The Concurrent Real-Time Task Model

Daniel Ejsing-Duun, Lisa Fontani, Jonas Finnemann Jensen, Jacob Haubach Smedegård, Lars Kærlund Østergaard

Department of Computer Science, Aalborg University, Denmark

Abstract. The Digraph Real-time Task model (DRT) was recently introduced as a new modeling formalism for uniprocessor schedulability analysis of real-time systems with sporadic jobs. We extend the syntactical expressiveness of the DRT model by introducing the Concurrent Real-time Task model (CRT), where parallel constructions are allowed within tasks. We show how the CRT model can be expressed through a grammar and then demonstrate how the concept of the demand bound function and utilization may be adapted to this formalism. Lastly, we argue that the pseudo-polynomial feasibility results for the DRT model also apply to this model and show how the demand bound function and utilization may be computed by the means of path abstraction.

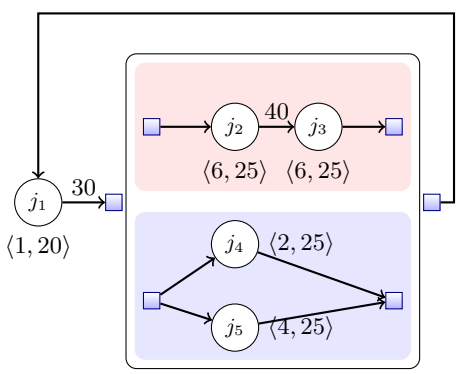
1 Introduction

Real-time systems are widely used in time critical application areas, such as embedded automative electronics, air traffic control systems, networked systems, etc. Such systems run various tasks that collect information and create events that must be handled in a timely manner. Hence, the response times of these systems must be guaranteed within strict time constraints. Every such task requires computation time and data resources to complete, however, with a limited amount of resources it becomes a challenge to order the tasks in such a way that each one satisfies its timing constraints.

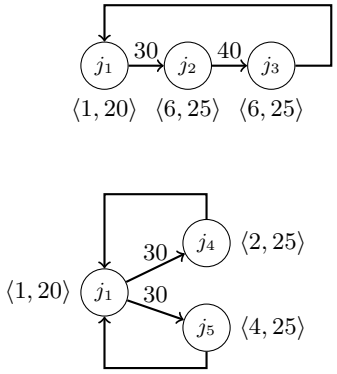
Scheduling of real-time systems has been a longstanding research area on its own. Within this area, a number of modeling formalisms have been developed to model and analyze real-time systems. Initially the main focus was on periodic task systems, where in 1973 Liu and Layland [6] formulated a necessary and sufficient criterion for uniprocessor feasibility and a utilization bound that required knowledge of execution times and unique periods of each task. Later these results were improved in other works that accounted for sporadic tasks [9] and moved onto multi-processor platforms [8], both of which resulted in an increase in the complexity from linear time to pseudo-polynomial. Later, task automata [3] have been proposed, allowing a higher level of expressiveness than the previous models at the price of higher complexity and undecidability in some cases.

In 2011 the Digraph Real-time Task model (DRT) was introduced by Stigge, Ekberg, Guan and Wang [12] for determining the feasibility of uniprocessor systems. This model is used for the modeling of sporadic tasks and offers a high

degree of expressiveness, while maintaining pseudo-polynomial complexity. The DRT modeling formalism uses a simplified set of assumptions to construct an abstract model of a system, while keeping the schedulability test tractable in practice. It requires that a program can be modeled as one or more independent tasks running concurrently. Each task can be modeled with a dependency graph of jobs containing a control flow of either serial or nondeterministic compositions. The model abstracts away from specific functionality of the jobs of each task and assumes that they adhere to a specific set of properties such as limited support for shared resources. A job is then an abstraction over a piece of code, consisting of a worst-case execution time and a deadline. The formal definition of the model is given in Appendix B.1.



(a) Example of a Concurrent Real-time Task system consisting of only one task. Nodes in the graph represent jobs, each labelled with an execution time and deadline. The edges in the graph represent dependencies between jobs and are labelled with minimum inter-release time where applicable.



(b) Example of a DRT system with two tasks. Nodes in the graph represent jobs, each labelled with an execution time and deadline. The edges in the graph represent dependencies between jobs and are labelled with minimum inter-release time where applicable.

Fig. 1: CRT and DRT models for the environmental sensor system.

We extend the DRT model by introducing the Concurrent Real-time Task model (CRT), which allows parallel job compositions within tasks that can be used to readily model concurrent systems with explicit synchronization on a uniprocessor. The motivation for this extension can be demonstrated through the following example, shown as a graph in Fig. 1a. Consider an environmental sensor which is activated for high and low temperature values. When a value within a specific range is observed, some preprocessing is done to determine the circumstances (j_1), after which, in parallel, data is sent to an off-site data store (j_2), leading to an acknowledgement of receival (j_3), while a cooling (j_4) or heating (j_5) device is activated. In addition to the constructions allowed in

the DRT model, this model allows parallel job composition and is therefore at least as expressive as the DRT, as shown in Appendix B.2.

In the original DRT model we cannot model parallel jobs within a task. Consequently, it is necessary to either duplicate the initialization code into two tasks, as shown in Fig. 1b, or create a third parallel task that contains the initialization code. However, both could cause an otherwise feasible system to no longer be feasible, if the initialization code is not able to run in parallel with the other tasks. By using our proposed parallel job construction we can model these three inter-dependent tasks as a single task that correctly encodes their ordering.

1.1 Related work

There are two main branches of focus in the schedulability reasearch: uniprocessor systems and multiprocessor systems. Liu and Leyland [6] introduced a periodic model for the feasibility analysis of uniprocessor real-time systems in 1973. As previously mentioned this work has since been extended to feasibility analysis of sporadic task models [9] and the DRT model.

Stigge, Ekberg, Guan and Wang have shown that the feasibility problem for an extension of the DRT model, EDRT [11], which contains inter-release time constraints between non-connected jobs in a task, remains pseudo-polynomial for a bounded number of constraints and is strongly coNP-hard for an arbitrary number of constraints.

The other branch of interest is that of multiprocessor scheduling. Algorithms for multiprocessor scheduling are divided into two approaches; partitioned and global. The partitioned algorithms assign tasks or jobs to a fixed processor, whereas global algorithms allow arbitrary migration of tasks or jobs between the processors. The general preemptive task allocation problem for multiprocessors is known to be NP-complete [4,2]. Moreover, no priority-driven scheduling algorithm, such as Earliest Deadline First (EDF), is known to be optimal for scheduling tasks on multiprocessors [7].

Our focus is strictly that of uniprocessor scheduling and hence we will not consider multiprocessors for the remainder of this paper.

Outline: This paper is structured as follows. The next section covers background theory such as jobs, deadlines, worst case execution time and the concept of feasibility. Section 3 presents the Concurrent Real-time Task (CRT) model with Section 4 covering feasibility analysis for CRT models. In Section 5 we present rules for calculating the utilization of CRT systems. Section 6 discusses the existence of an upper bound for the time for which schedule violations may occur in a system. The calculation of the demand bound function for CRT models is demonstrated in Section 7. Finally, Section 8 provides conclusions and Section 9 gives an overview of future work directions.

2 Background

In this paper we consider the problem of scheduling jobs in a real-time setting. A job is defined as a segment of code and is the smallest unit of a program that we shall consider. Every job has an associated execution time and deadline, and may be dependent on other jobs or events in a system. A job can only be released after all its dependencies have been fulfilled. A job that has been released is ready to begin execution and is waiting to be scheduled.

In this section we shall introduce basic concepts relating to jobs, present well-established categories into which jobs can be classified and explain the scheduling principle that the CRT model relies on.

2.1 Execution Time of a Job

The *execution time* of a job j, denoted by $e(j)$, is the amount of time that the job requires to complete its work on a single processor. The execution time may typically vary, depending on the architecture the program is running on, input data, current state of the cache, data locality, etc. Therefore it is impossible to accurately estimate the actual time any job will need to finish executing. Nevertheless, it is possible to approximate the execution time of a job by using either the best- or worst-case execution time (BCET/WCET)[7].

BCET is the minimum execution time required by a job, while WCET is the maximum execution time a job may need. Values for these two metrics are obtained through analysis, since through testing the actual best- or worst-case may never be observed. It is difficult to establish an accurate estimate of either BCET or WCET, since this requires intricate knowledge of the hardware and architecture used. Therefore, in practice, the values found are an under- and over-estimation, respectively.

2.2 Deadline of a Job

Since we are concerned with real-time systems, every job j can, besides an execution time, also have an associated *deadline*, $d(j)$. This deadline is relative to the release of the job and states how much time is allowed to pass after its release, before it must be finished.

Depending on the system this deadline can either be a hard or a soft deadline. The difference between these two types of deadlines refers to how useful the result is in case the deadline is violated. In case of a hard deadline, the execution has no value after the deadline, while the result of a job with a soft deadline still has some value depending on how late it is [7], i.e. if a few jobs with soft deadlines do not complete within their time constraints it is not critical to the overall performance or integrity of the system.

2.3 Minimum Inter-Release Time

A job may be dependent on other jobs in a system. Any two related jobs may have a minimum inter-release time of $x \in \mathbb{N}_{\geq 0}$ between them. This means that at least x time units must pass between the release of the first and the second job. Aside from this we have no way of predicting when the actual release of the job will occur.

Jobs with inter-release times between them can be organized into *tasks*. In turn, these independent tasks can then be grouped to form a *system*.

2.4 Categories of Jobs

In real-time scheduling there are generally three different types of jobs [7]: *periodic* jobs, *aperiodic* jobs and *sporadic* jobs. The difference lies in what can be asserted about their release times and deadlines.

Periodic jobs are continuously released exactly after a certain time interval since the last release has passed and have no dependencies to other jobs in a system. The deadlines associated with periodic jobs may be both hard or soft.

Aperiodic jobs, on the other hand, do not have a precise time for when they must be released based on previous releases of the job. An aperiodic job can be dependent on the release of other jobs. Minimum inter-release times are therefore used to reflect these dependencies in a task model. Aperiodic jobs may have either soft deadlines or no deadline. Thus, aperiodic jobs describe jobs that are not released at specific times and are not critical for the system.

Sporadic jobs have no fixed release time either and use minimum inter-release times to express dependencies. Sporadic jobs usually have hard deadlines and thereby may represent time critical jobs that are released by events.

2.5 Feasibility

A task system can produce different *release scenarios* depending on the tasks present in the system, the jobs in each task and the dependencies between these. A release scenario contains all the jobs released by a system with their associated release times, execution times and absolute deadlines.

A task system is said to be *uniprocessor feasible* if all release scenarios generated by the system can be executed on a preemptive uniprocessor platform such that no job misses its deadline.

When testing for feasibility we only use the WCET, since, if a set of jobs is feasible with respect to their WCET, the schedule for its worst-case execution will surely also be a safe schedule for all other executions, if all jobs are released at the same time, but execute for shorter periods.

The challenge when using WCET for feasibility analysis, aside from finding a realistic estimate, is that the execution time of the jobs in reality may be much smaller, meaning that a schedule deemed infeasible by a WCET feasibility analysis might actually be able to execute and finish without any deadline violation. The best way to minimize this problem is to optimize the WCET analysis.

2.6 Earliest Deadline First Scheduling

There are several well-known scheduling strategies that work under different assumptions and have a varying performance, depending on the type of system. One of these strategies is called *Earliest Deadline First* (EDF) scheduling, which assumes preemptable jobs. The principle of EDF is to always allow the released job with the smallest absolute deadline to execute on the processor. This means that any new job can preempt the current job from running and that the processor will never be idle while there are any released jobs. Since jobs are assumed to be preemptable, challenges may arise if two jobs under execution need to access the same resource. Other strategies and variations of EDF take this aspect into consideration.

Previous research has shown that EDF is optimal for a uniprocessor platform where preemption is allowed and context-switching is assumed to take negligible time [7]. A scheduling algorithm is *optimal* if, for any release scenario of a feasible system, it always produces a schedule such that all jobs meet their deadline. Because of this, we can rely on EDF as scheduling strategy whenever we find a uniprocessor feasible task system.

In this paper we limit our scope to scheduling sporadic jobs with hard deadlines, and focus on determining whether a task system is feasible. In the next section we shall introduce the Concurrent Real-time Task model, including its syntax and semantics.

