Abstract. Embedded systems consist of interacting components that are required to deliver a specific functionality under constraints on execution rates and relative time separation of the components. In this paper, we model an embedded system using concurrent processes interacting through synchronization. We assume that there are rate constraints on the execution rates of processes imposed by the designer or the environment of the system, where the execution rate of a process is the number of its executions per unit time. We address the problem of computing bounds on the execution rates of processes constituting an embedded system, and propose an interactive rate analysis framework. As part of the rate analysis framework we present an efficient algorithm for checking the consistency of the rate constraints. Bounds on the execution rate of each process are computed using an efficient algorithm based on the relationship between the execution rate of a process and the maximum mean delay cycles in the process graph. Finally, if the computed rates violate some of the rate constraints, some of the processes in the system are redesigned using information from the rate analysis step. This RATe ANalysis framework is implemented in a tool called RATAN. We illustrate by an example how RATAN can be used in an embedded system design.
1 Introduction

The design of embedded systems has become an increasingly difficult problem due to increasing design complexity and shortening time-to-market. Since such systems consist of interacting hardware and software components, the designers have to validate not only these components individually but also the interfaces between them. The design of such a system entails mapping a high level description of the system in a hardware description language into hardware and software so that all the constraints such as timing, resource, and power are satisfied. This mapping typically consists of three main stages. The system is captured in a high level description, and an architecture of the system with software and hardware parts is selected during the first stage. The second stage includes the independent design of hardware, software, and interface between them. The third stage is the integration and test stage. During this stage, a hardware prototype is built and the software is tested to make sure that the hardware-software interface functions correctly. However, interfacing software with hardware is likely to cause many timing constraint violations, and unfortunately, any such errors found in this stage are very costly to correct due to the amount of the commitment already done in the design. Consequently, it is becoming extremely important to take timing constraints into consideration at higher levels in the design flow. This paper is an attempt in this direction.

An embedded system consists of concurrent components interacting under timing constraints. It is customary to generate an intermediate graphical model of the system during the design of such systems as illustrated in Figure 1. We use a process graph as our model of an embedded system in which each component corresponds to a process. Processes are active concurrently, and synchronization between the processes occur according to a statically defined dependency relation between them. Timing constraints are usually in two forms: constraints on the execution rates of processes and constraints on the time separation between a pair of processes. In this paper, we are concerned with those on the execution rates, called rate constraints. Rate constraints are imposed by the designer to guarantee the conformance of the system to its environment. Our ultimate objective is to make sure that all the rate constraints are satisfied, which holds if each process executes at a rate as constrained. We propose an interactive framework, called rate analysis framework, to realize this objective. This framework includes four main steps. The first step checks the consistency of the rate constraints using an efficient algorithm. The second step finds bounds on the execution rate of each process using an efficient algorithm based on the relationship between the execution rate of a process and the maximum mean delay cycles in the process graph. The third step checks to see if the rate constraints are satisfied. The fourth step is redesigning some parts of the system and is attempted if the set of the rate constraints are found to be inconsistent during the first step or if there are some rate constraints that are not satisfied, i.e., violated, during the third step. After any redesigns, the above steps are repeated. This RATE ANalysis framework is implemented in a tool called RATAN. We illustrate by an example how RATAN can be used in
an embedded system design.

Figure 1: Interaction between rate analysis and synthesis in the design of an embedded system.

The problem of determining execution rates has been studied in several different contexts. Gupta and De Micheli [6] have examined the problem of rate analysis in embedded systems, but they consider very limited interaction/synchronization between the component processes. Further, their algorithms require the processes to be implemented only in a non-pipelined manner. Rate analysis has also been studied for asynchronous, concurrent systems modeled using timed Petri nets [2], [9], [12]. However, their analysis is based on restrictive assumptions, such as the vector of start times for the processes is a specially chosen vector. This assumption is not valid if an embedded system software is implemented as a set of co-routines where each co-routine consists of an initial process that is executed only once and a repeating body. Hulgaard et al. [7] have addressed the problem of finding tight bounds on the time interval between events in a process graph using implicit unfolding of the process graph. These bounds can be converted to bounds on the execution rates of processes by inversion. However, the process graphs that can be analyzed using their techniques need to be strongly connected or satisfy other restrictive conditions. Further, the proposed algorithms can be computationally expensive in the worst case.

Bacelli et al. [1] have used algebraic techniques for the analysis of event graphs arising from discrete event dynamic systems (DEDS). They have shown that several results from traditional theory of linear systems can be extended to the analysis of DEDSs. The key to this is the fact that the timing semantics of DEDS can be expressed as linear equations using a new algebra that replaces multiplication by addition and addition by the \( \max \) operation.

This paper not only proposes a rate analysis framework but also presents its underlying system model, theoretical basis, and implementation via RATAN. The main contributions of this paper can be summarized as follows:

- A two-level system model in which the top level models the processes in the system using
a process graph and the bottom level models each process using a sequencing graph. We introduce the concept of an enable signal to capture the synchronization between processes. This abstraction allows us to model systems with pipelined processes. The two-level model is useful in developing efficient algorithms for rate analysis.

- The notion of average execution rate of a process, defined as the inverse of the asymptotic mean of the sequence of time intervals between successive invocations of the process. We present a purely graph theoretic proof showing that the average execution rate is well defined for all processes in a finite process graph, irrespective of the initial start times of the processes. This proof also gives more accurate bounds on the periodicity of inter-execution times for a process than previously reported in the literature. We present efficient algorithms for computing bounds on the average execution rates of processes in a process graph.

- The rate analysis framework for using the bounds on process execution rates to interactively modify the design of an embedded system to satisfy all the rate constraints. Our rate analysis framework is implemented in a tool, RATAN. We demonstrate the utility of RATAN using an example.

The rest of the paper is organized as follows. Section 2 describes our two-level system model, focusing mainly on the process graph and the modeling of synchronization. Section 3 gives an overview of the rate analysis framework, defines the average execution rate and the main problems addressed in the paper. Each of the remaining sections examines one of these problems. Section 4 shows that the average execution rate is well defined and present efficient algorithms for computing bounds on the average execution rates of processes in a process graph. The proof of this result is fairly long and given in the appendix. Section 5 defines the notion of consistency of rate constraints and presents an efficient algorithm for consistency checking. Section 6 discusses how rate analysis can be used to debug rate constraint violations and redesign the embedded system to satisfy the rate constraints.

2 The System Model

We model an embedded system as a set of concurrent, interacting processes. Our model is a hierarchical two-level model. The top level is the process graph that captures the interaction between the processes. Each process is modeled using a sequencing graph [3] that represents the data and control flow dependences between the operations within a process. These sequencing graphs form the bottom level in our two-level representation.

In the process graph, \( G_P(V_P, E_P) \), each vertex \( p_i \) in \( V_P \) represents a process. An edge \((p_i, p_j)\) indicates that process \( p_i \) enables the execution of process \( p_j \) by sending an enable signal to process \( p_j \). Each edge, \((p_i, p_j)\), in the process graph is associated with a delay interval \( \Delta_{ij} = (d_{ij}, D_{ij}) \) that
bounds the time after the initiation of an execution of process $p_i$ when $p_j$ receives the enable signal from $p_i$. Let $\delta_{ij}$ be the actual delay after the start of an execution of $p_i$ when $p_j$ receives the enable signal from $p_i$, then $d_{ij} \leq \delta_{ij} \leq D_{ij}$. Figure 2 shows a simple process graph with two processes.

