NECESSARY AND SUFFICIENT CONDITIONS FOR DEADLOCK IN A MANUFACTURING SYSTEM

A Dissertation Presented to

The Faculty of the

Fritz J. and Dolores H. Russ

College of Engineering and Technology

Ohio University

In Partial Fulfillment

of the Requirements for the Degree

Doctor of Philosophy

by

Paul E. Deering

June, 2000
Acknowledgements

First, I would like to thank my advisor, Dr. Robert Judd, for giving me the opportunity to pursue a doctoral degree. His work ethic and pursuit of excellence will be motivational forces in everything I pursue from this day forth.

Secondly, I would like to thank Bryan Jordan for all of his support during the research and writing of this dissertation. It was greatly appreciated.

Thirdly, I would like to thank Dr. Robert Lipset for the many hours of advice and suggestions concerning this manuscript and many other things.

Finally, I would like to thank my parents and my other family members Mark, Karen, Kathy, Greg, Tim, Jacob, Mike and Mike for their constant moral support.
# Table of Contents

Acknowledgements ........................................................................................................ iii

List of Figures .................................................................................................................. viii

List of Tables .................................................................................................................. xi

Chapter 1 Introduction ...................................................................................................... 1

Chapter 2 Literature Review ............................................................................................ 3

2.1 Deadlock in Computer Operating Systems ............................................................... 3

2.2 Deadlock in Flexible Manufacturing Systems .......................................................... 4

2.2.1 Banaszak and Krogh ............................................................................................. 7

2.2.2 Hsieh and Chang ................................................................................................. 9

2.2.3 Zhou and DiCesare ............................................................................................. 11

2.2.4 Zhou ................................................................................................................... 13

2.2.5 Barkaoui and Abdallah ......................................................................................... 14

2.2.6 Venkatesh ........................................................................................................... 17

2.2.7 Fanti, Maione, Mascolo and Turchiano ............................................................... 19

2.2.8 Wysk, Joshi, and Yang ......................................................................................... 20

2.2.9 Cho, Kumaran and Wysk .................................................................................... 20

2.2.10 Judd and Faiz .................................................................................................... 24

2.2.11 Lipset, Deering and Judd ................................................................................... 26

Chapter 3 Previous Method - Judd and Faiz ................................................................. 28

3.1 Illustrative Example .................................................................................................. 28

3.2 Assumptions ............................................................................................................. 29

3.3 Modeling a Manufacturing System – The Wait Relation Graph .............................. 29
## 5.2 Chained Closed Paths ................................................................. 83
## 5.3 Complex Closed Paths ............................................................. 90
## 5.4 Main Results .............................................................................. 93

### Chapter 6 Results and Discussion .................................................... 95

#### 6.1 Results ......................................................................................... 95

- 6.1.1 Extending the Methods of Judd and Faiz ....................................... 95
- 6.1.2 Necessary and Sufficient Conditions for Deadlock .......................... 98

#### 6.2 Implementation ............................................................................ 98

#### 6.3 Comparison with Past Research Methods ...................................... 103

- 6.3.1 Banaszak and Krogh and Hsieh and Chang ...................................... 103
- 6.3.2 Zhou and DiCesare ...................................................................... 104
- 6.3.3 Barkaoui and Addallah .................................................................. 105
- 6.3.4 Fanti, Maione, Mascolo and Turchiano ........................................... 106
- 6.3.5 Cho, Kumaran and Wysk .............................................................. 107

#### 6.4 Conclusions .................................................................................. 109

#### 6.5 Recommendations ....................................................................... 110

### Chapter 7 References ....................................................................... 111

### Abstract ............................................................................................. 116
List of Figures

Figure 2.1 Example of primary deadlock ................................................................. 5
Figure 2.2 Example of impending deadlock .............................................................. 6
Figure 2.3 Manufacturing system for example 2.1 ..................................................... 8
Figure 2.4 Manufacturing system for example 2.4 .................................................... 12
Figure 2.5 Petri net for example 2.5 ........................................................................ 15
Figure 2.6 System for example 2.6 ........................................................................... 18
Figure 2.7 Manufacturing system for example 2.7 .................................................... 21
Figure 2.8 Heuristic moves part 2 in example 2.7 ..................................................... 22
Figure 2.9 Manufacturing system for example 2.8 .................................................... 22
Figure 2.10 Manufacturing system in impending deadlock ...................................... 24
Figure 2.11 Wait-Relation-Graph for example 2.9 .................................................. 25
Figure 3.1 Manufacturing system for Brackets, Incorporated ................................... 34
Figure 3.2 Double gripper replaced with single gripper ............................................ 35
Figure 3.3 Two simple circuits intersecting at a single capacity resource ............... 37
Figure 3.4 Two simple circuits intersecting at a multiple capacity resource .......... 37
Figure 3.5 Manufacturing system for example 3.3 .................................................. 40
Figure 3.6 Manufacturing system for example 3.4 .................................................. 41
Figure 3.7 Manufacturing system for example 3.5 .................................................. 43
Figure 3.8 Manufacturing system for example 3.6 .................................................. 44
Figure 3.9 Manufacturing system for example 3.7 .................................................. 46
Figure 4.1 Two simple circuits intersecting at a knot .............................................. 49
Figure 6.7 Manufacturing system for example 6.5 ............................................. 107

Figure 6.8 Manufacturing system for example 6.6 ............................................. 108
List of Tables

Table 2.1 Circuit parameters for example 2.9 .................................................. 26
Table 3.1 Circuit parameters for example 3.1 .................................................. 34
Table 3.2 Space computations for example 3.4 ............................................... 42
Table 3.3 Space computations for example 3.5 ............................................... 43
Table 4.1 Circuit parameters for example 4.1 .................................................. 50
Table 4.2 Circuit parameters for example 4.1 .................................................. 52
Table 4.3 Order computations for system in figure 4.2 ..................................... 53
Table 4.4 Space computations for example 3.5 with new order ....................... 54
Table 4.5 Space Conditions for Example 4.5 ................................................... 65
Table 4.6 Circuit parameters for example 4.6 .................................................. 70
Table 4.7 Process plans for example 4.7 .......................................................... 76
Table 4.8 Space of manufacturing system in state $n_0$ ..................................... 77
Table 4.9 Part movements to empty system in example 4.7 ............................. 78
Table 4.10 Process plans for example 4.8 ........................................................ 78
Table 4.11 Process plans for example 4.8 ........................................................ 79
Table 5.1 Process plans for example 5.1 .......................................................... 81
Table 5.2 Order and space computations for example 5.1 ............................. 82
Table 5.3 Process plans for example 5.4 .......................................................... 91
Table 5.4 Process plans for example 5.4 .......................................................... 94
Table 6.1 Process plans for example 6.1 .......................................................... 96
Table 6.2 Slack, order and space computations .............................................. 97
Table 6.3 Process plans for example 6.3 ......................................................... 104
Table 6.4 Process plans for example 6.4 .......................................................... 105
Table 6.5 Process plans for example 6.5 .......................................................... 106
Table 6.6 Process plans for example 6.6 .......................................................... 108
Table 6.7 Slack, order and space computations ................................................. 109
Chapter 1 Introduction

In recent years, researchers have shown considerable interest in methods for the design, scheduling, operation, and performance of flexible manufacturing systems (FMS). FMS are discrete event systems characterized by the availability of resources—e.g., robots, buffers, and machines—to produce a set of products. Raw parts, which belong to various product types, enter the system at discrete points in time and are processed concurrently, sharing a limited number of resources. In such a resource-sharing system, a situation may occur in which parts become permanently blocked. This is called deadlock.

Allowing a manufacturing system to enter only live states (deadlock-free states) and avoid any dead states (deadlocked states) can save both loss of production and labor costs as well as provide better resource utilization. Moving the wrong part in a live FMS can cause deadlock that can both cripple the entire manufacturing system and stall production. The only recourse is to manually resolve the deadlock and reset the FMS to a known state that is live. To prevent manual deadlock resolution in a FMS, a deadlock avoidance algorithm that can determine which parts to move must be incorporated into the controller of the FMS.

A challenge for the researcher is to develop a method that will allow the FMS to enter only live states and to avoid all dead states. All methods to date only provide sufficient conditions for a system to be live. In other words, these methods will avoid all dead states at the cost of preventing some live states. The goal of this dissertation is to precisely quantify both necessary and sufficient conditions for a manufacturing system to
be live. That is, describe a method that will absolutely conclude whether a system is live or dead.

The outline of this dissertation is as follows: Chapter 2 is the literature review. Since this research extends an earlier method by Judd and Faiz [16], Chapter 3 provides a more in-depth background of their research. Chapter 4 will prove sufficient conditions for a system to be live. Chapter 5 will prove sufficient conditions for a manufacturing system to be dead and show necessary and sufficient conditions for a manufacturing system to be live. Chapter 6 discusses the main results and compares these results against methods presented in the literature review.
Chapter 2 Literature Review

A rich collection of literature exists for resolving the problem of deadlock. The literature is basically divided into two major segments: deadlock relating to computer operating systems and deadlock in manufacturing systems. This chapter will introduce the general problem of deadlock from early literature on computer operating systems. We will then focus most on literature that has contributed significantly to the theory and application of resolving deadlock in a FMS.

2.1 Deadlock in Computer Operating Systems

Coffman, Elphick, and Shoshani [1] first characterized the conditions necessary for deadlock to exist in relation to computer operating systems as follows:

1. Tasks must claim exclusive control of the resource they require ("mutual exclusion" condition).
2. Tasks must hold resources already allocated to them while waiting for additional resources.
3. Resources cannot be forcibly removed from the tasks holding them until the resources are used to completion ("no preemption" condition).
4. A circular chain of tasks exists, such that each task holds one or more resources that are being requested by the next task in the chain ("circular wait" condition).

To avoid deadlocks, it is sufficient to assure that at least one of these four necessary conditions is not satisfied. The first three conditions are always present in computer systems and manufacturing systems that share resources. Deadlock prevention or
deadlock avoidance schemes in many methods focus on preventing the fourth condition from ever occurring (circular wait).

The above four conditions are necessary for deadlock to exist in computer and manufacturing systems. Although similar, enough differences exist between computer and manufacturing systems to prevent direct application of the results to manufacturing [2]. For example (i), the dynamics of a manufacturing system are usually slower than that of a computer operating system and thus, more computer time is available for implementing real-time control; and (ii) the request for resources in a manufacturing system is assumed to be fixed, finite, and sequential. In a computer system it is not. Since the future allocation of resources is known in a manufacturing system, this will allow better resource utilization in a manufacturing system, but as the cost of more complex deadlock algorithms.

2.2 Deadlock in Flexible Manufacturing Systems

Two types of deadlock occur in a manufacturing system. The most basic type is primary deadlock; this situation occurs when each part on a circuit requests the next resource in its process plan. This situation is illustrated in Figure 2.1.
Figure 2.1 Example of primary deadlock

Assume that part \(a\) in resource 1 has to go to resource 2, part \(b\) in resource 2 has to go to resource 3, and part \(c\) in resource 3 has to go to resource 1 before each part is completed. If resource 1, resource 2 and resource 3 can only hold one part at a time, no parts can move without intervention. Circuit 1 in Figure 1.1 is said to be in primary deadlock.

A more complex and difficult-to-detect type of deadlock is called \textit{impending} deadlock; this occurs when parts can move through the system but will terminate in primary deadlock after a finite number of moves. Consider the system shown in Figure 2.2 and assume that each resource can only hold one part. Assume that part \(a\) is occupying resource 1 and that part \(a\) first requires resource 2, then resource 3. Likewise, assume part \(b\) is occupying resource 3 and the next resource required by part \(b\) is resource 2 followed by resource 3. Although part \(a\) can move to resource 2, the system will terminate in primary deadlock on circuit 2. Part \(b\) can also move to resource 2, but primary deadlock will result on circuit 1.
The three basic approaches to solving the deadlock situation in manufacturing systems include: prevention; avoidance; and detection and resolution. The first two approaches (prevention and avoidance) ensure that deadlock will never occur; the last approach (detection and resolution) allows deadlock to occur. Deadlock prevention disallows deadlock by imposing constraints on how the resource requests are made in the system, ensuring that the necessary conditions for deadlock can never occur. An example of deadlock prevention [3, 4, 31] is to provide a large number of in-process buffers and uni-directional batching of jobs. This would ensure the circular wait condition would never occur. Deadlock detection and resolution methods [4-14] allow deadlocks to occur and are resolved after they occur by moving deadlocked parts to buffer spaces reserved for deadlock recovery procedures. Deadlock avoidance methods [2, 15-23] avoid deadlock by controlling the mix of parts in the system at any given time. Deadlock avoidance works by ensuring that the sufficient conditions for deadlock are not met. A part can be moved or introduced into the system only if the move does not cause deadlock. If a move is found to cause deadlock, then the move is not allowed to occur.
Graphs [4, 5, 16, 17, 19] and Petri nets [2, 15, 17, 20, 21, 22, 24] are the two major formalisms used to describe the deadlock problem in manufacturing. A graph is represented by $G = (R, A)$, where $R$ is a set of resources and $A$ is a set of arcs. Figure 2.1 and Figure 2.2 illustrate the use of graphs to represent a manufacturing system. A Petri net is represented by $N = (P, T, F, M_0)$, where $P$ is a set of places, $T$ is a set of transitions, $F$ is a set of directed arcs, and $M_0$ is the initial marking.

2.2.1 Banaszak and Krogh

Banaszak and Krogh [2] developed a deadlock avoidance algorithm (DAA) using Petri nets. In their research deadlock was identified as being caused by the circular wait relation between resources. This situation occurs when a process-enabled transition is unable to become resource-enabled. DAA is the real-time algorithm that controls the allocation of shared resources by limiting the set of transitions that can be fired for a given marking. This is accomplished by partitioning the resources in the production routes into shared and unshared subsequences, or zones, and then applying DAA, which consists of two restriction policies—DAA1 and DAA2. DAA1 allows a token to enter a new zone in the production sequences only when the capacity in the unshared subzone of the zone exceeds the number of tokens (jobs) currently in the zone. DAA2 assures that if a shared resource is being requested by the job, all of the shared resources in the remainder of the zone are available at that time. However, DAA restriction policies are too conservative. DAA is sufficient but not necessary to avoid deadlock. Consider the following example demonstrating DDA.
Example 2.1. The manufacturing system in Figure 2.3 produces one product $q$.

The resource sequence for $q$ is given by $r_q = \{I_3, O_3, I_1, O_1, I_2, O_2, I_1, O_1\}$. All resources in Figure 2.3 are of capacity one.

Notice that the resources $I_1$ and $O_1$ are the only shared resources in the production sequence. Clearly, the system is deadlock-free because if the token in $p_q(6)$ enters place $p_q(7)$ before the token in $p_q(0)$ enters place $p_q(3)$, the system will not deadlock. The production route zones are uniquely defined as $p_q = p_q(0)z_q^1z_q^2z_q^3p_q(9)$, where $z_q^1 = u_q^1$. 

![Figure 2.3 Manufacturing system for example 2.1](image)
\[ z_q^2 = s_q^2 u_q^2 \quad \text{and} \quad z_q^3 = s_q^3 \quad \text{with} \quad u_q^1 = \{p_q(1), p_q(2)\}, \quad s_q^2 = \{p_q(3), p_q(4)\}, \]

\[ u_q^2 = \{p_q(5), p_q(6)\}, \quad \text{and} \quad s_q^3 = \{p_q(7), p_q(8)\}. \]

The production route zones are shown in Figure 2.1. For this example, the initial markings of the system will be defined as \( M_0 = \{101001100\} \).

Under DAA1 and DAA2, transition \( t_1 \) and \( t_7 \) would be allowed to fire respectively. DAA1 or DAA2 would not allow firing of transition \( t_3 \), even though the firing of \( t_3 \) would not cause the system to deadlock. If DAA were actually implemented as a real-time control algorithm, deadlock would be avoided but at only the cost of under-utilized resources.