3 Concurrent Real-Time Task Model

The Concurrent Real-time Task model (CRT), which is an extension of the DRT model is, likewise, composed of arbitrarily many independent tasks running in parallel. Each task is a set of related jobs that encodes the possible control flow of a program. Nondeterministic choices between different execution paths or subtasks running in parallel can be expressed. Note that subtasks are restricted to not include any cycles.

Definition 1 (Concurrent Real-Time Task System). *A Concurrent Real-time Task system (CRT) $\mathcal{S}$ is a tuple $\mathcal{S} = (\tau, J, e, d)$, 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.*

$$\begin{aligned} \textit{Configuration:} \quad & \tau ::= T \mid T \parallel \tau \\ \textit{Task:} \quad & T ::= j \mid T_1 \langle x \rangle T_2 \mid S_1 \parallel S_2 \mid T_1 + T_2 \mid T^{\omega} \\ \textit{Subtask:} \quad & S ::= j \mid S_1 \langle x \rangle S_2 \mid S_1 \parallel S_2 \mid S_1 + S_2 \end{aligned}$$

Here $j \in J$ is a job and $x \in \mathbb{N}$.
We consider only well-formed configurations τ, where any job $j \in J$ occurs at most once. Moreover, CRT_τ, CRT_T and CRT_S denote the infinite sets containing all legal configurations, tasks and subtasks, respectively.

Here $T_1\langle x\rangle T_2$ is the sequential composition of tasks T_1 and T_2 with inter-release time x, $T_1 + T_2$ is the choice between tasks T_1 and T_2, $S_1 \parallel S_2$ is the parallel composition of subtasks S_1 and S_2, and T^ω is one or more iterations of T.

Example 1 The model represented in Fig. 1a can be expressed as the configuration τ_e consisting of the task $T_e = (j_1\langle 30\rangle((j_2\langle 40\rangle j_3) \parallel (j_4 + j_5)))^\omega$. The CRT system is then given by $\mathcal{S} = (\tau_e, J_e, e, d)$, where

$$J_e = \{j_1, j_2, j_3, j_4, j_5\}$$

$e(j_1) = 1$	$e(j_2) = 4$	$e(j_3) = 6$	$e(j_4) = 2$	$e(j_5) = 4$
$d(j_1) = 20$	$d(j_2) = 25$	$d(j_3) = 25$	$d(j_4) = 25$	$d(j_5) = 25$

For any CRT, the sequences of jobs that may be generated can be expressed through the notion of *execution paths*. An execution path carries information about the inter-release time between jobs and may contain parallel compositions. It does, however, not contain iterations nor choices, as these are replaced through system unfolding.

Definition 2 (Execution Path). *An execution path is a finite sequence of jobs that may be generated by a CRT. It is defined by the following grammar.*

$$p ::= j \mid p_1 \parallel p_2 \mid p_1\langle x\rangle p_2$$

Given a CRT $\mathcal{S} = (\tau, J, e, d)$, the (possibly infinite) set of execution paths through the whole expression τ is denoted $Paths(\tau)$ and defined in the rules below.

$$Paths(j) = \{j\} \tag{1}$$
$$Paths(T_1\langle x\rangle T_2) = \{p_1\langle x\rangle p_2 \mid p_1 \in Paths(T_1), p_2 \in Paths(T_2)\} \tag{2}$$
$$Paths(T_1 \parallel T_2) = \{p_1 \parallel p_2 \mid p_1 \in Paths(T_1), p_2 \in Paths(T_2)\} \tag{3}$$
$$Paths(T_1 + T_2) = Paths(T_1) \cup Paths(T_2) \tag{4}$$
$$Paths(T^\omega) = Paths(T) \cup \{p_1\langle 0\rangle p_2 \mid p_1 \in Paths(T), p_2 \in Paths(T^\omega)\} \tag{5}$$

The base case of an execution path through a job is just the job itself (Equation 1). In a sequential composition an execution path through the first task is followed by the inter-release time and an execution path through the second task (Equation 2). The execution path that goes through a parallel composition is given by the parallel composition of an execution path for the first subtask and an execution path for the second subtask (Equation 3). For any choice in T the execution path may go through one of the components, thus removing nondeterminism (Equation 4). When a T^ω subterm is present, an execution path may repeat T one or more times (Equation 5).

An execution may not be limited to execution paths through the entire task. For this reason we must be able to refer to all kinds of execution paths in the system, rather than limiting us to complete paths. Therefore, we will now define the prefix and suffix of paths.

The prefix of an execution path p is the set including any execution path that starts where p starts, but may end with any job of p. To construct these we define the prefix operation for execution paths as follows.

$$\begin{aligned} \mathit{prefix}(j) &= \{j\} \\ \mathit{prefix}(p_1 \langle x \rangle p_2) &= \mathit{prefix}(p_1) \cup \{p_1 \langle x \rangle p_2^f \mid p_2^f \in \mathit{prefix}(p_2)\} \qquad (6) \\ \mathit{prefix}(p_1 \mid\mid p_2) &= \{p_1^f \mid\mid p_2^f \mid p_1^f \in \mathit{prefix}(p_1), p_2^f \in \mathit{prefix}(p_2)\} \qquad (7) \\ &\quad \cup \mathit{prefix}(p_1) \cup \mathit{prefix}(p_2) \end{aligned}$$

To define the prefix of an execution path we note that the prefix of a sequential composition (Equation 6) may be composed of a prefix of the execution path up until the inter-release term of the expression, which consequently is not included in the resulting execution path. Alternatively, we may need to represent the entire execution path that occurs before the inter-release term and concatenate it with a prefix of the execution path that follows the inter-release time. In a parallel composition (Equation 7) there is no relation between the parallel branches and as such the prefix of a parallel composition can be either the prefix of either of the branches or the parallel composition of any prefix of any of the execution paths in both branches.

The suffix of an execution path p is the set including any execution path that ends where p does, but may start with any job of p. To construct these we define the suffix operation on execution paths as follows.

$$\begin{aligned} \mathit{suffix}(j) &= \{j\} \\ \mathit{suffix}(p_1 \langle x \rangle p_2) &= \mathit{suffix}(p_2) \cup \{p_1^f \langle x \rangle p_2 \mid p_1^f \in \mathit{suffix}(p_1)\} \\ \mathit{suffix}(p_1 \mid\mid p_2) &= \{p_1^f \mid\mid p_2^f \mid p_1^f \in \mathit{suffix}(p_1), p_2^f \in \mathit{suffix}(p_2)\} \\ &\quad \cup \mathit{suffix}(p_1) \cup \mathit{suffix}(p_2) \end{aligned}$$

Symmetrical arguments to the ones for the prefix operation hold for the suffix operation.

For a configuration τ we define $\mathit{AllPaths}(\tau)$ to be the set containing the prefixes of all suffixes of all execution paths going through the whole system. Hence, this set includes all possible execution paths for τ.

$$\mathit{AllPaths}(\tau) = \{p \mid p \in \mathit{prefix}(p'), p' \in \mathit{suffix}(p''), p'' \in \mathit{Paths}(\tau)\}$$

Example 2 Here we present examples of execution paths for the task T_e in Fig. 1a, previously described in Example 1.

An example of a path through T_e is given by:
$p_1 = j_1\langle 30\rangle((j_2\langle 40\rangle j_3) \mid\mid (j_5))\langle 0\rangle j_1\langle 30\rangle((j_2\langle 40\rangle j_3) \mid\mid (j_4))$.

Some of the possible prefixes of p_1 are:
$j_1\langle 30\rangle j_2,\ \ j_1\langle 30\rangle((j_2\langle 40\rangle j_3) \mid\mid (j_5)),\ j_1\langle 30\rangle((j_2\langle 40\rangle j_3) \mid\mid (j_5))\langle 0\rangle j_1\langle 30\rangle j_4,\ \ j_1$.

Some of the possible suffixes of p_1 are:
$j_2\langle 40\rangle j_3,\ \ j_4,\ j_5\langle 0\rangle j_1\langle 30\rangle((j_2\langle 40\rangle j_3) \mid\mid (j_4)),\ j_3$.

Finally, we can consider an execution path inside T_e that is not included in any of the other categories. For instance, the execution path consisting of a single job $p = j_2$.

3.1 Semantics

The semantics of a CRT $\mathcal{S} = (\tau, J, e, d)$ describes how jobs can be released in a system. A job release of $j \in J$ is described by a release tuple $\langle r, e, d\rangle$, where r denotes the absolute time at which the job was released, $e = e(j)$ is the execution time of the job and $d = r + d(j)$ is the absolute deadline for the job. We define an auxiliary function, which given an offset value y, recomputes release times and deadlines of release tuples for a release scenario σ.

$$\mathit{offset}(\sigma, y) = \{\langle r + y, e, d + y\rangle \mid \langle r, e, d\rangle \in \sigma\}$$

With this we can now define the collection of job releases for a given execution path.

Definition 3 (Release Scenario). *Let $\mathcal{S} = (\tau, J, e, d)$ be a CRT. A release scenario σ of an execution path p is a multiset of release tuples for the jobs in p, where either of the following holds*

a) $p = j \quad \textit{and} \quad \sigma = \{\langle c, e(j), d(j) + c\rangle\}$ *for some* $c \in \mathbb{R}_{\geq 0}$

If σ_1 is a release scenario for p_1 and σ_2 is a release scenario for p_2, then

b) $p = p_1\langle x\rangle p_2 \quad \textit{and} \quad \sigma = \sigma_1 \uplus \mathit{offset}(\sigma_2, x + \max\{r \mid \langle r, e, d\rangle \in \sigma_1\})$
c) $p = p_1 \mid\mid p_2 \quad \textit{and} \quad \sigma = \sigma_1 \uplus \sigma_2$

Here $\uplus$ is the multiset union [13] and c denotes an arbitrary delay on the release of a job.

We define the collection of release scenarios below.

Definition 4 (Set of Release Scenarios). *Given an execution path p, we define $RS(p)$ as the infinite set of release scenarios for p.*

Note that when a parallel execution of a task is finished, the definition requires that all the jobs within the parallel composition, excluding those eliminated by choices, have been released before the next job in the task can be released.
Also, it is defined that for an execution path p and any release scenario $\sigma \in RS(p)$, it holds that σ contains exactly one release tuple for every occurrence of a job in p.

Example 3 A subset of the release scenarios for the execution path $p = j_1\langle 30\rangle j_2$ of the model in Fig. 1a is given by:

$$RS(p) = \Big\{\{\langle 0, 1, 20\rangle, \langle 30, 4, 55\rangle\}, \{\langle 2.5, 1, 22.5\rangle, \langle 37.3, 4, 62.3\rangle\}, \dots\Big\}$$

Given a release scenario $\sigma \in RS(p)$ for some execution path p, the execution time of σ, $E(\sigma)$, denotes the total execution time required by job releases in σ. It is given by the sum of the execution times of all release tuples in the multiset (Equation 8). The inter-release time of σ, $I(\sigma)$, denotes the time that passes between the first release and the last release in the multiset (Equation 9). Lastly, the deadline of σ, $D(\sigma)$, denotes the time by which all jobs in σ must be completed. In other words, it is the latest deadline of any job in σ (Equation 10).

$$E(\sigma) = \sum_{\langle r,e,d\rangle \in \sigma} e \tag{8}$$

$$I(\sigma) = \max\{r \mid \langle r, e, d\rangle \in \sigma\} - \min\{r \mid \langle r, e, d\rangle \in \sigma\} \tag{9}$$

$$D(\sigma) = \max\{d \mid \langle r, e, d\rangle \in \sigma\} \tag{10}$$

Let $\mathcal{S} = (\tau, J, e, d)$ be a CRT and $p \in \mathit{AllPaths}(\tau)$ be an execution path of τ. We can calculate the execution time, tight lower bounds for the inter-release time and deadline of any release scenario $\sigma \in RS(p)$, by calculating these for the execution path p. The execution time of p, $E(p)$, is computed by the following rules.

$$E(j) = e(j) \tag{11}$$

$$E(p_1\langle x\rangle p_2) = E(p_1) + E(p_2) \tag{12}$$

$$E(p_1 \mid\mid p_2) = E(p_1) + E(p_2) \tag{13}$$

Likewise, the inter-release time of execution path p, denoted $I(p)$, is calculated as follows.

$$I(j) = 0 \tag{14}$$
$$I(p_1\langle x\rangle p_2) = I(p_1) + x + I(p_2) \tag{15}$$
$$I(p_1 \mid\mid p_2) = \max\{I(p_1), I(p_2)\} \tag{16}$$

Finally, we compute the deadline of execution path p, which is the earliest deadline for any release scenario $\sigma \in RS(p)$. We denote this by $D(p)$ and compute it as follows.

$$D(j) = d(j) \tag{17}$$
$$D(p_1\langle x\rangle p_2) = \max\{D(p_1), I(p_1) + x + D(p_2)\} \tag{18}$$
$$D(p_1 \mid\mid p_2) = \max\{D(p_1), D(p_2)\} \tag{19}$$

Theorem 1. *Given an execution path p, then for any release scenario $\sigma \in RS(p)$ it holds that*

a) $E(p) = E(\sigma)$,
b) $I(p) \leq I(\sigma)$,
c) $D(p) \leq D(\sigma)$.