Each process represents an independent thread of control. We assume that all the processes are concurrently active. The execution semantics of our model is as follows. Let $x_i(k)$ denote the time at which process $p_i$ starts its $(k+1)$st execution. In particular, $x_i(0)$ is the initial start time of $p_i$. Then, the following rules govern the subsequent executions of the processes:

- **(Independence)** Initially, each process $p_i$ starts its first execution at $x_i(0)$ independently of the other processes.

- **(AND causality)** A new instance of process $p_j$ starts executing after all its predecessors in the process graph have issued enable signals for $p_j$. Consequently,

$$x_i(k) = \max_{p_j \in \text{Pred}(p_i)} \{ x_j(k-1) + \delta_{ij} \},$$

or since $\delta_{ij} \in [d_{ij}, D_{ij}]$,

$$\max_{p_j \in \text{Pred}(p_i)} \{ x_j(k-1) + d_{ij} \} \leq x_i(k) \leq \max_{p_j \in \text{Pred}(p_i)} \{ x_j(k-1) + D_{ij} \}$$

where $k > 0$ and $\text{Pred}(p_i)$ is the set of predecessors of $p_i$ in the process graph.

### 2.1 Highlights

Our process graph model and the associated enable signal based communications/synchronization is general enough to represent most embedded system specifications. Here are some of the main highlights of our model:

- **Ability to model several different synchronization mechanisms:** The basic primitive for synchronization in our model is an enable signal from one process to another. A process starts executing once it has received an enable signal from all its predecessors in the process graph. This allows us to model systems with several different kinds of synchronization and communication paradigms in a unified manner. Some examples are:
1. **Operation-based Enable Signal**: Such an enable signal is issued by a process when it executes a particular operation in its sequencing graph. A vertex in the sequencing graph whose execution results in the generation of an enable signal is called an *enabling vertex*. The delay between the start of the execution of a process and the generation of an operation-based enable signal can vary for different executions of the process because it may be value dependent.

2. **Time-based Enable Signal**: Such an enable signal is generated after a fixed time following the start of the execution of a process. It is not associated with the execution of a particular operation in the process. Such enable signals are useful in modeling a pipelined process for which a new execution starts after a fixed delay.

- **Modeling Pipelined Processes**: A new instance of a pipelined process can start executing before its previous execution has finished. Our process graph model can handle such pipelined processes using a self-loop with delay less than the execution time of the process. Thus, the process graph model allows us to model multiple parallel and pipelined execution instances of a process. If the delay associated with a self-loop is larger than the execution time of the process then a new execution of the process cannot begin until the previous execution has terminated. Consequently, the process cannot be pipelined.

### 2.2 Limitations

In order to develop efficient algorithms for rate analysis, we need to impose some restrictions on the way enable signals can be issued by processes. These are not inherent to our process graph model, but are limitations introduced due to the algorithmic techniques we use for rate analysis. Basically, these limitations arise due to the requirement that *one execution of a process should generate exactly one enable signal for each of its successors in the process graph*. This results in the following restrictions on the generation of enable signals:

- **Enable signals should not be generated from inside loops in the sequencing graph**: Since loop bodies can be executed multiple times for one execution of a process, if there are enable vertices in a loop body then multiple enable signals would be generated for the same successor process in one execution. Rate analysis for such systems is an interesting problem that would require new techniques.

- **Enable signals should not be generated from conditionally executed parts of the sequencing graph for a process**: The presence of enable vertices in conditional branches would imply that when the conditional branch is not executed no enable signal would be generated for some successor process. At first glance, this appears to be a severe restriction on the expressive power of our process graph model. However, any deadlock free, strongly
connected process graph that uses our synchronization semantics must have this property. The possibility of deadlock in the presence of conditional enable signals is illustrated in the following example. It is possible to have conditional enable signals without causing deadlock if they span across two different strongly connected components in the process graph.

![Graph](image)

Figure 3: Two processes (with the corresponding sequencing graphs shown in the dotted regions). Process $p_1$ issues a conditional enable signal for process $p_2$.

Example 2.1. Consider the processes shown in Fig. 3. Process $p_1$ issues an enable signal for process $p_2$ from an enabling vertex in the FALSE branch of a conditional branch and process $p_2$ issues an unconditional enable signal for process $p_1$. If the computation ever takes the TRUE branch in process $p_1$, no enable signal is issued for $p_2$. Thus, $p_2$ is not invoked and consequently $p_1$ is not invoked, leading to a deadlock. The same scenario can occur in any strongly connected process graph that has enable vertices in conditional branches. $\square$

3 Rate Analysis Framework

In this section we define average execution rate, give a precise formulation of the problems that are addressed in the paper and then give a brief overview of our rate analysis framework.

3.1 Average Execution Rate

The rate of execution of a process is defined as the number of executions of the process per unit time. Let

$$x_i(k) = \text{time at which process } p_i \text{ starts executing for the } (k+1)\text{th time.} \quad (1)$$

We are concerned with systems that exhibit infinite behavior. Such systems behave in a cyclic fashion in steady state after the initial transients. The following limit has been used by researchers as a standard way of summarizing the performance of a process in such a system (see [1], [12]). We
use the same definition and define the average inter-execution time, $T_i$, and the average execution rate, $r_i$, of a process $p_i$ as

$$T_i = \lim_{n \to \infty} \frac{\sum_{k=0}^{n-1} x_i(k+1) - x_i(k)}{n} = \lim_{n \to \infty} \frac{x_i(n)}{n}, \quad \text{and} \quad r_i = T_i^{-1},$$

(2)

if the above limit exists. We show in Section 3.1 that for a finite process graph with finite edge delays, the above limit exists and can be efficiently computed. Thus, in this paper we focus on asymptotic rates of execution. Note that the time between successive invocations of a process (inter-execution time) is not constant, hence the instantaneous execution rate of a process keeps changing. The following example illustrates our definition of average rate of execution.

**Example 3.1.** Consider the process graph shown in Fig. 2. Assuming that the processes start executing at time 0, and the edge delays equal their lower bounds ($\delta_{12} = 1, \delta_{21} = 2$), the sequence of initiation times of the processes are:

Process $p_1 : 0, 2, 3, 5, 6, 8 \cdots$

Process $p_2 : 0, 1, 3, 4, 6, 7 \cdots$

Thus, the sequence of inter-execution time for both the processes are:

Process $p_1 : 2, 1, 2, 1 \cdots$

Process $p_2 : 1, 2, 1, 2 \cdots$

and the average inter-execution time is $3/2$. Thus, using the definition of average rate of execution given above, the execution rates of both the processes in this example are $2/3$. Notice that the limit in our definition of the average rate of execution exists in this case because both the sequences of inter-execution times are periodic (with period 2).

