Feasibility of Fork-Join Real-Time Task Graph Models: Hardness and Algorithms

JINGHAO SUN, NAN GUAN, YANG WANG, and QINGXU DENG,
Northeastern University, China
PENG ZENG, Chinese Academy of Sciences, China
WANG YI, Uppsala University, Sweden

In the formal analysis of real-time systems, modeling of branching codes and modeling of intratask parallelism structures are two of the most important research topics. These two real-time properties are combined, resulting in the fork-join real-time task (FJRT) model, which extends the digraph-based task model with forking and joining semantics. We prove that the EDF schedulability problem on a preemptive uniprocessor for the FJRT model is coNP-hard in the strong sense, even if the utilization of the task system is bounded by a constant strictly less than 1. Then, we show that the problem becomes tractable with some slight structural restrictions on parallel sections, for which we propose an exact schedulability test with pseudo-polynomial time complexity. Our results thus establish a borderline between the tractable and intractable FJRT models.

Categories and Subject Descriptors: C.2.2 [**Computer-Communication Networks**]: Network Protocols

General Terms: Design, Algorithms, Performance

Additional Key Words and Phrases: Fork-join, digraph-based task, EDF-schedulability, tractability

ACM Reference Format:
Jinghao Sun, Nan Guan, Yang Wang, Qingxu Deng, Peng Zeng, and Wang Yi. 2016. Feasibility of fork-join real-time task graph models: Hardness and algorithms. ACM Trans. Embed. Comput. Syst. 15, 1, Article 14 (February 2016), 28 pages.
DOI: http://dx.doi.org/10.1145/2809780

1. INTRODUCTION

A number of modeling formalisms have been developed to model and analyze real-time systems. Liu and Layland [1973] modeled a real-time system as a finite collection of independent recurrent tasks, each of which periodically generates an infinite sequence of jobs. This seminal work developed a polynomial-time feasibility test on uniprocessors. This model was then generalized in Ka and Mok [1983] to count sporadic tasks for which relative deadlines are different from release periods. The multi-frame (MF) model [Mok and Chen 1997] and generalized multi-frame (GMF) model [Baruah et al. 1999] further generalize periodic and sporadic tasks. MF and GMF models define several different types of jobs that may be generated by the same task and are used for modeling

This work is supported by the Natural Science Foundation of China, under grant no. 61300194, grant no. 61300022, and grant no. 61472072, the National Basic Pre-research Program of China 2014CB360509, and the Opening Project of Key Laboratory of Networked Control Systems of Chinese Academy of Sciences.
Authors' addresses: J. Sun, N. Guan (corresponding author), Y. Wang, and Q. Deng, School of Information Science and Engineering, Northeastern University, China; emails: jhsun@mail.dlut.edu.cn, guannan@ise.neu.edu.cn, 1510406@stu.neu.edu.cn, Dengqx@mail.neu.edu.cn; P. Zeng, Laboratory of Networked Control Systems, Shenyang Institute of Automation, Chinese Academy of Sciences, China; email: zp@sia.cn; W. Yi, Department of Information Technology, Uppsala University, Sweden; email: yi@it.uu.se.

DOI: http://dx.doi.org/10.1145/2809780

recurrent processes composed of straight-line code. Recent developments focus on nonlinear structures of the programming code. Specifically, modeling of the branch (conditional) property and modeling of the intratask parallelism (Fork-join) structure have become two of the most interesting research topics in the real-time community.

Conditional Real-Time Task Models. Several graph-based task models were proposed to formulate the conditional code and offer a high degree of expressiveness while maintaining pseudo-polynomial complexity. In the recurring real-time task (RRT) [Baruah 2003] and noncyclic RRT [Baruah 2010a] models, each task is characterized by a directed acyclic graph (DAG). Recurrent behavior is modeled by adding back edges from sink vertices to the unique source vertex. These models cannot express local looping or task mode switches. The digraph real-time task (DRT) [Stigge et al. 2011] models each task by an arbitrary digraph, for which the feasibility test is still tractable. The DRT model is considered the most general tractable conditional task model currently known, as it generalizes earlier tractable conditional task models.

Fork-Join Task Models. The trend of deploying real-time applications on parallel processing platforms poses new challenges to the real-time scheduling theory, which traditionally focuses on sequential tasks. There has been some recent work on the real-time scheduling of parallel tasks. In synchronous models, each task consists of a sequence of segments with synchronization points at the end of each segment. In addition, each segment of a task contains threads of execution of equal length. Several theoretical results have been provided for scheduling the synchronous parallel tasks on multiprocessors [Lakshmanan et al. 2010; Saifullah et al. 2011; Ferry et al. 2013; Chwa et al. 2013; Axer et al. 2013].

In this article, we aim to incorporate the conditional real-time model with the fork-join task model and investigate a more general task model, namely, the fork-join real-time task (FJRT) model, which allows parallel job compositions (fork-join structure) within digraph-based tasks. At the time of writing this article, several graph-based parallel task models have already been studied. Recently, Ejsing-Duun et al. [2013] extended the DRT model to a parallel version, called the concurrent real-time task (CRT) model. They defined the CRT model inductively and restricted it to models with well-formed structures (the formal definition is introduced in Section 2.2). For the noncyclic and well-formed CRT model, Ejsing-Duun et al. [2013] noted that the uniprocessor feasibility can be determined in pseudo-polynomial time. Stigge et al. [2013] introduced a more flexibly defined model than the CRT model, which is also an extension of the DRT model combined with fork-join structures and is called the fork-join real-time task (FJRT) model. Stigge et al. [2013] considered whether there exists an exact EDF schedulability test for the FJRT model, defined on uniprocessor systems with a pseudo-polynomial bound of its runtime, and left it as an open problem:

CONJECTURE 1.1 [STIGGE ET AL. 2013]. *For an FJRT task system $\mathcal{T}$ with a utilization not exceeding a constant c ($c < 1$), feasibility can be decided in pseudo-polynomial time*.

The open problem listed in this conjecture will be closed in this article. We first investigate the expressiveness of the FJRT model and claim that there exist some FJRT models that can produce job release scenarios that cannot be produced by any CRT model. We then separately study the FJRT model by considering whether it can be expressed by a CRT model. For the general FJRT model, which exceeds the CRT model's expressiveness, we prove the feasibility problem to be strongly coNP-hard even if the utilization is bounded by a constant strictly less than 1. We also consider the FJRT model with some slight structural restrictions on parallel sections, which can be viewed as the equivalence of the general CRT model that does not need to be well formed. For this restricted FJRT model, we propose a pseudo-polynomial algorithm for testing the feasibility. This article thus extends the theoretical results for the

well-formed CRT models in Ejsing-Duun et al. [2013] to those that are not well formed, and establishes a borderline between the intractable and tractable FJRT models.

1.1. Related Work

There are two main branches in the schedulability research: uniprocessor systems and multiprocessor systems. Liu and Layland [1973] investigated the periodic task model for the feasibility analysis of uniprocessor real-time systems. They formulated a necessary and sufficient criterion for uniprocessor feasibility and a utilization bound that required knowledge of execution times and unique periods of each task. As previously mentioned, this work was later extended to the sporadic task model [Ka and Mok 1983; Baruah et al. 1990]. For its analysis, Baruah et al. [1990] introduced a concept that was later called the demand bound function (DBF), and derived a closed-form expression to determine DBF(t) for a given time interval length t. Moreover, Baruah et al. [1990] also proposed a pseudo-polynomial bound T of time-interval lengths by using the concept of the utilization bound. The feasibility problem then can be solved by iteratively calculating DBF(t) for $t < T$. The concept of the demand bound function and the utilization was adapted to the GMF model [Baruah et al. 1999] and the algorithm for DBF computing is not as simple as it is for the sporadic task model. Baruah et al. [1999] presented a polynomial-time dynamic programming algorithm to compute DBF by using a skillful data structure that was called the "demand pair" in later work [Stigge et al. 2011]. This demand-pair structure was further applicable to the DAG- and DRT-based models [Baruah 2010b; Stigge et al. 2011]; the complexity of the demand computation algorithm increases from linear time to pseudo-polynomial. Ejsing-Duun et al. [2013] extended the DRT model to the CRT model and argued that the pseudo-polynomial feasibility test for the DRT model can also be applied to the well-formed CRT model, in which each job is strictly restricted to being dependent on at most one other job in the system. For the parallel tasks without this restriction, this article presents a pseudo-polynomial feasibility test.

Recently, a few analytic methods have been proposed for schedulability of parallel tasks on multiprocessor platforms. These methods broadly fall into two categories: direct and indirect. Indirect analysis methods [Lakshmanan et al. 2010; Saifullah et al. 2011; Nelissen et al. 2012; Ferry et al. 2013] share a principle of task decomposition. Each single parallel task is transformed into multiple independent sequential subtasks such that each individual subtask is assigned its own intermediate deadline. Schedulability analysis is then performed over intermediate deadlines after task decomposition at the expense of potentially incurring some nontrivial decomposition overheads. On the other hand, direct analysis methods [Kato and Ishikawa 2009; Baruah et al. 2012; Saifullah et al. 2012] performed analysis without task decomposition. Kato and Ishikawa [2009] considered only a certain type of intratask thread-level parallelism (i.e., gang scheduling), which allowed a fixed degree of parallelism: all threads run or none does. Baruah et al. [2012] and Saifullah et al. [2012] considered a more general DAG-based parallel task model.

Because our focus is strictly on uniprocessor scheduling, we will not consider multiprocessors for the remainder of this article. It should also be noted that, although the work of this article is conducted with a uniprocessor setting, the investigated feasibility analysis represents the fundamental problem of calculating the workload of the FJRT model. This is also necessary for any more complex problem settings, for example, in the scheduling of FJRT tasks on multiprocessor platforms.

1.2. Structure

This article is organized as follows. Section 2 presents the FJRT model and investigates its expressiveness. Section 3 presents a reduction from the 3SAT problem to prove that

the feasibility problem for the FJRT model is coNP-hard in the strong sense even if the utilization is bounded by a constant strictly less than 1. Section 4 focuses on a tractable FJRT model and analyzes its feasibility. Section 5 presents a case study of an Intelligent Traffic Control System (ITCS) to illustrate the usage of the FJRT model. Sections 6 and 7 offer discussions and conclusions, respectively.

2. TASK MODEL AND RELATIVE NOTATIONS

The task model is defined as first proposed in Stigge et al. [2013]. An FJRT task system $\mathscr{T} = \{\tau_1, \ldots, \tau_N\}$ consists of N independent tasks. A task τ is represented by a directed hypergraph $D = (V_\tau, A_\tau)$ with vertex and arc labels. A vertex $v_i \in V_\tau$ represents the type of jobs that τ can release and is labeled with the ordered pair $[e(v_i), d(v_i)]$, denoting the worst-case execution time $e(v_i)$ and the relative deadline $d(v_i)$. The hyperarcs of D represent the order in which jobs generated by τ are released. A hyperarc (U, V) is a sequence arc, with U and V being singleton sets of vertices, a fork arc with U being a singleton set, or a join arc with V being a singleton set. In all cases, the arcs are labeled with a nonnegative parameter $p(U, V)$ denoting the minimum job interrelease separation time. Note that this contains the DRT model as a special case if all hyperarcs are sequence arcs.

As an extension of the DRT model, the FJRT task system releases independent jobs, allowing concepts such as the utilization $U(\mathscr{T})$ and the demand bound function to be defined just as before (in, e.g., Stigge et al. [2011]), which are formally given next.

Definition 2.1 (*Demand Bound Function*). For an FJRT task τ and an interval length t, $\mathrm{DBF}_\tau(t)$ denotes the maximum cumulative execution requirement of jobs of τ with both the release time and deadline within an interval of length t over all job sequences generated by τ. Furthermore, for a task set $\mathscr{T}$,

$$\mathrm{DBF}(t) := \sum_{\tau \in \mathscr{T}} \mathrm{DBF}_\tau(t).$$

Definition 2.2 (*Utilization*). For an FJRT task τ and a task set $\mathscr{T}$, we define their utilizations:

$$U(\tau) := \lim_{t \to \infty} \frac{\mathrm{DBF}_\tau(t)}{t}; \qquad U(\mathscr{T}) := \sum_{\tau \in \mathscr{T}} U(\tau).$$

Semantics: A task executes by following a path through the hypergraph, triggering releases of associated jobs each time a vertex is visited. Whenever a fork arc $(\{u\}, \{v_1, \ldots, v_m\})$ is taken, m independent paths starting from v_1 to v_m, respectively, will be followed in parallel until joined by a corresponding join arc. In order for a join arc $(\{u_1, \ldots, u_n\}, \{v\})$ to be taken, all jobs associated with vertices $u_1, \ldots, u_n$ must have been released and sufficient time must have passed to satisfy the join arc label. Forking can be nested, that is, the m paths can lead to further fork arcs before being joined. Additionally, joining can also be nested, that is, the m parallel paths can be partially joined before they are totally joined together.