Also, there exists a release scenario $\sigma' \in RS(p)$, s.t. $I(p) = I(\sigma')$ and $D(p) = D(\sigma')$.

Proof. See Appendix A. □

To define whether a task can be scheduled on a preemptive uniprocessor, we state the requirements for feasibility of a CRT.

Definition 5 (Feasibility). *A CRT $\mathcal{S} = (\tau, J, e, d)$ is preemptive uniprocessor feasible if for all $p \in AllPaths(\tau)$ and for all $\sigma \in RS(p)$, σ is uniprocessor schedulable, s.t. no job misses its deadline.*

4 CRT Feasibility Analysis

In this section we explain the concept of the demand bound function and the utilization, which we will use to determine feasibility for a CRT system.

4.1 Demand Bound Function

The demand bound function (DBF) corresponds to the maximum execution demand that can be requested and must complete in any time interval of length t.

Definition 6 (Demand Bound Function). *Given a CRT $\mathcal{S} = (\tau, J, e, d)$ and an interval of length t, the demand bound function $dbf(\mathcal{S}, t)$ is the maximum execution demand of $\mathcal{S}$ in any time interval of length t.*

$$dbf(\mathcal{S}, t) = \max\{E(\sigma) \mid \sigma \in RS(p), p \in AllPaths(\tau), D(\sigma) \leq t\}$$

Note that since $E(\sigma) \in \mathbb{N}$ and because only a finite number of paths with different execution demands can yield release scenarios satisfying the condition $D(\sigma) \leq t$, the set $\{E(\sigma) \mid \sigma \in RS(p), p \in AllPaths(\tau), D(\sigma) \leq t\}$ is finite. Using Theorem 1 we can also define $dbf(\mathcal{S}, t)$ as follows.

$$dbf(\mathcal{S}, t) = \max\{E(p) \mid p \in AllPaths(\tau), D(p) \leq t\}$$

For details on how to compute the demand bound function, see Section 7.

Proposition 1. *A CRT $\mathcal{S} = (\tau, J, e, d)$ is infeasible if and only if for some t_f it holds that $dbf(\mathcal{S}, t_f) > t_f$.*

Proof. This result has been shown in [1] for all systems of independent tasks. The intuition is that if the execution demand is less than the length of the interval for any interval, then the execution demand can be satisfied. Conversely, if a CRT is infeasible, there must be an interval for which it cannot satisfy the execution demand. □

4.2 Utilization

The utilization of a CRT is the maximum amount of execution time the system can demand for each time unit in an infinite execution path. In other words, it is the growth of the demand bound function as time goes towards infinity.

Definition 7 (Utilization). *Given a CRT $\mathcal{S} = (\tau, J, e, d)$, the utilization $U(\mathcal{S})$ is the maximum execution demand of any path with a deadline of at most the length of the time interval, divided by the same length as time goes towards infinity.*

$$U(\mathcal{S}) = \lim_{t \to \infty} \frac{dbf(\mathcal{S}, t)}{t}$$

A utilization greater than 1 implies infeasibility, since jobs will begin to accumulate and at some point a job will miss its deadline.

Proposition 2. *A CRT $\mathcal{S}$ is infeasible if $U(\mathcal{S}) > 1$.*

Proof. From Definition 7, $U(\mathcal{S}) > 1$ implies the existence of some t, s.t. $\frac{dbf(\mathcal{S},t)}{t} > 1$. Rewriting this yields $dbf(\mathcal{S}, t) > t$, which by Proposition 1 implies infeasibility. □

We now give an inductive definition of utilization.

$$U_{\mathcal{S}}(j) = 0 \tag{20}$$
$$U_{\mathcal{S}}(T_1 \langle x \rangle T_2) = \max\{U_{\mathcal{S}}(T_1), U_{\mathcal{S}}(T_2)\} \tag{21}$$
$$U_{\mathcal{S}}(T_1 \mid\mid T_2) = U_{\mathcal{S}}(T_1) + U_{\mathcal{S}}(T_2) \tag{22}$$
$$U_{\mathcal{S}}(T_1 + T_2) = \max\{U_{\mathcal{S}}(T_1), U_{\mathcal{S}}(T_2)\} \tag{23}$$
$$U_{\mathcal{S}}(T^{\omega}) = \max\left\{U_{\mathcal{S}}(T), \sup\left\{\frac{E(p)}{I(p)} \mid p \in \mathit{Paths}(T)\right\}\right\} \tag{24}$$

Theorem 2. *Given a CRT $\mathcal{S} = (\tau, J, e, d)$, we have $U(\mathcal{S}) = U_{\mathcal{S}}(\tau)$.*

Proof. Given a CRT $\mathcal{S} = (\tau, J, e, d)$ we prove by structural induction on τ that $U(\mathcal{S}) = U_{\mathcal{S}}(\tau)$.

If $\tau = j$, the utilization is 0 (Equation 20), since no infinite path can exist in τ.

If $\tau = T_1 \langle x \rangle T_2$, then $U_{\mathcal{S}}(\tau) = \max\{U_{\mathcal{S}}(T_1), U_{\mathcal{S}}(T_2)\}$ (Equation 21). If we construct CRTs $\mathcal{S}_1 = (T_1, J, e, d)$ and $\mathcal{S}_2 = (T_2, J, e, d)$, then by the induction hypothesis we need only show $U(\mathcal{S}) = \max\{U(\mathcal{S}_1), U(\mathcal{S}_2)\}$. Sequential composition does not give rise to any infinite behaviour, unless it occurs in either T_1 or T_2. Thus, $U(\mathcal{S}) = \lim_{t\to\infty} \frac{\mathit{dbf}(\mathcal{S},t)}{t}$ is either

(i) infinite execution in $\mathcal{S}_1$ given by $\lim_{t\to\infty} \frac{\mathit{dbf}(\mathcal{S}_1,t)}{t} = U(\mathcal{S}_1)$ or
(ii) finite execution in $\mathcal{S}_1$ followed by infinite execution in $\mathcal{S}_2$, given by $\lim_{t\to\infty} \frac{\mathit{dbf}(\mathcal{S}_1,k)+\mathit{dbf}(\mathcal{S}_2,t)}{k+t}$, where k is a constant.

Note that the latter case (ii) can be written as $\lim_{t\to\infty} \frac{\mathit{dbf}(\mathcal{S}_2,t)}{t} = U(\mathcal{S}_2)$ since finite execution in $\mathcal{S}_1$ has no effect when t goes towards infinity.

If $\tau = T_1 \mid\mid T_2$ the utilization is $U_{\mathcal{S}}(T_1) + U_{\mathcal{S}}(T_2)$ (Equation 22). Again we can construct CRTs $\mathcal{S}_1 = (T_1, J, e, d)$ and $\mathcal{S}_2 = (T_2, J, e, d)$. By the induction hypothesis we only show $U(\mathcal{S}) = U(\mathcal{S}_1) + U(\mathcal{S}_2)$. Again, parallel composition does not give rise to any infinite behaviour, unless it occurs in either T_1 or T_2. However, infinite behaviour can continue in both T_1 and T_2 simultaneously. Thus we can rewrite $U(\mathcal{S})$ as follows.

$$\begin{aligned} U(\mathcal{S}) &= \lim_{t\to\infty} \frac{\mathit{dbf}(\mathcal{S},t)}{t} = \lim_{t\to\infty} \frac{\mathit{dbf}(\mathcal{S}_1,t) + \mathit{dbf}(\mathcal{S}_2,t)}{t} \\ &= \lim_{t\to\infty} \frac{\mathit{dbf}(\mathcal{S}_1,t)}{t} + \lim_{t\to\infty} \frac{\mathit{dbf}(\mathcal{S}_2,t)}{t} = U(\mathcal{S}_1) + U(\mathcal{S}_2) \end{aligned}$$

If $\tau = T_1 + T_2$ the utilization is $\max\{U_{\mathcal{S}}(T_1), U_{\mathcal{S}}(T_2)\}$ (Equation 23). Again we construct CRTs $\mathcal{S}_1 = (T_1, J, e, d)$ and $\mathcal{S}_2 = (T_2, J, e, d)$. By the induction hypothesis we have $U(\mathcal{S}_1) = U_{\mathcal{S}}(T_1)$ and $U(\mathcal{S}_2) = U_{\mathcal{S}}(T_2)$. Since choice composition does not give rise to any infinite behaviour, any such behaviour must take place in T_1 or T_2. Thus, $U(\mathcal{S}) = \max\{U(\mathcal{S}_1), U(\mathcal{S}_2)\}$.

If $\tau = T^\omega$ the utilization is the maximum of $U_{\mathcal{S}}(T)$ and $\frac{E(p)}{I(p)}$, where $p \in \mathit{Paths}(T)$ (Equation 24). Let $\mathcal{S}_1 = (T, J, e, d)$ be a CRT. Notice that any nested infinite behaviour in T is accounted for in $U_{\mathcal{S}}(T^\omega)$ with $U_{\mathcal{S}}(T)$, which by induction hypothesis is $U(T)$. This composition does, however, give rise to infinite behaviour, for instance an execution path $p \in \mathit{Paths}(T)$ can be repeated infinitely. Thus, $U(\mathcal{S}) \geq \frac{E(p)}{I(p)}$, for any $p \in \mathit{Paths}(T)$, as any path $p \in \mathit{Paths}(T)$ can be repeated infinitely.

An infinite path using T^ω may also consist of different execution paths through T, yet this will not give a higher value for $U(\tau)$. Consider the case where an infinite execution path p_{inf} using T^ω consists of the infinite repetition of $p_1, p_2 \in \mathit{Paths}(T)$, e.g. $p_{inf} = p_1\langle 0\rangle p_2\langle 0\rangle \ldots$. We now show by contradiction that $\frac{E(p_{inf})}{I(p_{inf})}$ will never be higher than $\frac{E(p_1)}{I(p_1)}$ and $\frac{E(p_2)}{I(p_2)}$. Thus, combinations of paths, such as it is done in p_{inf}, cannot yield a higher utilization. Assume $\frac{E(p_{inf})}{I(p_{inf})} = \frac{E(p_1)+E(p_2)}{I(p_1)+I(p_2)} > \max\left\{\frac{E(p_1)}{I(p_1)}, \frac{E(p_2)}{I(p_2)}\right\}$. We now write as follows.

$$
\begin{aligned}
&\frac{E(p_1)}{I(p_1)} < \frac{E(p_1) + E(p_2)}{I(p_1) + I(p_2)} \\
\Updownarrow& \\
&E(p_1) \cdot (I(p_1) + I(p_2)) < I(p_1) \cdot (E(p_1) + E(p_2)) \\
\Updownarrow& \\
&I(p_2) \cdot E(p_1) < I(p_1) \cdot E(p_2) \\
\Updownarrow& \\
&\frac{E(p_1)}{I(p_1)} < \frac{E(p_2)}{I(p_2)}
\end{aligned}
$$

The same can be done with $\frac{E(p_2)}{I(p_2)} < \frac{E(p_1)+E(p_2)}{I(p_1)+I(p_2)}$, yielding $\frac{E(p_2)}{I(p_2)} < \frac{E(p_1)}{I(p_1)}$. Thus we have a contradiction. This can be generalized to arbitrarily many execution paths. Thus, no infinite execution path consisting of combinations of different paths through T can yield a higher utilization than $\sup\left\{\frac{E(p)}{I(p)} \mid p \in \mathit{Paths}(T)\right\}$. □

5 Computation of Utilization

Because the set of execution paths $\mathit{Paths}(T)$ through a task T, used in Equation 24, may be infinite, the rules in Equations 20 to 24 cannot be used alone to compute the utilization.

