korea, in 1982, the M.S. degree from Drexel University, Philadelphia, PA, USA, in 1986, and the Ph.D. degree from the University of Pennsylvania, Philadelphia, in 1993. He is currently a Professor with the Graduate School of Information Security, Korea University, Seoul. His current research interests include real-time computing, formal methods, programming languages, process algebras, security, and secure software engineering.