3.2 Rate Constraints

We assume that the designer specifies upper and lower bounds on the average execution rates of all the processes in the system. Thus, each process, $p_i$, in the process graph is associated with a constraint interval, $I_i = [L_i, U_i]$. Constraint intervals for processes in a process graph can be arbitrary (since they are specified independently for each process by the designer). If there is no lower bound constraint on the rate of a process then $L_i = 0$, and if there is no upper bound constraint then $U_i = \infty$. For the specified rate constraints to be satisfied, the computed execution rate for each process must lie within its constraint interval, that is

$$L_i \leq r_i \leq U_i.$$

Usually the environment in which an embedded system works imposes rate constraints on various components (processes) of the system. So the rates at which certain events happen in the environment place execution rate constraints on the components of the embedded system that are supposed to process those events.
3.3 Problem Formulation

Given a description of an embedded system (process graph and the sequencing graphs of the processes), along with the associated rate constraints, the following problems need to be addressed in the rate analysis framework:

1. **Delay Analysis:** Compute bounds on the execution times of operations in the sequencing graphs, and use these estimates to find bounds on the delays of the edges in the process graph. Delay analysis requires computation of bounds on the time after the initiation of a process when it issues an enable signal for another process, and the estimation of the communication delay between the process generating the enable signal and the one receiving it. This problem has been addressed by [5], [10] and is not the focus of this paper.

2. **Consistency Checking of Rate Constraints:**
   This problem arises because the designer usually specifies the rate constraints independently for each process. A set of rate constraints is said to be *inconsistent* if they cannot be simultaneously satisfied for a given process graph topology, irrespective of the delay intervals on the process graph edges. Thus, if a set of rate constraints is inconsistent, the computed rate intervals can never satisfy all the rate constraints. Since consistency of a set of rate constraints is independent of the actual delay intervals in the process graph, we can state the consistency checking problem as follows:

   *Given a process graph and the rate constraint intervals associated with each process in the process graph, are the constraint intervals consistent?*

   Section 5 describes our consistency checking algorithm.

3. **Rate Analysis:**
   The rate analysis problem is stated as: *Given a description of an embedded system as a process graph and the associated sequencing graphs, find upper and lower bounds on the average execution rate of each process in the process graph.*

   So, rate analysis finds an interval \([r_l(p_i), r_u(p_i)]\) for each process \(p_i\) such that the average rate of execution of the process is guaranteed to lie in this interval. The rate constraints on \(p_i\) are said to be *satisfied* if the rate interval computed by rate analysis is contained in \([L_i, U_i]\). If the computed rate interval is not contained in \([L_i, U_i]\), then one or both of the rate constraints of \(p_i\) are *violated*. In Section 4 we present our algorithms for rate analysis.

4. **Rate Constraint Debugging and Process Redesign:** If rate analysis finds that some of the rate constraints are violated, the designer needs to redesign certain processes and/or change the process graph topology by altering the manner in which the processes communicate. The rate analysis tool needs to give adequate information about the cause of the rate constraint violation to help in this step. We discuss some possible approaches in Section 6.
Figure 4: An overview of our rate analysis framework.

Figure 4 shows our framework for rate analysis. The design specification is first translated from a high level description to a process graph along with the associated sequencing graphs. First, the rate constraints are checked for consistency. If the rate constraints are consistent, rate analysis is performed on the process graph. If all the computed rate intervals are contained in the intervals defined by the rate constraints, then the system satisfies all the rate constraints and no redesign is required. If a rate constraint is violated, the tool gives the user the reasons for the violation and this information can be used to redesign the relevant processes. Although consistency analysis of the rate constraints precedes rate analysis in the flow, we will discuss rate analysis before consistency analysis in this paper. This is motivated by the fact that many of the results required for describing our consistency checking algorithm are more natural to discuss in the context of rate analysis.
4 Rate Analysis

In this section we show that our notion of average execution rate is well defined and present algorithms for rate analysis of process graphs. For the sake of clarity, some of our results are stated for strongly connected process graphs. However, we do generalize our results to arbitrary process graphs and there is no limitation on the topology of the process graphs that our rate analysis algorithms can handle.

4.1 Existence of Average Execution Rate

In order to prove that our asymptotic definition of average execution rate is well defined, we first examine the case when the process graph is strongly connected (i.e., there is a path from each process $p_i$ to all other processes in the graph) and there is a unique delay, $\delta_{ij}$, associated with each edge in the process graph. The extension to the case when the process graph has several strongly connected components and the process graph edges have delay intervals associated with them will become clear when we discuss the algorithms for rate analysis for such process graphs in subsection 4.2.

We define the delay of a cycle $C$ in the process graph as

$$d(C) = \sum_{(i,j) \in C} \delta_{ij}. \quad (3)$$

The number of edges in a cycle $C$ is denoted by $|C|$. The mean delay of a cycle $C$ is given by

$$\frac{d(C)}{|C|}. \quad (4)$$

The maximum mean cycle delay is denoted by $\lambda$. A cycle is said to be critical if it has the maximum mean delay among all the cycles in the graph.

The following theorem establishes that our definition of execution rate in Eq. 2 is well defined, and can be related to the maximum mean cycle in the process graph. Weaker forms of this theorem can be found in [1, 2, 12, 13].

**Theorem 1.** Consider a strongly connected process graph and let $x_i(k)$, $k \geq 0$ be the sequence of time instances at which process $p_i$ executes. Then there exists some $N \geq 0$ such that the following are true:

1. The sequence $x_i(k) - x_i(k-1)$, $k \geq N$ is periodic with period equal to $L$, where $L$ is equal to the least common multiple of the lengths of all the critical cycles in the process graph.

2. For any $q \geq N$,

$$\frac{\sum_{j=1}^{q+L-1} x_i(j+1) - x_i(j)}{L} = \lambda,$$

where $\lambda$ is the maximum mean cycle delay in the process graph.
3. The average inter-execution time is well defined and

\[
T_i = \lim_{n \to \infty} \frac{\sum_{k=0}^{n-1} x_i(k+1) - x_i(k)}{n} = \lambda.
\]

The proof of this theorem is based on characterization of critical paths in the process graph and is given in the Appendix. The main idea used in the proof is that \(x_i(k)\) (the time when the \((k+1)\)th execution of process \(p_i\) starts) is determined by the delay of the longest path in the process graph with exactly \(k\) edges that ends at process \(p_i\), and that this path can be characterized using the maximum mean delay cycles in the process graph.

Notice that \(T_i\) is not average in the probabilistic sense. As this theorem shows, the inter-execution time of a process in a process graph may not stay constant but a sequence of its successive inter-execution times repeats itself with a certain period and \(T_i\) is the mean of the inter-execution times within such a sequence.

4.2 Algorithms for Rate Analysis

Theorem 1 shows that for a process graph with one strongly connected component (SCC), if all the edge delays are exact then the execution rate of all the processes is equal to the inverse of the maximum mean cycle delay in the process graph. In this section, we first show how to compute upper and lower bounds on execution rates of processes in the presence of uncertainty in the edge delays. Then, we give an efficient algorithm for computing execution rates for the case when the process graph is strongly connected (or equivalently has one SCC), and finally, we extend our algorithms to the case when the process graph has multiple SCCs. This case handles any process graph.

Handling Delay Intervals

We assume that we have a process graph, \(G_P\), with a delay interval associated with each edge. The following theorem allows us to compute the rate interval, \([r_l(p_i), r_u(p_i)]\), for each process \(p_i\) in the process graph from the delay intervals of the edges.

**Theorem 2.** (Monotonicity Theorem) Increase (decrease) in the delay of an edge cannot increase (decrease) the average execution rate of any process.