2.2.2 Hsieh and Chang

Hsieh and Chang [18] expanded upon Banaszak and Krogh's ideas [2] and developed a deadlock avoidance controller (DAC) also based on the Petri Net formalism. Their bottom-up approach allows for each job to be first represented by a Production Petri Net (PPN) according to its common manufacturing activities. Then, they synthesized each PPN by adding control places to form a Controlled Production Petri net (CPPN) and formulated a deadlock control policy that allows the occurrence of concurrent production events under a given dispatching policy. In order to complete this deadlock control policy, they decomposed the CPPN into controlled production subnets (CPSN) of individual job types, with each subnet completely capturing the interactions between shared resources. Then the minimal resource requirement (MRR) for each CPSN to determine if a control policy exists to keep the CPSN deadlock free. If a control
policy exists for each CPSN, then Hsieh and Chang use a sufficient validity test (SVT) to determine if the CPPN was covered. If it was not, then control policies for individual CPSN were modified and the system was re-tested using SVT. An analysis of the resulting control policy indicates that the DAC procedure is polynomial in complexity and has good potential for real-time applications. The DAC algorithm not only applies to a larger class of manufacturing systems but is less restrictive and allows more deadlock-free states than DAA. DAC provides sufficient conditions for a manufacturing system to be deadlock-free. Consider the following example.

**Example 2.2.** Consider the manufacturing system depicted in Figure 2.3. Assume each resource has capacity of 5, and that the initial marking for the Petri Net is $M_0 = \{001334010\}$. The question remain which transitions will DAC allow to fire.

DAC would allow the transitions $t_3$, $t_4$, $t_5$, and $t_6$ to fire. The firing of any of these transitions will not cause the system to deadlock. DAA would not fire $t_3$ under marking $M_0$. In essence, DAC identifies that at least one unit of free resource exists in shared places $p_q(3)$ and $p_q(4)$ and in unshared zone $u_q^2$. This allows the part in $p_q(2)$ to be pushed past places $p_q(3)$ and $p_q(4)$ and into the unshared zone $u_q^2$. DAC is an improvement of DAA, but is still not the optimal solution as illustrated by the next example.

**Example 2.3.** Consider the manufacturing system in Example 2.1 where all resources are of a capacity of one with $M_0 = \{101001100\}$. 
DAC would not let transition \( t_3 \) fire because the part in \( p_q(2) \) cannot be pushed to unshared zone \( u_q^2 \). The unshared zone \( u_q^2 \) is filled to capacity. Clearly, firing transition \( t_3 \) will not cause the system to deadlock in this case.

Two basic problems still exist with DAC and DAA: (1) neither will let a part to enter a new zone if the unshared portion of the zone is filled to capacity and (2) neither will let a part enter a shared zone unless all shared resources required by the part in the shared zone are free.

### 2.2.3 Zhou and DiCesare

To address the problem of allocating shared resources in a FMS to avoid deadlock, Zhou and DiCesare [21] developed two Petri nets structures Parallel Mutual Exclusion (PME) and Sequential Mutual Exclusion (SME). In these two representations, the places are divided into three mutually exclusive sets: \( A \) places, which represent operations or machines; \( B \) places, which represent the availability of fixed resources, such as robots and machines; and \( C \) places, which represent the availability of variable resources, such as raw material, fixtures, and pallets. The PME structure is used to represent resources shared by different independent processes. An ordered pair that consists of an initially marked place and a set of transition pairs defines a PME.

A Sequential Mutual Exclusion (SME) models a shared resource that is required sequentially by its processes. It can be defined as a sequential composition of several PMEs that share the same resource place. The liveness and reversibility of a net can be affected by an inappropriate distribution of initial tokens. If the number of initial tokens
in the SME does not exceed the token capacity, then the SME will remain live and reversible. Consider the following example.

**Example 2.4.** The system, depicted in Figure 2.4 as a 2-PME, is defined as $(p_{10}, \{(t_0, t_2), (t_4, t_6)\})$. It consists of three workstations: a drill, mill, and a lathe. The two independent processes share the mill and the lathe.

![Diagram of manufacturing system for example 2.4](image)

**Figure 2.4 Manufacturing system for example 2.4**

The parts are represented with tokens in the C-place set, $P_C = \{p_0, p_3\}$. The tokens in the B-place set, $P_B = \{p_7, p_8, p_9, p_{10}\}$, represent the availability of the fixed resources. Notice that the token in B place $p_{10}$ controls the allocation of both the lathe and mill. The operation A-place set is $P_A = \{p_1, p_2, p_4, p_5, p_6\}$. The initial markings for the system will be $M_0 = \{20011001101\}$. The system in Figure 2.2 is live, bounded, and reversible and, hence, deadlock-free. Deadlock is avoided because B-Place $p_{10}$ prevents
the only deadlock situation in which a part in process 1 is using the lathe while a part in process 2 is simultaneously using the mill.

The problem with this method is that it eliminates deadlock-free states. For example, suppose a situation requires only process 1 parts. Under the current control strategy, only one process 1 part is allowed in the system at any one time. Therefore, when transition $t_1$ fires and a process 1 part enters the lathe, it also seizes control of the idle mill. While the lathe is turning, the process 1 part of the mill will remain idle. When the lathe is finished, transition $t_2$ fires and the process 1 part is moved from the lathe to the mill. The lathe will then be idle while the mill processes the part, because transition $t_1$ is disabled. Transition $t_1$ will remain disabled until a token is returned to B-place $p_{10}$. When transition $t_3$ a token is returned to B-place $p_{10}$ fires and the part leaves the system. Thus, only one process 1 part can exist in the system at any one time. The same scenario occurs if only process 2 parts are required without demand for process 1 parts. Under these situations, system resources are under-utilized. If parts are required for only one process, then both the lathe and the mill can be utilized concurrently to produce parts.

2.2.4 Zhou

Zhou [22] expanded upon Zhou and DiCesare [21] and presented generalized versions of parallel and sequential mutual exclusion concepts. His research also introduced GPMEs and GSMEs. Unfortunately, the PMEs and SMEs [21] could only resolve problems where shared resources dealt with no choice operations. The research in [6, 22, 24] parts may have choice. For example, a robot can be in charge of unloading
between Machine 1 and two successive machines, Machine 2 or Machine 3. When the robot acquires a processed part from Machine 1, it can load the part on either Machine 2 or Machine 3.

2.2.5 Barkaoui and Abdallah

Barkaoui and Abdallah [15] developed a two-phase method to avoid deadlock in a FMS by adding control places to a structured Petri net. The first phase was to augment the Petri net by adding a "local control place" to each not-controlled minimal siphon. A siphon (originally called a deadlock) in a Petri net is a set of places which remains empty of tokens once it loses all tokens. It is said to be minimal if it does not contain any other siphons as a proper subset. Although a method to compute siphons is not presented in [15], several methods exist [26-30]. The first phase is called the "near-prevention" phase and prevents the places in a minimal siphon from becoming token-free. The second phase addresses the issue of controlling the control places augmented to the Petri Net in phase one. In [15] it is shown that the augmented Petri net necessarily reaches an unsafe marking deadlock occurs. An unsafe marking occurs when at least a "local control place" is token-free. The second phase serves to falsify this unsafe marking from occurring dynamically in the augmented net. This method is implemented by a Petri Net Controller (PNC) for which conflict transitions are solved according to a resource allocation policy. Consider the following example.

Example 2.5. Figure 2.5 shows a FMS, which produces two product types. The production sequences for each product are given by \( q_1 = r_1 r_2 r_3 r_4 r_5 r_1 \) and \( q_2 = r_3 r_4 \). To compute the siphons, the method presented in [26] produced 57 siphons of which only 4
are minimal siphons. The set of minimal siphons includes $D_1 = \{r_1, p_1, r_2, p_3\}$, $D_2 = \{r_2, r_3, p_4, p_5\}$, $D_3 = \{r_3, r_4, p_5, p_6\}$ and $D_4 = \{r_1, r_2, r_3, p_4, p_5\}$. The first phase is to augment the Petri net by adding control places such that these minimal siphons do not become token-free. Figure 2.5b shows the control places ($CD_1$, $CD_2$, $CD_3$, and $CD_4$) which must be augmented to the Petri net in Figure 2.5 to prevent siphons $D_1$, $D_2$, $D_3$, and $D_4$ from becoming token-free.

Figure 2.5 Petri net for example 2.5

Figure 2.5b Augmented control places
The second phase prevents impending deadlock from occurring dynamically. In essence, the second phase does not allow any new parts to enter the system as long as a control place(s) \((CD_1, CD_2, CD_3, \text{ and } CD_4)\) is token-free. Suppose transition \(t_1\) fires, resulting in place \(p_1\) becoming marked and \(r_1\) and \(CD_1\) becoming unmarked. This firing is analogous to a part \(q_1\) seizing resource \(r_1\). Since \(CD_1\) is unmarked, this marking is considered unsafe. The PNC will command firing transition \(t_2\) to return a token to \(CD_1\). Since each control place has at least one token, the resulting marking from firing \(t_2\) is considered a safe marking. However, suppose the following sequence of fired transitions fire \(t_3, t_4, t_8\). In this marking, parts \(q_1\) and \(q_2\) are processed by both resource \(r_4\) and resource \(r_3\), respectively, which make the situation unsafe because control place \(CD_2\) is token-free. This marking is, in fact, unsafe because if it enables transition \(t_1\) to fire, the resulting marking would be an impending deadlock. To avoid deadlock, therefore, the PNC will command the firing of either transition \(t_5\) or \(t_9\) to return a token to \(CD_2\).

The method presented in [15] is, therefore too conservative. Consider the situation in Figure 2.5a where part \(q_1\) was first introduced into the system. This occurred when transition \(t_1\) fires, resulting in place \(p_1\) becoming marked and unmarking places \(r_1\) and \(CD_1\). This marking is considered unsafe when, in fact, it is safe, because \(q_2\) parts can be processed from start-to-finish even though \(CD_1\) is token-free.
2.2.6 Venkatesh

Venkatesh [11] developed a model using the graphical formalism that captures the deadlock characteristics of a general discrete simulation system. In this model, resources may have capacities larger than unity, but the entities may only seize one unit of resource at any one time. Venkatesh [12] expanded upon earlier work [11] and allows entities to seize more than one unit of resource. Venkatesh makes a distinction between nonself-resolvable and self-resolvable deadlock. The latter of which is transient and may resolve itself over time as entities return resources in their possession. A nonself-resolvable deadlock requires external intervention. Venkatesh’s deadlock detection algorithm uses the concept of graph reduction. His reduction process is designed to distinguish between impending deadlock and transient deadlock. Basically, if the reduction procedure can eliminate all closed paths in the graph, then the system is not deadlocked. In this model if deadlock was detected, a spare buffer space was used to resolve the deadlock. Many of the functions required to implement the algorithm in SIMAN are introduced. Venkatesh [13] generalized the concepts in [12] so that the algorithm could be implemented in a real-time controller of a FMS. Consider the following example.

Example 2.6. In Figure 2.6, entities $e_1$ and $e_2$ hold one unit of resource, $r_1$ and $r_2$, respectively, and are currently requesting two units of resource $r_2$ and $r_1$. By assumption, entities $e_1$ and $e_2$ will not relinquish the resources they hold until they acquire the requested resource units. Additionally, entities $e_3$ and $e_4$ control one unit of resource $r_1$ and $r_2$, respectively. Entities $e_5$ and $e_6$ are waiting for a single unit of resource $r_1$ and $r_2$, respectively.
In this example, the reduction process would begin with the set of entities ($e_1$, $e_2$, $e_3$ and $e_4$). Initially, entity $e_3$ will return the unit of resource $r_1$ to its possession. Next, entity $e_4$ will return the unit of resource $r_2$ to its possession. Now, since neither entity $e_1$ nor entity $e_2$ can obtain its requested resource units and, further, since neither entity will relinquish the resource that it holds until it obtains the requested resource, the reduction process will end with entities $e_1$ and $e_2$ unable to satisfy their requests. This indicates impending deadlock, which would resolved by requiring one of the entities $e_1$ or $e_2$ to relinquish its hold on the resource. For the sake of this example, imagine entity $e_2$ relinquishes resource $r_2$, which it controls. This would allow entity $e_1$ to seize control of resource $r_2$ and allow entity $e_2$ to re-enter the system.

![Figure 2.6 System for example 2.6](image-url)
2.2.7 Fanti, Maione, Mascolo and Turchiano

Fanti, Maione, Mascolo and Turchiano [23] used graphical formalism to derive necessary and sufficient conditions for primary deadlock and characterized a situation called second level deadlock (SLD). They developed a digraph called the Working Procedure Digraph (similar to the Wait Relation Graph in [4]) that defines all potential interactions among the production mix and the sequences of resources required by each part in the mix. A Transition Digraph was developed to describe the current interactions by indicating both the resources currently held by jobs in process and the resources requested by the same parts in the next future. Using these two digraphs, they presented a theorem that proves necessary and sufficient conditions for a manufacturing system to be in primary deadlock. These two digraphs also allow for the characterization of second level deadlock, which are not immediate circular waits but will evolve into deadlock in the future. The researcher then present five restriction scheduling polices.

However, their policies address deadlock only. To them, a manufacturing system in impending deadlock is not considered deadlocked. Second level deadlock, is basically a system that is one part movement away from primary deadlock. Restriction policies RP1, RP2, and RP4 prevent deadlock from occurring by limiting the number of parts in the system based on the minimum capacity primary circuit in the Working Procedure Graph. Clearly, RP1, RP2 and RP4 are too restrictive and will prevent deadlock-free states from occurring. RP3, RP4 and RP5 use a simple single step look-ahead method. RP3 is the least restrictive method, but may transfer a deadlock-free system into an impending deadlock state. RP5 does prevent primary and impending deadlock from occurring by
restricting the number of parts allowed into a second level digraph. However, though Fanti and Maione et al. [23] claim to have derived necessary and sufficient conditions for deadlock, they have fallen short of deriving these conditions for both primary and impending deadlock states.

2.2.8 Wysk, Joshi, and Yang

Wysk, Joshi, and Yang [4] used graphical formalism to model a Flexible Manufacturing System to detect deadlock. To model a manufacturing system they developed a specialized directed graphical structure called a Wait Relation Graph (WRG). The nodes of a WRG represent the resources, and the arcs between resources represent the flow of parts through the system. A string manipulation procedure computes circuits. The WRG is constructed at each system state and a deadlock detection procedure (DDP) is invoked, which yields a set of control actions to detect first-level deadlock or primary deadlock. The DDP procedure is intended for use in conjunction with an extra storage buffer to recover from deadlock. DDP is a deadlock recovery strategy. The method is limited to systems that contain single capacity resources.

2.2.9 Cho, Kumaran and Wysk

Cho, Kumaran and Wysk [5] extended the results of [4] and developed a graphic-theoretic deadlock detection and resolution procedure in which a system status graph is used to represent the current state of the system. This status graph shows the routing of all parts currently in the system. It is a dynamic structure that analyzes for part flow
and/or impending part flow deadlock each time a part movement occurs. Part flow deadlock is primary deadlock. Impending part flow deadlocks will eventually result in part flow deadlock after a finite number of part movements.

Cho, Kumaran, and Wysk study introduces the concept of a bounded circuit. In general, a bounded circuit can be obtained by traversing the routings of the part on the reference node until a nonempty node is found. Then the routings of the part on the non-empty node can be traversed again until another non-empty node is found. This searching process continues until the reference node is met and a bounded circuit is obtained. Consider the following example.

**Example 2.7.** Consider a deadlock-free manufacturing system in Figure 2.7 which consists of five machines--A, B, C, D and E--with three parts in the system 1, 2, and 3. Parts 1, 2, and 3 currently occupy machines A, B, and E, respectively.