2.1. Structural Restrictions on the FJRT Model

In this article, we assume that the meaningful FJRT model has to satisfy the structural restriction that each fork vertex (say v_i) must match with a corresponding join vertex (say, v_j) such that once v_i has been visited, it cannot be revisited unless all paths that have been forked from v_i are eventually joined together by the matching join vertex v_j. Otherwise, the FJRT task system might be unfeasible because there always exists

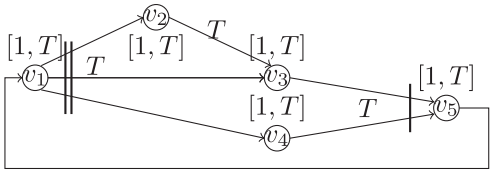


Fig. 1. A fork arc cannot be joined by a matching join.

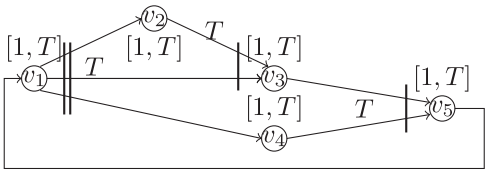


Fig. 2. The modified FJRT from Example 2.3.

an unbounded workload increase caused by an unmatched fork. To illustrate this, we construct a nonrestricted models, as follows.

Example 2.3. Figure 1 shows an FJRT task τ defined on the hypergraph $D = (V, A)$. There are five vertices in V; the ordered pair associated with each vertex is $[1, T]$. The arc set A consists of a fork arc, $(v_1, \{v_2, v_3, v_4\})$, a join arc, $(\{v_3, v_4\}, v_5)$, and two sequence arcs, (v_2, v_3) and (v_5, v_1). The corresponding period of each (hyper) arc is T.

As shown in Figure 1, three parallel paths can be forked from v_1: two visiting v_3 and the other visiting v_4. These parallel paths must be eventually joined by v_5 because v_5 is the only join vertex in D. It is easy to show that the two paths visiting v_3 might not be joined simultaneously. In other words, we can choose one of the paths visiting v_3, together with the path visiting v_4, to be joined by v_5, and leave the other path waiting at v_3. Without loss of generality, we assume that the path chosen to be joined by v_5 is (v_1, v_3) and the path that waits to release jobs is (v_1, v_2, v_3). Accordingly, the two paths forked from v_1, (v_1, v_3) and (v_1, v_4), are joined into v_5 and further re-visit the vertex v_1. By repeating these steps, we can travel along the loop path $(v_1, \{v_3||v_4\}, v_5, v_1)$ periodically, where $\{v_3||v_4\}$ represents the parallel composition in the loop. Note that one more path (v_1, v_2, v_3) is generated each time the loop path is traversed. If all the new generated paths (v_1, v_2, v_3) do not release any jobs and continue to wait at v_3 until the loop path is traversed T times, the number of jobs waiting to be released at v_3 will exceed $T + 1$. We will release these $T + 1$ jobs simultaneously. Because these jobs have a common deadline T, it is easy to show that there must exist at least one job that will miss the deadline. Thus, the task τ is unfeasible.

Example 2.3 would be restricted if the arcs entering into v_3 were join arcs, as shown in Figure 2. It is easy to see that each path forked from one vertex (e.g., v_1 in Figure 2) will not revisit v_1 unless it is joined together with all other paths that are forked from the same vertex. From this, we conclude that there is a sufficient condition for checking the unfeasible task system, provided next.

PROPOSITION 2.4. *The task is unfeasible if a fork vertex defined on the corresponding hypergraph can be visited before all paths forked from that vertex are eventually joined together.*

In the rest of this article, we consider only meaningful task systems, in which the counterfactual condition in Proposition 2.4 is excluded.

2.2. Expressiveness of the FJRT Model

Ejsing-Duun et al. [2013] recently proposed the CRT model, which can be viewed as the most related work to this article. In this section, we consider the CRT model to compare its expressiveness to that of our FJRT model and assert that any CRT task can be converted into an FJRT task. However, there exist some FJRT models that cannot be expressed by the CRT models. To show this, we first introduce the formal definition of the CRT model, then transform each atomic grammar rule of the CRT

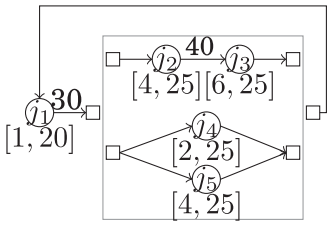


Fig. 3. An example CRT task τ.

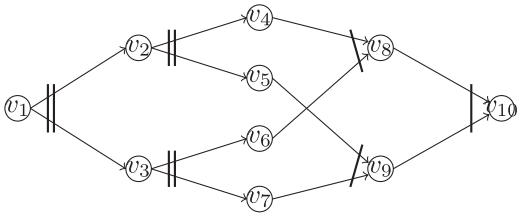


Fig. 4. An example FJRT task that cannot be expressed by a CRT model.

model into a regular substructure of the hypergraph, which can subsequently be used to construct an equivalent FJRT model for the original CRT model. Finally, we present the counterexample of the parallel task that can be expressed by the FJRT model but cannot be expressed by the CRT model.

Definition 2.5 (*Concurrent Real-Time Task System [Ejsing-Duun et al. 2013]*). A Concurrent Real-Time Task System (CRT) $\mathscr{S}$ is a tuple $\mathscr{S} = (\tau, J, e, d)$, where

—J is a finite set of jobs,
—e: $J \to \mathbb{N}$ is a mapping from jobs to worst-case execution times,
—d: $J \to \mathbb{N}$ is a mapping from jobs to relative deadlines, and
—τ is a configuration defined by the following grammar.
  —Configuration: $\tau ::= T \mid T \| \tau$
  —Task: $T ::= j \mid T_1 \langle x \rangle T_2 \mid S_1 \| S_2 \mid T_1 + T_2 \mid T^{\omega}$
  —Subtask: $S ::= j \mid S_1 \langle x \rangle S_2 \mid S_1 \| S_2 \mid S_1 + S_2$

Here, $j \in J$ is a job and $x \in \mathbb{N}$, $T_1 \langle x \rangle T_2$ is the sequential composition of tasks T_1 and T_2, with interrelease time x, $T_1 + T_2$ is the choice between tasks T_1 and T_2, $S_1 \| S_2$ is the parallel composition of subtasks S_1 and S_2, and T^{ω} is one or more iterations of T.

In contrast to the definition of our FJRT model, the CRT model is inductively defined by some regular grammar. The grammar of the configuration defines the task system, which can be viewed as a composition of arbitrarily many independent tasks running in parallel (intertask parallelism). Moreover, a nondeterministic choice between different execution paths or subtasks running in parallel can be expressed by the grammar of tasks and subtasks. Note that the subtasks defined in the CRT are restricted to not include any cycles. This article considers only acyclic parallel subtasks in the FJRT model.

Example 2.6. Figure 3 represents the CRT task $\tau = (j_1 \langle 30 \rangle ((j_2 \langle 40 \rangle j_3) \| (j_4 + j_5)))^{\omega}$, where $e(j_1) = 1$, $d(j_1) = 20$; $e(j_2) = 4$, $d(j_2) = 25$; $e(j_3) = 6$, $d(j_3) = 25$; $e(j_4) = 2$, $d(j_4) = 25$; and $e(j_5) = 4$, $d(j_5) = 25$. As shown in Figure 3, the vertices in the graph represent jobs, each labeled with an execution time and a deadline. The arcs in the graph represent dependencies between jobs and are labeled with their minimum interrelease time.

The combination of Theorem 2.7 and Example 2.8 proves that our digraph-based FJRT model is more expressive than the CRT models.

THEOREM 2.7. *The FJRT model is at least as expressive as the CRT model.*

PROOF. We prove by induction that each grammar rule listed in Definition 2.5 can be fully expressed by a substructure of the hypergraph. We do this by first introducing the prefix and suffix operations for tasks, as follows.

—For each job $j \in J$, we let prefix(j) = suffix(j) = j.
—For each $T := p_1\langle x\rangle T'$, where p_1 is the first job released in T, we let prefix(T) = p_1; similarly, for each $T := T'\langle x\rangle p_m$, suffix($T$) = p_m, where p_m is the last job of T.
—For two arbitrary tasks T_1 and T_2, to define the prefix and suffix of $T_1 + T_2$, we first define two auxiliary virtual jobs v and u with an execution time of 0. The original task $T_1 + T_2$ is then equivalent to $v\langle 0\rangle(T_1 + T_2)\langle 0\rangle u$. We let prefix($T_1 + T_2$) = v and suffix($T_1 + T_2$) = u.
—For two arbitrary subtasks S_1 and S_2, we focus on the prefix and suffix of $S_1 \| S_2$. To this end, we also define two auxiliary virtual jobs v and u with an execution time of 0. The original subtask $S_1 \| S_2$ is equivalent to $v\langle 0\rangle(S_1 \| S_2)\langle 0\rangle u$. We let prefix($S_1 \| S_2$) = v and suffix($S_1 \| S_2$) = u.
—prefix(T^ω) = prefix(T), and suffix(T^ω) = suffix(T).

We now propose an approach for expressing each atomic grammar rule by the substructure of the hypergraph.

—For each $j \in J$, we define a corresponding vertex v_j;
—For two arbitrary T_1 and T_2, let the subgraphs associated with T_1 and T_2 be $D(T_1)$ and $D(T_2)$, respectively.
  —To construct the subgraph of $T_1\langle x\rangle T_2$, we combine $D(T_1)$ and $D(T_2)$ by connecting suffix(T_1) to prefix(T_2). The additional arc from suffix(T_1) to prefix(T_2) is labeled with an interrelease time x.
  —The subgraph of $T_1 + T_2$ can be constructed by respectively adding arcs from prefix($T_1 + T_2$) to prefix(T_1) and prefix(T_2) and arcs from suffix(T_1) and suffix(T_2) to suffix($T_1 + T_2$). The additional arcs are labeled with interrelease times of 0.
  —Similarly, the subgraph of $S_1 \| S_2$ can be obtained by adding the fork arc (prefix($S_1 \| S_2$),{ prefix(S_1), prefix(S_2)}) and the join arc ({ suffix(S_1), suffix(S_2)}, suffix($S_1 \| S_2$)). The additional hyperarcs are labeled with interrelease times of 0.
—We can construct the subgraph of T^ωw by adding the arc from suffix(T^ω) to prefix(T^ω).

In summary, each atomic grammar rule can be fully expressed by a subgraph. Thereby, all the configurations induced by the grammar can be expressed by a hypergraph. That is, the task system modeled as a CRT can also be modeled as an FJRT, which completes the proof. □

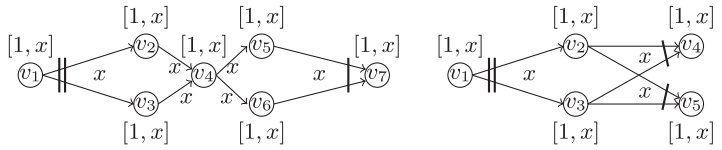
We also note that there exist some FJRT models that cannot be expressed by CRTs. This can be demonstrated through the following example, shown as a graph in Figure 4.

Example 2.8. The hypergraph $D = (V, A)$ in Figure 4 consists of ten vertices, three fork arcs, and three join arcs, defined as follows.

—Set of vertices: $V = \{v_1, \ldots, v_{10}\}$;
—Fork arcs: $(v_1, \{v_2, v_3\}), (v_2, \{v_4, v_5\}), (v_3, \{v_6, v_7\})$;
—Join arcs: $(\{v_4, v_6\}, v_8), (\{v_5, v_7\}, v_9), (\{v_8, v_9\}, v_{10})$.

The parallel task defined in this example cannot be modeled as a CRT because some parallel paths forked from different vertices need to be joined into one vertex. For example, the paths (v_2, v_4) and (v_3, v_6) forked, respectively, from vertices v_2 and v_3 should be joined into the vertex v_8.

We also give some tractable FJRT models. Note that only some well-formed CRTs, in which any job in J occurs in the configuration at most once, have been shown to be tractable in Ejsing-Duun et al. [2013]. In this article, we focus on the parallel tasks that do not need to be well formed. Two examples of these nontrivial parallel tasks are shown here.