**Proof.** From Theorem 1, we know that the average execution rate of a process in a strongly connected process graph is equal to the inverse of the maximum mean cycle delay in the process graph. Increasing (decreasing) the delay of an edge in the process graph increases (decreases) the mean delay of any cycle in the process graph. It follows that increasing (decreasing) the delay of an edge in the process graph cannot increase (decrease) the average execution rate of any process. \(\square\)
From the Monotonicity Theorem it follows that we can compute \( r_u(x) \) by setting all the edge delays to their lower bounds and \( r_l(x) \) by setting all the edge delays to their upper bounds. Hence, if the process graph is strongly connected then all the nodes have the same rate interval \([\lambda_u^{-1}, \lambda_l^{-1}]\) where \( \lambda_l \) and \( \lambda_u \) are the maximum mean cycle delays of the process graph computed by setting the edge delays of all the edges to their upper bounds and lower bounds respectively. Hence, the rate interval for a strongly connected process graph are computed by two invocations of the algorithm for computing the maximum mean delay cycle in a graph.

Notice that the lower and upper rate bounds are the tightest bounds on the average execution rate but they do not bound the best and worst case execution rates. If the latter are desired, our rate analysis framework would still apply; however, the average inter-execution time, which is used to compute the average execution rate, should be replaced with the best and worst case inter-execution times. Loose bounds on them are simply the minimum and maximum arc delays in the process graph. The tightest bounds on them are given by the time separation algorithm of [7].

**Rate Analysis : Single SCC**

Using the result in Theorem 1, we need to compute the maximum mean cycle delay to compute the average execution rate of processes in a SCC. The following theorem due to Karp [8] allows the design of an efficient algorithm for finding the maximum mean cycle delay. Its proof can be found in [1] (page 47).

**Theorem 3.** Consider a strongly connected graph \( G(V, E) \) with \( n \) nodes. Let \( s \) be an arbitrary chosen vertex. For every \( v \in V \), and every nonnegative integer \( k \), define \( D_k(v) \) as the maximum delay of a path of length \( k \) from \( s \) to \( v \); if no such path exists, then \( D_k(v) = -\infty \). Then the maximum cycle mean \( \lambda \) of \( G \) is given by

\[
\lambda = \min_{v \in V} \max_{0 \leq k < n} \frac{D_n(v) - D_k(v)}{n - k}.
\]

Thus, for a strongly connected process graph the average execution rates of all the processes are equal to the inverse of the maximum mean cycle delay, and they can be computed using Karp’s characterization of the maximum mean cycle delay in \( O(|V| \cdot |E|) \) time.

**Rate Analysis : Multiple SCCs**

We now consider a process graph that has several strongly connected components. We define the *component graph* of the process graph to be the graph in which there is a vertex for each SCC and an edge from \( u \) to \( v \) if and only if there is an edge from a vertex in the SCC represented by \( u \) to a vertex in the SCC represented by \( v \) in the process graph. Note that the component graph is a directed acyclic graph. The following is the key observation on which the algorithm for computing rate intervals for a process graph with multiple SCCs hinges.
RATEANALYSIS(G)

Find the maximal SCCs in G
for each SCC, C
    r_i(C) = RATE(C : t(x) = t_u(x) \forall x \in C)
    r_u(C) = RATE(C : t(x) = t_l(x) \forall x \in C)

Construct the component graph : GC(VC, EC)
    r_l(C_j) = \min\{r_l(C_j), \min_{(C_i, C_j) \in EC}\{r_l(C_i)\}\}
    r_u(C_j) = \min\{r_u(C_j), \min_{(C_i, C_j) \in EC}\{r_u(C_i)\}\}
end RATEANALYSIS

Figure 5: Algorithm for rate analysis for a process graph with multiple SCCs. RATE is the algorithm for computing $\lambda^{-1}$ using Karp’s characterization of $\lambda$.

Theorem 4. Let P and C be two maximal strongly connected components in a process graph with some edges from vertices in P to vertices in C (so P is a “producer” and C is a “consumer”). Let $[r_l(P), r_u(P)]$ and $[r_l(C), r_u(C)]$ be the rate intervals for P and C respectively, computed assuming that there are no edges between P and C. Then the actual rate interval for C is $[r_l(C), r_u(C)]$ where

\[
    r_l(C) = \min\{r_l(P), r_l(C)\}, \\
    r_u(C) = \min\{r_u(P), r_u(C)\}.
\]

Proof. Any cycle in P has a path to any process $p_i \in C$ through edges connecting P to C. Consequently, the k critical paths for $p_i$ (for sufficiently large values of k) are determined by the larger of the maximum mean delay cycle in P and C (refer to the proof of Theorem 1). Hence, the average execution rate of any process in C is the minimum of the rate computed assuming that there are no dependencies from P to C (this is the inverse of the maximum mean cycle delay in C) and the average execution rate of any process in P (this is the inverse of the maximum mean cycle delay in P). These observations along with the Monotonicity Theorem prove this theorem.

Using the above theorem we develop the algorithm in Fig. 5 for rate analysis of a process graph with multiple SCCs. The following example illustrates the steps involved in the rate analysis for process graphs.

Example 4.1. Consider the process graph shown in Fig. 6. The delay intervals for the edges are shown. Notice that we do not associate a delay interval with the edge connecting the two strongly connected components of the graph. This is because such inter-SCC edges are not part of any cycles in the process graph, and consequently their delay does not affect any execution rates. Let us now consider the steps involved in the rate analysis of the process graph in Fig. 6.
For $SCC_1$:
For computing $\eta$, set all the edge delays to their upper bounds.

$$\lambda_t = \text{maximum mean delay cycle in } SCC_1$$

$$= \max \left\{ \frac{20 + 18 + 6}{3}, \frac{6 + 20 + 10 + 20}{4} \right\}$$

$$= \max \left\{ \frac{44}{3}, 14 \right\}$$

$$= 14.67$$

The critical cycle is $(p_1 \rightarrow p_2 \rightarrow p_4 \rightarrow p_1)$ and

$$r_t = (14.67)^{-1} = 0.068$$

For computing $\nu$, we use the lower bounds on the edge delays.

$$\lambda_u = \text{maximum mean delay cycle in } SCC_1$$

$$= \max \left\{ \frac{10 + 9 + 2}{3}, \frac{2 + 4 + 3 + 10}{4} \right\}$$

$$= \max \left\{ 7, 4.75 \right\}$$

$$= 7$$

Thus,

$$r_u = (7)^{-1} = 0.142$$

Hence, the rate interval for $SCC_1$ is $[0.068, 0.142]$.