![Figure 2.7 Manufacturing system for example 2.7]

Machines C and D are currently empty. The labels on each arc represent the part routing of each part through the system. Each arc is labeled as $e(i,j)$, where $i$ represents the part identifier and $j$ represents the transition number of part $i$. Resource A will be the reference node; then the non-simple bounded circuit $v(A,1)~e(1,1)~v(B,2)~e(2,1)~v(D,0)~e(2,2)~v(E,3)~e(3,1)~v(D,0)~e(3,2)~v(C,0)~e(3,3)~v(B,2)~e(3,4)~v(A,1)$ can be generated from
definition 2 in [5]. Since the common-node $v(B,2)$ is occupied, the heuristic clears it by moving part 2 at machine B to machine D. The resulting graph is depicted in Figure 2.8.

![Figure 2.8 Heuristic moves part 2 in example 2.7](image)

As can be seen, parts 2 and 3 cannot exit the system; thus the system in Figure 2.8 is deadlocked. When the heuristic clears the common node, it moves the system into a part flow deadlock. Hence, even though this method concludes deadlock, the original system is, in fact, deadlock-free.

**Example 2.8.** Consider the manufacturing system in Figure 2.9, which consists of three machines--A, B, and C--with two parts in the system. Parts 1 and 2 currently occupy machines A and C, respectively. Machine B is empty. According to definition 2 in [5], a circuit starting at a reference node is a bounded circuit if one of the following conditions is held for all subsequences: $e(r,s) \lor (a,b) e(r',s')$ in the circuit: 1) if $r=r'$, then $s'=s+1$, or 2) if $r\neq r'$, then $r'=b$ ($\neq 0$) and $s'=1$. The bounded circuit in Figure 2.7 is $v(A,1) e(1,1) v(B,0) e(1,2) v(C,2) e(2,1) v(B,0) e(2,2) v(A,1)$.

![Figure 2.9 Manufacturing system for example 2.8](image)
A simple bounded circuit is a simple circuit. A simple bounded circuit represents a part flow deadlock if, and only if, all the nodes are occupied and committed in a cyclic fashion. Theorem 1 of [5] proves that a simple bounded circuit is necessary and sufficient for part flow deadlock. A non-simple bounded circuit is a bounded circuit with two or more repeated nodes other than the reference node (the first node of the non-simple bounded circuit). Theorem 2 of [5] proves that a non-simple bounded circuit with one empty common node represents an impending part flow deadlock if all nodes other than the common node are occupied. The bounded circuit depicted in Figure 2.7 is a non-simple bounded circuit with one empty common node and represents an impending part flow deadlock. A common node is a node in the bounded circuit that is repeated two or more times and is not the reference node. If a common node(s) is occupied in the non-simple bounded circuit, then a heuristic is suggested to empty the common nodes to determine deadlock. However, this method is limited because it only provides sufficient conditions for deadlock; therefore, the method predicts deadlock in certain cases when, in fact, the system in not actually deadlocked.

Deering later discovered Cho et al. [5] have a more serious problem with the concept using non-simple bounded circuits to detect deadlock. In [5], they claimed that a non-simple bounded circuit starting at a reference node is a necessary condition for impending part flow deadlock. In other words, a circuit that represents an impending part flow deadlock must be a non-simple bounded circuit. For example, the system in Figure 2.10 is a status graph that is in impending deadlock and does not contain a non-simple bounded circuit. After contacting Cho, he stated $v(A,1) \ e(1,1) \ v(B,2) \ e(2,1) \ v(C,0) \ e(2,2)$
$v(B,0) e(1,2) v(C,3) e(3,1) v(D,0) e(3,2) v(A,1)$ is a bounded circuit when, in fact, it is not a non-simple bounded circuit. The problem with this bounded circuit is that subsequence $e(2,2) v(B,0) e(1,2)$ does not adhere to the Definition 2 of a bounded circuit in [5]. Cho stated that Definition 2 in [5] should be appropriately modified but never stated what the modification should be.

![Diagram of manufacturing system in impending deadlock](image)

**Figure 2.10** Manufacturing system in impending deadlock

### 2.2.10 Judd and Faiz

To address the problem of deadlock Judd and Faiz [16] expanded upon the original formulation proposed by Wysk [4]. This new formulation creates a Wait Relation Graph WRG which models FMS containing resources with arbitrary capacities. This method shows that deadlock avoidance can be reduced to satisfying a set of linear inequalities. In essence, *space* can be calculated on each closed path within the FMS. Therefore, if the space of every closed path in the FMS is greater than zero, then this condition is sufficient for deadlock to be avoided. In essence, if the space of any closed subgraph within the FMS is zero, then the system is in deadlock. The space calculation is dependent upon on the *slack* and *order* of each closed path and the space inequalities are
computed by determining the slack of each closed subgraph. The slack of a closed subgraph is the number of free resource units on the closed subgraph. Slack is computed by subtracting the number of committed arcs from the total resource capacity of the closed subgraph. The order of a closed path is one less than the number of simple circuits that share a knot, which occurs when two or more distinct closed paths intersect at a single resource of unit capacity. The space is defined as the slack minus the order. As long as the space of each closed path in [16] is greater than zero, sufficient conditions exist to avoid deadlock. The following example illustrates the theory.

**Example 2.9.** Let a manufacturing system be composed of four resources--\( r_1, r_2, r_3 \) and \( r_4 \)--all with a capacity of one. Suppose the system manufactures two parts, \( a \) and \( b \), as specified by the following resource sequence: \( \text{res}(a) = r_4, r_2, r_1, r_2; \) \( \text{res}(b) = r_3, r_2, r_1, r_2, r_3 \). The WRG for this system is illustrated in Figure 2.11. The system consists of two simple circuits, \( c_1 \) and \( c_2 \), specified as follows: \( c_1 = r_1, r_2, r_1; \) \( c_2 = r_2, r_3, r_2 \). Assume the parts \( a \) and \( b \) are in the first resource of their respective production sequences.

![Figure 2.11 Wait-Relation-Graph for example 2.9](image-url)
It is easy to see that \( c_1 \cap c_2 = \{ r_2 \} \); thus, \( r_2 \) is a knot resource. Since both circuits contain a space greater than zero, the system is not in deadlock. The analysis for this system is portrayed in Table 2.1.

<table>
<thead>
<tr>
<th>Circuit</th>
<th>Order</th>
<th>Slack</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>( c_1 )</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>( c_2 )</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>( c_1 \cup c_2 )</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

Unfortunately, the problem with this method is that it is too conservative and eliminates many deadlock-free states. The method in [16] would not allow part \( a \) to move from resource \( r_4 \) to \( r_2 \) because after that move, \( \text{space}(c_1 \cup c_2, n) = 0 \). Clearly, moving part \( a \) from resource \( r_4 \) to \( r_2 \) would not cause the system to deadlock. This is because if part \( a \) did move into resource \( r_2 \), then part \( a \) could move to \( r_1 \), back to \( r_2 \), and then out of the system. Part \( b \) could then propagate through its production sequence and the system would be cleared. If a sequence of part movements can be found to clear the system, then obviously the system is not in deadlock.

2.2.11 Lipset, Deering and Judd

Lipset, Deering and Judd [32] expanded upon the methods proposed by Judd and Faiz [16]. In this research we redefined the order of a knot, defined a special state called an evaluation state, and defined the concept of order reduction. The approach was to put the system into an evaluation state and then compute the order. The approach taken in [32] to quantify both necessary and sufficient conditions for deadlock is similar to the
approach taken in this dissertation. Although similar, the results of this dissertation improve the definition of an evaluation state presented in [32], eliminated the need for order reduction, and corrected some of the proofs.

All of the methods presented in this literature provide only sufficient conditions for a system to be live. The goal of this dissertation is to precisely quantify both necessary and sufficient conditions for deadlock to occur in a manufacturing system. To accomplish this I will expand upon the research by Judd and Faiz [16]. The following chapter will provides a more in-depth background of their research.
Chapter 3  Previous Method - Judd and Faiz

This chapter starts with an illustrative example of a typical manufacturing system modeled in Judd and Faiz's research. It will explain the assumptions, terminology, definitions and the Wait Relational Graph (WRG) formalism used to accurately model a manufacturing system of this class and include a section with several examples illustrating the theory. The chapter concludes with a section outlining proposed improvements to the method of Judd and Faiz.

3.1  Illustrative Example

This research models Brackets, Incorporated, a hypothetical manufacturing system. Brackets, Incorporated produces mounting brackets for digital metering devices that are installed in numerous computer devices. The company has a small manufacturing system consisting of a mill, drill, lathe and two robots. The two robots are used to transport parts to and from the mill, drill and lathe. The manufacturing system is equipped to produce three different bracket types. The product process plans for the three bracket types are as follows:

Bracket A – Robot1, Mill, Robot1, Drill, Robot2.
Bracket B – Robot2, Lathe, Robot1, Mill, Robot1
Bracket C – Robot1, Drill, Robot2.

The resources in the system are the robots, mill, drill and lathe. Robot1 is equipped with a double gripper; therefore, it has the capacity to hold two parts. All other resources in the system can accommodate only one part at a time. The manufacturing system is
capable of sharing the available resources and processing any of the three bracket types concurrently. Thus, mixtures of different product types can be run concurrently in the system. Each resource performs a specific operation and cannot be preempted before its operation is complete.

### 3.2 Assumptions

The following assumptions will be made about manufacturing systems:

1. Each operation uses just one resource.
2. All operations and process plans are fixed, finite and sequential.
3. No operation can precede or succeed itself. (i.e., no self-loops)
4. No branching of operations are allowed in a process plan. (i.e., no choice)

### 3.3 Modeling a Manufacturing System – The Wait Relation Graph

The Wait Relation Graph (WRG) formalism will be used to model manufacturing systems in this research. This section begins by presenting a series of definitions introducing the Wait Relation Graph. This section concludes with the Wait Relation Graph representation of the manufacturing system presented of Brackets, Incorporated as presented in Section 3.1.

#### 3.3.1 Definitions

*Definition 3.1.* The set $R$ represents the set of resources in the system.

*Definition 3.2.* The set $P$ represents the set of products the system can produce.
Definition 3.3. For each product \( p \in P \), the process plan \( \text{plan}(p) = r_1r_2 \ldots r_m \) defines the sequence of resources that are required to produce \( p \). Resource \( r_m \) is the terminal resource for product \( p \).

Definition 3.4. A part is an instance of a product that flows through the system.

Definition 3.5. The set \( Q \) represents the set of parts currently in the system.

Definition 3.6. The Wait Relation Graph (WRG) \( G = (V,A) \) associated with a manufacturing system consists of a set of vertices and arcs. Each vertex represents a resource; that is, \( V=R \). A directed arc is drawn from vertex \( r_1 \) to vertex \( r_2 \), if \( r_2 \) immediately follows \( r_1 \) in at least one process plan. Each arc will be labeled with the part(s) that will flow through it.

Definition 3.7. A subgraph \( G_i = (R_i,A_i) \subset G \) of a WRG consists of a subset of the resources and arcs of \( G \) so that all the arcs in \( A_i \) connect resources in \( R_i \). The union (intersection), denoted by \( G_1 \cup G_2 \) \( (G_1 \cap G_2) \), of two subgraphs is the union (intersection) of the component resource and arc sets.

Definition 3.8. A path \( P = (R_p,A_p) \) is a subgraph whose resources and arcs can be ordered in the list

\[ r_1a_1r_2a_2r_3a_3 \ldots a_{n-1}r_n \]

where each arc in the list connects the resources on either side. When specifying a path, writing the arcs is redundant; therefore, only the resources will be enumerated when a path is defined.
Definition 3.9. A simple path is a path with no repeated elements in the ordered list.

Definition 3.10. A closed path is a path with the same first and last element.

Definition 3.11. A simple circuit is a closed path with no repeated elements in the ordered list except the first and last element.

Definition 3.12. Let \( q \in Q \) and \( p \in P \). The function \( \text{class}(q) = p \) returns the product \( p \) to which part \( q \) belongs.

Definition 3.13. The function \( n(q) \) returns a positive integer that represents the position in \( \text{plan}(\text{class}(q)) \) of the operation that is currently processing \( q \). When a new part \( q \) is added to the system, then \( n(q) = 1 \). As the part is moved from resource to resource according to its plan, \( n(q) \) is incremented until it reaches the end of its plan and exits the system.

Definition 3.14. The state \( n \) of a manufacturing system is a vector containing the current \( n(q) \) for all \( q \in Q \). Let \( N \) denote the set of all possible states for a given manufacturing system.

Definition 3.15. Each resource \( r \in R \) has a capacity of \( \text{cap}(r) \) units that can perform the required operations. This is termed the capacity of the resource. The capacity function can be extended to a set of resources, that is

\[
\text{cap}(R_i) = \sum_{r \in R_i} \text{cap}(r), \quad \text{for any } R_i \subseteq R.
\]
Definition 3.16. Let \( a \in A \) and \( r \in R \). The function \( \text{tail}(a) = r \) returns the resource at the tail of the given arc; the function \( \text{head}(a) = r \) returns the resource at the head of the arc.

Definition 3.17. A unit of the resource \( r = \text{tail}(a) \) is said to be committed to arc \( a \) if it is processing a part \( q \) whose next resource in its process plan is \( \text{head}(a) \). It is important to note that the number of resource units committed to the outgoing arcs of \( r \) can be less than the number of busy units. This happens when some of the busy units are being used for terminal operations. A resource unit is free if it is not committed to an arc; by this definition a busy unit which is not committed is still termed free. A resource is free if any of its units are free. A resource is empty if it contains no parts.

Definition 3.18. The commitment function \( \text{com}(a, n) \) returns the number of resource units that are committed to arc \( a \) when the system is in state \( n \). The commitment function is extended to a set of arcs as follows:

\[
\text{com}(A_i, n) = \sum_{a \in A_i} \text{com}(a, n), \quad \text{for any } A_i \subseteq A
\]

Definition 3.19. The slack of any subgraph \( G_i = (R_i, A_i) \subseteq G \) is given by

\[
\text{slack}(G_i, n) = \text{cap}(R_i) - \text{com}(A_i, n). \tag{3.1}
\]

Definition 3.20. The function \( \text{converge}(r) \) returns true if more than one arc enters resource \( r \). A resource \( r \) is said to be a non-converging resource if \( \text{converge}(r) \) is false.

Definition 3.21. A part is enabled if either the next resource in its process plan is free or the part is in the last step of its process plan.
Definition 3.22. Consider a subgraph of a WRG, which represents a manufacturing system. Assume that the system is in state $n_0$; there exists an arc $a$ such that resource $r_2 = \text{head}(a)$ is free; and the part in the resource $r_1 = \text{tail}(a)$ is committed to $a$. Then, when $r_1$ finishes its operation, this part can be moved to resource $r_2$. This process is called propagation. The symbol $n_k$ is used to denote the state of the system after the $k^{th}$ propagation.

Definition 3.23. A state $n$ of a manufacturing system is live if a sequence of part movements exists to empty the system.

Definition 3.24. A state $n$ of a manufacturing system is dead, or deadlocked, if it is not live.

Lemma 3.1. Let $G_1$ and $G_2$ be two subgraphs of a WRG $G$ in state $n$. Then

$$\text{slack}(G_1 \cup G_2, n) = \text{slack}(G_1, n) + \text{slack}(G_2, n) - \text{slack}(G_1 \cap G_2, n).$$

Proof. See Lemma 4.2 in Judd and Faiz [16].

To clarify the definitions presented, the following examples are provided.

3.3.2 Examples of the Wait Relation Graphs

Example 3.1. The Wait Relation Graph, depicted in Figure 3.1, represents the manufacturing system at Brackets, Incorporated. Recall that the process plans for the three bracket types are as follows:

Bracket A – Robot1, Mill, Robot1, Drill, Robot2.