(a) FJRT task τ_1. (b) FJRT task τ_2

Fig. 5. Two example FJRT tasks that can be modeled by the CRTs without the well-formed structure.

Example 2.9. The hypergraph associated with the task τ_1 is given in Figure 5(a), and the hyperarcs are defined here.

—Fork arc: $(v_1, \{v_2, v_3\})$;
—Sequential arcs: $(v_2, v_4), (v_3, v_4), (v_4, v_5), (v_4, v_6)$;
—Join arc: $(\{v_5, v_6\}, v_7)$.

Here, all the hyperarcs are labeled with interrelease times of x, and each vertex v_j corresponding to the job j is labeled with $[1, x](j = 1, \ldots, 7)$. This FJRT task can be expressed by the CRT configuration $\tau_1 = 1\langle x\rangle((2\langle x\rangle 4\langle x\rangle(5 + 6))\|(3\langle x\rangle 4\langle x\rangle(5 + 6)))\langle x\rangle 7$.

Example 2.10. The FJRT task τ_2 can be expressed by the hyper-graph in Figure 5(b), which consists of five vertices, one fork arc, $(v_1, \{v_2, v_3\})$, and two join arcs, $(\{v_2, v_3\}, v_4)$ and $(\{v_2, v_3\}, v_5)$. Here, all the hyper-arcs are labeled with inter-release times of x, and all the vertices are labeled with $[1, x]$. The corresponding CRT model for τ_2 is $1\langle x\rangle(((2\|3)\langle x\rangle 4) + ((2\|3)\langle x\rangle 5))$.

It is easy to show that both examples can be expressed by CRTs but neither is well formed. In Section 4, we show that the feasibility problem for the CRT that is not well formed can be solved in pseudo-polynomial time.

3. HARDNESS OF ANALYSIS OF THE FJRT MODEL

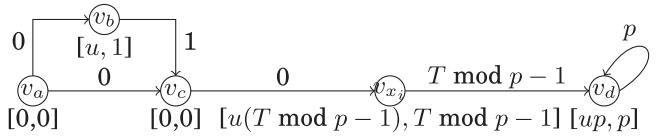
To show the strong coNP-hardness of the feasibility problem for the FJRT model, we provide a reduction from the 3SAT problem that is known to be NP-Complete in the strong sense [Cook 1971]. We first introduce the 3SAT problem.

Let $X = \{x_1, \ldots, x_m\}$ be a finite set of variables taking values in the set {true, false}. Let C be a collection of disjunction clauses on the variables of X such that $\forall C_j \in C$, $C_j = \vee_{k=1}^{3} c_{jk}$, where each c_{jk} is a distinct literal, that is, either a variable or the negation of a variable. Without loss of generality, we assume that no clause contains both a variable and its negation and $|C| = n$. Let E be a conjunction of all clauses in C (i.e., $E = C_1 \wedge C_2 \wedge \cdots \wedge C_n$). The problem of deciding whether there exists at least one assignment of values to each variable in X such that E is satisfiable is called the 3SAT problem.

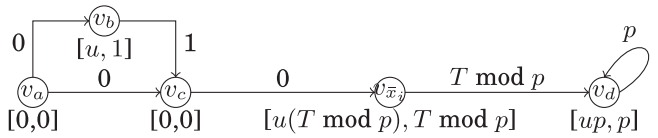
The intuitive idea of our construction is as follows. Given an instance E of 3SAT, we construct a task set $\mathscr{T}$ with the following properties:

(1) A witness of the falsity of the condition $\mathrm{DBF}(t) \leq t$ for an unfeasible task $\mathscr{T}$ will give a satisfying assignment for E, and vice versa.
(2) The number of vertices in the task of $\mathscr{T}$ and all involved values need to be polynomially bounded in the size of E.
(3) Given a constant $c < 1$, we must be able to construct $\mathscr{T}$ such that $U(\mathscr{T}) \leq c$.

The second requirement is necessary to establish the coNP-hardness in the strong sense. The last requirement is also necessary because we usually restrict ourselves to a class of task sets with a utilization bounded by a constant $c < 1$. We want to show that, for any choice of c, the problem stays strongly coNP-hard.



(a) The subgraph g_{x_i} corresponds to x_i.



(b) The subgraph $g_{\bar{x}_i}$ corresponds to $\bar{x}_i$.

Fig. 6. Subgraphs are constructed to cooperate with the variable x_i and its negation.

A more formal construction is introduced as follows. We first define c as a constant strictly less than 1 and let p be a constant integer greater than 1. We now define a fixed time interval length T such that $T \bmod p \neq 0$. Furthermore, we also define a parameter u as follows.

$$u = \frac{c' + \frac{\epsilon}{T}}{n}. \tag{1}$$

Here, n is the number of conjunctions of clauses in C, and

$$c' + \frac{\epsilon}{T} \leq c \tag{2}$$

$$\epsilon = \min\left\{\frac{T}{nT-1}c', T(c-c'), \frac{T}{T+1}(1-c')\right\}. \tag{3}$$

For each variable $x_i \in X$ (or its corresponding negation $\bar{x}_i$), we construct a subgraph g_{x_i} (or $g_{\bar{x}_i}$), as shown in Figure 6. There is a starting vertex v_a in g_{x_i} (or $g_{\bar{x}_i}$). Starting from v_a, we traverse v_b and enter into v_c, corresponding to an assignment of true to x_i (or $\bar{x}_i$), or directly enter into v_c without passing through the vertex v_b, corresponding to an assignment of false to x_i (or $\bar{x}_i$). Moreover, the parameters p_{x_i} and $p_{\bar{x}_i}$ used in graphs g_{x_i} and $g_{\bar{x}_i}$ are defined as follows. Note that p_{x_i} is a nonnegative integer because $T \bmod p \neq 0$.

—$p_{x_i} = T \bmod p - 1$;
—$p_{\bar{x}_i} = T \bmod p$.

As shown in Figure 6(a), in the graph g_{x_i}, the vertex v_b is labeled with an ordered pair $[u, 1]$, and the associated arc from v_b to v_c is labeled with a period of 1. The vertex v_{x_i} is labeled with $[up_{x_i}, p_{x_i}]$, and the vertex v_d is labeled with $[up, p]$. Moreover, the arc from v_{x_i} to v_d is associated with a period p_{x_i}, and the self-loop arc of v_d is labeled with a period p. Without special definition, the remaining vertices and arcs in g_{x_i} are all labeled with an ordered pair $[0, 0]$ and a period of 0, respectively. Additionally, the subgraph $g_{\bar{x}_i}$ corresponding to $\bar{x}_i$ has the same topology as g_{x_i}, as shown in Figure 6(b). The only difference for $g_{\bar{x}_i}$ is that the vertex $v_{\bar{x}_i}$ is labeled with $[up_{\bar{x}_i}, p_{\bar{x}_i}]$ and the arc $(v_{\bar{x}_i}, v_d)$ is labeled with $p_{\bar{x}_i}$.

Now, consider a clause $c_{j1} \vee c_{j2} \vee c_{j3}$; we construct three subgraphs $g_{c_{jk}}$, $1 \leq k \leq 3$, one corresponding to each of the three literals in the clause. Denoting the maximum demand requirement of the job sequence generated from the subgraph $g_{c_{jk}}$ for a time interval T as $\mathrm{DBF}_{c_{jk}}(T)$, the relationship between the literal c_{jk} and its corresponding subgraph $g_{c_{jk}}$ is given in the following lemma.

LEMMA 3.1. *For a given time interval length T, $DBF_{c_{jk}}(T) = uT$ if $c_{jk} = true$. Otherwise, $DBF_{c_{jk}}(T) \leq uT - u$.*

PROOF. For each literal c_{jk} ($k = 1, 2, 3$), we will separately consider two possibilities: that c_{jk} corresponds to either a variable or the negation of a variable.

(1) If c_{jk} is a variable, without loss of generality, we assume that $c_{jk} = x_i$. Then, the corresponding subgraph $g_{c_{jk}}$ is the same as g_{x_i}, as shown in Figure 6(a). Additionally, we consider that x_i = true, which makes c_{jk} true. In this case, the corresponding path in $g_{c_{jk}}$ is $\pi = (v_a, v_b, v_c, v_{x_i}, v_d, v_d, \ldots)$. Accordingly, for a given T, the maximum demand of π can be calculated as:

$$\begin{aligned}\mathrm{DBF}_{c_{jk}}(T) &= u + u(T \bmod p - 1) + up\left\lfloor \frac{T - (T \bmod p - 1) - 1}{p} \right\rfloor \\ &= u(T \bmod p) + up\left\lfloor \frac{T - T \bmod p}{p} \right\rfloor \\ &= u(T \bmod p) + uT - u(T \bmod p) = uT.\end{aligned}$$

In contrast, when x_i = false, c_{jk} = false. Then, the corresponding path is $\pi = (v_a, v_c, v_{x_i}, v_d, v_d, \ldots)$, and the demand of π can be bounded by:

$$\begin{aligned}\mathrm{DBF}_{c_{jk}}(T) &= u(T \bmod p - 1) + up\left\lfloor \frac{T - (T \bmod p - 1)}{p} \right\rfloor \\ &= u(T \bmod p - 1) + up\left\lfloor \frac{T - T \bmod p + 1}{p} \right\rfloor \\ &= u(T \bmod p - 1) + up\left(\frac{T - T \bmod p}{p}\right) + up\left\lfloor \frac{1}{p} \right\rfloor \\ &= u(T \bmod p) - u + uT - u(T \bmod p) \quad (\because p \geq 2) \\ &= uT - u.\end{aligned}$$

(2) Suppose that c_{jk} is the negation of the variable x_i, that is, $c_{jk} = \bar{x}_i$; then, the corresponding subgraph $g_{c_{jk}}$ is equal to $g_{\bar{x}_i}$, as shown in Figure 6(b). If x_i = true, then c_{jk} = false. In this case, the corresponding path in $g_{c_{jk}}$ is $\pi = (v_a, v_b, v_c, v_{\bar{x}_i}, v_d, v_d, \ldots)$, and the associated demand can be bounded by:

$$\begin{aligned}\mathrm{DBF}_{c_{jk}}(T) &= u + u(T \bmod p) + up\left\lfloor \frac{T - (T \bmod p) - 1}{p} \right\rfloor \\ &= u + u(T \bmod p) + up\left(\frac{T - T \bmod p}{p} - 1\right) \\ &= u + u(T \bmod p) + uT - u(T \bmod p) - up \\ &= uT - up + u \leq uT - u. \quad (\because p \geq 2).\end{aligned}$$

Otherwise, c_{jk} = true if x_i = false. The corresponding path in $g_{c_{jk}}$ is $\pi = (v_a, v_c, v_{\bar{x}_i}, v_d, v_d, \ldots)$, and the associated demand can be calculated as follows.

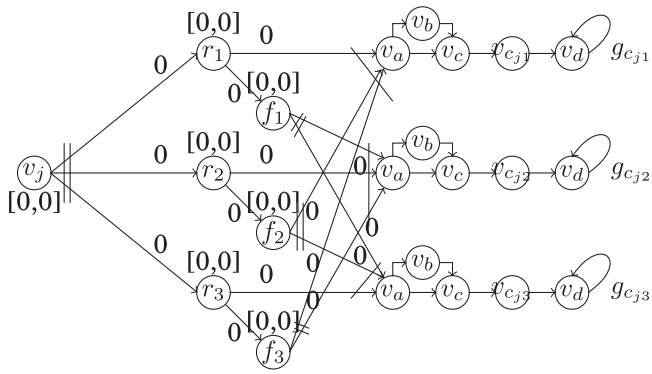


Fig. 7. The subgraph G_j that corresponds to the clause C_j contains three conditional subgraphs.

$$\begin{aligned}\mathrm{DBF}_{c_{jk}}(T) &= u(T \bmod p) + up\left\lfloor \frac{T - T \bmod p}{p} \right\rfloor \\ &= u(T \bmod p) + up\left(\frac{T - T \bmod p}{p}\right) \\ &= u(T \bmod p) + uT - u(T \bmod p) = uT.\end{aligned}$$

In summary, for a given T, if $c_{jk} = \text{true}$, then $\mathrm{DBF}_{c_{jk}}(T) = uT$; otherwise, $\mathrm{DBF}_{c_{jk}}(T) \leq uT - u$. □

For each clause $C_j \in C$, the corresponding structure G_j is given in Figure 7. The starting vertex v_j splits into three parallel paths, each of which starts at the vertex r_k $(1 \leq k \leq 3)$ and contains two conditional branches. The first is composed of the subgraph $g_{c_{jk}}$. The second consists of one fork vertex, f_k, which is associated with a fork arc connecting the other two subgraphs, that is, $g_{c_{jk}}$ and $g_{c_{jk}}$ (we have $k = 2, k = 3$ if $k = 1$). Moreover, the vertex v_a in each subgraph $g_{c_{jk}}$ has a join arc $(\{r_k, f_k, f_k\}, v_a)$. All additional vertices and arcs in G_j are labeled with zero parameters. This structure ensures that the maximum path demand $\mathrm{DBF}_j(T)$ of G_j can be bounded by uT for a given time interval length T, as shown in the following lemma.