For $SCC_2$:

$$\eta = \left[ \max \left\{ \frac{20 + 8 + 4}{3}, \frac{10 + 5 + 6 + 8 + 4}{5} \right\} \right]^{-1} = 0.094$$

and

$$r_u = \left[ \max \left\{ \frac{7 + 5 + 1}{3}, \frac{4 + 3 + 3 + 5 + 1}{5} \right\} \right]^{-1} = 0.231$$
So, the rate interval for \( SCC_2 \) is \([0.094, 0.231]\). Notice that the rate intervals of the two SCCs overlap. The rate analysis was carried out assuming that the two SCCs are completely disjoint. The rates in \( SCC_1 \) are not affected by the edge from \( SCC_1 \) to \( SCC_2 \), however, the rate interval for \( SCC_2 \) needs to take into consideration the rate interval of the “producer” SCC. In fact, the actual rate interval for \( SCC_2 \) is

\[
\left[ \min\{0.094, 0.068\}, \min\{0.231, 0.142\} \right] = \left[ 0.068, 0.142 \right]
\]

Thus, the rate interval of \( SCC_2 \) is the same as that of \( SCC_1 \). Notice that for the sake of clarity of exposition, we have computed the maximum mean cycle delay using explicit enumeration of the cycles in this example. In our implementation, instead of explicit cycle enumeration, we use Karp’s characterization of maximum mean cycle delay to compute the rates. □

5 Consistency Checking

A set of rate constraints is said to be inconsistent if they cannot be simultaneously satisfied for a given process graph topology, irrespective of the delay intervals on the process graph edges. Thus, if a set of rate constraints is inconsistent, the computed rate intervals can never satisfy all the rate constraints. Note that a consistent set of constraints may not be satisfied because some of the computed rate intervals may not be contained in the corresponding constraint intervals. Theorem 4 gives sufficient conditions for a set of rate constraints to be inconsistent. Before proceeding further with the consistency analysis, we need a few definitions. Consider \( n \) rate constraint intervals in the form of \( I_i = [L_i, U_i] \) for \( 1 \leq i \leq n \).

- Their minimum is \( \min_i \{I_i\} = [\min_i \{L_i\}, \min_i \{U_i\}] \).
- Their intersection is \( \bigcap_i \{I_i\} = [\max_i \{L_i\}, \min_i \{U_i\}] \).
- Given intervals \( I_i \) and \( I_j \), \( I_i < I_j \), if \( U_i < L_j \).

Now consider a strongly connected component in the process graph, \( C_j \). Then

- The intersection of the rate intervals of the processes in \( C_j \) is \( I_j^C = \bigcap_{p_i \in C_j} I_i \).
- The minimum of all the intersections that belong to \( C_j \) and all of its predecessors is

\[
I_j^{\min} = \min \{ I_j^C, \min_{C_i \in \text{pred}(C_j)} \{ I_i^{\min} \} \}, \tag{5}
\]

and for a component with no predecessors, \( I_j^{\min} = I_j^C \). More intuitively, \( I_j^{\min} \) is obtained by propagating the \( I_j^C \) intervals in the component graph in topological order (this is similar to the propagation of rate intervals during rate analysis).

**Theorem 5.** A set of rate constraints, \( S \), that define the constraint intervals \( I_i \) for each process \( p_i \) in a process graph \( G_P \), is inconsistent if any of the following conditions is satisfied.
1. There exists a strongly connected component, \( C_j \), of the process graph such that
\[
I_{j}^\cap = \Phi,
\]

2. There exists a component \( C_j \) for which
\[
I_{j}^{\\min} < I_{j}^\cap.
\]

Proof.

1. From Theorem 1, Theorem 2 and Theorem 4, we know that the computed rate interval of all processes in a strongly connected component, \( C_j \), of \( G_P \) is identical. Let \([r_l(C_j), r_u(C_j)]\) be the computed rate interval. If \( I_{j}^\cap = \Phi \), then there must be some process \( p_i \in C_j \) for which \([r_l(C_j), r_u(C_j)]\) is not contained in \( I_i \) (otherwise the intersection of the \( I_i \)'s contains \([r_l(C_j), r_u(C_j)]\) and is not empty). Hence, the rate constraints for \( p_i \) are violated. Since the above argument does not depend on any particular choice of the delay ranges of edges in \( G_P \), it follows that if \( I_{j}^\cap = \Phi \) for any \( C_j \), then \( S \) is inconsistent.

2. From Theorem 4, we know that the average execution rate of a SCC is constrained by the average execution rate of its predecessors in the component graph. In particular, the rate interval for a process is the minimum of its own rate interval (assuming no dependencies across SCCs), and the rate intervals of its predecessors. If we assume that rate constraints are satisfied for all SCCs that are predecessors of \( C_j \), then the largest possible rate interval for \( C_j \) is equal to \( I_{j}^{\\min} \). Further, if rate constraints are satisfied for all the processes in \( C_j \), then the largest possible rate interval for \( C_j \) is \( I_{j}^\cap \). Consequently, if
\[
I_{j}^{\\min} < I_{j}^\cap,
\]
then any assignment of delay intervals that satisfies the rate constraints at all the predecessors of \( C_j \) will violate the rate constraints at some process in \( C_j \). Hence, the constraint set \( S \) is inconsistent.

The above theorem gives us a characterization of a set of inconsistent rate constraints. Computing \( I_{j}^\cap \) for all the SCCs in the process graph takes time linear in the size of the process graph. Hence, the first condition in this theorem can be checked in linear time. Also, the component graph can be constructed in linear time, and \( I_{j}^{\\min} \) can be computed by a topological traversal of the component graph. So, the second condition can also be checked in linear time. Thus, Theorem 5 yields a linear time algorithm for checking whether the specified rate constraints are consistent.
Figure 7: (a) Process graph used in Example 5.1. (b) The corresponding component graph.

**Example 5.1.** Consider the process graph along with the constraint intervals for all the processes shown in Fig. 7.

\[
I_1^0 = [0.5, 1.0] \cap [0.75, 12.0] \\
= [0.75, 1.0] \\
I_2^0 = [10.0, 30.0] \cap [5.0, 25.0] \cap [7.0, 20.0] \\
= [10.0, 20.0]
\]

Also,

\[
I_1^{\text{min}} = I_1^0 = [0.75, 1.0],
\]

and

\[
I_2^{\text{min}} = \min([0.75, 1.0], [10.0, 20.0]) = [0.75, 1.0].
\]

Since [0.75, 1.0] < [10.0, 20.0], the given rate constraints are inconsistent due to the condition 2 in Theorem 5. More intuitively, the rate constraints on process \( p_1 \) force the constraint interval for \( SCC_1 \) to be at most 1. Further, since the constraint intervals for all the processes in \( SCC_2 \) are strictly larger than 5, the constraint interval for \( SCC_2 \) has a lower bound of 5. Hence, condition 2 in Theorem 5 is satisfied and the rate constraints are inconsistent. □

## 6 Rate Constraint Debugging and Process Redesign

Our algorithms for rate analysis and for checking consistency of rate constraints can be used as a part of a system for interactive debugging of embedded system designs. Fig. 4 shows one such framework. The design specification is first translated from a high level description to a process graph along with the associated sequencing graphs. First, the rate constraints are checked for consistency. If the rate constraints are consistent, rate analysis is performed on the process graph. If all the computed rate intervals are contained in the intervals defined by the rate constraints, then the system satisfies all the rate constraints and no redesign is required.
If a rate constraint is violated, the system gives the user the reasons for the violation along with the critical cycles responsible for the violation. This information can be used to redesign the processes that are involved in the critical cycles. If an upper bound rate constraint is violated for process \( p_i \), then it means that \( p_i \) executes faster than required by the constraint. This situation is easier to remedy because additional delay can be introduced on some of the process graph edges to slow down the execution rate of \( p_i \). In this case, the “best” places where additional delay can be introduced to avoid the constraint violation can be identified by the rate analysis program. If a lower bound rate constraint is violated, the program outputs all the critical cycles in the process graph that affect the rate at the process where the constraint violation occurs. Potential candidates for pipelining are identified by the self-loops in the critical cycles responsible for rate constraint violation. Pipelining will reduce the delay of the self-loop and possibly avoid the constraint violation. In general, the tool outputs the bottlenecks in the current design (in the form of critical cycles) and the designer then needs to redesign the processes involved to reduce the delay intervals on the edges involved in the critical cycles. In some cases it may even be necessary to alter the communication and synchronization between the component processes to change the topology of the process graph.