In Equation 24 we say that the utilization of T^ω is the execution demand divided by the accumulated inter-release time for any execution path through T. However, by considering the utilization of T, it is enough to consider only execution paths going through subcycles in T once. Using the following rules we can compute the execution demand and inter-release time in one tuple for each path through a task T, where subcycles in T are not repeated.

$$ut(j) = \{\langle e(j), 0\rangle\} \tag{25}$$

$$ut(T_1\langle x\rangle T_2) = \{\langle e_1 + e_2, i_1 + i_2 + x\rangle \mid \langle e_1, i_1\rangle \in ut(T_1), \langle e_2, i_2\rangle \in ut(T_2)\} \tag{26}$$

$$ut(T_1 \mid\mid T_2) = \{\langle e_1 + e_2, \max\{i_1, i_2\}\rangle \mid \langle e_1, i_1\rangle \in ut(T_1), \langle e_2, i_2\rangle \in ut(T_2)\} \tag{27}$$

$$ut(T_1 + T_2) = ut(T_1) \cup ut(T_2) \tag{28}$$

$$ut(T^\omega) = ut(T) \tag{29}$$

The correctness of the rules for the computation of $ut(T)$ can be proven using structural induction. We will now see, how utilization tuples allow us to calculate the utilization of a CRT with a T^ω.

Lemma 1. *The utilization of T^ω, $U_{\mathcal{S}}(T^\omega)$, can be computed as* $\max\left\{U_{\mathcal{S}}(T), \max\left\{\frac{e}{i} \mid \langle e, i\rangle \in ut(T), i \neq 0\right\}\right\}$

Proof. The only case where $\frac{E(p)}{I(p)}$ can be greater than $\frac{e}{i}$, where $p \in Paths(T)$ and $\langle e, i\rangle \in ut(T)$, is if there is some subcycle in T, s.t. repeating this subcycle infinitely many times yields a higher utilization. It holds that $U_{\mathcal{S}}(T) \geq \frac{E(p)}{I(p)}$, where $p \in Paths(T)$, since the sequential composition of two cycles cannot yield a greater utilization, see proof of Theorem 2. Consequently, we need not consider the case of infinite paths in T having a greater utilization than $U_{\mathcal{S}}(T)$ or $\frac{e}{i}$, where $\langle e, i\rangle \in ut(T)$. □

Example 4 Recall the environmental sensor in Fig. 1a. A relevant question is now how many sensors we can run in parallel on a single processor. We know that if the utilization is greater than 1 then the system is not feasible so we can start by calculating the utilization for one task.

$$\begin{aligned} U_{\mathcal{S}}(T_e) = &\quad U_{\mathcal{S}}((j_1\langle 30\rangle((j_2\langle 40\rangle j_3) \mid\mid (j_4 + j_5)))^\omega) \\ = &\quad \max\{U_{\mathcal{S}}((j_1\langle 30\rangle((j_2\langle 40\rangle j_3) \mid\mid (j_4 + j_5)))), \\ &\qquad \frac{e}{i} \mid \langle e, i\rangle \in ut((j_1\langle 30\rangle((j_2\langle 40\rangle j_3) \mid\mid (j_4 + j_5)))^\omega)\} \end{aligned}$$

These can be calculated separately as:

$$\begin{aligned} ut(j_1) &= \{\langle 1, 0\rangle\} \\ ut(j_2\langle 40\rangle j_3) &= \{\langle 10, 40\rangle\} \\ ut(j_4 + j_5) &= ut(j_4) \cup ut(j_5) \\ &= \{\langle 2, 0\rangle, \langle 4, 0\rangle\} \\ ut((j_2\langle 40\rangle j_3) \mid\mid (j_4 + j_5)) &= \{\langle 12, 40\rangle, \langle 14, 40\rangle\} \\ ut(j_1\langle 30\rangle((j_2\langle 40\rangle j_3) \mid\mid (j_4 + j_5))) &= \{\langle 13, 70\rangle, \langle 15, 70\rangle\} \\ U_{\mathcal{S}}((j_1\langle 30\rangle((j_2\langle 40\rangle j_3) \mid\mid (j_4 + j_5)))) &= 0 \end{aligned}$$

Thus $U_{\mathcal{S}}(T_e) = \frac{15}{70} \approx 0.22$. This means that we can at most run four sensors on the same processor, as multiplying this value by more than 4 will yield a utilization greater than 1.

Now that we have established a method for calculating the utilization, we state the following complexity result.

Theorem 3. *Given a CRT $\mathcal{S} = (\tau, J, e, d)$ the utilization is computable in pseudo-polynomial time.*

Proof. Following Theorem 2, it holds that $U(\mathcal{S}) = U_{\mathcal{S}}(\tau)$, and given Lemma 1, $U_{\mathcal{S}}(\tau)$ is computable. Notice that the rules for $U_{\mathcal{S}}(\mathcal{S})$, with the exception of $U_{\mathcal{S}}(T^{\omega})$, are defined inductively on the structure of τ, so these can be evaluated in polynomial time. Using Lemma 1, $U_{\mathcal{S}}(T^{\omega})$ can be computed with $ut(T)$. Since $ut(T)$ is defined inductively on the structure of T, a subterm will never be processed more than once. This implies that $e \leq \sum_{j \in J} e(j)$ for all $\langle e, i \rangle \in ut(\tau)$, since the execution time of a job cannot be included in $\langle e, i \rangle$ more than once. The same argument applies to the occurrences of inter-release times, meaning that the amount of tuples, and hence the calculation of utilization, is pseudo-polynomial in the input $\mathcal{S}$. □

6 Upper Bound for Demand Bound Function

As previously mentioned, the utilization of a CRT is the maximum growth of the demand bound function for the CRT, as presented in [12]. The demand bound function of a CRT is defined for all time units, so if we want to use it to determine feasibility we need to check $dbf(\mathcal{S}, t) < t$ for all t (Proposition 1). Consequently, to determine feasibility, we must establish a bound for the DBF for which the feasibility test may be resolved in a finite number of steps. We now demonstrate how the utilization may be exploited to determine this bound for the demand bound function. First we observe that any path in τ may be expressed on the following form.

Lemma 2. *Given a CRT $\mathcal{S} = (\tau, J, e, d)$, where the configuration τ only consists of one task T ($\tau = T$), any path $p \in AllPaths(\tau)$ can be written on the form*

$$p_s^1 \langle x_1 \rangle p_c^1 \langle y_1 \rangle \dots \langle x_n \rangle p_s^n \langle y_n \rangle p_c^n$$

where no job occurs more than once in all p_s^i, for $0 < i \leq n$, and $p_c^i \in Paths(T_1^{\omega})$ for some subterm T_1^{ω} of T.

Proof. If a job occurs more than once in the execution path, it must be inside some subterm T_1^ω of T and by the structural restrictions of subtasks (Definition 1), a parallel composition cannot contain an iteration. So, for any parallel subterm $S_1 \parallel S_2$ of T, S_1 and S_2 do not contain any iterations. Thus, an execution path $p \in \mathit{AllPaths}(\tau)$ will never have the form $p_c^1 \parallel p_c^2 \ldots$, since all the parallel compositions in p can be in either p_s^i or p_c^i for some i, $0 < i \leq n$. □

With Lemma 2 we can now use the utilization to find an upper bound for the demand bound function. We first prove this for one CRT task.

Lemma 3. *For a CRT $\mathcal{S} = (\tau, J, e, d)$ and time interval length t, where the configuration τ only consists of one task T ($\tau = T$), the following holds.*

$$\mathit{dbf}(\mathcal{S}, t) \leq U(\mathcal{S}) \cdot t + \sum_{j \in J} e(j)$$

Proof. Given a time interval of length t, let p be an execution path s.t. $E(p) = \mathit{dbf}(\mathcal{S}, t)$ and $D(p) \leq t$. Note that from Definition 6, it follows that p is a path with the maximum execution demand in any time interval of length t.

To prove Lemma 3 we show that $E(p) \leq U(\mathcal{S}) \cdot t + \sum_{j \in J} e(j)$. Following Lemma 2, we can write p on the form $p = p_s^1 \langle x_1 \rangle p_c^1 \langle y_1 \rangle \ldots \langle x_n \rangle p_s^n \langle y_n \rangle p_c^n$, where for all i, $0 < i \leq n$, no job occurs more than once in p_s^i and $p_c^i \in \mathit{Paths}(T_1^\omega)$ for some T_1^ω construction in T. In other words, p_c^i for $0 < i \leq n$, are the subpaths in p that repeat a cycle T_1^ω one or more times.

As the order in which the execution time is summed does not matter, we can write $E(p)$ as Equation 30. We know that each job $j \in J$ can at most occur once in each of the subpaths p_s^1 to p_s^n, giving us Equation 31. From Equation 24, in the description of how to compute utilization, it follows that $U(\mathcal{S})$ is the maximum of $\frac{E(p')}{I(p')}$, for any $p' \in \mathit{Paths}(T_1)$, where T_1^ω is in T. Note that p_c^1 to p_c^n are all in $\mathit{Paths}(T_1^\omega)$ from some T_1^ω in T, so $U(\mathcal{S}) \geq \frac{E(p_c^i)}{I(p_c^i)}$ where $0 < i \leq n$. This allows us to write Equation 32. From the observation that $\sum_{i=1}^{n} I(p_c^i) \leq I(p)$, we get Equation 33. With the fact that $I(p) \leq D(p) \leq t$, we can now write Equation 34, which proves Lemma 3.

$$E(p) = \sum_{i=1}^{n} E\left(p_s^i\right) + \sum_{i=1}^{n} E\left(p_c^i\right) \tag{30}$$

$$\leq \sum_{j \in J} e(j) + \sum_{i=1}^{n} E\left(p_c^i\right) \tag{31}$$

$$\leq \sum_{j \in J} e(j) + \sum_{i=1}^{n} U(\mathcal{S}) \cdot I\left(p_c^i\right) \tag{32}$$

$$\leq \sum_{j \in J} e(j) + U(\mathcal{S}) \cdot I(p) \tag{33}$$

$$\leq \sum_{j \in J} e(j) + U(\mathcal{S}) \cdot t \tag{34}$$

□

We can extend this result to a CRT task system as follows.

Theorem 4. *For a CRT $\mathcal{S} = (\tau, J, e, d)$ and time interval length t, we have*

$$dbf(\mathcal{S}, t) \leq U(\mathcal{S}) \cdot t + \sum_{j \in J} e(j)$$

Proof. To prove Theorem 4 we observe that τ is of the form $\tau = T_1 \,||\, \ldots \,||\, T_n$, see Definition 1.

From Definition 1 we know that each job $j \in J$ is only used in τ once. Thus j is only used in one task T_i where $0 < i \leq n$. This allows us to construct disjoint subsets $J_1, \ldots, J_n \subseteq J$, s.t. only jobs from J_1 occur in T_1 and $J_i \cap J_j = \emptyset$ for $i \neq j$.

We now construct CRTs $\mathcal{S}_i = (T_i, J_i, e, d)$, for all i where $0 < i \leq n$. The execution demand for parallel constructions is given by the sum of the execution times of the individual components. Thus, we can rewrite $dbf(\mathcal{S}, t)$ as in Equation 35. Since every CRT $\mathcal{S}_i$, for $0 < i \leq n$, only has one task, Lemma 3 allows us to write Equation 36.

As J_1 to J_n are disjoint, we can rewrite this as Equation 37. From Equation 22, in the description of how to compute utilization, we know that $U(\mathcal{S}) = \sum_{i=1}^{n} U(\mathcal{S}_i)$, which allows us to write Equation 38, proving Theorem 4.

$$dbf(\mathcal{S}, t) = \sum_{i=1}^{n} dbf(\mathcal{S}_i, t) \tag{35}$$

$$\leq \sum_{i=1}^{n} \left(U(\mathcal{S}_i) \cdot t + \sum_{j \in J_i} e(j) \right) \tag{36}$$

$$\leq t \cdot \sum_{i=1}^{n} U(\mathcal{S}_i) + \sum_{j \in J} e(j) \tag{37}$$

$$\leq t \cdot U(\mathcal{S}) + \sum_{j \in J} e(j) \tag{38}$$

□

This leads to the following result.