Bracket B – Robot2, Lathe, Robot1, Mill, Robot1

Bracket C – Robot1, Drill, Robot2.
Assume that five parts, $Q = \{a_1, a_2, a_3, b_1, b_2\}$, are currently being processed in the system. Let $\text{class}(a_i) = A$ and $\text{class}(b_i) = B$. No parts currently exist in the system of product type $C$. Suppose the current state $n_0$ of the system is $n_0 = [n(a_1), n(a_2), n(a_3), n(b_1), n(b_2)] = [4, 3, 2, 2, 1]$.

![Figure 3.1 Manufacturing system for Brackets, Incorporated](image)

Notice that every resource in the system is processing a part. The system consists of two simple circuits, $c_1 = \text{Mill}, \text{Robot1}, \text{Mill}$ and $c_2 = \text{Robot1}, \text{Drill}, \text{Robot2}, \text{Lathe}, \text{Robot1}$. Table 3.1 shows the capacity, commitment and slack computations for $c_1$, $c_2$ and $c_1 \cup c_2$. It must be noted that the part in Robot1 does not commit any arcs on $c_1$ since the next resource required by part $a_2$ is the drill.

<table>
<thead>
<tr>
<th>Subgraph</th>
<th>Capacity</th>
<th>Commitment</th>
<th>Slack</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_1$</td>
<td>3</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>$c_2$</td>
<td>5</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>$c_1 \cup c_2$</td>
<td>6</td>
<td>5</td>
<td>1</td>
</tr>
</tbody>
</table>
Clearly, this system is *live* since a set of part movements (propagations) exists so that the manufacturing system can be emptied. The next example will show that replacing the double gripper on Robot1 with a single gripper will deadlock the system.

*Example 3.2.* Consider the manufacturing system in Example 3.1 where the double gripper of Robot1, has been replaced with a single gripper as shown in Figure 3.2. Suppose that the current state $n_0$ of the system is $n_0 = [n(a_1), n(a_2), n(b_1), n(b_2)] = [4, 2, 2, 1]$. The system consists of two simple circuits, $c_1 = \text{Mill, Robot1, Mill}$ and $c_2 = \text{Robot1, Drill, Robot2, Lathe, Robot1}$.

![Figure 3.2 Double gripper replaced with single gripper](image)

The system depicted in Figure 3.2 is *dead*. If the double gripper had replaced the single gripper on Robot1, this situation could have been avoided.

### 3.4 Research of Judd and Faiz

The research of Judd and Faiz [16] presents a method to avoid deadlock in a manufacturing system by satisfying a set of linear inequalities based upon static constraints of the WRG. As long as these constraints are satisfied, the manufacturing
system will remain live. The theory begins by establishing the necessary and sufficient conditions to avoid primary deadlock in a simple circuit. Next, sufficient conditions are proven to avoid deadlock in WRGs that consist of two simple circuits intersecting at a single resource. Sufficient conditions for deadlock are then derived for larger and more general manufacturing systems by altering live WRGs, adding simple circuits and paths and then proving that these systems can remain live. The theory finally ends with a theorem describing the sufficient conditions to avoid deadlock in any manufacturing system.

3.4.1 Primary Deadlock

Definition 3.25. A closed path $c$ in a WRG $G$ is in primary deadlock in state $n$ if $\text{slack}(c, n) = 0$.

If a manufacturing system represented by WRG $G = (R, A)$ consists of only a simple circuit, then the system is deadlock-free if, and only if, it is in a state contained in

$$N = \{n : \text{slack}(G, n) > 0\}.$$ 

For the complete proof see Theorem 4.1 and 4.2 in [16].

Deadlock will be avoided as long as the capacity of the resources on the simple circuit is greater than the number of committed arcs. In other words, if a simple circuit contains at least one free resource, then the system is guaranteed to be live.

The next section will examine the interaction between circuits.
3.4.2 Interaction between Two Circuits

An interesting phenomenon happens when two simple circuits are joined by a single capacity resource as opposed to a multiple capacity resource. Consider the two manufacturing system depicted in Figure 3.3 and Figure 3.4. Here, two simple circuits are joined by resource \( r_0 \), and all parts are committed to their outgoing arcs. In Figure 3.3 and Figure 3.4, the labels indicating which parts flows through each arc have been left off for simplicity.

Figure 3.3 Two simple circuits intersecting at a single capacity resource

Assume that all resources in both systems are of a capacity of one except where \( \text{cap}(r_0) = 2 \) and part \( q_5 \) is committed to arc \( a_1 \) (see Figure 3.4). Further assume in both systems that if \( q_1 \) is moved to resource \( r_0 \), it will be committed to arc \( a_1 \) and if part \( q_2 \) is moved to resource \( r_0 \), it will be committed to arc \( a_2 \).

Figure 3.4 Two simple circuits intersecting at a multiple capacity resource
Even though, resource \( r_0 \) is free in both manufacturing systems Figure 3.3 shows a dead system and Figure 3.4 a live one. In Figure 3.3, if either part \( q_1 \) or \( q_2 \) is moved into \( r_0 \), then primary deadlock will result on circuit \( c_2 \) or \( c_1 \), respectively. In Figure 3.4, if part \( q_1 \) were moved into \( r_0 \), then primary deadlock will result on circuit \( c_2 \). However, moving part \( q_2 \) will not cause deadlock since this move will allow the parts to propagate along circuit \( c_2 \).

**Definition 3.26.** A graph \( G \) is in *impending deadlock* if parts exist that can be propagated; however, the parts in circuit \( c \subset G \) must enter primary deadlock after a finite number of propagations.

Judd and Faiz present sufficient conditions for avoiding deadlock in the two manufacturing systems portrayed in Figure 3.3 and Figure 3.4. If two simple circuits \( c_1 \) and \( c_2 \) intersect at a common resource \( r \) and \( \text{cap}(r) = 1 \), then the set of live states is given by

\[
N_L = \left\{ n : \text{slack}(c_1, n) > 0 \text{ and } \text{slack}(c_2, n) > 0 \right\}.
\]

For the complete proof, see Theorem 4.3 in [16].

Judd and Faiz state that if the intersection of two simple circuits \( c_1 \) and \( c_2 \) is \( c_1 \cap c_2 = (R, A) \) where \( \text{cap}(R) > 1 \), then the set of live states is given by

\[
N_L = \left\{ n : \text{slack}(c_1, n) > 0 \text{ and } \text{slack}(c_2, n) > 0 \right\}.
\]

For the complete proofs, see Theorem 4.4 and Theorem 4.5 in [16].
The live states of a WRG consisting of two simple circuits depend on the structure of the intersection. This motivated the following two definitions:

**Definition 3.27.** Let \( c_1 \) and \( c_2 \) be any two closed paths in a WRG of a manufacturing system. If \( c_1 \cap c_2 \) consists of exactly one resource with a capacity of one, then this resource is called a *knot* with respect to \( c_1 \cup c_2 \).

**Definition 3.28.** Let \( c \) be any closed path in a WRG of a manufacturing system. The order of the path, denoted by \( \text{order}(c) \), is one less than the number of simple circuits on the path that share a knot with another simple circuit on the path. The order of a simple circuit is zero.

In Judd and Faiz, the order of a subgraph is related to its structure. The system represented in Figure 3.2 contains two simple circuits. Simple circuits \( c_1 \) and \( c_2 \) are of order zero, and closed path \( c_1 \cup c_2 \) is of order one. The system depicted in Figure 3.3 also contains two simple circuits. Simple circuits \( c_1 \) and \( c_2 \) are of order zero and closed path \( c_1 \cup c_2 \) is also of order zero. Resource \( r_0 \) is a knot in Figure 3.3. The system in Figure 3.4 contains no knots.

### 3.4.3 General Wait Relation Graphs

In order to extend the results of the previous section to general WRGs, a graph combining a series of simple circuits and paths was constructed. It shows that if \( C_G \) is the set of closed paths in a WRG then

\[
N_L = \{ n : \text{slack}(c, n) - \text{order}(c) > 0 \ \forall c \in C_G \}
\]

represents a set of live states. This motivated the following definition:
Definition 3.29. Let $c$ be a closed path in a WRG $G$ of a manufacturing system in state $n$. The free space on a closed path $c$ is the difference between the slack and the order; that is,

$$\text{space}(c, n) = \text{slack}(c, n) - \text{order}(c) \forall c \in C_G$$

where $C_G$ is the set of all closed paths in $G$.

The main result of [16] proved that a manufacturing system is live in all states defined by the set

$$N_L = \{ n : \text{space}(c, n) > 0 \forall c \in C_G \}$$

For complete detail of this proof, see Theorem 4.8 in [16].

3.5 Examples of the Theories of Judd and Faiz

The following is a simple example demonstrating the method of Judd and Faiz.

Example 3.3. The manufacturing system depicted in Figure 3.5 is composed of three resources $r_1$, $r_2$, and $r_3$, all with unit capacity. The system has two simple circuits: $c_1 = r_1r_2r_1$ and $c_2 = r_2r_3r_2$. Suppose that the system manufactures two products $p_1$ and $p_2$ specified by the following process plans: $\text{plan}(p_1) = r_1r_2r_3$ and $\text{plan}(p_2) = r_1r_2r_1$. Let $\text{class}(a) = p_1$ and $\text{class}(b) = p_2$. Suppose that the current state $n$ of the system is $n = [n(a), n(b)] = [1, 1]$.

![Figure 3.5 Manufacturing system for example 3.3](image-url)
According to [16], the order \( (c_1 \cup c_2) = 1 \) and slack \( (c_1 \cup c_2, n) = 1 \). Therefore, \( \text{space}(c_1 \cup c_2, n) = 0 \); this method will conclude that the system is dead.

**Example 3.4.** The manufacturing system depicted in Figure 3.6 is composed of five resources \( r_1, r_2, r_3, r_4 \) and \( r_5 \), all with unit capacity. The system consists of three simple circuits \( c_1 = r_1r_2r_1, c_2 = r_2r_3r_4r_2, \) and \( c_3 = r_4r_5r_3r_4 \). Suppose that the system manufactures two products -- \( p_1 \) and \( p_2 \) -- specified by the following process plans: \( \text{plan}(p_1) = r_1r_2r_3r_4r_5\) and \( \text{plan}(p_2) = r_5r_3r_4r_2r_1 \). Let \( \text{class}(a) = p_1 \) and \( \text{class}(b) = p_2 \). Suppose that the current state \( n \) of the system is \( n = [n(a_1), n(b_1), n(b_2)] = [1, 3, 1] \).

Notice that \( c_2 \cap c_3 = \{r_3, r_4\} \) is a path; thus, the system only contains one knot, i.e. \( c_1 \cap c_2 = \{r_2\} \). Table 3.2 shows the slack, order and space computations. Therefore, the system is live, since the space of all closed paths is greater than zero.
Table 3.2  Space computations for example 3.4

<table>
<thead>
<tr>
<th>Circuit</th>
<th>Slack</th>
<th>Order</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_1$</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$c_2$</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>$c_3$</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>$c_1 \cup c_2$</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$c_2 \cup c_3$</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>$c_1 \cup c_2 \cup c_3$</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

The following example demonstrates a situation where Judd and Faiz's method fails to determine if a given state is live.

Example 3.5. The manufacturing system depicted in Figure 3.7 is composed of five resources $r_1$, $r_2$, $r_3$, $r_4$ and $r_5$, all with unit capacity. The system consists of three simple circuits $c_1 = r_1r_2r_1$, $c_2 = r_2r_3r_2$, and $c_3 = r_3r_3$. Suppose that the system manufactures three products — $p_1$, $p_2$ and $p_3$ — specified by the following process plans:

plan($p_1$) = $r_1r_2r_4$, plan($p_2$) = $r_1r_2r_4$, plan($p_3$) = $r_3r_3r_1$. Let class($a$) = $p_1$, class($b$) = $p_2$, and class($c$) = $p_3$. Suppose that the current state $n$ of the system is $n = [n(a), n(b), n(c)] = [1, 1, 1]$. 
Figure 3.7 Manufacturing system for example 3.5

The system only contains one knot, i.e. $r_2$. Even though the system is actually live, the method does not conclude that it is live, since $\text{space}(c_1 \cup c_2 \cup c_3, n) = 0$. Table 3.3 shows the slack, order and space computations for the system.

Table 3.3 Space computations for example 3.5

<table>
<thead>
<tr>
<th>Circuit</th>
<th>Slack</th>
<th>Order</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_1$</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$c_2$</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>$c_3$</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$c_1 \cup c_2$</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$c_1 \cup c_3$</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$c_2 \cup c_3$</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$c_1 \cup c_2 \cup c_3$</td>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

While the research of Judd and Faiz avoided all dead states, it eliminated many live ones.
3.6 Expanding the Method of Judd and Faiz

One of the problems with Judd and Faiz's method is that the order of a closed path is based on the structure of the WRG. A better way to define order would be to base it on the system state. Consider the following example, identical to Example 3.3 except for a minor change in the process plans of products $p_1$ and $p_2$.

Example 3.6. The manufacturing system depicted in Figure 3.8 is composed of three resources, $r_1$, $r_2$, and $r_3$, all with unit capacity. The system has two simple circuits $c_1 = r_1 r_2 r_1$, $c_2 = r_2 r_3 r_2$. Suppose the system manufactures two products $p_1$ and $p_2$ -- specified by the following process plans: \( \text{plan}(p_1) = r_1 r_2 r_1 \) and \( \text{plan}(p_2) = r_3 r_2 r_3 \). Let class($a$) = $p_1$ and class($b$) = $p_2$. Suppose that the current state $n$ of the system is $n = [n(a), n(b)] = [1, 1]$.

![Diagram of manufacturing system](image)

Figure 3.8 Manufacturing system for example 3.6

According to the method of Judd and Faiz, the system in Example 3.6 is dead. Judd and Faiz make this conclusion by computing as \( \text{order}(c_1 \cup c_2) = 1 \) and therefore, conclude \( \text{space}(c_1 \cup c_2, n) = 0 \). Clearly, the system is live because part $a$ can propagate through its process plan followed by part $b$ to empty the system. Judd and Faiz's method, therefore, fails to distinguish between systems like this one and others where parts crisscross through a knot among circuits.
In this dissertation, the order of a knot will be redefined to be one only when parts crisscross through a knot, otherwise it will be zero. Redefining the order in this manner will allow more live states to be detected. My research will also extend Judd and Faiz’s methods by describing both the necessary and sufficient conditions for deadlock. Judd and Faiz only provide sufficient conditions for live states. My research, will introduce a special system state called an evaluation state that allows us to precisely qualify the necessary and sufficient conditions for deadlock. My research will show that by evaluating the space on any manufacturing system in an evaluation state, one can absolutely conclude whether the system is live or dead.

A final problem exists in the proofs of Judd and Faiz. Theorem 4.3 to Theorem 4.5 and Lemma 4.3 in [16] assume that an enabled part exists to propagate on the closed path with the least slack. This is not a valid assumption. The following is a counterexample.

**Example 3.7.** Consider the manufacturing system in state \( n \), depicted in Figure 3.9. The system consists of two single capacity resources and one multiple capacity resource. Suppose that parts \( b_1, b_2 \) and \( c \) are committed arcs \( a_1, a_2, \) and \( a_3 \), respectively. In Figure 3.9, the labels indicating which parts flow through each arc have been left off for simplicity. According to Theorem 4.4 in [16], if the system is in any of the following states the system is live:

\[
N = \left[ \text{slack}(c_1, n) > 0 \& \text{slack}(c_2, n) > 0 \right] \\
&\left[ \& \text{slack}(c_1 \cup c_2, n) > 0 \right].
\]
The problem with the proof of Theorem 4.4 in [16] is that it assumes one can always propagate on the circuit with the least slack first. In this case, \( \text{slack}(C_1, n) = 1 \) and \( \text{slack}(C_2, n) = 2 \). Clearly, this assumption is incorrect since \( C_1 \) does not contain an enabled part. Although this assumption about the existence of an enable part is incorrect, the hypothesis of the proof is correct. The proofs in the following chapter will solve this problem.