LEMMA 3.2. *For a given time interval T and a graph G_j, $DBF_j(T) = uT$ if at least one literal in the corresponding clause C_j is true. Otherwise, $DBF_j(T) \leq uT - u$.*

PROOF. We first show that there is at most one subgraph that should be traversed in C_j. Otherwise, without loss of generality, suppose that $g_{c_{j1}}$ and $g_{c_{j2}}$ are both chosen to be traversed. For the subgraph $g_{c_{j1}}$, we observe that there are three join arcs entering into the vertex v_a, one of which is from the branch (r_2, f_2). Obviously, the conditional structure associated with r_2 precludes entrance into the vertex v_a of $g_{c_{j2}}$ if the arc (r_2, f_2) is passed through. That is, the subgraph $g_{c_{j2}}$ will not be chosen because $g_{c_{j1}}$ has been chosen for traversal, which contradicts the assumption. Thus, we can choose only one subgraph to be traversed.

According to Lemma 3.1, if there is a literal, denoted as c_{jk}, that is true, $C_j = \text{true}$. We can traverse the corresponding subgraph $g_{c_{jk}}$ and obtain a path demand uT as the maximum path demand of G_j. Otherwise, all three literals in C_j are false ($C_j = \text{false}$), and then, traversing any one of the three subgraphs in G_j will lead to a maximum demand bounded by $uT - u$. This completes the proof. □

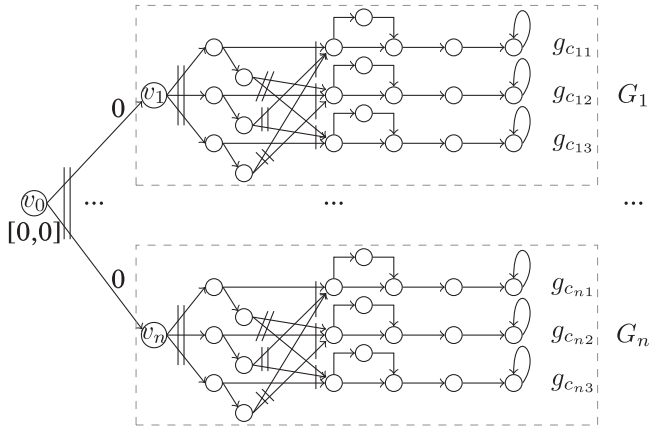


Fig. 8. Example for the construction of the task τ_2 from a given 3SAT instance E.

We construct the task set $\mathscr{T}$ from a 3SAT, E, with m variables and n clauses, as follows. The task set contains two tasks:

—The first task τ_1 contains just one vertex, labeled with $[T(1-c'), T]$, without any arcs.
—The second task τ_2 consists of n substructures, G_j, $(j = 1, \ldots, n)$, each of which corresponds to a clause in C. The vertex v_0 is labeled with $[0, 0]$; the associated fork arc $(v_0, \{v_1, \ldots, v_n\})$ is labeled with an interarrival time of 0 (see Figure 8).

By now, the task set $\mathscr{T}$ cannot be considered yet as a complete reduction from an instance E of the 3SAT problem. We consider two literals c_{jk} and $c_{lk'}$ within different clauses C_j and C_l, which correspond to the same variable or its negation. Without loss of generality, we let $c_{jk} = x_i$ and $c_{lk'} = \bar{x}_i$. The assignments of the values to the variables x_i in literals c_{jk} and $c_{lk'}$ should be consistent with each other. Accordingly, the paths in subgraphs $g_{c_{jk}}$ and $g_{c_{lk'}}$ should make the same decision on whether to visit their corresponding v_b. For expression convenience, we further use $g_{c_{jk}}.v_x$ and $g_{c_{lk'}}.v_x$ to designate, respectively, the vertices v_x of subgraphs $g_{c_{jk}}$ and $g_{c_{lk'}}$ $(x = a, b, c)$. Here, a consistent decision means that the vertices $g_{c_{jk}}.v_b$ and $g_{c_{lk'}}.v_b$ must be visited when x_i = true; otherwise, both of these vertices should not be visited when x_i = false. Meanwhile, the inconsistent decision is not forbidden in the current version of τ_2. For example, to obtain the maximum path demands for both subgraphs $g_{c_{jk}}$ and $g_{c_{lk'}}$, we should visit $g_{c_{jk}}.v_b$ and not visit $g_{c_{lk'}}.v_b$ at all. In this case, the path demands of subgraphs $g_{c_{jk}}$ and $g_{c_{lk'}}$ are both equal to uT, which indicates that $c_{jk} = c_{lk'}$ = true and further leads to a contradiction: $x_i = \bar{x}_i$ = true.

To preclude this inconsistent case, we add a new vertex o_k into the subgraph $g_{c_{jk}}$ of G_j and generate one fork arc from $g_{c_{jk}}.f_k$ to o_k and another fork arc from $g_{c_{jk}}.v_c$ to o_k, as shown in Figure 9. We also modify the structure G_l with the same operations. Finally, we add a join arc from o_k to $v_{c_{lk'}}$ and its symmetric join arc from $o_{k'}$ to $v_{c_{jk}}$. All the vertices and arcs are labeled with zero parameters. With this modification, the paths of the subgraphs corresponding to the same variable or its negation will not collide with each other, as shown in Lemma 3.3.

LEMMA 3.3. *For the subgraphs corresponding to the same variable or its negation, the paths on these subgraphs must make the same decision on whether to visit their own vertices v_b if at least one of these subgraphs is chosen for the maximum demand.*

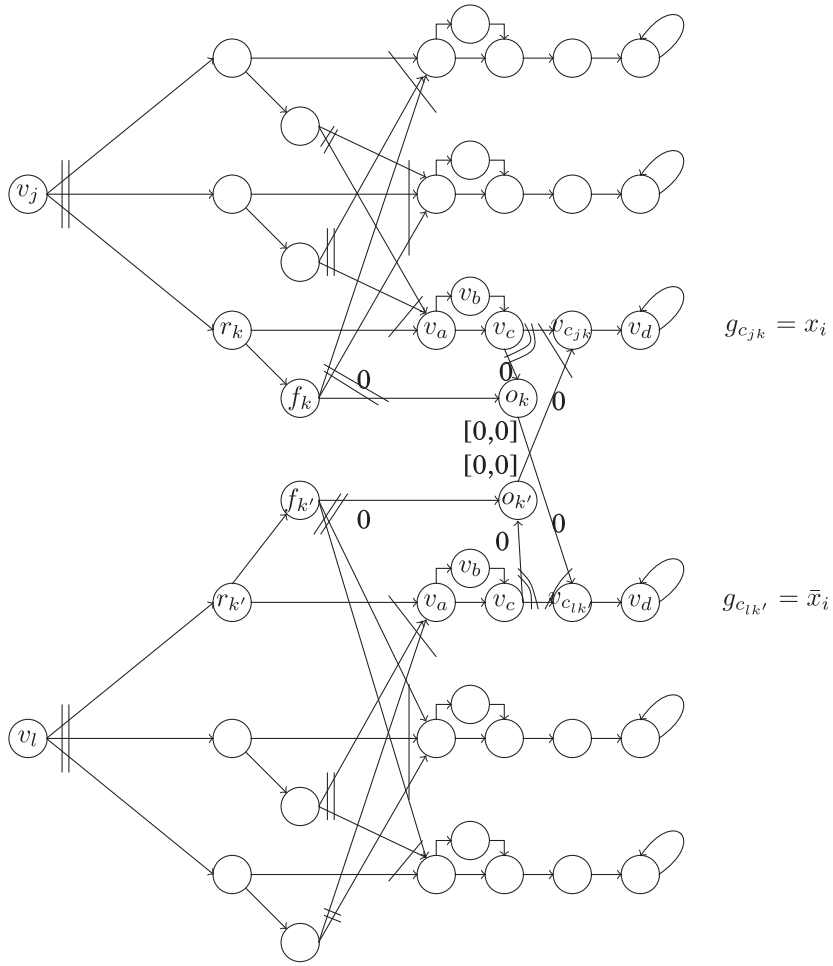


Fig. 9. Modification of subgraphs $g_{c_{jk}}$ and $g_{c_{lk'}}$ to preclude the inconsistent case.

PROOF. Without loss of generality, we consider only two subgraphs, $g_{c_{jk}}$ and $g_{c_{lk'}}$, and assume that $g_{c_{jk}}$ corresponds to x_i and that $g_{c_{lk'}}$ corresponds to $\bar{x}_i$. We can distinguish among three possibilities concerning the subgraphs $g_{c_{jk}}$ and $g_{c_{lk'}}$, depending on whether none, one, or both are chosen for traversal. We now consider each possibility separately.

(1) If the subgraphs $g_{c_{jk}}$ and $g_{c_{lk'}}$ are both chosen to be traversed, without loss of generality, we assume that the decisions on how to traverse these two subgraphs are inconsistent. For example, we should visit $g_{c_{jk}}.v_b$ and not visit $g_{c_{lk'}}.v_b$. As shown in Figure 9, it is easy to see that we can pass through $(g_{lk'}.v_a, g_{lk'}.v_c)$ and enter into the vertex $v_{c_{lk'}}$ at the time instant of 0. However, there is another join arc $(o_k, v_{c_{lk'}})$; thus, we cannot further generate a job from $v_{c_{lk'}}$ until the path $\pi = (g_{c_{jk}}.v_a, g_{c_{jk}}.v_b, g_{c_{jk}}.v_c, o_k)$ from $g_{c_{jk}}$ enters into $v_{c_{lk'}}$ by passing through the join arc $(o_k, v_{c_{lk'}})$. Observe that the earliest time at which the path π can reach $v_{c_{lk'}}$ is 1. This implies that we have to wait for one time unit at the vertex $v_{c_{lk'}}$ if we do not visit $g_{c_{lk'}}.v_b$. In this case, it is better to visit $g_{c_{lk'}}.v_b$, which leads to no waiting time

and an extra demand u. Therefore, this contradicts the maximum demand if the paths associated with $g_{c_{jk}}$ and $g_{c_{lk'}}$ are inconsistent.

(2) Suppose that there is only one subgraph chosen to be traversed; without loss of generality, we assume that $g_{c_{jk}}$ is traversed in G_j and $g_{c_{lk'}}$ is not chosen for traversal in G_l. In this case, the path of $g_{c_{lk'}}$ is not important to $g_{c_{jk}}$ because the branch $(f_{k'}, o_{k'})$ is traversed instead of the vertex $g_{c_{lk'}}.v_a$. Thus, it is necessary to ensure that the path in C_l will not affect the choice of the path prefix in $g_{c_{jk}}$. Observe that the path from $r_{k'}$ of $g_{c_{lk'}}$ to $v_{c_{jk}}$ costs no time. Then, the path from $g_{c_{lk'}}$ can reach $v_{c_{jk}}$ at the time instant of 0. This indicates that the earliest time at which the job can be generated from $v_{c_{jk}}$ depends only on the path from r_k to $g_{c_{jk}}.v_c$.

(3) None of the subgraphs $g_{c_{jk}}$ and $g_{c_{lk'}}$ are chosen for traversal. In this case, the vertices $v_{c_{jk}}$ and $v_{c_{lk'}}$ will not be traversed. It is not necessary to consider this trivial case.

In summary, the modified task model of τ_2 can preclude the inconsistent path prefixes of the subgraphs that correspond to the same variable or its negation. This completes the proof. □

After the modification for the subgraphs associated with each variable x_i and its corresponding negation, we now show that the task set $\mathscr{T}$ is unfeasible if and only if E is satisfiable.