Corollary 1. *A CRT $\mathcal{S} = (\tau, J, e, d)$, where $U(\mathcal{S}) < 1$, is preemptive uniprocessor feasible if and only if $dbf(\mathcal{S}, t) \leq t$, for all $t < B$, where*

$$B = \frac{\sum_{j \in J} e(j)}{1 - U(\mathcal{S})}$$

Proof. We shall show by contradiction that if $dbf(\mathcal{S}, t) < t$ for all $t < B$, then $\mathcal{S}$ is feasible. Assume that $\mathcal{S}$ is infeasible, $U(\mathcal{S}) < 1$ and $dbf(\mathcal{S}, t) < t$ for all $t < B$. Then, by Proposition 1, there is some t_f, s.t. $dbf(\mathcal{S}, t_f) > t_f$ (Equation 39) and by assumption we have $t_f \geq B$. Using Theorem 4 we can write this as in Equation 40. This can be rewritten through Equation 40 to Equation 42. As $U(\mathcal{S}) < 1$ we have that $(1 - U(\mathcal{S}))$ is strictly positive, which allows us to write Equation 44. This is the same as to state that $t_f < B$, contradicting the initial assumption that $dbf(\mathcal{S}, t) < t$ for all $t < B$.

$$t_f < dbf(\mathcal{S}, t_f) \tag{39}$$

$$t_f < U(\mathcal{S}) \cdot t_f + \sum_{j \in J} e(j) \tag{40}$$

$$t_f - U(\mathcal{S}) \cdot t_f < \sum_{j \in J} e(j) \tag{41}$$

$$(1 - U(\mathcal{S})) \cdot t_f < \sum_{j \in J} e(j) \tag{42}$$

$$t_f < \frac{\sum_{j \in J} e(j)}{1 - U(\mathcal{S})} \tag{43}$$

$$\tag{44}$$

We know that a CRT $\mathcal{S}$ is feasible only if $dbf(\mathcal{S}, t) \leq t$, for all $t < B$, because by Proposition 1 $dbf(\mathcal{S}, t) > t$ implies infeasibility. □

7 Computation of the Demand Bound Function

We can use the different types of execution paths in a CRT, described in Example 2, to compute the demand bound function. Given an interval of length t, we are interested in the maximum execution demand of any execution path with a deadline less than t. Observe that some execution paths have the same values for execution time and deadline. This fact is exploited in [12] by using demand tuples for representing all paths with the same execution demand and deadline in a compact abstraction. For our computations, a demand tuple $\langle e, i, d\rangle$ is an abstraction of any path p with $e = E(p)$, $i = I(p)$ and $d = D(p)$. The set of all demand tuples over all possible paths with time bound t of a task T is denoted by $DT(T, t)$:

$$DT(T,t) = \{\langle E(p), I(p), D(p)\rangle \mid p \in AllPaths(T), D(p) \leq t\}$$

Given a CRT $\mathcal{S} = (\tau, J, e, d)$ and an interval length t, the demand bound function $dbf(\mathcal{S}, t)$ (Definition 6) can therefore be computed using $DT(T, t)$ as follows.

$$dbf(\mathcal{S},t) = \max\{e \mid \langle e, i, d\rangle \in DT(\tau, t)\}$$

7.1 Computation of Demand Tuples

In order to compute $DT(T, t)$ we introduce some auxiliary functions to generate tuples up to some bound t. These will be used to compute demand tuples for the different kinds of paths. We define these functions as follows.

$$DT_{tot}(T,t) = \{\langle E(p), I(p), D(p)\rangle \mid p \in Paths(T), D(p) \leq t\} \tag{45}$$

$$DT_{suf}(T,t) = \{\langle E(p), I(p), D(p)\rangle \mid p \in suffix(p'), p' \in Paths(T), D(p) \leq t\} \tag{46}$$

$$DT_{pre}(T,t) = \{\langle E(p), I(p), D(p)\rangle \mid p \in prefix(p'), p' \in Paths(T), D(p) \leq t\} \tag{47}$$

The first function, $DT_{tot}(T, t)$, denotes all the demand tuples that quantify over any path through T with a deadline below bound t. This function is not concerned with paths that start or end inside T. Since we also need to have tuples for these, i.e. for paths that are prefixes or suffixes of some path going all the way through T, we make use of $DT_{pre}(T, t)$ and $DT_{suf}(T, t)$ to quantify over these, respectively.

Furthermore we define two functions which compute merges of sets of tuples. First, we can define the parallel merge function m_{par} which computes the parallel combination of tuples up to some given bound t.

$$\mathrm{m}_{par}(dt_1, dt_2, t) = \{\langle e_1 + e_2, i, d\rangle \mid i = \max\{i_1, i_2\}, d = \max\{d_1, d_2\}, d \leq t, \langle e_1, i_1, d_1\rangle \in dt_1, \langle e_2, i_2, d_2\rangle \in dt_2\}$$

The function m_{par} takes the two sets of tuples dt_1 and dt_2 that each corresponds to one of the two branches T_1 or T_2 in a parallel construction. When

these tuples are merged, a new set of tuples is constructed, where each tuple represents one or more paths within the expression $T_1 \mid\mid T_2$ that has a deadline below bound t. Note that the formulas used to calculate the execution time, inter-release time and deadline of the new tuple correspond to the ones used to calculate the same values for a path.

Similarly we can define a sequential merge function, m_{seq}, that merges two sets of tuples in a sequential composition.

$$\mathrm{m}_{seq}(dt_1, dt_2, x, t) = \{\langle e_1 + e_2, i_1 + x + i_2, d\rangle \mid \; d = \max\{d_1, i_1 + x + d_2\}, d \leq t, \\ \langle e_1, i_1, d_1\rangle \in dt_1, \langle e_2, i_2, d_2\rangle \in dt_2\}$$

The function m_{seq} merges demand tuples from dt_1 with tuples from dt_2 to reflect that the paths they represent were concatenated sequentially with the inter-release time x in between.

We can now compute $DT(T, t)$ with the help of the auxiliary functions defined above.

$$DT(j, t) = \{\langle e(j), 0, d(j)\rangle, \langle 0, 0, 0\rangle\} \tag{48}$$

$$DT(T_1\langle x\rangle T_2, t) = \mathrm{m}_{seq}\big(DT_{suf}(T_1, t), DT_{pre}(T_2, t), x, t\big) \\ \cup DT(T_1, t) \cup DT(T_2, t) \tag{49}$$

$$DT(T_1 \mid\mid T_2, t) = \mathrm{m}_{par}\big(DT(T_1, t), DT(T_2, t), t\big) \tag{50}$$

$$DT(T_1 + T_2, t) = DT(T_1, t) \cup DT(T_2, t) \tag{51}$$

$$DT(T^\omega, t) = DT(T, t) \cup \mathrm{m}_{seq}(DT_{suf}(T, t), DT_{pre}(T^\omega, t), 0, t) \tag{52}$$

Following Equation 48 we say that any path within a task, consisting of a single job j is either the empty path represented by $\langle 0, 0, 0\rangle$, or the path consisting of j denoted by $\langle e(j), 0, d(j)\rangle$.

In Equation 49 we state that any path within a sequential composition of T_1 and T_2 with inter-release time x, is a path within either T_1 or T_2, or a sequential merge of any path starting in T_1 and ending in T_2 with inter-release time x between the two parts.

In Equation 50 we say that any path within a task consisting of a parallel composition, of subtasks T_1 and T_2, is a parallel merge of any path within T_1 with any path within T_2.

In Equation 51 we state that any path within a task, consisting of the choice between T_1 and T_2, is any path within either T_1 or T_2 satisfying the bound t. In Equation 52 we state that any path within a task, consisting of the repetition of T, is any path within T or any path starting in T sequentially merged with any path ending in T^ω.

In Equations 45, 46 and 47 we defined DT_{tot}, DT_{suf} and DT_{pre} in terms of paths. To compute the demand tuples for each of these functions we will now present a set of rules for each function.

$$DT_{tot}(j,t) = \{\langle e(j), 0, d(j)\rangle\} \tag{53}$$
$$DT_{tot}(T_1\langle x\rangle T_2,t) = \mathrm{m}_{seq}\big(DT_{tot}(T_1,t), DT_{tot}(T_2,t), x, t\big) \tag{54}$$
$$DT_{tot}(T_1 \mid\mid T_2,t) = \mathrm{m}_{par}\big(DT_{tot}(T_1,t), DT_{tot}(T_2,t), t\big) \tag{55}$$
$$DT_{tot}(T_1 + T_2,t) = DT_{tot}(T_1,t) \cup DT_{tot}(T_2,t) \tag{56}$$
$$DT_{tot}(T^{\omega},t) = DT_{tot}(T,t) \cup \mathrm{m}_{seq}(DT_{tot}(T,t), DT_{tot}(T^{\omega},t), 0, t) \tag{57}$$

For a task consisting of a job (Equation 53), this task is simply the path consisting of the single job, abstracted by a single tuple.

In Equation 54 and Equation 55 we say that the paths going through a sequential or parallel composition, are the sequential or parallel merges of paths through the subtasks, respectively.

For the choice between two tasks (Equation 56) we say that any path through either task is a path through the choice.

In Equation 57 we say that a path through a task, consisting of the repetition of T, is any path through T or any path through T sequentially merged with any path through T^{ω}.

The following rules for the function DT_{pre} compute demand tuples for all paths $p \in \mathit{prefix}(T)$ for a task T.

$$DT_{pre}(j,t) = \{\langle e(j), 0, d(j)\rangle, \langle 0,0,0\rangle\} \tag{58}$$
$$DT_{pre}(T_1\langle x\rangle T_2,t) = DT_{pre}(T_1,t) \cup \mathrm{m}_{seq}(DT_{tot}(T_1,t), DT_{pre}(T_2,t), x, t) \tag{59}$$
$$\begin{aligned} DT_{pre}(T_1 \mid\mid T_2,t) = {} & \mathrm{m}_{par}\big(DT_{pre}(T_1,t), DT_{pre}(T_2,t), t\big) \\ & \cup DT_{pre}(T_1,t) \cup DT_{pre}(T_2,t) \end{aligned} \tag{60}$$
$$DT_{pre}(T_1 + T_2,t) = DT_{pre}(T_1,t) \cup DT_{pre}(T_2,t) \tag{61}$$
$$DT_{pre}(T^{\omega},t) = \mathrm{m}_{seq}\big(DT_{tot}(T^{\omega},t), DT_{pre}(T,t), 0, t\big) \cup DT_{pre}(T,t) \tag{62}$$

Similarly, rules for the function DT_{suf}, which computes demand tuples for all paths $p \in \mathit{suffix}(T)$ for a task T, are presented below.

$$DT_{suf}(j,t) = \{\langle e(j), 0, d(j)\rangle, \langle 0,0,0\rangle\} \tag{63}$$
$$DT_{suf}(T_1\langle x\rangle T_2,t) = \mathrm{m}_{seq}\big(DT_{suf}(T_1,t), DT_{tot}(T_2,t), x\big) \cup DT_{suf}(T_2,t) \tag{64}$$
$$\begin{aligned} DT_{suf}(T_1 \mid\mid T_2,t) = {} & \mathrm{m}_{par}\big(DT_{suf}(T_1,t), DT_{suf}(T_2,t), t\big) \\ & \cup DT_{suf}(T_1,t) \cup DT_{suf}(T_2,t) \end{aligned} \tag{65}$$
$$DT_{suf}(T_1 + T_2,t) = DT_{suf}(T_1,t) \cup DT_{suf}(T_2,t) \tag{66}$$
$$DT_{suf}(T^{\omega},t) = DT_{suf}(T,t) \cup \mathrm{m}_{seq}(DT_{suf}(T,t), DT_{tot}(T^{\omega},t), 0, t) \tag{67}$$

The rules for computation of DT_{pre} and DT_{suf} are quite similar. The rules for a single job (Equation 58 and Equation 63) are the same, where the empty path is represented by $\langle 0,0,0\rangle$ and the path consisting of j is represented by $\langle e(j), 0, d(j)\rangle$.

For sequential composition (Equation 59 and Equation 64) we consider the sequential merge of any path that goes through either the first or last task, with any path that either starts or ends in the last or the first task, respectively.

The parallel composition of two subtasks T_1 and T_2 is the parallel merge of tuples representing paths starting or ending in the subtasks, as noted in Equation 60 and Equation 65, respectively.