Figure 3.9 Manufacturing system for example 3.7
Chapter 4 Sufficient Conditions for a System to be Live

The objective of this chapter is to expand the sufficient conditions proposed by Judd and Faiz [16]. In this chapter, refining the definition of order will broaden the set of live states. The approach used to prove sufficient conditions in this chapter is different from that of Judd and Faiz, which used an inductive proof to show sufficient conditions for a system to be live. The approach here is algorithmic. This approach assures there will always be an enabled part, so that when the part is moved it will not cause the system to deadlock. Since there exist an enabled part always capable of moving, and all process plans are finite, then the manufacturing system can be emptied.

The first section of this chapter proves necessary and sufficient conditions for a simple circuit to be live; the next section introduces the new definition of order, followed by a section that presents some preliminary results. The subsequent section consists of a set of lemmas and theorems that are employed to prove the main results. The concluding section provides several examples.

4.1 Primary Deadlock in a Simple Circuit

A necessary condition for a system to be dead is for the system to contain a closed path. The basic component of a closed path is a simple circuit. This section will prove necessary and sufficient conditions for deadlock in such a system. Consider a WRG $G$ that only contains a single simple circuit in the following theorem:

Theorem 4.1. If a manufacturing system represented by WRG $G = (R, A)$ consists only of a simple circuit, then the system is live if, and only if, it is in a state contained in
\[ N = \{ n : \text{slack}(G, n) > 0 \} \]

Proof. If the system is live, then \( \text{slack}(G, n) > 0 \). To prove this, the contrapositive will be proven. Suppose that the \( \text{slack}(G, n) = 0 \). Then from equation (3.1), \( \text{cap}(R) = \text{com}(A, n) \). This indicates that there are no free resources on the simple circuit, and all the arcs are committed to resources on the circuit. None of the parts on the simple circuit are enabled, and the system cannot be emptied. Thus, the system is in primary deadlock. Next, we need to show that if \( \text{slack}(G, n) > 0 \), then the system is live. This state must include at least one free resource since \( \text{cap}(R) > \text{com}(A, n) \). Let this free resource be \( r_1 \). Clearly, either a part in \( r_1 \) can exit the system, or a part from \( r_2 \), the resource immediately preceding \( r_1 \), can move. Parts can move in this manner until all the parts complete their process plans and have left the system. Therefore, if \( \text{slack}(G, n) > 0 \), then the system is live.

4.2 Redefining Order

This section will redefine order. These new definitions will allow for the identification of more live states than in the method presented in Judd and Faiz, where the order of a knot is strictly structural. This new definition considers the fact that the order of a knot is actually state dependent. Consider the following example.

Example 4.1. Suppose that there are parts in all the resources shown in Figure 4.1, except \( r_6 \). Each of the parts in the other resources is committed to the outgoing arc of that resource. Assume that the part \( q_i \) in resource \( k \) is committed to arc \( a_i \). Finally,
assume that all parts in circuit $c_1$ must flow into circuit $c_2$ before completion. Call this state $n$. This state may, or may not, be in deadlock, depending on the ultimate destination of the parts in the resources on circuit $c_2$.

Case 1: Suppose all the parts in resources on $c_2$ must flow into $c_1$, then $n$ is a dead state.

Case 2: Suppose the parts in the resources on $c_2$ never enter $c_1$. The parts on $c_2$ just flow around $c_2$ and eventually exit the system. Then clearly, the system is live.

In Figure 4.1, the labels indicating which parts flow through each arc have been left off the graph for simplicity.

![Figure 4.1 Two simple circuits intersecting at a knot](image)

Part $q_1$ in $k$ committed to arc $a_1$

Table 4.1 depicts the slack, order, and space computations as per [16] for example 4.1. Since the order, slack, and space computations are equal in both cases, the method presented by Judd and Faiz cannot distinguish between the two.
Table 4.1 Circuit parameters for example 4.1

<table>
<thead>
<tr>
<th>Case 1</th>
<th>Circuit</th>
<th>Order</th>
<th>Slack</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$c_1$</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>$c_2$</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>$c_1 \cup c_2$</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Case 2</th>
<th>Circuit</th>
<th>Order</th>
<th>Slack</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$c_1$</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>$c_2$</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>$c_1 \cup c_2$</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

To help distinguish between the two cases, order is made a function of the system state. In case 1 above, a part on $c_1$ must enter $c_2$ and a part on $c_2$ must enter $c_1$. This situation is referred to as part crossing. In this situation, the order of the knot will be defined as one. However, case 2, the parts on $c_2$ stay on $c_2$ until completion and never cross into $c_1$. There are no parts that need to crisscross through knot $k$. The order of the knot in this case will be defined as zero. The following definitions help identify this situation.

**Definition 4.1.** Let $c_1$ and $c_2$ be two closed paths in a WRG $G$. Path $c_1$ is connected to $c_2$ if $c_1 \cap c_2 \neq 0$ and a part currently exists in the system that must propagate from $c_1$ to $c_2$ without leaving $c_1 \cup c_2$.

**Definition 4.2.** Given two closed paths $c_1$ and $c_2$, then $c_1$ and $c_2$ are cross connected if $c_1$ is connected to $c_2$ and $c_3$ is connected to $c_1$.

The next definition defines the order of a knot.
Definition 4.3. Let the closed path $c$ in state $n$ consist of two closed paths, $c_1$ and $c_2$, such that $c = c_1 \cup c_2$ and $c_1 \cap c_2 = k$ where $k$ is a knot. The order of knot $k$ with respect to the closed path $c$ in state $n$ is defined as

$$\text{order}(k, c, n) = \begin{cases} 
1, & \text{if } c_1 \text{ and } c_2 \text{ are cross connected.} \\
0, & \text{otherwise.}
\end{cases}$$

The order of any simple circuit is zero.

The next definition defines the order of a closed path.

Definition 4.4. Let $c$ be a closed path in a WRG $G$ in state $n$ that contains $m$ knots. Then, the order of $c$ is given by

$$\text{order}(c, n) = \sum_{i=1}^{m} \text{order}(k_i, c, n).$$

In Case 1 of the example, closed path $c_1$ is connected to $c_2$ and $c_2$ is connected to $c_1$; thus, the two circuits are cross-connected through knot $k$. Therefore, the order of knot $k$, with respect to the closed path $c_1 \cup c_2$, is one in state $n$. In Case 2, the closed path $c_1$ is connected to $c_2$, but $c_2$ is not connected to $c_1$. Therefore, the order of knot $k$ with respect to closed path $c_1 \cup c_2$ is zero in state $n$. Table 4.2 illustrates the slack, order and space computations for Example 4.1, incorporating this new definition of order. By changing the definition of the order it is possible to distinguish between the two cases in Example 4.1.
Table 4.2 Circuit parameters for example 4.1

<table>
<thead>
<tr>
<th>Case 1</th>
<th>Circuit</th>
<th>Order</th>
<th>Slack</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$c_1$</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>$c_2$</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>$c_1 \cup c_2$</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Case 2</th>
<th>Circuit</th>
<th>Order</th>
<th>Slack</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$c_1$</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>$c_2$</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>$c_1 \cup c_2$</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

A further implication of this new definition of order is that the order of a knot is either zero or one. In Judd and Faiz, the order is defined as one less than the number of simple circuits on the path that share the knot (see Example 3.5.). In that example, the system consisted of three simple circuits intersecting at a knot resource in which all three simple circuits are cross connected (see Figure 4.2.). The order($c_1 \cup c_2 \cup c_3$) = 2 per Judd and Faiz. According to Definition 4.3 the order($r_2, c_1 \cup c_2 \cup c_3, n$) = 1. The following is an explanation of this special structure and order computation.

![Figure 4.2 Manufacturing system example 3.5](image-url)
The structure of the system in Figure 4.2 is referred to as a corolla. Corollas are structures that contain parts that cross between closed paths through a knot in a cyclically cross-connected manner. The closed paths of the corolla are called leaves. The system in Figure 4.2 is a three-leaf corolla with leaves $c_1$, $c_2$, and $c_3$ which are cyclically cross connected as follows: $c_1 \rightarrow c_2 \rightarrow c_3 \rightarrow c_1$. This means that there exists a part in $c_1$ which must enter $c_2$, and there exists a part in $c_2$ which must enter $c_3$, and there exists a part in $c_3$ which must enter $c_1$ through a knot. Table 4.3 depicts the order computation of each closed path in Figure 4.2. Notice that only one case shows the order is one. This is because closed path $c_1 \cup c_2$ is cross-connected with $c_3$. Obviously, $c_2 \cup c_3$ is cross-connected with $c_1$, and $c_1 \cup c_3$ is also cross connected with $c_2$.

**Table 4.3 Order computations for system in figure 4.2**

<table>
<thead>
<tr>
<th>Closed Path</th>
<th>Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_1$</td>
<td>0</td>
</tr>
<tr>
<td>$c_2$</td>
<td>0</td>
</tr>
<tr>
<td>$c_3$</td>
<td>0</td>
</tr>
<tr>
<td>$c_1 \cup c_2$</td>
<td>0</td>
</tr>
<tr>
<td>$c_1 \cup c_3$</td>
<td>0</td>
</tr>
<tr>
<td>$c_2 \cup c_3$</td>
<td>0</td>
</tr>
<tr>
<td>$(c_1 \cup c_2) \cup c_3$</td>
<td>1</td>
</tr>
</tbody>
</table>

Clearly, using this new definition of order, the system in Example 3.5 would be declared as live. These values have been recomputed in Table 4.4.
Table 4.4 Space computations for example 3.5 with new order

<table>
<thead>
<tr>
<th>Circuit</th>
<th>Slack</th>
<th>Order</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_1$</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$c_2$</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>$c_3$</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$c_1 \cup c_2$</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>$c_1 \cup c_3$</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$c_2 \cup c_3$</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>$c_1 \cup c_2 \cup c_3$</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

4.3 Preliminary Results

This section presents some preliminary lemmas and corollaries required in the following sections. The first lemma will show that the commitment of a subgraph cannot be increased due to part propagation.

Lemma 4.1. Let $G_i = (R_i, A_i)$ be any subgraph in a WRG, and assume that the graph is in state $n_0$. Assume that an enabled part $q$ exists in $G_i$. If $n_1$ is the resulting state of the system after $q$ is propagated, then propagating $q$ results in

$$\text{com}(A_i, n_1) \leq \text{com}(A_i, n_0)$$  (4.1)

Proof: See Lemma 4.1 in [16].

The following corollary will show that the slack of any subgraph cannot decrease due to part propagation along an arc within the subgraph.

Corollary 4.1. Let $G_i = (R_i, A_i)$ be a subgraph of a WRG and

$$\text{slack}(G_i, n_0) > 0.$$
Assume that the graph $G_1$ has an enabled part $q$ committed along arc $a$. Let $n_1$ be the resulting state of the system after $q$ is propagated. Then,

$$\text{slack}(G_1, n_1) \geq \text{slack}(G_1, n_0).$$  \hspace{1cm} (4.2)

**Proof:** Combining (3.1) and (4.1) results in the desired result.

The following lemma will show that the order cannot increase due to part propagation.

**Lemma 4.2.** Let $G_1$ be a subgraph of a WRG and assume the graph is in state $n_0$. Assume that $G_1$ contains an enabled part. Let $n_1$ be the resulting state of the system after propagation, then the order of any knot on $G_1$ cannot increase.

**Proof:** All connected closed paths are known in state $n_0$. This implies that the order of all knots is known in state $n_0$. After propagating a part, all the connected paths will either remain connected or become disconnected. That is, propagation cannot create a new set of connected closed paths. Clearly, if the set of connected closed paths cannot increase, the order cannot increase with respect to any of these connected closed paths. Therefore, the order of $G_1$ cannot increase.

The following corollary will show that the space of any subgraph cannot decrease due to part propagation along an arc within the subgraph.

**Corollary 4.2.** Let $G_1$ be a subgraph of a WRG in state $n_0$ and

$$\text{space}(G_1, n_0) > 0$$

Assume that the subgraph has an enabled part $q$ committed to an arc $a$. Let $n_1$ be the resulting state of the system after part $q$ is propagated along $a$. Then
Proof: Substituting (3.2) into (4.2), we can write

\[ \text{slack}(G_t, n_t) - \text{order}(G_t, n_t) \geq \text{slack}(G_t, n_0) - \text{order}(G_t, n_0). \]  

(4.3)

Then, combining (4.3) and Lemma 4.2, we can conclude

\[ \text{space}(G_t, n_t) \geq \text{space}(G_t, n_0) > 0 \]

The following two lemmas provide some preliminary results required by Lemma 4.5.

**Lemma 4.3.**

\[ \sum_{i=2}^{m} (-1)^i mC_i = m - 1 \]

where \( mC_i \) is the number of combinations of \( m \) items taken \( i \) at a time.

**Proof.** From the binomial theorem we have

\[ (x + y)^m = \sum_{i=0}^{m} mC_i x^i y^{m-i}. \]  

(4.4)

Substituting \( x = -1 \) and \( y = 1 \) into (4.4), we obtain

\[ \sum_{i=2}^{m} (-1)^i mC_i = mC_1 - mC_0 = m - 1. \]  

The next lemma will show that the order of a knot of any closed path is greater than or equal to the order of the same knot with respect to a smaller subgraph.

**Lemma 4.4.** Let \( c_1 \subseteq c \) be closed paths and \( K = \{\text{knots on } c_1\} \). Then for any state \( n \),
order\( (k, c, n) \geq \text{order}(k, c_1, n) \quad \forall k \in K \). \quad (4.5)

**Proof.** \( c \) contains all of the resources and parts that \( c_1 \) does. Suppose that \( k \in K \) exists such that \( \text{order}(k, c_1, n) = 1 \). Then, two closed paths on \( c_1 \) that are cross-connected through \( k \) must exist. These paths also exist on \( c \) and must be cross-connected; therefore, \( \text{order}(k, c, n) = 1 \). Now suppose that \( k \in K \) exists such that \( \text{order}(k, c, n) = 0 \). Two cross-connected closed paths may exist on \( c \) where one or both are not part of \( c_1 \). Therefore, \( \text{order}(k, c, n) \) may be either 0 or 1.

The following example will help clarify Lemma 4.4.

**Example 4.2.** Suppose that the manufacturing system in Figure 4.3 is in state \( n \) and has part counter flow, where the parts of type \( a \) flow to the right and parts of type \( b \) flow to the left.

![Figure 4.3 Manufacturing system for example 4.2](image)

From Figure 4.3 we have \( c_1 \subset c \). Both \( k_1 \) and \( k_2 \) are knots with respect to closed paths \( c \) and \( c_1 \). Since part \( a_2 \) is not on \( c_1 \), we can conclude no parts crisscross through \( k_1 \) with respect to \( c_1 \); therefore, the order of \( k_1 \), with respect to \( c_1 \), is zero. Since both
parts $a_2$ and $b$ are in $c$, we can conclude that some parts cross through $k_1$ in $c$, and the order of $k_1$ with respect to $c$ is one. Therefore,

$$\text{order}(k_1, c, n) > \text{order}(k_1, c_1, n).$$

Since both parts $a_i$ and $b$ reside on closed paths $c$ and $c_1$ (which cross through $k_2$) we can conclude that the order of $k_2$ is one with respect to both $c$ and $c_1$. Therefore,

$$\text{order}(k_2, c, n) = \text{order}(k_2, c_1, n).$$

If $K = \{k_1, k_2\}$, we can conclude that (4.4) holds.

The following lemma is used to prove Lemma 4.9.