LEMMA 3.4. *If E is satisfiable, then $DBF(T) > T$.*

PROOF. Let E be satisfiable and consider some satisfying assignment. The task set $\mathscr{T}$ contains two tasks, in which τ_1 has only one vertex with $[T(1-c'), T]$ and τ_2 consists of n substructures, each of which corresponds to a clause in C. Considering each clause C_j formed as the disjunction of three logical variables or the negations of the variables, such as $c_{j1} \vee c_{j2} \vee c_{j3}$, there exists at least one variable in the clause evaluating as true. Without loss of generality, we assume that $c_{jk} =$ is true. In the execution of τ_2, the corresponding subgraph G_j implies a conditional structure such that only one subgraph can be chosen for traversal. Suppose that we choose $g_{c_{jk}}$ for traversal. Note that, for a given time interval length T, according to Lemma 3.2, the demand of the path generated from $g_{c_{jk}}$ is uT because the corresponding c_{jk} is true. Therefore, the total demand of the parallel paths associated with all the substructures is equal to nuT for a given T. That is, $\mathrm{DBF}_{\tau_2}(T) \geq nuT$. Moreover, we also have that $\mathrm{DBF}_{\tau_1}(T) = T(1-c')$. The demand bound function of $\mathscr{T}$ is $\mathrm{DBF}(T) \geq T(1-c') + nuT = T + \epsilon$. This completes the proof. □

LEMMA 3.5. *If E is not satisfiable, then $\forall t \geq 0 : DBF(t) \leq t$.*

PROOF. Assume that E is not satisfiable; we want to show that $\mathrm{DBF}(t) \leq t$ for all $t \geq 0$.

We first consider the case when $t = T$. Because E is not satisfiable, there must exist at least one clause in C that is false. Without loss of generality, we assume that only one clause of C, say C_j, is false and the other $n-1$ clauses in C are true. According to Lemma 3.2, the path demand corresponding to G_j is at most $uT - u$, and the total path demand corresponding to the other $n-1$ clauses equals $(n-1)uT$. Thus, the demand bound of τ_2 is given as follows.

$$\mathrm{DBF}_{\tau_2}(T) \leq nuT - u. \tag{4}$$

Moreover, we have that $\mathrm{DBF}_{\tau_1}(T) = T(1-c')$. The demand of $\mathscr{T} = \{\tau_1, \tau_2\}$ can be bounded as follows.

$$\begin{aligned} \mathrm{DBF}(T) &= \mathrm{DBF}_{\tau_1}(T) + \mathrm{DBF}_{\tau_2}(T) && (5)\\ &\le T(1-c') + nuT - u \quad \text{(substitute Equation (4) into Equation (5))} && (6)\\ &\le T + \epsilon - u. && (7) \end{aligned}$$

Combining Equations (1), (3), and (7), $\mathrm{DBF}(T)$ can be further bounded by T: $\mathrm{DBF}(T) \le T$ because $u = \frac{c' + \frac{\epsilon}{T}}{n}$ and $\epsilon \le \frac{T}{nT-1}c'$ holds.

For $t \in [0, T)$: The task τ_i does not count in such an interval because its deadline is T. Furthermore, the task τ_2 has the demand bounded by nut, as shown in the proof of Lemma 3.2. Thus, $\mathrm{DBF}(t) = nut = t(c' + \epsilon/T)$. According to Equation (3), we can conclude that $\mathrm{DBF}(t) \le t$ because $\epsilon \le T(c - c')$ holds.

For $t \in (T, \infty)$: The task τ_1 can release its job and contributes $T(1 - c')$ to $\mathrm{DBF}(t)$. Furthermore, the demand of the task τ_2 can be bounded by nut. We have $\mathrm{DBF}(t) \le nut + T(1-c') = (c' + \frac{\epsilon}{T})t + T(1-c')$. According to Equation (3), we can further conclude that $\mathrm{DBF}(t) \le t$ because $\epsilon \le \frac{T}{T+1}(1 - c')$ holds. This completes the proof. □

Lemmas 3.4 and 3.5 indicate the first property we proposed earlier such that there exists a proper reduction from the 3SAT problem. Furthermore, all the values in the task system $\mathcal{T}$ that we constructed are bounded by a polynomial in the values in the instance E of 3SAT. Finally, we will show that the utilization of the FJRT task set $\mathscr{T}$ is bounded by a constant c. On the one hand, the utilization of τ_1 is 0, with τ_1 being acyclic. On the other hand, the utilization of τ_2 is at most $nu = c' + \frac{\epsilon}{T} \le c$ because we can traverse at most one subgraph $g_{c_{jk}}$ in each substructure G_j, and the utilization of each $g_{c_{jk}}$ is exactly u, which can be calculated by considering the cyclic part of $g_{c_{jk}}$, as shown in Figure 6. Therefore, the utilization of the entire task system is bounded by a constant c. We now present the main result of this section in Theorem 3.6.

THEOREM 3.6. *For any constant $c < 1$, the feasibility problem for the FJRT task set τ with $U(\tau) \le c$ is coNP-hard in the strong sense.*

4. THE FJRT MODEL BASED ON HIERARCHY DIGRAPHS

In the previous section, we constructed a special FJRT system $\mathscr{T}$ and proved that the associated feasibility problem is a coNP-hard problem in the strong sense. During the construction phase, we noted that the fork arcs that come from different vertices can be joined into one vertex. For example, as shown in Figure 7, the arcs separately forked from f_k and $f_{k'}$ are joined into one vertex $g_{c_{jk''}}.v_a$ (we have that $k'' = 3$ if $k = 1$ and $k' = 2$). Intuitively, these arcs jump from one parallel section to the other and finally join. We refer to this type of fork arcs as jumping arcs. Although the jumping arcs that send important control information between the parallel sections can model the communication behaviors of many concurrent parallel, real-time systems, we believe that the jumping arc is one of the main factors that significantly affects the complexity of the feasibility problem. In the rest of this article, we focus on a particular restricted form of the FJRT model, in which the control is not allowed to jump between parallel sections. In other words, the arcs forked from different vertices are forbidden to be joined into one vertex. We first introduce the corresponding definition and relative notations in the next section.

4.1. Hierarchy Task Graph and Relative Notations

A parallel task τ is characterized by a hierarchy task graph $D = (V_\tau, A_\tau)$, where the set V_τ of vertices represents the types of jobs that can be released by the task τ, and the arcs in A_τ indicate the order in which jobs generated by τ are released. The terminology of *hierarchy* means that the vertex is allowed to embed a fork-join structure, which is

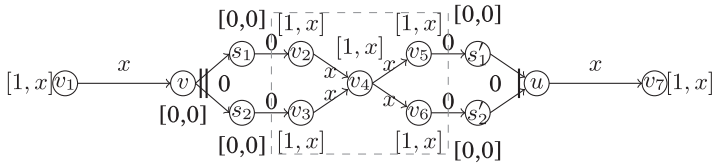


Fig. 10. The equivalent transformation of τ_1 defined in Example 2.9.

used to express parallel jobs. Thereby, we distinguish two types of vertices in V_τ by using different notations:

—If the vertex contains a fork-join structure, we call it a generalized "vertex." Obviously, the generalized "vertex" is not a real vertex, but it can be viewed as a subgraph embedded within D.
—Otherwise, the vertex that generates only one sequential job at one time is called a simple vertex.

Because the simple vertex has been defined previously in Section 2, in what follows, we will merely introduce the parameters associated with the generalized "vertex."

The generalized "vertex" is specified as a DAG $G = (V, E)$. Each vertex v_i of G might correspond to a sequential job that is released by τ if v_i is a simple vertex; otherwise, a generalized "vertex" v_i will correspond to a parallel job. An example of the structure of G is shown in Figure 10. We first denote a fork vertex v and a join vertex u of G, then define several source and sink vertices in G. Note that the numbers of source and sink vertices should be the same according to Proposition 2.4. Without loss of generality, we assume that there are m source and sink vertices, which are separately denoted as s_i and s'_i $(i = 1, \ldots, m)$. Specifically, the number m of source (sink) vertices in Figure 10 is 2. The fork and join arcs of G are separately represented as (v, S) and (S', u), where $S = \{s_1, \ldots, s_m\}$ and $S' = \{s'_1, \ldots, s'_m\}$. These hyperarcs are labeled, respectively, with the nonnegative integers $p(v, S)$ and $p(S', u)$, which denote the minimum job interrelease separation times.

Moreover, for convenience of analysis, the DAG of the generalized "vertex" is further restricted such that the fork, join, source and sink vertices in the DAG cannot be generalized. We consider only this restricted generalized "vertex" in the rest of this article. For the nonrestricted generalized "vertex," we perform its transformation into an equivalent (restricted) one. We illustrate this transformation by using very basic examples, to follow.

Example 4.1. The example task τ_1 in Figure 5(a) corresponds to a hierarchy task graph D consisting of only one generalized "vertex." The fork-join structure of this generalized "vertex" is defined as a DAG G that contains a fork vertex v_1 and a join vertex v_7. Moreover, vertices v_2 and v_3 can be viewed as two source vertices in G, and vertices v_5 and v_6 are the sink vertices in G. The equivalent transformation of τ_1 is given in Figure 10.

We first add two simple vertices v and u as the new fork and join vertices, respectively, then add two new arcs (v_1, v) and (u, v_7). We further add two simple vertices s_1 and s_2 as source vertices, then add arcs (s_1, v_2) and (s_2, v_3). We also add two simple vertices s'_1 and s'_2 as sink vertices, then add arcs (v_5, s'_1), and (v_6, s'_2). All newly added vertices are labeled with $[0, 0]$. The arcs (v_1, v) and (u, v_7) are labeled, respectively, with the interreleased separations that used to correspond to the hyperarcs $(v_1, \{v_2, v_3\})$ and $(\{v_5, v_6\}, v_7)$. All other added arcs are labeled with 0.

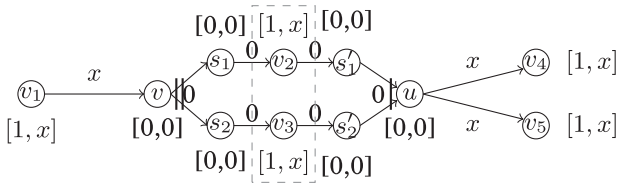


Fig. 11. The equivalent transformation of τ_2 defined in Example 2.10.

Example 4.2. The example task τ_2 in Figure 5(b) is equivalent to the FJRT shown in Figure 11. Simple vertices v and u are added as the fork and join vertices, respectively. Simple vertices s_1 and s_2 are added as two source vertices, and simple vertices s_1' and s_2' are added as two sink vertices. New arcs (s_1, v_2), (s_2, v_3), (v_2, s_1'), and (v_3, s_2') are added with an interreleased separation of 0. Moreover, the arc (v_1, v) is added, which is labeled with the interreleased separation $p(v_1, \{v_2, v_3\})$. Two arcs (u, v_4) and (u, v_5) are also added; their labels are $p(\{v_2, v_3\}, v_4) = x$ and $p(\{v_2, v_3\}, v_5) = x$, respectively.

These two examples show that the tasks previously modeled as CRTs in Examples 2.9 and 2.10 can also be described by the FJRT models based on hierarchy digraphs (short for the FJRT-HD model). This result is presented as the following theorem.

THEOREM 4.3. *The FJRT-HD model is at least as expressive as the CRT model.*

PROOF. The FJRT-HD model, which contains the DRT model as a special case if there is no generalized "vertex," is allowed to express the sequential, conditional, and loop compositions of tasks. Thus, it is trivial to show that the atomic grammar rules $T_1\langle x\rangle T_2$, $T_1 + T_2$, and T^ω can be expressed by the FJRT-HD model. Additionally, the parallel composition of subtasks S_1 and S_2, that is, $S_1 \| S_2$, can be modeled as a generalized "vertex," where the fork and join vertices are prefix$(S_1 \| S_2)$ and suffix$(S_1 \| S_2)$, respectively; the source vertices are prefix(S_1) and prefix(S_2); and the sink vertices are suffix(S_1) and suffix(S_2). In summary, all atomic grammar rules can be expressed by the FJRT-HD model. Therefore, the FJRT-HD model is at least as expressive as the CRT model. □

Although Theorem 4.3 cannot imply that the FJRT-HD is strictly more expressive than the CRT model, it is worth introducing the FJRT-HD model mainly because of the following reasons. On the one hand, Ejsing-Duun et al. [2013] restricted themselves to a well-formed CRT model, in which each job is forbidden to occur in the configuration more than once. The existing pseudo-polynomial time result in Ejsing-Duun et al. [2013] for the well-formed setting cannot reveal the conclusion that the general CRT model is also tractable. On the other hand, Examples 4.1 and 4.2, together with Theorem 4.3, conclude that each CRT model, regardless of whether it is well formed, can be equivalently expressed by an FJRT-HD model. In this section, we will prove that the FJRT-HD model is tractable, which implies that the CRT model is also tractable even if it is not well formed.