Choice is constructed using the same principle as for DT_{tot} (Equation 61 and Equation 66), and repetition can be defined using DT_{tot} (Equation 62 and Equation 67). This recursive rule will be the only source of infinite sets.

Example 5 Recall the environmental sensor in Fig. 1a. In Example 4 we calculated the utilization of this model to be 0.22. To calculate the demand bound function for the model in which two sensors are run on the same uniprocessor we can establish the value of the bound that we need to compute the demand bound function up to.

$$B = \frac{\sum_{j \in J} e(j)}{1 - U(\mathcal{S})} = \frac{34}{1 - 0.44} \approx 60$$

We can now calculate the demand bound function for $\mathcal{S}_2 = T_e \mid\mid T_e$ up to the bound of 60. This is shown in Fig. 2a. Note that the path $((j_3 \mid\mid j_5)\langle 0\rangle j_1\langle 30\rangle(j_2 \mid\mid j_5)) \mid\mid ((j_3 \mid\mid j_5)\langle 0\rangle j_1\langle 30\rangle(j_2 \mid\mid j_5))$ gives rise to the highest demand within the bound. Since the value of the demand bound function is less than the value of time function, the system is feasible.

Similarly, we can calculate the demand bound function for $\mathcal{S}_2 = T_e \mid\mid T_e \mid\mid T_e$. This is shown in Fig. 2b. In this case the demand bound function intersects the time function, and thus the system is infeasible. The first violation is obtained for $t = 25$ where the value of DBF reaches 33. This value can be obtained by releasing j_3, j_5 and j_1 in all three tasks simultaneously.

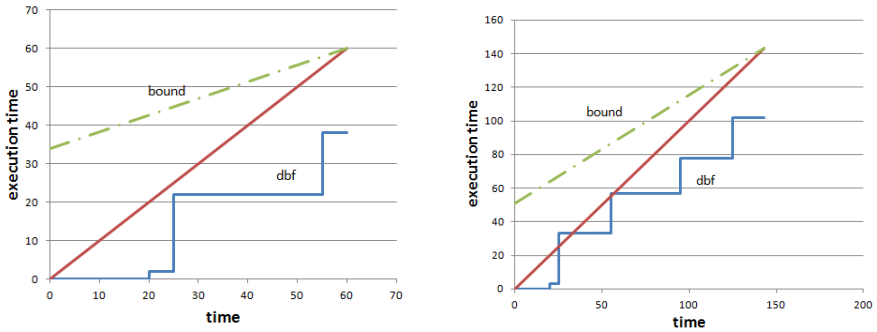
We will now state our main theorem.

Theorem 5. *Given CRT $\mathcal{S}$, where $U(\mathcal{S}) < 1$, feasibility is decidable in pseudo-polynomial time.*

Proof (Sketch). Using Corollary 1, there is an upper bound B, meaning that we need only check $dbf(\mathcal{S}, t) \leq t$ for all $t \leq B$. Given Theorem 3 the utilization $U(\mathcal{S})$, and by implication B, is computable in pseudo-polynomial time. Given this, checking $dbf(\mathcal{S}, t) \leq t$ for all $t \leq B$ can be done in pseudo-polynomial time, assuming the demand bound function is computable in polynomial time.

Rules for DT, DT_{tot}, DT_{pre} and DT_{suf} can be proven correct by structural induction; the arguments for correctness are present in the explanations of this

section. Furthermore, all demand tuples have an execution demand, an inter-release time and a deadline less than t, where $t \leq B$, meaning that we have less than B^3 tuples to consider. Note that the execution demand technically could be larger than t and B, but if such a tuple were encountered, no further execution would be necessary, since the CRT $\mathcal{S}$ would be infeasible. □



(a) Demand bound function for the configuration $T_e \mid\mid T_e$.

(b) Demand bound function for the configuration $T_e \mid\mid T_e \mid\mid T_e$.

Fig. 2: Graphs showing the demand bound function for Example 5.

7.2 Prototype Implementation

The functions and auxiliary functions, presented in previous sections for computation of utilization and demand bound function, generally operate as a set of rules applied to the syntactic structure of a CRT model. These rules can be easily implemented in any purely functional language, where pattern matching makes the structure trivial to operate on.

For this paper we built a minimalistic implementation of the algorithms in Haskell. Minimalistic in the sense that it was a direct translation of the functions presented in this paper to Haskell functions. The prototype implementation allowed us to verify feasibility for CRT models, i.e. the examples shown in this paper.

The implementation can be improved with an optimization that removes *dominated* tuples. We already know that a demand tuple can be discarded if it has already been considered. The basic idea is to extend this to discard tuples that are dominated by one that was already computed. The intuition is that if a tuple is dominated then it does not yield any new information. This optimization was proposed in [12]. In the following we define in the same way the concept of domination in general terms.

Given demand tuples $dt = \langle e(j), i(j), d(j) \rangle$, $dt' = \langle e(j'), i(j'), d(j') \rangle$, we say that dt dominates dt' if:

$$e(j) \geq e(j') \ \wedge \ d(j) \leq d(j') \ \wedge \ j = j'$$

Moreover, if the tuples are sorted by inter-release time (or deadline) in ascending order as they are generated, it is trivial to implement the optimization using domination presented above. We also claim that there is a pseudo-polynomial algorithm that only uses a polynomial amount of memory, suggesting that checking for feasibility of CRT models is in PSPACE.

8 Conclusion

We have considered uniprocessor feasibility testing and proposed the Concurrent Real-time Task model (CRT) that is syntactically more expressive than the Digraph Real-time Task model (DRT), since CRT models allow parallel compositions of jobs within the same task.

With this model it is possible to more closely model real world systems, where jobs require explicit use of synchronization in the control flow. We have shown how to calculate utilization for CRTs in pseudo-polynomial time and used the concept of the demand bound function to determine feasibility of CRTs. We have demonstrated how an upper bound for the demand bound function may be computed for CRTs. Finally, we can determine feasibility for CRTs in pseudo-polynomial time, similar to the DRT model.

9 Future Work

We conclude this paper with possible directions for future work. For instance, it is of interest to formally prove that CRT models are strictly more expressive than DRT models. It is also worth investigating the expressiveness of CRT models compared to Extended DRT models (EDRT), presented in [11]. Preliminary investigation suggests that EDRT models are strictly more expressive, however, formal proofs to support this claim are necessary.

Moreover, it can be beneficial to remove the structural restrictions on CRTs, i.e. allow loop constructions in subtasks, eliminating the need for the third syntactic category. As far as computation of demand bound function is concerned this will not be problematic, but it may, nevertheless, significantly complicate the computation of utilization.

In relation to utilization, we consider improving the complexity of the algorithm. The rules for the computation of utilization, presented earlier, are heavily inspired by similar efforts for DRT models [12], which describe a pseudo-polynomial algorithm for utilization. Yet, in [5] a similar polynomial time algorithm for maximum ratio cycles is presented; an algorithm that seems likely to be adaptable for the computation of the utilization for DRT models. This suggests that a polynomial time algorithm for finding the utilization may exist for both DRT and CRT models.

The complexity of the feasiblity problem is also an important question for both CRT and DRT models. We believe feasibility for CRT models to be co-NP-hard and for DRT models to be co-NP-complete. We consider proving these statements as an obvious direction for future work.

Another problem left open both by this paper, [12] and [11], is how to decide feasiblity for CRT or DRT models with a utilization of exactly one. In fact, it is even possible that the feasibility problem, for both models with a utilization of one, is undecidable.

References

1. Sanjoy Baruah, Deji Chen, Sergey Gorinsky, and Aloysius Mok. Generalized multiframe tasks. *Real-Time Systems*, 17:5–22, 1999.
2. Robert I. Davis and Alan Burns. A survey of hard real-time scheduling for multiprocessor systems. *ACM Comput. Surv.*, 43(4):35, 2011.
3. Elena Fersman, Pavel Krcál, Paul Pettersson, and Wang Yi. Task automata: Schedulability, decidability and undecidability. *Inf. Comput.*, 205(8):1149–1172, 2007.
4. J.D. and Ullman". Np-complete scheduling problems. *Journal of Computer and System Sciences*, 10(3):384 – 393, 1975.
5. Eugene L. Lawler. *Combinatorial Optimization: Networks and Matroids*, chapter Chapter 13, pages 94–97. Holt, Rinehart and Winston, New York, USA, 1976.
6. C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, 1973.
7. Jane W.S. Liu. *Real-time Systems*. Prentice Hall, 2000.
8. Aloysius K. Mok and Deji Chen. A multiframe model for real-time tasks. *IEEE Trans. Software Eng.*, 23(10):635–645, 1997.
9. Aloysius Ka-Lau Mok. *Fundamental design problems of distributed systems for the hard-real-time environment*. Massachusetts Institute of Technology, 1977.
10. Michael Sipser. *Introduction to the theory of computation*. Thomson Course Technology, 2006.
11. M. Stigge, P. Ekberg, N. Guan, and Wang Yi. On the tractability of digraph-based task models. In *Real-Time Systems (ECRTS), 2011 23rd Euromicro Conference on*, pages 162 –171, july 2011.
12. Martin Stigge, Pontus Ekberg, Nan Guan, and Wang Yi. The digraph real-time task model. In *IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 71–80, 2011.
13. Apostolos Syropoulos. Mathematics of multisets. In *WMP*, pages 347–358, 2000.

A Proof of Theorem 1

Theorem 1. *Given an execution path p, for any release scenario $\sigma \in RS(p)$ it holds that:*

a) $E(p) = E(\sigma)$,
b) $I(p) \leq I(\sigma)$,
c) $D(p) \leq D(\sigma)$.

Also, there exists a release scenario $\sigma' \in RS(p)$ s.t. $I(p) = I(\sigma')$ and $D(p) = D(\sigma')$.

Proof. We prove Theorem 1 by structural induction on the execution path p. We start by proving Theorem 1a.

When the execution path p consists of only one job, $p = j$, the execution time of p is equal to $e(j)$ (Equation 11). In this case the release scenario σ is defined as in Definition 3a and $E(\sigma) = e(j)$. So, for $p = j$ we have that $E(p) = E(\sigma)$.

Consider the rule $p = p_1\langle x\rangle p_2$; the execution time of p is given by $E(p_1) + E(p_2)$ (Equation 12). In this case the release scenario σ is given by Definition 3b. By the induction hypothesis assume that $E(p_1) = E(\sigma_1)$ and $E(p_2) = E(\sigma_2)$. Since σ includes all the tuples from σ_1 and the tuples from σ_2, where the inter-release time and deadline have changed, then in this case it also holds that $E(p) = E(\sigma)$.

Consider the rule $p = p_1 \mid\mid p_2$; the execution time of p is again given by $E(p_1) + E(p_2)$ (Equation 13). In this case the release scenario σ is given by Definition 3c. By the induction hypothesis assume that $E(p_1) = E(\sigma_1)$ and $E(p_2) = E(\sigma_2)$. Again σ includes exactly all the tuples from σ_1 and σ_2, therefore $E(p) = E(\sigma)$.

Thus, we have proven Theorem 1a and proceed to prove Theorem 1b.

Consider $p = j$; the inter-release time of p is 0 (Equation 14). The release scenario σ only includes one tuple of the form $\langle r, e, d\rangle$ (Definition 3a), so Equation 9 can be rewritten to $I(\sigma) = r - r = 0$. Therefore we conclude that $I(p) = I(\sigma)$ in this case.