**Lemma 4.5.** Let $c_1, c_2, c_3, \ldots, c_m$ be $m$ closed paths joined together by a knot $k$, that is, they form the leaves of a corolla. Define $c = \bigcup_{i=1}^{m} c_i$. Assume that the corolla is in state $n$. Then,

$$\text{space}(c, n) \leq \left( \sum_{i=1}^{m} \text{space}(c_i, n) \right) - (m-1) - \text{order}(k, c, n).$$

**Proof.** Substituting the results of Lemma 3.1 into (3.2), we can write

$$\text{space}(c, n) = \text{slack}(c, n) - \text{order}(c, n)$$

$$= \sum \text{slack}(c_i, n) - \sum_{\text{over all } c_i \text{'s} \text{ taken 2 at a time}} \text{slack}(\cap c_i, n)$$

$$+ \sum_{\text{over all } c_i \text{'s} \text{ taken 3 at a time}} \text{slack}(\cap c_i, n) - \ldots - \text{order}(c, n)$$

Since $c_1 \cap c_2 \cap c_3 \ldots \cap c_m = \cap c_i = \{k\}$, we can conclude that all the $c_i$'s intersect at a single resource of capacity one; therefore,

$$\text{slack}(\cap c_i, n) = 1$$
and

$$ \sum_{i} \text{slack}(c_i, n) = m \cdot C_i $$ \hspace{1cm} (4.7)

Combining (4.6), (4.7) and Lemma 4.3, we obtain

$$ \text{space}(c, n) = \text{slack}(c, n) - \text{order}(c, n) $$

$$ = \sum \text{slack}(c_i, n) - m \cdot C_2 + m \cdot C_3 - \ldots - \text{order}(c, n) $$

$$ = \sum \text{slack}(c_i, n) - (m - 1) - \text{order}(c, n) $$

$$ = \sum \text{slack}(c_i, n) - (m - 1) - \sum \text{order}(k', c, n) $$ \hspace{1cm} (4.8)

$$ = \sum \text{slack}(c_i, n) - (m - 1) - \sum \text{order}(k', c, n) - \sum \text{order}(k', c, n) $$

$$ - \sum \text{order}(k', c, n) - \ldots - \sum \text{order}(k', c, n) - \text{order}(k, c, n) $$

Finally, combining (4.8) and Lemma 4.4, we derive the desired result of

$$ \text{space}(c, n) = \sum \text{slack}(c_i, n) - (m - 1) - \sum \text{order}(k', c, n) - \sum \text{order}(k', c, n) $$

$$ = \sum \text{slack}(c_i, n) - (m - 1) - \sum \text{order}(k', c, n) - \sum \text{order}(k', c, n) $$

$$ - \sum \text{order}(k', c_3, n) - \ldots - \sum \text{order}(k', c_n, n) - \text{order}(k, c, n) $$

$$ = \sum \text{slack}(c_i, n) - (m - 1) - \text{order}(c_1, n) - \text{order}(c_2, n) $$

$$ - \text{order}(c_3, n) - \ldots - \text{order}(c_n, n) - \text{order}(k, c, n) $$

$$ = \sum \text{space}(c_i, n) - (m - 1) - \text{order}(k, c, n) $$

4.4 Sufficient Conditions

This section develops sufficient conditions for a manufacturing system can be live. Given a manufacturing system with space on all closed paths greater than zero, the strategy, find an enabled part such that, when it is propagated, the space on all closed
paths in $G$ will remain greater than zero. If such a part exists, a sequence of part movements exists that will empty the system. Therefore, the original system state must be live.

Examples will be used throughout this section to help conceptualize the theory.

**Definition 4.5.** A part $q$ in WRG $G$ can be shifted to resource $r$ if it can be propagated to $r$ without propagating any other part in $G$.

**Example 4.3.** Consider the manufacturing system in Figure 4.4, which has part counter-flow. Part $a$ flows to the right and part $b$ flows to the left. Both parts $a$ and $b$ can be shifted to resource $r_1$.

![Figure 4.4 Shifting parts in a Wait-Relation-Graph](image)

**Definition 4.6.** Let $\text{plan}(\text{class}(q)) = r_1 r_2 \ldots r_m$ be the process plan for part $q$ in a WRG $G$. Then $q$ can freely exit if part $q$ can be shifted to $r_m$.

**Example 4.4.** Let the WRG $G$ in Figure 4.5 be in state $n$. 
The manufacturing system is composed of six resources $r_1, r_2, r_5, r_4, r_5$ and $r_6$, all with unit capacity except for $r_1$ which has capacity two. The system contains three closed paths, $c_1$, $c_2$ and $c_1 \cup c_2$. Let $C_G = \{c_1, c_2, c_1 \cup c_2\}$. Suppose that the system produces three products, $p_1$, $p_2$ and $p_3$, specified by the following process plans: $\text{plan}(p_1) = r_1r_2r_5r_5$, $\text{plan}(p_2) = r_5r_2$, and $\text{plan}(p_3) = r_5r_6$. Assume that parts $a$, $b$, and $c$ belong to product classes $p_1$, $p_2$ and $p_3$, respectively. Suppose that the system is in state $n = [n(a_1), n(a_2), n(a_3), n(b), n(c)] = [2,1,1,1,1]$. The question that remains is what sequences of part movements may be employed and empty the system to determine whether state $n$ is live.

The approach to empty the manufacturing system in Figure 4.5 may be as follows: first, remove any parts that have a free exit. That is, remove part $c$ from $G$, since it can freely exit $G$. Part $a_1$ can then be propagated to resource $r_3$ and then to $r_4$. Now, part
b can be removed from the system, since it has a free exit. Part \( a_1 \) can then be removed from the system, since it is no longer blocked by \( b \). Finally, remove parts \( a_2 \) and \( a_3 \), one part at a time, since both of these have free exits. This is only one of the many possible sequences of part movements the method may use to empty the manufacturing system in order to conclude that the original state \( n \) is live.

An outline of the section below is as follows: section 4.4.1 addresses parts that can freely exit the system; section 4.4.2 addresses parts that reside on closed paths; and section 4.4.3 presents the main results.

### 4.4.1 Part that can Freely Exit

This section will show that any part freely exiting a WRG \( G \) will not cause the space of any closed path in \( G \) to decrease.

**Theorem 4.2.** Given any part \( q \) in a WRG \( G \) that can freely exit, then removing part \( q \) from \( G \) will not decrease the space of any closed path in \( G \).

**Proof.** Let \( n_0 \) be the state of \( G \) before part \( q \) is removed from \( G \). Let \( n_1 \) be the state of \( G \) after part \( q \) is removed from \( G \). Since part \( q \) no longer commits an arc in state \( n_1 \), then by Corollary 4.2, we can conclude

\[
\text{space}(c, n_1) \geq \text{space}(c, n_0) \quad \forall c \in C_G
\]

where \( C_G \) is the set of all closed paths in \( G \).

The next section addresses part movements on closed paths in WRG \( G \).
4.4.2 Propagating Parts on Closed Paths

The approach to emptying the parts on the closed paths will be accomplished using three separate methods:

1. Parts that propagate to a non-converging resource on a closed path.
2. Parts that propagate to a converging resource where all closed paths have space greater than one.
3. Parts that propagate into a converging resource where one or more closed paths have space equal to one.

4.4.2.1 Propagating Parts into Non-Converging Resources on a Closed Path

This section will address propagating parts into non-converging resources.

Theorem 4.3. Let $C_G$ be the set of closed paths in a non-empty WRG $G$ in state $n_0$. Let $a_0$ be the arc part $q$ commits in state $n_0$. Assume that $r = \text{head}(a_0)$ is a non-converging resource. If

$$\text{space}(c, n_0) > 0 \ \forall c \in C_G$$

then

$$\text{space}(c, n_1) > 0 \ \forall c \in C_G.$$  

Proof. Propagating part $q$ to the non-converging resource $r$ cannot commit $q$ to any new circuits. Therefore, the space on any $c \in C_G$ in state $n_1$ can never change. \hfill \blacksquare

4.4.2.2 Propagating Parts on Closed Paths with Space Greater than One

The following theorem will show that parts propagating on closed paths with space greater than one will not cause space to become zero of any closed path.
**Theorem 4.4.** Let \( C_G \) be the set of closed paths in a non-empty WRG \( G \) in state \( n_0 \). Let \( a_o \) be the arc part \( q \) commits in state \( n_0 \) and \( r = \text{head}(a_o) \). Assume that propagating a part \( q \) in \( G \) to resource \( r \) commits arc \( a_l \) in state \( n_l \). If

\[
\text{space}(c, n_0) > 1 \quad \forall c \in C \quad \text{and} \quad \text{space}(c, n_0) > 0 \quad \forall c \in C_G - C
\]

where \( C = \{c \in C_G : a_o \not\in c \text{ and } a_l \in c\} \) then

\[
\text{space}(c, n_l) > 0 \quad \forall c \in C_G.
\]

**Proof.** Two cases will be considered.

**Case 1.** \( c \in C_G - C \). Since either \( a_o \in c \) or \( a_l \not\in c \) we can conclude

\[
\text{space}(c, n_l) \geq \text{space}(c, n_0) > 0 \quad \forall c \in C_G - C.
\]

**Case 2.** \( c \in C \). According to Lemma 4.2, propagating part \( q \) cannot increase the order of closed path \( c \). By propagating part \( q \) to \( r \) and committing to arc \( a_l \) in state \( n_l \) the space on \( c \) will decrease by one. Therefore, we can write

\[
\text{space}(c, n_l) = \text{space}(c, n_0) - 1 > 0 \quad \forall c \in C.
\]

The next example will clarify Theorem 4.3.

**Example 4.5.** Assume the manufacturing system in Figure 4.6 is in state \( n_0 \), which has part counter-flow. Part of type \( a \) flow to the right and part of type \( b \) flow to the left. If we wish to determine if part \( a \) can be propagated to \( r_2 \), let state \( n_1 \) be the state of the system after the propagation.
Table 4.5 depicts the resulting space conditions before and after shifting part \( a \) to \( r_2 \).

Clearly, the results of Theorem 4.4 are satisfied, where \( C = \{c_2\} \) and 
\[
C_G - C = \{c_1, c_1 \cup c_2\}.
\]

<table>
<thead>
<tr>
<th>Subgraph</th>
<th>Space in ( n_0 )</th>
<th>Space in ( n_1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( c_1 )</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>( c_2 )</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>( c_1 \cup c_2 )</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

We now consider part movements in a WRG \( G \) that contains enabled parts for which the properties specified in Theorem 4.3 and Theorem 4.4 do not exist.

4.4.2.3 Shifting Parts on Closed Paths with Space Equal to One

This section presents an algorithm that will determine which part can be safely shifted into a converging resource \( r \) where some of the closed paths containing \( r \) have space equal to one. The part will be selected so that when it is shifted to \( r \), it will not decrease the space of any closed path in the WRG to zero. The present section contains a series of definitions, lemmas and a theorem. These will be introduced by way of an illustrative example to help the reader conceptualize the theory.
The following example will be used to demonstrate the definitions and lemmas presented in this section.

**Example 4.6** Let the WRG $G$ in Figure 4.7 be in state $n$. The manufacturing system is composed of six resources, $r_1, r_2, r_3, r_4, r_5$ and $r_6$, all with unit capacity. The system contains seven closed paths. Let $C_G$ represent the set of closed paths in $G$, i.e.

$$C_G = \{c_1, c_2, c_3, c_1 \cup c_2, c_1 \cup c_3, c_2 \cup c_3, c_1 \cup c_2 \cup c_3\}.$$

Suppose that the system manufactures three products, $p_1, p_2$ and $p_3$, specified by the following process plans: $\text{plan}(p_1) = r_1 r_2 r_3$, $\text{plan}(p_2) = r_3 r_4 r_5$, and $\text{plan}(p_3) = r_5 r_6 r_1$. Assume that parts $a$, $b$, and $c$ belong to product classes $p_1, p_2$ and $p_3$ respectively. Suppose that the system is in state $n = [n(a), n(b_1), n(b_2), n(c)] = [1, 2, 1, 1]$. We wish to determine which part can be propagated to resource $r_6$ so that the space of all closed paths in $C_G$ remain greater than zero.

![Figure 4.7 Manufacturing system for example 4.6](image)

Figure 4.7 Manufacturing system for example 4.6

The only parts that can be shifted to resource $r_6$ are parts $a$, $b_1$, and $c$. This motivates the next definition.

**Definition 4.7.** Let WRG $G$ contain a free resource $r$. Then define the set
\[ Q_s(r) = \{ q : q \text{ can be shifted to } r \}. \]

Therefore, the set \( Q_s(r_6) = \{ a, b, c \} \) represents the set of parts that can be shifted to \( r_6 \) in Figure 4.7.

The following definition specifies a fundamental closed path and a fundamental set. The union of any two elements in a fundamental set is not a fundamental closed path.

**Definition 4.8.** Let \( C \) be a set of closed paths. If \( c \in C \) has the property
\[
    c' \subset c \quad \forall c' \in C \text{ and } c' \neq c,
\]
then \( c \) is a **fundamental closed path** with respect to \( C \). If all the elements of \( C \) are fundamental closed paths, then \( C \) is called a **fundamental set**. The function
\[
    \overline{C} = \text{reduce}(C)
\]
returns the largest subset of \( C \) that is a fundamental set. Note that every element \( c \in C \) can be expressed as the union of one or more elements of \( \overline{C} \).

The heart of this method is the following algorithm.

**Algorithm 4.1** Let WRG \( G \) be in state \( n_0 \). Let \( C_G \) be all the closed paths in \( G \). Suppose that
\[
    \text{space}(c, n_0) > 0 \quad \forall c \in C_G,
\]
and \( r \) is a free resource in \( G \) such that \( \text{space}(c, n_0) = 1 \) and \( c \) contains \( r \). The algorithm in Figure 4.8 will find a part \( q^* \) on a circuit in \( C_G \) such that when \( q^* \) is shifted to \( r \) resulting in state \( n^* \), then the
\[
    \text{space}(c, n^*) > 0 \quad \forall c \in C_G.
\]
Algorithm

1. \( C_0 = \{ c \in C_G : \text{space}(c, n_0) = 1, \text{and } r \in R_e \text{ where } c = (R_e, A_e) \} \)

   // By assumption \( C_0 \) can not be empty

2. \( \overline{C}_0 = \text{reduce}(C_0) \)

3. \( i = 1 \)

4. \( \overline{c}_i \in \overline{C}_{i-1} \)

5. Let \( q_i \in Q_*(r) \) such that \( q_i \) is on \( \overline{c}_i \)

6. Let \( n_i \) be the state of the system after \( q_i \) is shifted to \( r \)

7. If \( \text{space}(c, n_i) > 0 \forall c \in \overline{C}_{i-1} \) then go to step 11

8. \( \overline{C}_i = \overline{C}_{i-1} - \{ \overline{c}_i \} \)

9. \( i = i + 1 \)

10. Go to step 4

11. \( q^* = q_i \)

12. \( n^* = n_i \)

13. End

---

Figure 4.8 Listing of algorithm 4.1
Returning to Example 4.6, we wish to determine what part can be shifted to \( r_6 \) so that the space of all the closed paths in \( C_G \) remains greater than zero. The following is one of the possible sequence of steps Algorithm 4.1 might use to determine \( q^* \) and \( n^* \).