All our algorithms for checking the consistency of rate constraints and for rate analysis have been implemented in C++ as a tool called RATAN. The following example illustrates the utility of this interactive rate analysis framework in redesigning a system to meet specified rate constraints.

Example 6.1. Consider the process graph in Fig. 8 containing 18 processes and 32 edges. The interval of real numbers on an edge is the delay interval associated with that edge. This process graph has 6 strongly connected components labeled \( C_0, C_2, \ldots C_5 \), and grouped with a dashed circle. The darker lines between components are also edges in the process graph. Since the delay intervals on these edges are not used in rate analysis, they are not shown in the figure. There can be more than one edge between two components, but only one is enough to show the precedence relation between a pair of components. We will now trace the steps involved in interactive rate analysis (as shown in Fig. 4) on the process graph in Fig. 8.

<table>
<thead>
<tr>
<th>SCC</th>
<th>( r_l )</th>
<th>( r_u )</th>
<th>Critical cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td>( C_0 )</td>
<td>0.043</td>
<td>0.500</td>
<td>0 ( \rightarrow ) 0</td>
</tr>
<tr>
<td>( C_1 )</td>
<td>0.167</td>
<td>0.250</td>
<td>1 ( \rightarrow ) 2 ( \rightarrow ) 1</td>
</tr>
<tr>
<td>( C_2 )</td>
<td>0.200</td>
<td>0.500</td>
<td>3 ( \rightarrow ) 5 ( \rightarrow ) 6 ( \rightarrow ) 7 ( \rightarrow ) 4 ( \rightarrow ) 3</td>
</tr>
<tr>
<td>( C_3 )</td>
<td>0.050</td>
<td>0.083</td>
<td>8 ( \rightarrow ) 8</td>
</tr>
<tr>
<td>( C_4 )</td>
<td>0.067</td>
<td>0.200</td>
<td>12 ( \rightarrow ) 14 ( \rightarrow ) 13 ( \rightarrow ) 12</td>
</tr>
<tr>
<td>( C_5 )</td>
<td>0.059</td>
<td>0.100</td>
<td>17 ( \rightarrow ) 17</td>
</tr>
</tbody>
</table>

Table 1: Step 2 - before rate propagation

1. **Step 1:** [Check if the given rate constraints are consistent] For simplicity, Fig. 8 does not show the rate constraints for each process. Suppose that the given rate constraints
Figure 8: The process graph used in Example 6.1.

are consistent, and the constraint interval for C4 is \([0.05, 0.10]\) (this is the intersection of the constraint intervals of all the processes in C4).

2. **Step 2:** [Rate analysis] Tables 1 and 2 give the computed rate bounds for each component before and after rate propagation. These tables also show the critical cycle(s) that determine a particular rate bound. Notice that the critical cycle that determines the lower bound can be different from the one that determines the upper bound (for example C2 in Table 2). Also, after rate propagation, the cycles that determine the rate bound for a component may be contained in a predecessor of the component in the component graph.

3. **Step 3:** [Check if the computed rate constraints are satisfied] The lower bound rate constraint for C4 is not satisfied since the computed rate interval is \([0.043, 0.083]\), and the constraint interval is \([0.05, 0.10]\). This means that we need to speed up some processes on the critical cycle that constrains the lower bound to be less than 0.05. Assume that all other rate constraints are satisfied.

4. **Steps 4, 5:** [Pipeline a process in a critical cycle responsible for violating a rate constraint and re-do rate analysis] The critical cycle that determines the rate lower bound of 0.043 for C4 is \(0 \rightarrow 0\). This suggests that by speeding up process 0, we can avoid violation of the rate constraint. Indeed, if the process 0 is pipelined and the delay on the \(0 \rightarrow 0\) self-loop is reduced to \([2, 20]\), the rate interval for C4 becomes \([0.05, 0.083]\) and the violated rate constraint is now satisfied. Tables 3 and 4 show the rate intervals for the modified process graph (after pipelining process 0) before and after rate propagation respectively.

5. **Step 6:** [Check if the computed rate constraints are satisfied] We now check if all the rate constraints are satisfied. In this case, all the rate constraints are satisfied because
the only effect of pipelining process 0 is to increase the rate lower bounds for components C0, C2, C3, C4 and C5. If there were still some violated rate constraints, we would have incrementally pipelined processes on critical paths to reduce delays on critical self-loops. If even pipelining fails to prevent constraint violation, more drastic redesign of the processes on the critical paths is called for. In fact, even the communication/synchronization patterns of the processes may need to be changed in order to alter the topology of the process graph.

This example illustrates that using our interactive rate analysis, we can explore bottlenecks in the current design and interactively modify the design to obtain one that satisfies all the rate constraints. Some of the modifications suggested by the interactive system are non-trivial to discover otherwise. For instance, to remove the constraint violation at C4 consisting of processes 12 through 16, process 0 needs to be pipelined. □
7 Conclusions

In this paper we have examined the problem of estimating execution rates of concurrent processes modeled as sequencing graphs that interact through process level synchronization. We show that the average execution rate can be efficiently computed using a relation between average execution rate and the critical cycles in the process graph. A linear time algorithm was developed for checking the consistency of a set of rate constraints. We have developed a tool, RATAN, for using our rate analysis algorithms for debugging violations of rate constraints.

Our future plans are to seek effective integration of RATAN in the design of embedded systems. Another interesting area of research is to study the rate analysis problem under more general timing semantics, such as allowing some processes to start executing as soon as they receive one enable signal, i.e., OR causality. We also plan to study the rate analysis problem when enable vertices are allowed inside loops and conditional branches in sequencing graphs.

Acknowledgments

The first author’s work was partially supported by NSF under grant MIP 92-22408 and by Silicon Graphics. We also acknowledge support from NSF Career Award 95-01615 and a grant from NSF Engineering Research Center ECD 89-43166.

Appendix: Proof of Theorem 1

In order to prove Theorem 1, we need a few preliminary results.