Consider the case where $p = p_1\langle x\rangle p_2$; the inter-release time of p is given by $I(p_1) + I(p_2) + x$ (Equation 15). In this case the release scenario σ is given by Definition 3b. By the induction hypothesis assume that $I(p_1) \leq I(\sigma_1)$ and $I(p_2) \leq I(\sigma_2)$. The inter-release time of a release scenario depends on the difference between the tuple with the greatest release time and the tuple with the smallest release time (Equation 9). We can then write the following:

$$I(\sigma_1) = r_{1_{max}} - r_{1_{min}} \tag{68}$$

$$I(\sigma_2) = r_{2_{max}} - r_{2_{min}} \tag{69}$$

where $r_{1_{max}}$ and $r_{1_{min}}$ are the maximum and minimum release times of the tuples in σ_1, in the order mentioned. Likewise, $r_{2_{max}}$ and $r_{2_{min}}$ are the maximum and

minimum release times of the tuples in σ_2, respectively. The release scenario σ includes the unchanged tuples from σ_1 as well as the tuples from σ_2, where the release times are increased by $r_{1_{max}} + x$. Since all the tuples in σ_2 have their inter-release time increased by the same value the relation between them remains the same. Thus, we can express the inter-release time of σ below.

$$I(\sigma) = \max\{r_{2_{max}} + r_{1_{max}} + x, r_{1_{max}}\} - \min\{r_{1_{min}}, r_{2_{min}} + r_{1_{max}} + x\}$$

Clearly, $r_{2_{max}} + r_{1_{max}} + x \geq r_{1_{max}}$ and $r_{1_{min}} \leq r_{2_{min}} + r_{1_{max}} + x$, so the inter-release time of σ becomes the following:

$$\begin{aligned} I(\sigma) &= r_{2_{max}} + r_{1_{max}} + x - r_{1_{min}} = r_{2_{max}} + x + I(\sigma_1) \\ &\geq r_{2_{max}} - r_{2_{min}} + x + I(\sigma_1) = I(\sigma_2) + x + I(\sigma_1) \geq I(p) \end{aligned}$$

Note that $I(p) = I(\sigma)$, when $r_{2_{min}} = 0$, $I(\sigma_2) = I(p_2)$ and $I(\sigma_1) = I(p_1)$.

Consider the parallel case $p = p_1 \mid\mid p_2$; the inter-release time of p is the maximum inter-release time between $I(p_1)$ and $I(p_2)$ (Equation 16). In this case σ is given by Definition 3c and the inter-release time of σ_1 and σ_2 are again given by Equations 68-69. The release scenario σ includes exactly the same tuples that are in σ_1 and σ_2. So, the inter-release time of σ (Equation 9) can be written as the following.

$$I(\sigma) = \max\{r_{1_{max}}, r_{2_{max}}\} - \min\{r_{1_{min}}, r_{2_{min}}\}$$

Thus, we have the following four cases.

1. $r_{1_{max}} \geq r_{2_{max}}$ and $r_{1_{min}} \leq r_{2_{min}}$, $I(\sigma) = r_{1_{max}} - r_{1_{min}} = I(\sigma_1)$, $I(\sigma) \geq I(\sigma_2)$
2. $r_{1_{max}} > r_{2_{max}}$ and $r_{1_{min}} > r_{2_{min}}$, $I(\sigma) = r_{1_{max}} - r_{2_{min}} > I(\sigma_1)$, $I(\sigma) > I(\sigma_2)$
3. $r_{1_{max}} < r_{2_{max}}$ and $r_{1_{min}} < r_{2_{min}}$, $I(\sigma) = r_{2_{max}} - r_{1_{min}} > I(\sigma_1)$, $I(\sigma) > I(\sigma_2)$
4. $r_{1_{max}} \leq r_{2_{max}}$ and $r_{1_{min}} \geq r_{2_{min}}$, $I(\sigma) = r_{2_{max}} - r_{2_{min}} = I(\sigma_2)$, $I(\sigma) \geq I(\sigma_1)$

By the induction hypothesis assume that $I(p_1) \leq I(\sigma_1)$ and $I(p_2) \leq I(\sigma_2)$. Hence the inter-release time for p will either be $I(\sigma_1)$ or $I(\sigma_2)$, which in any case will be less than or equal to $I(\sigma)$. Note that $I(p) = I(\sigma)$ when both the first case holds together with $I(\sigma_1) \geq I(\sigma_2)$ and the second case holds together with $I(\sigma_2) \geq I(\sigma_1)$.

Finally, we will prove Theorem 1c.

Here we first consider the execution path that only includes one job, i.e. $p = j$. The deadline of the path is $d(j)$ (Equation 17). In this case the release scenario σ is defined as in Definition 3a and only contains one tuple with deadline $d(j) + c$, where $c \in \mathbb{R}_{\geq 0}$.

The deadline of a release scenario is given by the largest deadline of the tuples (Equation 10). So, in this case $D(p) = d(j) \leq D(\sigma)$.

Now consider the path $p = p_1\langle x \rangle p_2$; the deadline of the path is given by the maximum between the deadline of p_1 and $I(p_1) + x + D(p_2)$ (Equation 18).

Any release scenario will be given by Definition 3b. Let $d_{1_{max}}$ and $d_{2_{max}}$ be the deadlines of the tuples with maximum deadline in σ_1 and σ_2, respectively. Then $D(\sigma_1) = d_{1_{max}}$ and $D(\sigma_2) = d_{2_{max}}$. Any release scenario σ contains all the tuples from σ_1 and all the tuples from σ_2, where the last ones have their deadlines increased by $x+r_{1_{max}}$, where $r_{1_{max}}$ is the maximum release time among the tuples in σ_1. Because all tuples in σ_2 have been increased by the same value, their ordering is preserved, hence we can write the deadline of σ as the equation below.

$$D(\sigma) = \max\{d_{1_{max}}, d_{2_{max}} + x + r_{1_{max}}\}$$

Now assume by the induction hypothesis that $D(p_1) \leq D(\sigma_1)$ and $D(p_2) \leq D(\sigma_2)$. We have already shown that $I(p_1) \leq I(\sigma_1)$, so we can rewrite Equation 18 by first replacing the inter-release time and then the deadlines by those from the release scenarios.

$$D(p) \leq \max\{D(p_1), I(\sigma_1) + x + D(p_2)\} \leq \max\{D(\sigma_1), I(\sigma_1) + x + D(\sigma_2)\}$$

We also know that $I(\sigma_1) = r_{1_{max}} - r_{1_{min}}$ (Equation 9). It follows that

$$D(p) \leq \max\{d_{1_{max}}, d_{2_{max}} + x + r_{1_{max}} - r_{1_{min}}\} \leq D(\sigma)$$

Note also that $D(p) = D(\sigma)$ when $r_{1_{min}} = 0$, $D(p_1) = D(\sigma_1)$ and $D(p_2) = D(\sigma_2)$.

Finally, consider when $p = p_1 \mid\mid p_2$. The deadline of this path is given by Equation 19. In this case σ is given by Definition 3c. Let $d_{1_{max}}$ and $d_{2_{max}}$ be the deadlines of the tuples with the maximum deadlines in σ_1 and σ_2, respectively. Thus, the deadline of σ is given by $\max\{d_{1_{max}}, d_{2_{max}}\}$ (Equation 10). By the induction hypothesis assume that $D(p_1) \leq D(\sigma_1)$ and $D(p_2) \leq D(\sigma_2)$. We can rewrite Equation 19.

$$D(p) \leq \max\{D(\sigma_1), D(\sigma_2)\} \leq D(\sigma)$$

Note that $D(p) = D(\sigma)$ when $D(p_1) = D(\sigma_1)$ and $D(p_2) = D(\sigma_2)$. □

B Expressiveness of CRTs

In this section we shall investigate the expressiveness of CRTs. We assert that CRT systems are at least as expressive as DRTs. That is, for every DRT there exists a CRT that yields similar sets of release scenarios. In addition, we claim that there exist CRTs that can produce job release scenarios that cannot be produced by any DRT. Finally, we compare the CRT model to the EDRT model and claim that the number of global constraints and the size of the graph grow exponentially with the number of jobs in parallel subtasks, as well as the number of parallel subtasks.

We can now define the concept of release scenario similarity, which is used to establish an expressiveness criterion between sporadic job models.

Definition 8 (Release Scenario Similarity). *Given two release scenarios σ_1 and σ_2, we say that σ_1 and σ_2 are similar if for any release tuple $(r_1, e_1, d_1) \in \sigma_1$, where $e_1 \neq 0$, there exists a release tuple $(r_2, e_2, d_2) \in \sigma_2$, s.t. $r_1 = r_2, e_1 = e_2, d_1 = d_2$ and contrariwise.*

Definition 9 (Expressiveness). *Let C_1 and C_2 be classes of sporadic job models. We then say that C_1 is at least as expressive as C_2, $C_2 \preceq C_1$, if for any $\tau \in C_2$ there exists some $\tau' \in C_1$ such that any release scenario of τ is similar to a release scenario of τ'.*

If $C_1 \preceq C_2$ and $C_2 \not\preceq C_1$, we write $C_1 \prec C_2$ and say that C_2 is strictly more expressive than C_1.

To prove that the CRT model is at least as expressive as the DRT model we shall now summarize the semantics of the DRT model.

B.1 Digraph Real-Time Task Model

The DRT model [12] consists of a set of independent tasks.

Definition 10 (DRT Task). *A DRT task T_{DRT} is a 5-tuple $T_{DRT} = (\mathcal{V}, \mathcal{E}, \rho, e, d)$, where*

- *$\mathcal{V}$ is a finite set of jobs,*
- *$\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is a finite set of edges,*
- *$\rho : \mathcal{E} \to \mathbb{N}$ is the inter-release times of the edges,*
- *$e : \mathcal{V} \to \mathbb{N}$ is the worst case execution times of the jobs and*
- *$d : \mathcal{V} \to \mathbb{N}$ is the relative deadlines of the jobs.*

Definition 11 (DRT). *A Digraph Real-time Task system τ_{DRT} is a finite set of tasks.*

The release sequence of a DRT task T_{DRT} is given by a sequence of release tuples (r, e, d), where each tuple represents the release of a job j at absolute time r (the release time) with the WCET $e = e(j)$ and absolute deadline $d = d(j) + r$ for a job in T_{DRT}. A release sequence $\sigma_{T_{DRT}} = [(r_1, e_1, d_1), (r_2, e_2, d_2), \ldots, (r_n, e_n, d_n)]$ is valid if and only if there exists a path $\pi = (v_1, v_2, \ldots)$ in T_{DRT} s.t. for $i, 1 \leq i \leq n$ it holds that

a) $e_i = e(v_i)$,
b) $d_i = r_i + d(v_i)$ and
c) $r_{i+1} - r_i \geq \rho(v_i, v_{i+1})$.

A release scenario for a DRT can be constructed using a multiset containing all tuples in the release sequence.

B.2 Expressiveness

We now consider the DRT model to compare its expressiveness to that of the CRT model. Any DRT task T_{DRT} can be converted into a CRT. To show this we use a similar approach to that of converting a deterministic finite automaton into a generalized nondeterministic finite automaton and then to a regular expression [10]. In this case we transform the DRT model into a generalized DRT (GDRT) model, which can subsequently be converted into a task expression in the CRT model syntax.

Definition 12 (GDRT). *A GDRT G is a tuple $(\mathcal{V}, \mathcal{E}, J, \delta, e, d)$, where*

- *$\mathcal{V}$ is a finite set of vertices,*
- *$\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is a finite set of edges,*
- *J is a finite set of jobs,*
- *$\delta : \mathcal{E} \to CRT_T$ is a mapping from edges to legal CRT tasks,*
- *$e : J \to \mathbb{N}$ is the worst case execution times of the jobs and*
- *$d : J \to \mathbb{N}$ is the relative deadlines of the jobs.*

We denote a job that is released by a release tuple, which is defined similarly as for DRTs. The release sequence generated by a GDRT G is given, similarly to the DRT model, by a sequence of release tuples. A release sequence $\sigma_{T_{DRT}} = [(r_1, e_1, d_1), (r_2, e_2, d_2), \dots (r_n, e_n, d_n)]$ is valid if there exists a path $\pi = (v_a, v_b, v_c, \dots)$ which can generate a CRT task $T = \delta(v_a, v_b)\langle 0 \rangle \delta(v_b, v_c)\langle 0 \rangle \dots$ with an execution path $p = j_1\langle x_1 \rangle j_2 \langle x_2 \rangle j_3, \dots$ s.t. $p \in Paths(T)$ and it holds for $i, 1 \leq i \leq n$ that

a) $e_i = e(j_i)$,
b) $d_i = r_i + d(j_i)$ and
c) $r_{i+1} - r_i \geq x_i$.

As for DRT models, we can construct a release scenario by constructing the multiset containing all release tuples in the release sequence.