4.2. Feasibility Analysis of the FJRT-HD Model

For our proposed parallel task model, we are concerned with the corresponding preemptive uniprocessor feasibility problem:

Definition 4.4 (*Feasibility*). An FJRT-HD set $\mathcal{T}$ is preemptive uniprocessor feasible if and only if all job sequences generated by $\mathcal{T}$ can be executed on a preemptive uniprocessor platform such that all jobs meet their deadlines.

We note that the feasibility problem for the DRT model can be solved within pseudo-polynomial time. In what follows, we will show that this technique can be further applied to the analysis of the FJRT-HD model, and the time complexity remains pseudo-polynomial.

In attempting to adapt the approach in Stigge et al. [2011] to the FJRT-HD setting, we propose a transformation method for each given FJRT-HD task to transform each task into an equivalent DRT task with pseudo-polynomial size. The main challenge that we may face in the transformation is how to deal with the generalized "vertex," which is hierarchically defined in a nested structure. In what follows, we are concerned with the parallel path abstraction of the generalized "vertex" and keep this useful information for further use (refer to Section 4.2.2).

4.2.1. Computing Parallel Path Demands in the Generalized "Vertex". For a given generalized "vertex," we can distinguish among two possibilities concerning the associated DAG $G = (V, A)$, depending on whether or not it contains generalized "vertices." We consider each possibility separately here.

Case A. If the DAG G consists only of simple vertices. In the DAG G, we let v be the fork vertex and u be the join vertex and assume that there are m source vertices and m sink vertices, denoted as $s_1, \ldots, s_m$ and $s'_1, \ldots, s'_m$, respectively. The remaining vertices in G, which are connected between the source and sink vertices, are denoted as $v_1, \ldots, v_n$. Each job sequence generated in G corresponds to an execution path that starts at the fork vertex v, then forks into m parallel paths starting from s_1 to s_m. Denote by $\pi(s_i, -)$ the parallel path that begins with the i-th source vertex s_i, such as $(s_i, v_{[1]}, \ldots, v_{[l]})$. We further define the execution demand and deadline of $\pi(s_i, -)$ as follows.

—Execution demand: $e(\pi(s_i, -)) := \sum_{j=1}^{l} e(v_{[j]})$;

—Deadline: $d(\pi(s_i, -)) := \sum_{j=1}^{l-1} p(v_{[j]}, v_{[j+1]}) + d(v_{[l]})$.

For a fixed time-interval length t, we focus on the maximum execution demand of the paths with a deadline of at most t. More formally, we further propose the concept of the path demand as follows.

Definition 4.5 (*Path Demand*). For a given source vertex s_i, an arbitrary vertex v_l and a fixed time-interval length t, the path demand $e(s_i, v_l, t)$ denotes the maximum cumulative execution demand of jobs generated along the path starting at s_i and ending at v_l, and the associated deadline is bounded by the length t:

$$e(s_i, v_l, t) = \max\{e(\pi(s_i, v_l)) | \pi(s_i, v_l) \text{ is an arbitrary path from } s_i \text{ to } v_l, \text{ and } d(\pi(s_i, v_l)) \leq t\}.$$

The path demand $e(s_i, v_l, t)$ can be calculated by using the following recursive formula:

$$e(s_i, v_l, t) = \max\{e(s_i, v_j, t - d(v_l) - p(v_j, v_l) + d(v_j)) + e(v_l) | (v_j, v_l) \in A\}. \quad (8)$$

The correctness of this formula can be discussed in a twofold manner. First, we show that there is a path with a deadline of at most t that starts at s_i and ends at v_l. Let $t' = t - d(v_l) - p(v_j, v_l) + d(v_j)$. It is known that the path demand $e(s_i, v_j, t')$ corresponds to at least one path $\pi(s_i, v_j)$ from s_i to v_j with a deadline of at most t'. Because v_j is denoted as a predecessor vertex of v_l, we now combine the path $\pi(s_i, v_j)$ and the arc (v_j, v_l) into a new path $\pi(s_i, v_l) = (s_i, \ldots, v_j, v_l)$. Obviously, the deadline of $\pi(s_i, v_l)$ can be calculated as $t = t' + d(v_l) + p(v_j, v_l) - d(v_j)$, which implies that there is a path from s_i to v_j with a deadline of at most t.

We now show that $e(s_i, v_l, t)$ is the maximum demand of the path from s_i to v_l with a deadline of at most t. Suppose that $\pi^*(s_i, v_l)$ is the optimal path with a deadline of at most t, which begins with s_i and ends at v_l, and $e(\pi^*(s_i, v_l))$ is its demand. Then, we have $e(\pi^*(s_i, v_l)) > e(s_i, v_l, t)$. Additionally, let v_j be such a predecessor vertex of v_l on the path $\pi^*(s_i, v_l)$. Let $\pi^*(s_i, v_j)$ be the subpath of $\pi^*(s_i, v_l)$ from s_i to v_j with a deadline of at most $t' = t - p(v_j, v_l) - d(v_l) + d(v_j)$. Therefore, by the induction, we have $e(\pi^*(s_i, v_j)) \leq e(s_i, v_j, t')$ because $e(s_i, v_j, t')$ denotes the maximum demand of the paths between s_i and v_j with a deadline of at most t'. This means that $e(\pi^*(s_i, v_l)) = e(\pi^*(s_i, v_j)) + e(v_l) \leq e(s_i, v_j, t') + e(v_l) \leq e(s_i, v_l, t)$. In summary, we have $e(\pi^*(s_i, v_l)) = e(s_i, v_l, t)$. This completes the discussion.

Furthermore, for a given source vertex s_i and a fixed time interval t, we denote $e(s_i, t)$ as the maximum cumulative execution demand of the paths starting at s_i with a deadline of at most t:

$$e(s_i, t) = \max\{e(s_i, v_l, t) | v_l \text{ is an arbitrary vertex of the DAG } G\}.$$

This concept, used as an abstraction of the single path, can be further extended for parallel paths. We now consider all parallel paths in the DAG G and define the parallel path demand as follows.

Definition 4.6 (*Parallel Path Demand*). For a fixed time interval length t, the parallel path demand $e(t)$ denotes the maximum cumulative execution demand of jobs generated along the parallel paths, respectively, starting from source vertices s_1 to s_m, and the associated deadline is bounded by a length t, which can be calculated by:

$$e(t) = \sum_{i=1}^{m} e(s_i, t).$$

In particular, for the parallel paths that end at their corresponding sink vertices and that finally need to be joined into join vertex u, the associated parallel path demand is defined as $e'(t) = \sum_{i=1}^{m} e(s_i, s_i', t)$.

LEMMA 4.7. *For each constant $T \in \mathbb{N}$, the number of path demands $e(s_i, v_l, t)$, $e(s_i, t)$ and $e(t)$ with $t \leq T$ is bounded polynomially in T, m, and n, which are the numbers of the source vertices and simple vertices in the DAG G.*

PROOF. The proof is trivial. All path demands $e(s_i, v_l, t)$ are in $\mathbb{N}_{\leq m} \times \mathbb{N}_{\leq n} \times \mathbb{N}_{\leq T}$, all path demands $e(s_i, t)$ are in $\mathbb{N}_{\leq m} \times \mathbb{N}_{\leq T}$, and all parallel path demands $e(t)$ ($e'(t)$) are in $\mathbb{N}_{\leq T}$, leaving altogether $O((nm + m + 1)T)$ possibilities. □

Case B. If the generalized "vertex" (say, v) is defined as a nested structure, which contains some generalized "vertices." We will transform each generalized "vertex" v_i into a subgraph G_i that only contains simple vertices; the transformation procedure **Transform()** is given in Algorithm 1. Before approaching this procedure, we first define a parameter for each generalized "vertex" v_i, namely, the diameter D_i of v_i, which is denoted as the longest length of the paths from the fork vertex v to the join vertex u. It is easy to see that the diameter D_i is pseudo-polynomially bounded because D_i is much smaller than the summation of the interreleased separations labeled on the inner arcs defined in v_i. We then denote two sets of parallel path demands of v_i as $E_i = \{e(t) | t \leq D_i\}$ and $E_i' = \{e'(t) | t \leq D_i\}$. For each generalized "vertex" v_i, the equivalent subgraph G_i contains $|E_i| + |E_i'| + 2$ vertices.

As shown in Algorithm 1, we define a source vertex μ_i in each subgraph G_i, which accepts the arcs that formerly entered into v_i, and denote a sink vertex μ_i', from which the outter arcs of v_i leave. Vertices μ_i and μ_i' are labeled with an execution time of 0 and a deadline of 0. Without loss of generality, the elements of E_i and E_i' are sorted in

ALGORITHM 1: Transform(ν)

Input: a generalized "vertex" ν that has a nested structure;
Output: an equivalence generalized "vertex" ν' that contains only simple vertices.

```
1: for each i in [1, n] do
2:     if vertex v_i is a generalized "vertex" then
3:         Set pred_i := {v_l|for each(v_l, v_i) the precedure arc of v_i defined in ν};
4:         Set succ_i := {v_j|for each(v_i, v_j) the successor arc of v_i defined in ν};
5:         Generate two vertices μ_i and μ'_i labeled with an ordered pair [0,0];
6:         while pred_i ≠ ∅ do
7:             predVertex := Pop(pred_i);
8:             generate arc (predVertex, μ_i) labeled with a period 0;
9:         end while
10:        while succ_i ≠ ∅ do
11:            succVertex := Pop(succ_i);
12:            Generate arc (μ'_i, succVertex) labeled with a period 0;
13:        end while
14:        j:=1;
15:        while E_i ≠ ∅ do
16:            e(t_j) = Pop(E_i);
17:            Generate a new vertex v_j labeled with [e(t_j), t_j];
18:            Generate a new arc (μ_i, v_j) labeled with t_j;
19:            j++;
20:        end while
21:        j:=1;
22:        while E'_i ≠ ∅ do
23:            e'(t'_j) = Pop(E'_i);
24:            Generate a new vertex v'_j labeled with [e'(t'_j), t'_j];
25:            Generate two arcs (μ_i, v'_j) and (v'_j, μ'_i) labeled with 0 and t'_j respectively;
26:            j++;
27:        end while
28:    end if
29: end for
```

increasing order. We then denote the j-th element in E_i as $e(t_j)$, where t_j is the time interval length corresponding to the parallel path demand $e(t_j)$. Similarly, the j-th demand in E'_i is denoted as $e(t'_j)$, and t'_j is its corresponding time-interval length. For each $e(t_j) \in E_i$, we define a vertex v_j in G_i, which is labeled with an execution time of e_j and a deadline of d_j, where $e_j = e(t_j)$ and $d_j = t_j$. We then add the arc (μ_i, v_j) with an interreleased separation of t_j into G_i. Moreover, for each $e(t'_j) \in E'_i$, we define a vertex v'_j in G_i, which is labeled with an execution time of $e'_j = e'(t'_j)$ and a deadline of $d'_j = t'_j$. Two arcs, (μ_i, v'_j) and (v'_j, μ'_i), are added into G_i. The interreleased separations labeled with (μ_i, v'_j) and (v'_j, μ'_i) are t'_j and 0, respectively.

Note that the parallel path demands in E_i and E'_i are assumed to be previously calculated before Algorithm 1 runs. In fact, E_i and E'_i cannot be directly calculated by the recursive formulas listed in Case A if the generalized "vertex" in ν also has a nested structure. Note that Algorithm 1 can be used only for the generalized "vertex" nested at a depth of 1. However, Algorithm 1 can be used as the core procedure to solve the deeper problem for the transformation of the generalized "vertex" nested at arbitrary depths. We will show this in Algorithm 2 in the next section.

4.2.2. Transformation of the FJRT-HD into the DRT Model. By inductively computing the path demands in each generalized "vertex" v_i and then forming the transformation of

v_i, an FJRT-HD task τ can be equivalently cast into a DRT task τ'. The transformation method is shown in Algorithm 2.