<table>
<thead>
<tr>
<th>Inputs ( C_g = {c_1, c_2, c_3, c_1 \cup c_2, c_1 \cup c_3, c_2 \cup c_3, c_1 \cup c_2 \cup c_3} ) and ( r_6 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Steps</td>
</tr>
<tr>
<td>1. ( C_0 = {c_1, c_3, c_1 \cup c_3, c_1 \cup c_2 \cup c_3} )</td>
</tr>
<tr>
<td>2. ( \overline{C_0} = {c_1, c_3} )</td>
</tr>
<tr>
<td>3. ( i = 1 )</td>
</tr>
<tr>
<td>4. Let ( \overline{c}_i = c_3 )</td>
</tr>
<tr>
<td>5. Let ( q_i = c )</td>
</tr>
<tr>
<td>6. Let ( n_1 ) be the state of the system after part ( c ) is shifted to ( r_6 )</td>
</tr>
<tr>
<td>7. If ( \text{space}(c, n_1) &gt; 0 ) for all ( c \in \overline{C_0} ) // This evaluates to FALSE since ( \text{space}(c_1, n_1) = 0 )</td>
</tr>
<tr>
<td>8. ( \overline{C}_1 = \overline{C_0} - {\overline{c}_1} = {c_1, c_3} - {c_3} = {c_1} )</td>
</tr>
<tr>
<td>9. ( i = 2 ) ( \quad ) // Go to step 4 in algorithm</td>
</tr>
<tr>
<td>10. Let ( \overline{c}_2 = c_1 )</td>
</tr>
<tr>
<td>11. Let ( q_2 = a )</td>
</tr>
<tr>
<td>12. Let ( n_2 ) be the state of the system after part ( a ) is shifted to ( r_6 )</td>
</tr>
<tr>
<td>13. If ( \text{space}(c, n_2) &gt; 0 ) for all ( c \in \overline{C}_1 ) // This evaluates to TRUE we goto step 11 in algorithm</td>
</tr>
<tr>
<td>14. ( q^* = a )</td>
</tr>
<tr>
<td>15. ( n^* = n_2 )</td>
</tr>
</tbody>
</table>

Therefore, part \( a \) is shifted to resource \( r_6 \) to form state \( n^* \). Table 4.6 shows the order and space computation before (state \( n_0 \)) and after the shift (state \( n^* \)).
Table 4.6 Circuit parameters for example 4.6

<table>
<thead>
<tr>
<th>Circuit</th>
<th>Order</th>
<th>Space in $n_0$</th>
<th>Space in $n^*$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_1$</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>$c_2$</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>$c_3$</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$c_1 \cup c_2$</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>$c_1 \cup c_3$</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>$c_2 \cup c_3$</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>$c_1 \cup c_2 \cup c_3$</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Notice that the space of all closed paths in state $n^*$ remains greater than zero. The remainder of this section will prove the correctness of Algorithm 4.1.

The next two lemmas prove that a part in step 5 of Algorithm 4.1 exists.

**Lemma 4.6.** If there is a closed path $c = (R,A)$ in a WRG $G$ in state $n$ such that $\text{space}(c,n) = 1$, then $c$ must contain a part.

**Proof.** We prove this by contradiction. Assume $c$ contains no parts. Since $c$ contains no parts, we have $\text{com}(A,n) = 0$; therefore, we can write

$$\text{order}(c,n) = \text{cap}(R) - 1.$$  

Clearly, this cannot be the case since self-loops are not permitted and any closed path requires at least two resources that are not knots; thus, we can write

$$\text{order}(c,n) \leq \text{cap}(R) - 2.$$  

Hence this is a contradiction.
Lemma 4.7. Let $C_G$ be a closed WRG in state $n$ with at least one closed path $c$ such that $\text{space}(c,n) = 1$. Let $r$ be a free resource on $c$. Then there exists a part $q \in Q_s(r)$ on $c$.

Proof. By Lemma 4.6, circuit $c$ contains a part. An input arc exists to $r$ on $c$. By following this arc backwards, eventually an alternate part must be encountered before encountering $r$ again. Therefore, a part $q$ in $c$ exists which can be shifted into $r$. ■

Corollary 4.3. A part $q \in Q_s(r)$ in step 5 of Algorithm 4.1 exists.

The next lemma will prove that Algorithm 4.1 terminates.

Lemma 4.8. Algorithm 4.1 terminates.

Proof. Clearly, $\bar{C}_0$ is not empty and is finite. In step 8 of Algorithm 4.1, an element is subtracted from set $\bar{C}_{i-1}$. Steps 4 through 10 can be repeated until $|\bar{C}_{i-1}| = 1$. In this case, the condition in Step 7 must be satisfied, since according to Corollary 4.2, propagating a part on a closed path cannot decrease its space. ■

Returning to Example 4.6, note that the space of any element in the fundamental set $\bar{C}_0$, namely $c_1$ and $c_3$ did not decrease as a result of the shift. The following lemma will prove in the general case, the space of any element in the fundamental set $\bar{C}_0$ will not decrease as a result of the shifting part $q^*$ recommended by Algorithm 4.1.

Lemma 4.9. Shifting part $q^*$ in $C_G$ to resource $r$ will not decrease the space of any element in the fundamental set $\bar{C}_0$. 
Proof. We prove this by contradiction. Assume shifting \( q^* \) to \( r \) from state \( n_0 \) to \( n^* \) will decrease the space of some \( \bar{c}_j \in \bar{C}_0 \). Let \( \bar{c}_i \) be the circuit in the step that terminates the algorithm. Since, by the check in Step 7, the space of all \( c \in \bar{C}_i \) must be greater than zero, then \( c_j \) must have been removed from \( \bar{C}_0 \) in an earlier step. Define \( C = \{c_j, c_{j+1}, \ldots, c_i\} \). Now, Algorithm 4.1 terminates by identifying a part \( q^* \) that will be shifted from closed path \( \bar{c}_j \) to \( r \). As a result of the shift, it is assumed that \( \text{space}(\bar{c}_j, n^*) \) went to zero. This implies that \( \bar{c}_i \) is connected to \( \bar{c}_j \). The operation of Algorithm 4.1 guarantees that path \( \bar{c}_{k-1} \) is connected to \( \bar{c}_k \) for \( j \leq k \leq i \). Combining this observation with the previous one, we have

\[
\bar{c}_i \rightarrow \bar{c}_j \rightarrow \bar{c}_{j+1} \rightarrow \ldots \bar{c}_i
\]

where \( \rightarrow \) indicates "connected to". Therefore, resource \( r \) is an order-one knot and the elements of \( C \) are leaves that form a corolla; see Figure 4.9.

![Image of Corolla with leaves]

Figure 4.9 Corolla with leaves

Now

\[
\text{space}(c, n_0) = 1 \quad \forall c \in C
\] (4.9)
and since the circuits in \( c \) are cyclically connected then

\[
\text{order}(r, c^*, n_0) = 1,
\]

where \( c^* = \overline{c}_j \cup \overline{c}_{j+1} \cup \ldots \cup \overline{c}_i \). Substituting (4.9) and (4.10) in to the results of Lemma 4.5, we can write

\[
\text{space}(c^*, n_0) = \sum_{i=1}^{m} \text{space}(c_i) - (m-1) - \text{order}(r, c^*, n_0)
= m - (m-1) - 1
= 0
\]

This contradicts the hypothesis of Algorithm 4.1 that \( \text{space}(c, n_0) > 0 \ \forall c \in C_G \).

Returning to Example 4.6, it is noted that the space of any element in the set \( C_0 \) did not decrease as a result of the shift. The following lemma will prove, in general, that the space of any element in the set \( C_0 \) will not decrease as a result of the shift.

**Lemma 4.10.** Shifting \( q^* \) on a closed graph in \( C_G \) to resource \( r \) in Algorithm 4.1 will not decrease the space of any element in the set \( C_0 \) in state \( n^* \).

**Proof.** By Lemma 4.9, \( \text{space}(c, n^*) > 0 \ \forall c \in \overline{C}_0 \). We need to show that the space of \( c \in C_0 - \overline{C}_0 \) will not decrease. Since \( c \) is the union of two or more elements in \( \overline{C}_0 \), then propagating \( q^* \) to resource \( r \) cannot commit any additional arcs on \( c \). Therefore, the space of \( c \) can never decrease.

The following theorem will prove, in general, that the space of all closed paths in WRG \( G \) must remain greater than zero.

**Theorem 4.5.** Shifting \( q^* \) identified by Algorithm 4.1 on a closed graph in \( C_G \) to resource \( r \) will not decrease the space of any closed path to zero in \( G \) in state \( n^* \).
Proof. By Lemma 4.9, space(c, n*) > 0 ∀c ∈ C_0; therefore, we still need to show that the space of c ∈ C_G - C_0 will not decrease. Assume c = (R_c, A_c). We have two cases to consider.

Case 1. r ∈ R_c. Since c ∉ C_0 then space(c, n_0) > 1. Propagating q* to resource r can only commit at most one arc on c and, by Lemma 4.2, the order of c cannot increase; therefore, we can conclude

space(c, n*) ≥ space(c, n_0) - 1 > 0.

Case 2. r ∉ R_c. Propagating q* to r will not commit any arcs on c; therefore,

space(c, n*) = space(c, n_0) > 1.

The next section presents the main result.

4.4.3 Main Result

The theorem in this section proves that if all closed paths of a WRG G have space greater than zero, G is live. It does this by determining a series of part propagations and shifts to empty the system.

Theorem 4.6. Let C_G be the set of all closed paths in a non-empty WRG G in state n. If

space(c, n) > 0 ∀c ∈ C_G,

then G is live.

Proof. Since the space of all simple circuits of G must be greater than zero, a free resource r exists in G. Therefore, since G is not empty, an enabled part q in G exists. We have four cases to consider in order show all possibilities.
Case 1. Part $q$ has a free exit. According to Theorem 4.2, removing part $q$ from the system will not decrease the space of any closed path in $G$. Therefore, moving part $q$ out of the system cannot reduce the space of any closed path in $G$ to zero.

Next assume that part $q$ can be propagated to $r$, and $n_1$ is the state of the system after shifting the propagation. Let $a_0$ be the arc part $q$ commits in state $n_0$ and $a_i$ be the arc part $q$ commits in state $n_1$.

Case 2. Resource $r$ is a non-converging resource. According to Theorem 4.3 the space on any closed path in $C_G$ can not decrease. Therefore, propagating part $q$ to resource $r$, we can conclude

$$\text{space}(c, n_1) > 0 \quad \forall c \in C_G.$$ 

The next two cases assume that resource $r$ is a converging resource and part $q$ is being shifted onto a closed path in $C_G$. Define $C = \{c \in C_G : a_0 \not\in c \text{ and } a_i \in c\}$.

Case 3. Space $(c, n_0) > 1 \quad \forall c \in C$. According to Theorem 4.4, shifting part $q$ on any circuit in $C$, we can conclude that

$$\text{space}(c, n_1) > 0 \quad \forall c \in C_G.$$ 

Case 4. Space $(c, n_0) = 1$ for some $c \in C_G$. By Theorem 4.5, Algorithm 4.1 will identify a part $q^*$ that can be shifted to resource $r$ such that

$$\text{space}(c, n_1) > 0 \quad \forall c \in C_G.$$ 

Applying case 1, case 2, case 3, and case 4 at each step, the space of all the closed paths in $G$ will remain strictly positive; therefore, there will always be an empty resource.
into which a part can be propagated. (That is, primary deadlock can never occur.). Since all the process plans are finite, this process can continue until all parts exit G.

Therefore, a series of part movements exists to empty G. Hence, the system in its original state $n$ is live.

Theorem 4.6 proves sufficient conditions for a system to be live. As presented in an example in the next section (see Example 4.8), there are systems that have space equal to zero that are live. The next chapter will address systems of this type.

### 4.5 Examples

**Example 4.7.** Let the WRG $G$ in Figure 4.10 be in state $n_0$. Suppose the process plans for parts $a$, $b$ and $c$ are as presented in Table 4.7. Let $c_1 = r_1r_4r_2r_1$, $c_2 = r_2r_3r_4r_2$, and $c_3 = r_4r_5r_6r_4$. Assume that the state of the system is $n_0 = [n(a_1), n(a_2), n(b), n(c)] = [2, 1, 2, 1]$.

<table>
<thead>
<tr>
<th>Part</th>
<th>Process Plan</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>$r_2r_4r_2r_2$</td>
</tr>
<tr>
<td>$b$</td>
<td>$r_2r_3r_4r_6$</td>
</tr>
<tr>
<td>$c$</td>
<td>$r_6r_4r_2$</td>
</tr>
</tbody>
</table>
Table 4.8 shows the space computations.

Table 4.8 Space of manufacturing system in state $n_0$

<table>
<thead>
<tr>
<th>Subgraph</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_1$</td>
<td>1</td>
</tr>
<tr>
<td>$c_2$</td>
<td>2</td>
</tr>
<tr>
<td>$c_3$</td>
<td>2</td>
</tr>
<tr>
<td>$c_1 \cup c_2$</td>
<td>1</td>
</tr>
<tr>
<td>$c_1 \cup c_3$</td>
<td>2</td>
</tr>
<tr>
<td>$c_2 \cup c_3$</td>
<td>2</td>
</tr>
<tr>
<td>$c_1 \cup c_2 \cup c_3$</td>
<td>1</td>
</tr>
</tbody>
</table>

Clearly, the space of all closed paths is greater then zero. The manufacturing system is *live* according to Theorem 4.4. Table 4.9 shows one possible sequence of moves *Theorem 4.6* might use to empty the system and conclude that the system is live.
Example 4.8. Theorem 4.6 can conclude a system is live if the space of all closed paths is greater than zero. If the space is zero the system may be live or dead. Consider the following two cases.

Case 1: Suppose that the system in Figure 4.11 has the following process plans depicted in Table 4.10.

Assume that the system is in state $n = [a, b, c] = [1,1,1]$. 

Table 4.9 Part movements to empty system in example 4.7

<table>
<thead>
<tr>
<th>Part Movement</th>
<th>Apply Theorem</th>
<th>Resulting State after move</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_1$ to $r_4$</td>
<td>4.5</td>
<td>3 1 2 1</td>
</tr>
<tr>
<td>$a_2$ to $r_1$</td>
<td>4.3</td>
<td>3 2 2 1</td>
</tr>
<tr>
<td>$a_1$ free exit</td>
<td>4.2</td>
<td>- 2 2 1</td>
</tr>
<tr>
<td>$a_2$ free exit</td>
<td>4.2</td>
<td>- - 2 1</td>
</tr>
<tr>
<td>$b$ to $r_4$</td>
<td>4.4</td>
<td>- - 3 1</td>
</tr>
<tr>
<td>$b$ to $r_5$</td>
<td>4.3</td>
<td>- - 4 1</td>
</tr>
<tr>
<td>$c$ free exit</td>
<td>4.2</td>
<td>- - 4 -</td>
</tr>
<tr>
<td>$b$ free exit</td>
<td>4.2</td>
<td>- - - -</td>
</tr>
</tbody>
</table>

Table 4.10 Process plans for example 4.8

<table>
<thead>
<tr>
<th>Part</th>
<th>Process Plan</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>$r_3 r_4 r_5 r_6$</td>
</tr>
<tr>
<td>$b$</td>
<td>$r_3 r_4 r_5 r_6$</td>
</tr>
<tr>
<td>$c$</td>
<td>$r_3 r_4 r_5 r_6$</td>
</tr>
</tbody>
</table>
Figure 4.11 Manufacturing system for example 4.8, Case 1

The space of all closed paths in Figure 4.11 is greater than zero except for the closed path which contains the entire system; that is, \( \text{space}(c_1 \cup c_2 \cup c_3 \cup c_4, n_0) = 0 \). Even though the space is zero the system is live. Now consider a WRG with the same structure except for a different system state and part routings.

Case 2: Suppose the system in Figure 4.12 has the following process plans as depicted in Table 4.11.

<table>
<thead>
<tr>
<th>Part</th>
<th>Process Plan</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a )</td>
<td>( r_1r_2r_3r_4 )</td>
</tr>
<tr>
<td>( b )</td>
<td>( r_5r_4r_6 )</td>
</tr>
<tr>
<td>( c )</td>
<td>( r_6r_4r_5r_1 )</td>
</tr>
</tbody>
</table>

Assume that the system is in state \( n = [n(a), n(b), n(c)] = [1,1,1] \).
As in case 1, the space of all closed paths in Figure 4.11 is greater than zero, except for the closed path that contains the entire system; that is, $\text{space}(c_1 \cup c_2 \cup c_3 \cup c_4, n_0) = 0$.

In this case, the system is dead. The space condition cannot distinguish between the two cases. The next chapter will introduce an evaluation state that will partially resolve these two cases.
Chapter 5 Sufficient Conditions for a System to be Dead

The previous chapter proved sufficient conditions for a manufacturing system to be live; that is, if the space of all closed paths in a manufacturing system is greater than zero, then the system is live. This section of the dissertation will determine sufficient conditions for a dead manufacturing system. Unfortunately, this cannot be proven in the general case, since there is insufficient information in the WRG to determine these conditions. However, when the system is in a special system state called an evaluation state, it can be shown that a manufacturing system is dead if one of the closed paths equals zero. The following example will demonstrate this more clearly.