We define the \( k \)-critical paths for process \( p \) to be the paths whose delay determines \( x_i(k) \). A

<table>
<thead>
<tr>
<th>SCC</th>
<th>( r_i )</th>
<th>( r_n )</th>
<th>Critical cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td>C0</td>
<td>0.050</td>
<td>0.500</td>
<td>0 ( \rightarrow ) 0</td>
</tr>
<tr>
<td>C1</td>
<td>0.167</td>
<td>0.250</td>
<td>1 ( \rightarrow ) 2 ( \rightarrow ) 1</td>
</tr>
<tr>
<td>C2</td>
<td>0.050</td>
<td>0.250</td>
<td>0 ( \rightarrow ) 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 ( \rightarrow ) 2 ( \rightarrow ) 1</td>
</tr>
<tr>
<td>C3</td>
<td>0.05</td>
<td></td>
<td>0 ( \rightarrow ) 0</td>
</tr>
<tr>
<td>C3</td>
<td></td>
<td>0.083</td>
<td>8 ( \rightarrow ) 8</td>
</tr>
<tr>
<td>C4</td>
<td>0.05</td>
<td></td>
<td>0 ( \rightarrow ) 0</td>
</tr>
<tr>
<td>C4</td>
<td></td>
<td>0.083</td>
<td>8 ( \rightarrow ) 8</td>
</tr>
<tr>
<td>C5</td>
<td>0.05</td>
<td></td>
<td>0 ( \rightarrow ) 0</td>
</tr>
<tr>
<td>C5</td>
<td></td>
<td>0.083</td>
<td>8 ( \rightarrow ) 8</td>
</tr>
</tbody>
</table>

Table 4: Step 4.5 - after rate propagation
$k$-critical path for $p_i$ is a maximum delay path with exactly $k$ edges ending at $p_i$. For any $k$-critical path $\pi : p_j = v_0, v_1, \cdots, v_k = p_i$, the delay of the path is $$d(\pi) = x_j(0) + \sum_{i=0}^{k-1} \delta_{v_i, v_{i+1}} = x_i(k).$$

**Lemma 1** The $k$-critical paths for all processes remain unchanged when we change the delay on all the edges using the transformation $\delta'_{ij} = \delta_{ij} + x_j(0) - x_i(0)$ and the initial vector to $x'(0) = [00 \cdots 0]$. Further, this transformation also leaves the critical cycles and the maximum mean cycle delay in the process graph unchanged.

**Proof.** The length of a path $\pi : p_j = v_0, v_1, v_2, \cdots, v_k = p_i$ using the new edge costs is

$$d'(\pi) = \sum_{i=0}^{k-1} \delta'_{v_i, v_{i+1}}$$

$$= \sum_{i=0}^{k-1} (\delta_{v_i, v_{i+1}} + x_v(0) - x_{v_{i+1}}(0))$$

$$= \sum_{i=0}^{k-1} \delta_{v_i, v_{i+1}} + x_j(0) - x_i(0)$$

$$= d(\pi) - x_i(0)$$

Since, the length of all the paths ending at $p_i$ gets decreased by $x_i(0)$, the set of $k$-critical paths remains unchanged.

For any cycle $C : p_j = v_0, v_1, v_2, \cdots, v_{k-1}, v_k = p_j$,

$$d'(C) = \sum_{i=0}^{k-1} \delta'_{v_i, v_{i+1}}$$

$$= \sum_{i=0}^{k-1} (\delta_{v_i, v_{i+1}} + x_v(0) - x_{v_{i+1}}(0))$$

$$= d(C) + \sum_{i=0}^{k-1} (x_{v_i}(0) - x_{v_{i+1}}(0))$$

$$= d(C) + (x_{v_0}(0) - x_{v_k}(0))$$

$$= d(C)$$

Since the delay of all the cycles remains unchanged, it is clear that the critical cycles and maximum mean cycle delay also remain unchanged. $\blacksquare$

Henceforth, we will assume that the transformation on edge delays in Lemma 1 has been performed and the vector of initial start times is zero. The following three lemmas establish
important characteristics of “sufficiently long” $k$-critical paths that enable us to prove the theorem establishing the relation between the average rate of execution and the maximum mean cycle delay of the process graph. In the following lemmas we assume that the process graph has $m$ critical cycles : $C_1, C_2, \cdots, C_m$ each with mean delay $\lambda$.

**Lemma 2** A $k$-critical path, $\pi$, for process $p_i$ that includes a vertex of a critical cycle $C_j$ has less than $\text{lcm}(|C_j|, |C|)/|C|$ occurrences of any non-critical cycle $C$. Further, there exists a $k$-critical path for $p_i$ that has fewer than $\text{lcm}(|C_j|, |C'|)/|C'|$ occurrences of any cycle $C'$ other than $C_j$.

**Proof.** Let $S_j$ be the delay of the critical cycle $C_j$ (this is the sum of the edge delays of the edges in $C_j$) and let $S$ be the delay of the non-critical cycle $C$. Since $C$ is not critical, its mean delay is less than that of $C_j$. Hence,

$$\frac{S_j}{|C_j|} > \frac{S}{|C|},$$

or

$$\frac{\text{lcm}(|C_j|, |C|) \cdot S_j}{|C_j|} > \frac{\text{lcm}(|C_j|, |C|) \cdot S}{|C|}. \quad (6)$$

Consider any path ending at $p_i$ passing through a vertex of $C_j$ that has more than $\text{lcm}(|C_j|, |C|)/|C|$ occurrences of $C$. Due to the relation in the inequality 6 we can replace $\text{lcm}(|C_j|, |C|)/|C|$ occurrences of $C$ with $\text{lcm}(|C_j|, |C|)/|C_j|$ occurrences of $C_j$, with an accompanying increase in the total delay of the path. Notice that this replacement does not alter the total number of edges in the path. Consequently, a $k$-critical path for process $p_i$ that contains a vertex of critical cycle $C_j$ will have fewer than $\text{lcm}(|C_j|, |C|)/|C|$ occurrences of a non-critical cycle $C$.

Since we have already bounded the number of occurrences of non-critical cycles in $\pi$, we only need to show the existence of a $k$-critical path where the occurrences of critical cycles other than $C_j$ is bounded. Let $C'$ be another critical cycle in $\pi$ with delay $S'$. Then,

$$\frac{S_j}{|C_j|} = \frac{S'}{|C'|},$$

or

$$\frac{\text{lcm}(|C_j|, |C'|) \cdot S_j}{|C_j|} = \frac{\text{lcm}(|C_j|, |C'|) \cdot S'}{|C'|}.$$

Hence, the path, $\pi'$, obtained by repeatedly replacing $\text{lcm}(|C_j|, |C'|)/|C'|$ occurrences of $C'$ by $\text{lcm}(|C_j|, |C'|)/|C_j|$ occurrences of $C_j$, for all critical cycles, $C'$, in $\pi$ has the same length and delay as $\pi$. Consequently, $\pi'$ is a $k$-critical path for process $p_i$ that has the property required in the lemma. $\square$

A $k$-critical path that contains a bounded number of occurrences of all cycles except a critical cycle $C_j$ is defined to be in *canonical form with respect to $C_j$*. Note that a $k$-critical path can be transformed to canonical form with respect to any of the critical cycles that it touches. The part
Figure 9: A canonical path with critical cycle $C_j$.

of a $k$-critical path that is canonical with respect to $C_j$, which is not included in the complete occurrences of $C_j$, is referred to as the *stem* of the canonical $k$-critical path (see Fig. 9). Note that the stem of a $k$-critical path that is canonical with respect to $C_j$ has bounded length because it can have at most $|C_j| - 1$ occurrences of any cycle.

Figure 10: Canonical paths with respect to a non-critical cycle $C'$ and a critical cycle $C_j$ ending at $p_i$.

**Lemma 3** There exists a constant $N$ such that for $k \geq N$, there is a $k$-critical path for process $p_i$ that is canonical with respect to some critical cycle $C_j$.

*Proof.* We will first show that for sufficiently large $k$, the $k$-critical path for $p_i$ will touch a critical cycle. Suppose there is no $k$-critical path for $p_i$ that touches a critical cycle. Let $\pi'$ be a $k$-critical path that uses only non-critical cycles. Let $C'$ be the cycle with maximum mean delay used in $\pi'$ and let $\lambda' < \lambda$ be its mean delay.