ALGORITHM 2: Transform the FJRT-HD task τ into an Equivalence DRT Task τ'

Input: an FJRT-HD task τ defined in the hierarchy task graph $D = (V_\tau, A_\tau)$;
Output: an equivalence DRT task τ' defined on the graph $D' = (V_{\tau'}, A_{\tau'})$.

```
1: Set stackGV := ∅;
2: for each vertex v in V_τ do
3:     if v is a generalized "vertex" then
4:         Push v to stackGV;
5:     end if
6: end for
7: while stackGV ≠ ∅ do
8:     v := GetTop(stackGV);
9:     furtherNested := false;
10:    for each vertex ν in v do
11:        if ν is a generalized "vertex" then
12:            Push ν to stackGV;
13:            furtherNested := true;
14:        end if
15:    end for
16:    if furtherNested = false then
17:        v := Pop(stackGV);
18:        Transform(v);
19:    end if
20: end while
```

The procedure in Algorithm 2 can be used to transform the FJRT-HD task that contains the generalized "vertex," which can be nested at arbitrary depths. In this procedure, we use a stack **stackGV** to maintain the order in which the generalized "vertices" should be transformed by procedure **Transform()**. It is easy to see that the generalized "vertex" nested at a smaller depth has a higher priority to be transformed. Thus, the generalized "vertex" nested at a depth of 1 will be transformed first; after that, the nested depths of the other generalized "vertices" in stackGV might be decreased accordingly. Note that the procedure Transform() can be used only for the generalized "vertex" nested at a depth of 1 (See Line 18 in Algorithm 2). Moreover, we also note that each generalized "vertex" in V_τ will not be pushed into stackGV more than twice. Thus, the time complexity of Algorithm 2 is certainly pseudo-polynomial.

LEMMA 4.8. *For an FJRT-HD task τ and its transformation τ', their demand bound functions coincide, that is, $\forall t \geq 0 : DBF_\tau(t) = DBF_{\tau'}(t)$.*

PROOF. For a given time interval t, we assume that the demand bound function of τ DBF_τ corresponds to a path in the hierarchy graph D, denoted by $\pi^* = (v_1, \ldots, v_l)$, where the execution demand $e(\pi^*)$ and the relative deadline $d(\pi^*)$ of π^* fulfill the following: $e(\pi^*) = \mathrm{DBF}_\tau(t)$ and $d(\pi^*) \leq t$. We iteratively construct a corresponding path π' based on the following rules.

—The simple vertices in π^* should stay in the same position of π' without any changes;
—If π^* contains some generalized "vertices" v_i, assume that the corresponding job triple is (r_i, e_i, d_i), where r_i is the starting time of traversal of v_i, e_i is the demand of the parallel paths forked in v_i, and $d_i - r_i$ is the deadline of v_i, which can be computed as the maximum length of the parallel paths forked in v_i. In what follows, we separately

consider the cases in which the generalized "vertex" v_i is and is not the ending vertex of π^*.

—If π^* traverses v_i as a middle vertex, v_i corresponds to a sequential path $(\mu_i, v'_{t_1}, \mu'_i)$ of π', where $t_1 = \max\{\tilde{t} | e'(\tilde{t}) \in E' \wedge \tilde{t} \leq d_i - r_i\}$. The demand of the path $(\mu_i, v'_{t_1}, \mu'_i)$ can be calculated as $e(\mu_i)+e(v'_{t_1})+e(\mu'_i) = e'(t_1)$ because the execution times assigned to the vertices μ_i, μ'_i, and v'_{t_1} are $e(\mu_i) = e(\mu'_i) = 0$ and $e(v'_{t_1}) = e'(t_1)$, respectively, according to the procedure in Algorithm 1.

—Otherwise, if π^* ends at v_i, v_i corresponds to a sequential path (μ_i, v_{t_2}) of π', where $t_2 = \max\{\tilde{t} | e(\tilde{t}) \in E \wedge \tilde{t} \leq d_i - r_i\}$. The demand of the path (μ_i, v_{t_2}) is $e(\mu_i) + e(v_{t_2}) = e(t_2)$.

Note that e_i is the demand of the parallel paths that are forked in v_i with a deadline of at most $d_i - r_i$, and according to Definition 4.6, $e'(t_1)$ and $e(t_2)$ are defined as the maximum demands of these parallel paths. Thus, we have $e'(t_1) \geq e_i$ and $e(t_2) \geq e_i$. Moreover, we also note that $t_1 \leq d_i - r_i$ and $t_2 \leq d_i - r_i$ hold. Therefore, the sequential path in π' has a larger demand and a smaller length than its corresponding generalized "vertex."

In summary, the path π' of τ' will have a larger demand and a smaller length than its corresponding path π^* of τ because the generalized "vertices" of π^* are replaced by the corresponding sequential paths in π'. That is, $e(\pi^*) \leq e(\pi')$. Moreover, it is easy to see that $\mathrm{DBF}_{\tau'}(t) \geq e(\pi')$, and according to the assumption, we have that $\mathrm{DBF}_{\tau}(t) = e(\pi^*)$. Thus, we can conclude that $\mathrm{DBF}_{\tau}(t) \leq \mathrm{DBF}_{\tau'}(t)$.

Additionally, we assume that the demand bound function $\mathrm{DBF}_{\tau'}(t)$ of τ' corresponds to the path π'. If π' passes some subpaths, such as $(\mu_i, v'_{t'}, \mu'_i)$ (or $(\mu_i, v_{t'})$), we let these subpaths be defined on the subgraph G_i, which is the transformation of the generalized "vertex" v_i. The corresponding path π^* of τ will pass v_i in the same position, and the visit of v_i corresponds to the set of parallel paths in v_i with the total execution demand $e(t')$ and the deadline t'. We note that such parallel paths always exist in v_i according to Definition 4.6. It is easy to show that the execution time and deadline of the corresponding path π^* can be restricted to being the same as those of the path π'. This implies that $\mathrm{DBF}_{\tau}(t) \geq \mathrm{DBF}_{\tau'}(t)$ for each given time-interval length t, which completes the proof. □

Given this lemma, the main theorem follows directly because the results from Stigge et al. [2011] can be applied to the set of transformed tasks.

THEOREM 4.9. *For a constant $c < 1$, the feasibility for all FJRT-HD task sets $\mathscr{T}$ based on hierarchy digraphs with $U(\mathscr{T}) \leq c$ can be solved in pseudo-polynomial time.*

PROOF. Given an FJRT-HD system $\mathscr{T}$, we apply the described transformation to all tasks to obtain the corresponding DRT system $\mathscr{T}'$. Lemma 4.8 guarantees that their demand bound functions coincide, which implies that their utilizations are also the same. Thus, we can apply the main result from Stigge et al. [2011], guaranteeing that the demand computation problem will be solved for $\mathscr{T}'$ in pseudo-polynomial time if the following two conditions hold:

—The number of vertices in $\mathscr{T}'$ and the values in $\mathscr{T}'$ (vertex and arc labels) are pseudo-polynomially bounded in the description of $\mathscr{T}$.

—The transformation itself runs in pseudo-polynomial time.

For the first property, Lemma 4.7 shows that the number of new vertices per generalized "vertex" is pseudo-polynomially bounded. Moreover, all labels of the new vertices and arcs associated with each generalized "vertex" v_i are bounded by the diameter of v_i, which is much less than the summation of all the interreleased separations labeled on

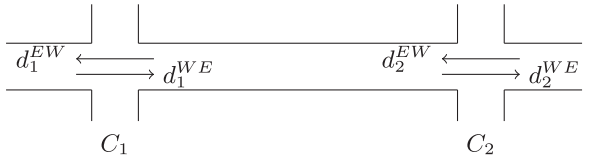


Fig. 12. Traffic flow in the ITCS case study.

the arcs nested in v_i. The second property is trivially satisfied because creating each of these vertices and arcs (and their labels) can be done in $O(1)$ time, which can be seen as an inexpensive operation. □

5. MODELING WITH FJRT: A CASE STUDY

In this section, we demonstrate the usage of FJRT by modeling an Intelligent Traffic Control System (ITCS). The ITCS was originally modeled by Simulink/Stateflow [Angermann 2007]. As we will see, the Simulink/Stateflow model of the ITCS is rather complicated and difficult to read. Then, we model the ITCS by FJRT, which gives a much simpler and more intuitive representation of the system. From the FJRT model, readers can easily understand the essential structure of the system.

In the ITCS of urban road networks, traffic signal lamps work in collaboration for certain optimization targets, for example, minimum energy consumption and emissions. For simplicity, we focus on two adjacent crossings C_1 and C_2 to illustrate the traffic flow control problem. Without loss of generality, traffic flows only in the west-east direction are considered, as shown in Figure 12.

The traffic flow control system periodically produces global control decisions in real time depending on the traffic flow observed at both crossings C_1 and C_2, then sends them to the lamp at each crossing. To decide the color of the traffic signal lamp at a crossing for the next period, the control system needs to analyze and predict the traffic flows in both directions. Traffic control at each crossing consists of the following parts:

—**Preprocessing**. Sensors are settled at each crossing to acquire real-time raw data of traffic flow information. Preprocessing of the sensor data is conducted as the first step of the traffic-control routine, in order to improve the data quality (including, e.g., filtering, normalization, and cleansing). With the preprocessed data, it also decides whether the current traffic flow at this crossing is heavy or light. If it is heavy, then the next period of signal control will be triggered after a relatively longer delay since a more frequent switching of traffic signal causes higher overhead. Otherwise, the delay for triggering the next period of signal control decision is shorter (there are 4 traffic levels in the original system design, but for simplicity we consider only 2 levels, light and heavy, in this case study).

—**Heavy Traffic Control**. If heavy traffic is recognized in the earlier step, the traffic flow data after preprocessing is then submitted to heavy traffic analysis. We use C_1 as an example to explain the Heavy Traffic Analysis component at each crossing. The traffic of both directions is analyzed:

  —**West-to-East Heavy Traffic Analysis (WtoE-H)**. The analysis of this direction depends on the preprocessed data of d_1^{WE}. The analysis program WtoE-H is finished in at most 3s relative to its trigger time (considering the competition of CPU time by workload of other functionalities), that is, the relative deadline is 3s.

  —**East-to-West Heavy Traffic Analysis (EtoW-H)**. The analysis of this direction depends on the preprocessed data of d_1^{EW} and the west-to-east analysis results at crossing C_2 (based on d_2^{EW}). The analysis program EtoW is triggered when input

from the preprocessed data from both crossings is ready. Similar to West-to-East Heavy Traffic, the preprocessing program at each crossing is required to finish in at most 3s, thus data from the preprocessing program of crossing C_1 is set to be ready 3s after the preprocessing program is triggered. On the other hand, the WtoE-H program at each crossing is required to be finished at most 4s after it is triggered (the relative deadline is 4s). The EtoW-H program also takes at most 4s and has a relative deadline of 4s.

—**Signal Decision**. The traffic flow analysis results of both directions are merged to make the final traffic signal decision. After the control decision program is triggered, the system will wait for 120s to trigger data preprocessing for the next period.

—**Light Traffic Control**. The analysis for light traffic is similar to the heavy-traffic case. Both the West-to-East Light Traffic Analysis (WtoE-L) and East-to-West Light Traffic Analysis (EtoW-L) are conducted, and the analysis results are merged to make the final traffic signal decision. The WtoE-L and EtoW-L programs are also finished in at most 4s. However, there are two differences from the case of heavy traffic: (1) the delay for triggering the next period of signal control is 60s (instead of 120s). (2) The EtoW-L analysis does *not* depend on the analysis results of the west-to-east traffic at C_2, since the delay for triggering the next period of signal control is shorter, and there is not enough time for the traffic at C_2 to propagate to C_1 in the current control period.

Figure 13 shows the Simulink/Stateflow model of the ITCS described earlier. We assume that 100000 ticks corresponds to 1s. Note that the Simulink/Stateflow model does not explicitly express the relative deadline constraints of each program. These constraints are annotated as auxiliary information in the model. (A detailed explanation of this Simulink/Stateflow model is omitted due to space limit.)

Figure 14 shows the modeling of the ITCS by a single FJRT task. The task contains two parts, G_1 and G_2, which models the control of crossing C_1 and C_2, respectively. The dashed vertices are dummy job types with both WCET and relative deadline being 0, which are merely used to help to model certain graph structures. All other vertex (solid circles) have implicit deadlines, that is, the relative deadline of a vertex equals the minimal interrelease separation among all its outgoing edges. The WCET of each vertex is also omitted in the figure for simplicity. Since two subgraphs are symmetrical, in the following, we explain only G_1, and use v_j to denote $G_1.v_j$.