Definition 13 (Similarity). *A DRT task T_{DRT} and a GDRT G are said to be similar if for any valid release scenario $\sigma_{T_{DRT}}$ of T_{DRT} there exists a valid release scenario σ_G of G, s.t. $\sigma_{T_{DRT}}$ is similar to σ_G and contrariwise.*

To translate a DRT task into a CRT we introduce the *CONVERT(T_{DRT})* algorithm (see Algorithm 1). The algorithm takes a DRT task $T_{DRT} = (\mathcal{V}_\mathcal{T}, \mathcal{E}_\mathcal{T}, \rho, e, d)$ as input and outputs a CRT task that is similar to T_{DRT}.

Algorithm 1 *CONVERT*(T_{DRT})

1. We introduce a new job, j_ϵ, s.t. $e(j_\epsilon) = 0, d(j_\epsilon) = 0$. We can now convert the DRT task $T_{DRT} = (\mathcal{V}_\mathcal{T}, \mathcal{E}_\mathcal{T}, \rho, e, d)$ into a GDRT $G = (\mathcal{V}, \mathcal{E}, J, \delta, e, d)$ as follows.
   - $\mathcal{V} = \mathcal{V}_\mathcal{T} \cup \{v_{start}, v_{end}\}$
   - $\mathcal{E} = \mathcal{E}_\mathcal{T} \cup \{(v_{start}, v_k) \mid v_k \in \mathcal{V} \setminus \{v_{start}\}\} \cup \{(v_k, v_{end}) \mid v_k \in \mathcal{V} \setminus \{v_{end}\}\}$
   - $J = \mathcal{V}_\mathcal{T} \cup \{j_\epsilon\}$
   - $\delta(v_i, v_j) = \begin{cases} j_i \langle x \rangle j_\epsilon & \text{if } (v_i, v_j) \in \mathcal{E}_\mathcal{T}, \text{ where } j_i = v_i, x = \rho(v_i, v_j) \\ j_\epsilon & \text{if } v_i = v_{start}, v_j \in \mathcal{V} \setminus \{v_{start}\} \\ j_\epsilon & \text{if } v_j = v_{end}, v_i \in \mathcal{V} \setminus \{v_{end}\} \end{cases}$
2. If G contains only two vertices, v_i and v_j, output ($\delta(v_i, v_j)$,J,e,d).
3. Select a vertex $v_t \in \mathcal{V} \setminus \{v_{start}, v_{end}\}$.
4. Construct a new GDRT $G' = (\mathcal{V}', \mathcal{E}', J, \delta', e, d)$, where
   - $\mathcal{V}' = \mathcal{V} \setminus \{v_t\}$
   - $\mathcal{E}' = \mathcal{E} \setminus \{(v_t, v_i) \mid (v_t, v_i) \in \mathcal{E}, v_i \in \mathcal{V}\} \setminus \{(v_i, v_t) \mid (v_i, v_t) \in \mathcal{E}, v_i \in \mathcal{V}\} \cup \{(v_i, v_j) \mid (v_i, v_t) \in \mathcal{E}, (v_t, v_j) \in \mathcal{E}\}$
   - For all $v_i, v_j \in \mathcal{V}, i \neq j$, assign a new value to

$$\delta'(v_i, v_j) = \begin{cases} T_1 \langle 0 \rangle T_3 + T_4 & \text{if } \{(v_i, v_t), (v_t, v_j), (v_i, v_j)\} \subseteq \mathcal{E}, (v_t, v_t) \notin \mathcal{E} \\ T_1 \langle 0 \rangle T_3 & \text{if } \{(v_i, v_t), (v_t, v_j)\} \subseteq \mathcal{E}, \{(v_t, v_t), (v_i, v_j)\} \not\subset \mathcal{E} \\ \left(T_1 \langle 0 \rangle (T_2)^\omega \langle 0 \rangle T_3 + T_1 \langle 0 \rangle T_3\right) + T_4 & \text{if } \{(v_i, v_t), (v_t, v_j), (v_t, v_t), (v_i, v_j)\} \subseteq \mathcal{E} \\ \left(T_1 \langle 0 \rangle (T_2)^\omega \langle 0 \rangle T_3 + T_1 \langle 0 \rangle T_3\right) & \text{if } \{(v_i, v_t), (v_t, v_j), (v_t, v_t)\} \subseteq \mathcal{E}, (v_i, v_j) \notin \mathcal{E} \\ T_4 & \text{else} \end{cases}$$

   where $T_1 = \delta(v_i, v_t)$, $T_2 = \delta(v_t, v_t)$, $T_3 = \delta(v_t, v_j)$ and $T_4 = \delta(v_i, v_j)$
5. Set $G = G'$ and go to step 2.

Note that the loop composition, T^ω, means at least one iteration is performed. Because of this a choice equal to not taking it is added in the third and fourth line of the δ function in step 4. As a result, the final expression will have at least one added choice for every loop in the graph.

The output of the algorithm is a CRT task expression equivalent to the DRT task. We will now consider the correctness of the algorithm, using the similarity between the release sequences of DRT and GDRT models.

Definition 14 (DRT-CRT Similarity). *A DRT task T_{DRT} and a CRT $\mathcal{S}$ are said to be similar if for any valid release scenario $\sigma_{T_{DRT}}$ of T_{DRT} there exists a valid release scenario $\sigma_{\mathcal{S}}$ of $\mathcal{S}$, s.t. $\sigma_{T_{DRT}}$ is similar to $\sigma_{\mathcal{S}}$ and contrariwise.*

Lemma 4. *Let $T_{DRT} = (\mathcal{V}_\mathcal{T}, \mathcal{E}_\mathcal{T}, \rho, e, d)$ be a DRT task and the CRT $\mathcal{S} = (\tau, J, e, d)$ be the output from CONVERT(T_{DRT}), then $\mathcal{S}$ is similar to T_{DRT}.*

Proof. We prove by induction that the resulting CRT $\mathcal{S}$ of *CONVERT(T_{DRT})* is similar to the input DRT T_{DRT}. We do this by showing

(i) that the GDRT G created from T_{DRT} in the base step (step 1 of *CONVERT*) is similar to T_{DRT},
(ii) the reduced GDRT G' created from G in the induction step (step 4 of *CONVERT*) is similar to G, and
(iii) and the resulting CRT $\mathcal{S}$ in the final step (step 2) is similar to G.

Base Step (i)
Let $T_{DRT} = (\mathcal{V}_T, \mathcal{E}_T, \rho, e, d)$ be the input for *CONVERT(T_{DRT})* and $G = (\mathcal{V}, \mathcal{E}, J, \delta, e, d)$ be the GDRT created in step 1.

If we have a path $p = (j_1, j_2, \ldots, j_n)$ in T_{DRT}, then by the construction of $\mathcal{V}$ and $\mathcal{E}$ in step 1 the same path exists in G. Given the construction of δ, p in G will yield the following CRT task $T = j_1\langle x_1\rangle j_\epsilon\langle 0\rangle j_2\langle x_2\rangle j_\epsilon\langle 0\rangle \ldots \langle 0\rangle j_n\langle 0\rangle j_\epsilon$. Ignoring empty jobs and inter-release times of zero we get $T = j_1\langle x_1\rangle j_2\langle x_2\rangle \ldots \langle x_{n-1}\rangle j_n$, where $x_i = \rho(j_i, j_{i+1})$. Given the semantics for release scenarios of DRT and GDRT models it can be observed that p gives rise to the same release scenarios for T_{DRT} and G.

Note that for any path p' in G, the same path also exists in T_{DRT}, with the exception of paths prefixed and suffixed with v_{start} or v_{end}. However, since edges to and from v_{start} and v_{end} only carry empty jobs these will have no effect on the similarity. Thus, T_{DRT} and G are similar, and (i) holds.

Induction Step (ii)
Let $G = (\mathcal{V}, \mathcal{E}, J, \delta, e, d)$ be the GDRT before step 4 and $G' = (\mathcal{V}', \mathcal{E}', J, \delta', e, d)$ be the reduced GDRT created, where v_i was removed. We now show that for any path $p = (v_1, v_2, \ldots, v_i, \ldots)$ in G, there is a corresponding path in G'. We have two cases.

(a) The path π does not traverse v_i.
(b) The path π traverses v_i.

For case (a) we do not have to show anything, since the path has not changed. In case (b) the path $p' = (v_1, v_2, \ldots, v_{i-1}, v_{i+1}, \ldots)$ gives rise to the same release scenarios as p. Given the semantics for G', p' yields a CRT task $T' = \delta(v_1, v_2)\langle 0 \rangle \delta(v_2, v_3) \ldots \delta(v_{i-1}, v_{i+1}) \ldots$, since the edge between v_{i-1} and v_{i+1} provided by $\delta(v_{i-1}, v_{i+1})$ contains both the label from v_{i-1} to v_i, as well as the label from any edge that might have been going from v_i to itself and also the label on the edge from v_i to v_{i+1}.

Notice that for any release scenario of a path p' in G', there is a path p in G, s.t. there is a similar release scenario of p. This can be shown with the same arguments as were used to show that cases (a) and (b) hold, with the additional detail that a cycle term in a CRT task might be replaced with a self-loop on v_i in G.

Final Step (iii)

Note that v_{start} and v_{end} is never removed in step 4. Thus, when a GDRT has been computed in step 2, we have $G = (\mathcal{V}, \mathcal{E}, J, \delta, e, d)$ where $\mathcal{V} = \{v_{start}, v_{end}\}$. We will now show that $\mathcal{S} = (\delta(v_{start}, v_{end}), J, e, d)$ is similar to G. The only path through G is $\pi = (v_{start}, v_{end})$, which yields the CRT task $T_{CRT} = \delta(v_{start}, v_{end})$. It can be observed that $\mathcal{S}$ and T_{CRT} have the same syntactic structure, and given that e and d are the same as in G, the models $\mathcal{S}$ and G must necessarily be similar.

□

We are now ready to state the following theorem.

Theorem 6. *The class of CRT models C_{CRT} is at least as expressive as the class of DRT models C_{DRT}, i.e. $C_{DRT} \preceq C_{CRT}$.*

Proof. By Lemma 4 we know that we can create a similar CRT task for every DRT task, and by parallel composition we can create a CRT for every DRT. □

Claim. The class of CRT models C_{CRT} is strictly more expressive than the class of DRT models C_{DRT}, i.e. $C_{DRT} \prec C_{CRT}$.

Consider a CRT $\mathcal{S} = (\tau_{CRT}, J, e, d)$, where $\tau_{CRT} = T_1$, $T_1 = j_1\langle x_1 \rangle j_2 \langle x_2 \rangle j_3 \mid\mid j_4$, $J = \{j_1, j_2, j_3, j_4\}$, $x_1, x_2 \in \mathbb{N}$. If we attempt to represent this CRT with different tasks in a DRT we cannot preserve the inter-release times, x_1 or x_2, regardless of how the task is broken up, since the resulting tasks are independent. Consequently, the DRT model will have a larger set of release scenarios than the CRT. Now if we retain the jobs within the same task, a starting point for obtaining the same job release scenarios is to ensure the same set of execution paths. This can be obtained by considering all possible interleavings of the paths in the parallel subterm. However, for the interleaving sequence (j_2, j_4, j_3), it is not possible to keep track of the inter-release time x_2 between j_2 and j_3, since there is no edge connecting the two jobs. As a result, this solution will again either yield a larger or smaller set of release scenarios, based on the constraints chosen for the edges connecting the three jobs.

Claim. The class of EDRTs [11] C_{EDRT} is at least as expressive as the class of CRTs, $C_{CRT} \preceq C_{EDRT}$.

A CRT may be converted into an EDRT by eliminating every parallel composition between tasks and converting each of them into an independent task in the EDRT task system. From now on we will focus on the conversion of a single CRT task into an EDRT task. Since the EDRT must encode all the possible execution paths in the parallel constructions of the CRT, it will need to include all the possible interleavings of the branches. This gives an exponential growth in the size of the graph. Global constrainsts are needed to maintain the inter-release time between the sequential compositions of jobs in a parallel subterm. Each of these sequential compositions requires one global constraint for each interleaving, where one or more jobs separate the previously connected jobs. The number of interleavings then depends on the following: the number of branches in the parallel composition, where the subterm that should be converted is located and the number of jobs in each branch. For each parallel composition there is at least one sequence, where the job is in between the two jobs connected in the sequential composition. When there are more jobs in the same subterm, the number of interleavings depends on the possible interleavings of this subterm, including all the other parallel compositions, besides the one that is being considered. Once again this number is exponential for the same reasons stated earlier. This holds for all sequential subterms in a parallel composition, therefore giving an exponential number of global constraints.