Using the result in Lemma 2 applied to the path $\pi$, we claim that $\pi$ must be canonical with respect to $C'$. Hence,

$$d(\pi') = w' + n' \cdot \lambda' \cdot |C'|,$$

where $w'$ is the delay of the stem of $\pi'$ and $n'$ is the number of occurrences of $C'$ in $\pi'$. Further, if the length of the stem is $l'$, then $k = l' + n' \cdot |C'|$. 

25
Consider any critical cycle, $C_j$. Since the process graph is strongly connected there is a path from $C_j$ to $p_i$. Consider the path, $\pi$, of length $k$ consisting of $n$ occurrences of $C_j$ along with a stem of delay $W$ and length $l$ ending at $p_i$ (see Fig. 10). Then,

$$d(\pi) = w + n \cdot \lambda \cdot |C_j|,$$

and $k = l + n \cdot |C_j|$. Now, $d(\pi) > d(\pi')$, if

$$n > \frac{w - w' + (l - l') \cdot \lambda'}{|C_j| \cdot (\lambda - \lambda')}.$$

Consequently, there exists a constant $N$ such that for $k \geq N$, we get a contradiction to the claim that all the $k$-critical paths for $p_i$ avoid all the critical cycles. Hence, for sufficiently large $k$, there is a $k$-critical path for $p_i$ that touches a critical cycle $C_j$. Now, using Lemma 2, we can claim that we can transform this $k$-critical path to one that is canonical with respect to $C_j$. $\square$

Let $C_1, C_2, \ldots, C_m$ be the critical cycles in $G_P$ and let $\mathcal{L} = \text{lcm}(|C_1|, |C_2|, \ldots, |C_m|)$.

**Lemma 4** For all $k \geq N$, there are $k$- and $(k + \mathcal{L})$-critical path that are canonical with respect to the same critical cycle $C_j$ and the stems of the canonical $k$- and $(k + \mathcal{L})$-critical paths are identical. Thus,

$$x_i(k + \mathcal{L}) = x_i(k) + \mathcal{L} \cdot \lambda, \forall k \geq N.$$

**Proof.** From Lemma 3 we know that for $k \geq N$, there exists a $k$-critical path for any process $p_i$ in the process graph that is canonical with respect to some critical cycle $C_j$. Notice that the length of the stem of a canonical $k$-critical path is bounded (using the result of Lemma 2). So, there are only a finite number of different stems. This observation plays a crucial role in this proof.

Consider a critical cycle $C_j$. Let $\pi_{k,r}$ be the longest path with $k$ edges ending at $p_i$ that is canonical with respect to $C_j$ and let $\pi_{k,r}^*$ denote the stem of this canonical path. The canonical $k$-critical path for $p_i$ is the longest of all these paths. Let $C_{\phi(j)}$ be the critical cycle with respect to which the $k$-critical path is canonical, where $k \equiv j \pmod{\mathcal{L}}$ and $\phi$ provides a mapping from $j$ to a critical cycle (the justification for this mapping follows from this lemma). Consider $\pi_{k+\mathcal{L},r}$ : the longest path with $k + \mathcal{L}$ edges that is canonical with respect to $C_j$. Since $k$ is large, and the lengths of the stems $\pi_{k,r}^*$ and $\pi_{k+\mathcal{L},r}$ are bounded, any stem that is reachable from $p_i$ in $k + \mathcal{L}$ steps is also reachable in $k$ steps. Thus, if the stem $\pi_{k+\mathcal{L},r}$ has more delay than $\pi_{k,r}^*$ then we could replace the stem of $\pi_{k,r}$ by the stem of $\pi_{k+\mathcal{L},r}$ to obtain a new path with $k$ edges and larger delay than $\pi_{k,r}$. This contradicts the definition of $\pi_{k,r}$. It follows that $\pi_{k,r}$ and $\pi_{k+\mathcal{L},r}$ have the same stem. Consequently,

$$d(\pi_{k+\mathcal{L},r}) = d(\pi_{k,r}) + \mathcal{L} \cdot \lambda.$$

Thus, if $C_{\phi(j)}$ is the critical cycle with respect to which the $k$-critical path is canonical, then the $(k + \mathcal{L})$-critical path will also be canonical with respect to $C_{\phi(j)}$ and as observed earlier both
these paths have the same stem. Since the delays of these canonical critical paths determine \( x_i(k) \) and \( x_i(k + \mathcal{L}) \), it follows that

\[
x_i(k + \mathcal{L}) = x_i(k) + \mathcal{L} \cdot \lambda.
\]

With the above results in hand, we are now ready to prove the theorem that shows that the average execution rate of processes is well defined irrespective of the initial vector of start times.

**Theorem 1.** Consider a strongly connected process graph and let \( x_i(k) \), \( k \geq 0 \) be the sequence of time instances at which process \( p_i \) executes. Then there exists some \( N \) such that the following are true:

1. The sequence \( x_i(k) - x_i(k - 1) \), \( k \geq N \) is periodic with period equal to \( \mathcal{L} \), where \( \mathcal{L} \) is equal to the least common multiple of the lengths of all the critical cycles in the process graph.

2. For any \( q \geq N \),

\[
\frac{\sum_{j=q}^{q+\mathcal{L}-1} x_i(j+1) - x_i(j)}{\mathcal{L}} = \lambda,
\]

where \( \lambda \) is the maximum mean cycle delay in the process graph.

3. The average inter-execution time is well defined and

\[
T_i = \lim_{n \to \infty} \frac{\sum_{k=0}^{n-1} x_i(k+1) - x_i(k)}{n} = \lambda.
\]

**Proof.**

1. From Lemma 4 we know that for \( k \geq N \), \( x_i(k + \mathcal{L}) = x_i(k) + \mathcal{L} \cdot \lambda \) and \( x_i(k + 1 + \mathcal{L}) = x_i(k + 1) + \mathcal{L} \cdot \lambda \). Consequently,

\[
x_i(k + 1) - x_i(k) = x_i(k + 1 + \mathcal{L}) - x_i(k + \mathcal{L}).
\]

Hence, the sequence of inter-execution times for any process \( p_i \) is periodic with period equal to \( \mathcal{L} \).

2. When consecutive terms in the sequence of inter-execution times are added, the sum telescopes, consequently

\[
\frac{\sum_{j=q}^{q+\mathcal{L}-1} x_i(j+1) - x_i(j)}{\mathcal{L}} = \frac{x_i(q + \mathcal{L}) - x_i(q)}{\mathcal{L}} = \frac{\mathcal{L} \cdot \lambda}{\mathcal{L}} = \lambda.
\]
3. Let $W$ be the sum of the first $N$ terms of the sequence of inter-execution times. After the first $N$ terms the sequence is periodic with period $L$ and the sum of the terms in one period is $L \cdot \lambda$. Therefore,

$$
\lim_{n \to \infty} \frac{\sum_{k=0}^{n-1} x_i(k + 1) - x_i(k)}{n} = \lim_{t \to \infty} \frac{W + t \cdot \lambda \cdot L}{N + t \cdot L} = \lim_{t \to \infty} \frac{W/t + \lambda \cdot L}{N/t + L} = \lambda
$$

References