First, v_1 corresponds to the preprocessing functionality, which has two out-going edges as it decides whether the traffic is heavy or light. If the traffic is light, it takes the branch to the dummy vertex v_2, which forks the analysis functionality of both directions (v_4 corresponds to EtoW-L and v_5 corresponds to WtoE-L). The final control decision for light traffic (v_8) is made based on the analysis results of both EtoW-L and WtoE-L, so the outgoing edges from v_4 and v_5 join to trigger v_8. A delay of 60s is inserted before triggering the next control period in the case of light traffic, so the edge between v_8 and v_1 is marked with 60. If the traffic is heavy, among the two outgoing edges, the branch to v_3 is taken. v_3 forks the analysis functionality of both directions for heavy traffic (v_7 corresponds to EtoW-H and v_6 corresponds to WtoE-H). Two edges from v_6 and v_7 join to trigger the control decision for heavy traffic v_9, and after a delay of 120s, v_1 in the next control period is triggered. The EtoW-H functionality needs input of both the preprocessed data from v_1 and the east-to-west traffic analysis result from the other crossing C_2. On the other hand, the west-to-east traffic (either heavy or light) analysis results are sent to C_2 for the same reason. The dummy node v_{10} is triggered by edges from either v_5 (analysis of light traffic) or v_6 (analysis of heavy traffic). The outgoing

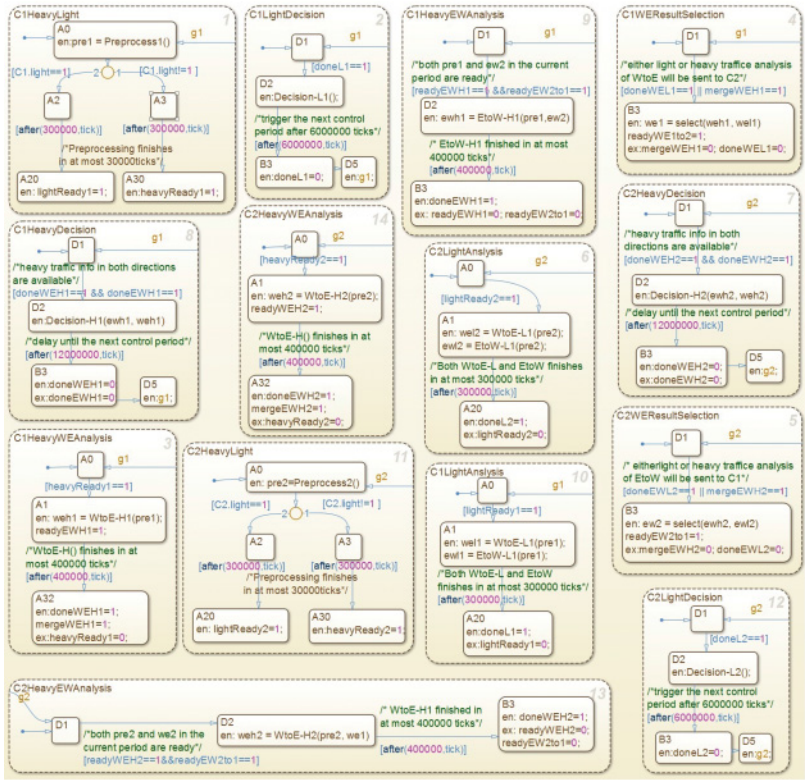


Fig. 13. Simulink/Stateflow Model of the ITCS.

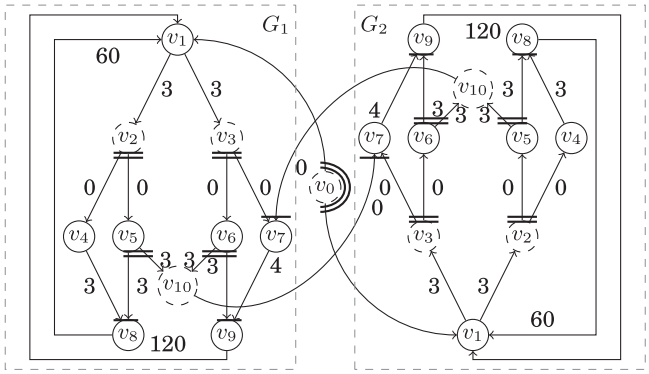


Fig. 14. The FJRT model of the ITCS system.

edge of v_{10} goes into the other subgraph G_2, joining with the edge from $G_2.v_3$ to trigger $G_2.v_6$, the heavy west-to-east traffic analysis at C_2.

Note that the introduced model is a simplification of the realistic ITCS. Particularly, we assume that the workload of the whole ITCS (including both crossings) is executed on a uniprocessor platform (the analysis center). In reality, the data preprocessing is

performed on a dedicated processor at each crossing and the preprocessed data are then uploaded to the analysis center.

6. DISCUSSIONS

6.1. Fixed-Priority Scheduling

In this article, we focus on the EDF schedulability analysis of the FJRT models, which is based on the demand bound functions (DBF). The insight of this article can also be used for the analysis of fixed-priority scheduling. The major difference is that, while the analysis of EDF is based on DBF, the analysis of fixed-priority scheduling is based on request bound functions (RBF) [Stigge and Yi 2013]. RBF is defined in a similar way as DBF, but includes the released workload immediately (instead of at the deadline of the workload as in DBF). The techniques developed for DBF in this article can be very well reused to calculate RBF, and thus support schedulability analysis of fixed-priority scheduling. However, it should be noted that fixed-priority scheduling is strongly coNP-hard for even the most simple graph-based task models, and the RBF-based analysis is inherently overapproximated with graph-based task models.

Moreover, we prove that the FJRT model is untractable in most general cases. However, we also show that a restricted form of the FJRT model based on hierarchy digraphs, called FJRT-HD, permits a tractability of the feasibility problem. In this article, the feasibility problem of the FJRT-HD model is solved through a transformation from FJRT-HD to DRT, and then the schedulability test method for DRT can be directly reused. Note that the FJRT-HD can be equivalently cast to a DRT, and the schedulability test for DRT proposed by Stigge et al. [2011] is an exact one. Therefore, the schedulability test approach for FJRT-HD proposed in this article is exact, which can produce the exact bound for the demand of an FJRT-HD model and can exactly determine whether a given FJRT-DH task system is schedulable.

6.2. Multiprocessor Scheduling

Multiprocessor platforms are more suitable to explore the parallelism of FJRT task systems. The major effort in the analysis is to calculate the DBFs of the graphs, and the uniprocessor EDF schedulability analysis can be viewed as a byproduct of the DBF calculation. Existing analysis techniques of multiprocessor EDF scheduling (for both simple sporadic tasks and advanced parallel tasks) are also based on DBFs [Saifullah et al. 2011; Lakshmanan et al. 2010; Baruah 2014; Baruah and Baker 2008; Baruah and Fisher 2005]. Therefore, the analysis techniques of this article can be viewed as a starting point toward the analysis of the FJRT model on multiprocessor platforms.

7. CONCLUSIONS

In this article, we have investigated a new DRT-based fork-join task model that allows high expressiveness in the modeling of conditional programming codes and intratask parallelism in task-level parallel embedded systems. The new model overcomes the restrictions of the previous models, which mostly focused on the sequential programming code. This article first investigated the most general FJRT model, previously introduced by Stigge et al. [2013]. The feasibility problem of this model has been stated in Stigge et al. [2013] as an open problem. We constructed a reduction from 3SAT to show that this problem is coNP-hard in the strong sense even if the system utilization is bounded by a constant that is strictly less than 1. Moreover, we also studied the FJRT model based on the hierarchy digraph and showed that the corresponding feasibility can be determined within pseudo-polynomial time. Thus, we presented a borderline between the intractable and tractable FJRT models.

REFERENCES

Anne Angermann. 2007. *Matlab-Simulink-Stateflow: Grundlagen, Toolboxen, Beispiele [mit CD-ROM]*. Oldenbourg Verlag.

Philip Axer, Sophie Quinton, Moritz Neukirchner, Rolf Ernst, Bjorn Dobel, and Hermann Hartig. 2013. Response-time analysis of parallel fork-join workloads with real-time constraints. In *2013 25th Euromicro Conference on Real-Time Systems (ECRTS'13)*. IEEE, 215–224.

Sanjoy Baruah. 2010a. The non-cyclic recurring real-time task model. In *2010 IEEE 31st Real-Time Systems Symposium (RTSS'10)*. IEEE, 173–182.

Sanjoy Baruah. 2010b. Preemptive uniprocessor scheduling of non-cyclic GMF task systems. In *2010 IEEE 16th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'10)*. IEEE, 195–202.

Sanjoy Baruah. 2014. Improved multiprocessor global schedulability analysis of sporadic DAG task systems. In *2014 26th Euromicro Conference on Real-Time Systems (ECRTS'14)*. IEEE, 97–105.

Sanjoy Baruah, Vincenzo Bonifaci, Alberto Marchetti-Spaccamela, Leen Stougie, and Andreas Wiese. 2012. A generalized parallel task model for recurrent real-time processes. In *2012 IEEE 33rd Real-Time Systems Symposium (RTSS'12)*. IEEE, 63–72.

Sanjoy Baruah, Deji Chen, Sergey Gorinsky, and Aloysius Mok. 1999. Generalized multiframe tasks. *Real-Time Systems* 17, 1, 5–22.

Sanjoy Baruah and Nathan Fisher. 2005. The partitioned multiprocessor scheduling of sporadic task systems. In *26th IEEE International Real-Time Systems Symposium, 2005 (RTSS'05).*. IEEE, 9–pp.

Sanjoy K. Baruah. 2003. Dynamic-and static-priority scheduling of recurring real-time tasks. *Real-Time Systems* 24, 1, 93–128.

Sanjoy K. Baruah and Theodore P. Baker. 2008. Global EDF schedulability analysis of arbitrary sporadic task systems. In *ECRTS*. 3–12.

Sanjoy K. Baruah, Aloysius K. Mok, and Louis E. Rosier. 1990. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Proceedings of the 11th Real-Time Systems Symposium, 1990.*IEEE, 182–190.

Hoon Sung Chwa, Jinkyu Lee, Kieu-My Phan, Arvind Easwaran, and Insik Shin. 2013. Global EDF schedulability analysis for synchronous parallel tasks on multicore platforms. In *2013 25th Euromicro Conference on Real-Time Systems (ECRTS'13)*. IEEE, 24–34.

Stephen A. Cook. 1971. The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing*. ACM, New York, NY, 151–158.

Daniel Ejsing-Duun, Lisa Fontani, Jonas Finnemann Jensen, Jacob Haubach, and Lars Kærlund Østergaard Smedegård. 2013. The concurrent real-time task model. Aalborg University, Denmark, Tech. Rep, 2014, 1–35. Retrieved December 17, 2015 from http://www.lets.dk/downloads/CRT.pdf.

David Ferry, Jing Li, Mahesh Mahadevan, Kunal Agrawal, Christopher Gill, and Chenyang Lu. 2013. A real-time scheduling service for parallel tasks. In *2013 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS'13)*. IEEE, 261–272.

Aloysius Ka and Lau Mok. 1983. *Fundamental design problems of distributed systems for the hard-real-time environment*. Vol. 1. MIT Thesis, Cambridge, MA.

Shinpei Kato and Yutaka Ishikawa. 2009. Gang EDF scheduling of parallel task systems. In *30th IEEE Real-Time Systems Symposium, 2009 (RTSS'09).* IEEE, 459–468.

Karthik Lakshmanan, Shinpei Kato, and Ragunathan Rajkumar. 2010. Scheduling parallel real-time tasks on multi-core processors. In *2010 IEEE 31st Real-Time Systems Symposium (RTSS'10)*. IEEE, 259–268.

Chung Laung Liu and James W. Layland. 1973. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM* 20, 1, 46–61.

Aloysius K. Mok and Deji Chen. 1997. A multiframe model for real-time tasks. *IEEE Transactions on Software Engineering* 23, 10, 635–645.

Geoffrey Nelissen, Vandy Berten, Joël Goossens, and Dragomir Milojevic. 2012. Techniques optimizing the number of processors to schedule multi-threaded tasks. In *2012 24th Euromicro Conference on Real-Time Systems (ECRTS'12)*. IEEE, 321–330.

Abusayeed Saifullah, Kunal Agrawal, Chenyang Lu, and Christopher Gill. 2011. Multi-core real-time scheduling for generalized parallel task models. In *2011 IEEE 32nd Real-Time Systems Symposium (RTSS'11)*. IEEE, 217–226.

Abusayeed Saifullah, David Ferry, Kunal Agrawal, Chenyang Lu, and Christopher Gill. 2012. Real-time scheduling of parallel tasks under general DAG model. Washington University in St Louis, USA, Tech. Rep. WUCSE-2012-14, 2012.

Martin Stigge, Pontus Ekberg, Nan Guan, and Wang Yi. 2011. The digraph real-time task model. In *2011 17th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'11)*. IEEE, 71–80.

Martin Stigge, Pontus Ekberg, and Wang Yi. 2013. The fork-join real-time task model. *ACM SIGBED Review* 10, 2, 20–20.

Martin Stigge and Wang Yi. 2013. Combinatorial abstraction refinement for feasibility analysis. In *2013 IEEE 34th Real-Time Systems Symposium (RTSS'13)*. IEEE, 340–349.

Received September 2014; revised April 2015; accepted July 2015