Example 5.1. Let the WRG $G$ in Figure 5.1 be in state $n$. Suppose that the process plans for parts $a$, $b$ and $c$ appear as presented in Table 5.1. Assume that the state of the system is $n = [n(a), n(b), n(b)] = [1, 1, 1]$. Table 5.2 depicts the order and space computations for this system.

Table 5.1 Process plans for example 5.1

<table>
<thead>
<tr>
<th>Part</th>
<th>Process Plan</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>$r_2r_3r_4$</td>
</tr>
<tr>
<td>$b$</td>
<td>$r_4r_3r_1$</td>
</tr>
<tr>
<td>$c$</td>
<td>$r_1r_2r_3r_1$</td>
</tr>
</tbody>
</table>
Since the space of the union between $c_1$ and $c_2$ is zero, the method previously presented in Chapter 4 cannot conclude whether the system is live or dead. My revised method will show that the order in which parts flow through order-one knots is required to describe sufficient conditions for a dead system. For example, knowing that parts $a$ and $b$ must pass through $r_3$ before any other parts can leave $c_1$ and $c_2$ and that the space of the union between $c_1$ and $c_2$ is zero, will allow researchers to know that the system in Figure 5.1 to be dead.

This chapter will contain four main parts: the first section shows necessary and sufficient conditions which render basic closed path as dead; the next two sections show sufficient conditions for deadlock of chained and complex closed paths. The main result is presented in the final section.
5.1 Basic Closed Paths

Definition 5.1. A basic closed path \( c \) is a closed path in a WRG \( G \) in state \( n \) such that \( \text{order}(c, n) = 0 \).

Theorem 5.1. Given a basic closed path \( c = (R, A) \) in state \( n \). If \( \text{space}(c, n) = 0 \) then \( c \) is dead.

Proof. Since \( c \) is a basic closed path, then \( \text{order}(c, n) = 0 \). We can write

\[
\text{space}(c, n) = \text{slack}(c, n) = \text{cap}(R) - \text{com}(A, n) .
\]

Substituting \( \text{space}(c, n) = 0 \) into (5.1) we can write

\[
\text{cap}(R) = \text{com}(A, n) .
\]

From (5.2) we can conclude that all parts in \( R \) are committed to arcs in \( R \). No enabled parts exist. Therefore, \( c \) is dead.

Theorem 4.5 and Theorem 5.1 allow use to we can conclude that space greater than zero of a basic closed path is necessary and sufficient for the system to be live. The next section addresses a particular closed path that contains order-one knots.

5.2 Chained Closed Paths

This section defines a chained closed path and introduces a special state called an evaluation state. A series of definitions, some lemmas and a theorem will prove that if a chained closed path is in an evaluation state and its space is equal to zero, then the chained closed path is \textit{dead}. 
Definition 5.2. A chained closed path $c$ is a closed path containing one or more order-one knots with respect to $c$, such that $c$ can be decomposed into a set of basic closed paths which intersect at only the order-one knots.

The following is a simple example of a chained closed path:

Example 5.2. Consider the manufacturing system in Figure 5.2. Assume that all $a$ part types flow to the right from $c_1$ to $c_3$, and that all $b$ part types flow to the left from $c_3$ to $c_1$. In this state, resources $r_2$ to $r_3$ are order-one knots. The manufacturing system can be decomposed into three simple circuits, $c_1$, $c_2$, and $c_3$. Let $c = c_1 \cup c_2 \cup c_3$. In this example, $c$ is a chained closed path, since $c$ can be decomposed into basic closed paths so that each circuit intersects each other at only the order-one knots (i.e. $c_1 \cap c_2 = r_2$ and $c_2 \cap c_3 = r_3$).

![Figure 5.2 A chained closed path](image)

The following example will help the reader conceptualize the need for an evaluation state:

Example 5.3. Suppose that a part exists in all the resources shown in Figure 5.3 except $r_4$. Each part is committed to the outgoing arc of its resource. Assume that all part $a$ types must flow to circuit $c_2$ before completion and parts $d_1$ and $d_2$ must flow to...
circuit $c_1$ before completion. Call this state $n$. This state may, or may not, be dead, depending on the ultimate destination of part $b$ in the resource $r_7$.

Case 1. Suppose part $b$ must move to resource $r_4$ and then to $r_5$ and exit the system. Clearly, in this case, state $n$ is a live state.

Case 2. Suppose part $b$ must flow to $r_4$ and commit to circuit $c_1$. Then state $n$ is a dead state.

![Diagram](image)

Figure 5.3 Manufacturing system for example 5.3

To distinguish and to evaluate these two cases, the dynamics of the part crossing through the knot should be analyzed more closely. Notice that in both cases, all part $a$'s must cross knot $r_4$ before any other part on $c_1$ can leave $c_1$. But the part crossing dynamics are different on circuit $c_2$ in the two cases. Notice that in case 1, part $b$ can leave circuit $c_2$ before part $d_1$ must cross knot $k$. In other words, a resource may become free on $c_2$ before part $d_1$ must cross the knot. In this state, we conclude that state $n$ is not in an evaluation state. The method in the previous chapter cannot determine if deadlock
exists by computing the space in state $n$. Notice that in Case 2, part $b$ must cross knot $r_4$ before any other part can leave $c_2$. In this situation, no part can escape $c_2$ before the crossing must occur. The state of the system in case 2 is considered to be an evaluation state. These ideas motivated the following definitions.

**Definition 5.3.** Let $c_1$ and $c_2$ be two closed paths in a WRG $G$ such that $c_1 \cap c_2 = k$ where $k$ is an order-one knot. If a part $q$ on $c_1$ propagates to $k$ and commits to an arc on $c_2$, then $q$ is said to cross knot $k$.

**Definition 5.4.** A basic closed path in a WRG $G$ is always in an evaluation state.

**Definition 5.5.** An empty chained closed path in state $n$ is in an evaluation state.

**Definition 5.6.** Let a non-empty chained closed path $c$ be in state $n$. A chained closed path $c$ can be divided into two closed paths, $c_1$ and $c_2$, at any order-one knot $k$ such that $c_1 \cap c_2 = k$. Then chained closed path $c$ is in an evaluation state if

1. all order-one knots are empty, and

2. for each order-one knot $k$, two parts, $q_1$ and $q_2$ exist, such that
   a. part $q_1$ must cross from $c_1$ to $c_2$ before any other part can leave $c_1$, and $c_2$ is in an evaluation state after the move; and
   b. part $q_2$ must cross from $c_2$ to $c_1$ before any other part can leave $c_2$, and $c_1$ is in an evaluation state after the move.

The system in Example 5.2 is in an evaluation state. Resources $r_2$ and $r_3$ are order-one knots. For order-one knot $r_2$, part $a$ must cross from $c_1$ to $c_2 \cup c_3$ before any other part can leave $c_1$, and part $b$ must cross $c_2 \cup c_3$ to $c_1$ before any other part can leave.
For order-one knot $r_3$, part $a$ must cross from $c_1 \cup c_2$ to $c_3$ before any other part can leave $c_1 \cup c_2$ and part $b$ must cross $c_3$ to $c_1 \cup c_2$ before any other part can leave $c_3$. After moving either part $a$ or part $b$, both $c_2 \cup c_3$ and $c_1 \cup c_2$ are in evaluation states.

The next lemma will show how the parts are committed when a chained closed path is in an evaluation state.

Lemma 5.1. Given a chained closed path $c = (R, A)$ in a WRG $G$ that is in an evaluation state $n$, $\text{space}(c, n) = 0$ if, and only if, all order-one knots are empty in $c$ and all other resources in $c$ are filled and committed to resources on $c$.

Proof. Let $R_k = \{ r : \text{order}(r, c, n) = 1 \}$ and $R_n = R - R_k$. Let $A_k$ and $A_n$ be the sets containing the committed arcs that the parts in $R_k$ and $R_n$ commit respectively in state $n$. Since $\text{space}(c, n) = 0$, we can write

\[
\text{order}(c, n) = \text{slack}(c, n) = \text{cap}(R) - \text{com}(A, n) = \text{cap}(R_k) - \text{com}(A_k, n) + \text{cap}(R_n) - \text{com}(A_n, n) \tag{5.3}
\]

Since $c$ is in an evaluation state, all order-one knots are free. Since an empty resource does not commit any arcs, we have

\[
\text{com}(A_k, n) = 0. \tag{5.4}
\]

Since the capacity of a knot is one, we can conclude

\[
\text{cap}(R_k) = \text{order}(c, n). \tag{5.5}
\]

Substituting (5.4) and (5.5) into (5.3) we have

\[
\text{cap}(R_n) = \text{com}(A_n, n) \tag{5.6}
\]
From (5.6), we can conclude that no free resources exist on \( c \) except for the order-one knots.

We still need to show that if the only free resources in \( c \) are the order-one knots and all other resources in \( c \) are filled and committed to resources on \( c \) then \( \text{space}(c, n) = 0 \).

We can write

\[
\text{space}(c, n) = \text{slack}(c, n) - \text{order}(c, n)
\]

\[
= \text{cap}(R) - \text{com}(A, n) - \text{order}(c, n) \tag{5.7}
\]

\[
= \text{cap}(R_k) - \text{com}(A_k, n) + \text{cap}(R_n) - \text{com}(A_n, n) - \text{order}(c, n)
\]

Since \( c \) is in an evaluation state, all order-one knots are free. Since an empty resource does not commit any arcs, we have

\[
\text{com}(A_k, n) = 0. \tag{5.8}
\]

Since the capacity of all knots are one, we can conclude

\[
\text{cap}(R_k) = \text{order}(c, n). \tag{5.9}
\]

Substituting (5.8) and (5.9) into (5.7), we have

\[
\text{space}(c, n) = \text{cap}(R_n) - \text{com}(A_n, n) \tag{5.10}
\]

But since all resources in \( R_n \) are filled and committed to resources on \( c \), then

\[
\text{cap}(R_n) = \text{com}(A_n, n). \tag{5.11}
\]

Substituting (5.11) into (5.10), we can conclude

\[
\text{space}(c, n) = 0
\]

The next two lemmas are preliminary results be required to prove the final theorem of this section.
Lemma 5.2. Given a chained closed path \( c \) that is in an evaluation state \( n \), if \( \text{space}(c, n) = 0 \) then a part \( q \) exists such that when it is moved, it will fill an order-one knot and commit an outgoing arc of that knot on \( c \).

Proof. According to Lemma 5.1, all resources are filled except the order-one knots. Let \( r \) be any order-one knot on \( c \). Since \( r \) is a knot, we can define \( c = c_1 \cup c_2 \) and \( c_1 \cap c_2 = r \). Without loss of generality, assume that part \( q \) is on \( c_1 \). Since state \( n \) is in an evaluation state, a part on \( c_1 \) exists that must cross from \( c_1 \) to \( c_2 \) before any other part can leave \( c_1 \). If part \( q \) is that part, a part \( q \) exists that when moved that will fill an order-one knot and commit to an outgoing arc of the order-one knot.

Lemma 5.3. Given a non-empty chained closed path \( c \) that is in an evaluation state \( n_0 \), if \( \text{space}(c, n_0) = 0 \) then propagating any part will create a chained closed path \( c_2 \) such that \( c_2 \subset c \) and \( \text{space}(c_2, n_1) = 0 \).

Proof. Let \( q_1 \) be an enabled part that can propagate into an order-one knot \( k \). Let \( c_1 \) and \( c_2 \) be defined such that \( c = c_1 \cup c_2 \) and \( c_1 \cap c_2 = k \). By Lemma 5.2, propagating \( q_1 \) will fill a knot \( k \) and commit to \( c_2 \) in state \( n_1 \). Clearly, since \( c = c_1 \cup c_2 \) and \( c_1 \neq \emptyset \) we can conclude \( c_2 \subset c \). Clearly, knot \( k \) is not a knot with respect to \( c_2 \). According to Definition 5.6, closed path \( c_2 \) is in an evaluation state. Also, all order-one knots are empty on \( c_2 \) and all other resources are filled and committed to \( c \). Therefore, by Lemma 5.1 \( \text{space}(c_2, n_1) = 0 \).

Definition 5.7. If any subgraph in a WRG \( G \) is dead, then \( G \) is dead.
Theorem 5.2. Given a non-empty chained closed path \( c \) that is in an evaluation state \( n_o \), if \( \text{space}(c, n_o) = 0 \), then \( c \) is dead.

Proof. Lemma 5.2 states an enabled part \( q \) exists which when propagated will fill an order-one knot. According to Lemma 5.3, propagating part \( q \) will create \( c^* \) such that \( c^* \subseteq c \) and \( \text{space}(c^*, n_o) = 0 \). If \( c^* \) does not contain a knot, we can conclude by Theorem 5.1 that \( c \) is dead. Recursive application of this theorem to \( c^* \) will eventually produce a basic closed path that is dead. Then by Definition 5.7, \( c \) is dead.

5.3 Complex Closed Paths

Closed paths that are not basic closed paths or chained closed paths are classified as complex closed paths. This section will introduce complex closed paths. It will also be shown, if a complex closed is in an evaluation state and it contains a path with space equal to zero, then this is sufficient for determining if the system is dead.

We will first define a complex closed path and its various components, then follow these definitions with an example.

Definition 5.8. A complex closed path is a closed path that contains one or more order one knots that is not a chained closed path.

Definition 5.9. A complex path can be decomposed into two paths, one being a chained closed path and the other is called the auxiliary closed path. The intersection of the auxiliary closed path intersects and the chained closed path must contain one or more order one knots of the chained path.
**Definition 5.10.** A bypass path is the portion of the auxiliary path that does not intersect the chained closed path.

**Definition 5.11.** The first arc on the bypass path is a bypass arc.

Consider the following example.

**Example 5.4.** Suppose that the system in Figure 5.4 has the following parts and process plans as depicted in Table 5.3.

<table>
<thead>
<tr>
<th>Part</th>
<th>Process Plan</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>r_1r_2r_3r_4</td>
</tr>
<tr>
<td>b</td>
<td>r_4r_2r_5r_6</td>
</tr>
<tr>
<td>d</td>
<td>r_3r_5r_6r_1</td>
</tr>
</tbody>
</table>

Assume that the system is in state \( n = [n(a_1), n(a_2), n(b), n(d_1), n(d_2)] = [3,1,1,3,2] \).

The system consists of three simple closed paths: \( c_1 = r_1r_2r_3r_4 \), \( c_2 = r_2r_5r_6r_2 \), and \( c_3 = r_2r_5r_6r_1r_2 \). The order \( r_2, c_1 \cup c_2, n \) = 1. Clearly, the manufacturing system in Figure 5.4 is not a basic closed path. The system cannot be a chained closed path either since
(c_1 \cup c_2) \cap c_3 \text{ is not a knot. According to Definition 5.8, the system in Figure 5.4 is a complex closed path. The complex closed path can be decomposed into a chained closed path, (i.e., } c_1 \cup c_2, \text{ and an auxiliary closed (i.e., } c_3). \text{ Closed path } c_3 \text{ is an auxiliary closed path since the intersection of } c_3 \text{ and the chained closed path } c_1 \cup c_2 \text{ contain the order-one knot } r_3. \text{ The simple path } r_3 r_5 r_6 \text{ is a bypass path that joins } c_1 \text{ and } c_2 \text{ together. Arc } a_6 \text{ on resource } r_1 \text{ is a bypass arc since it leaves } r_1 \text{ along the auxiliary closed path } c_3.

We next define the evaluation state for a complex closed path.

Definition 5.12 A complex closed path in a WRG G is in an evaluation state if its bypass arcs are not committed.

Definition 5.13. An empty subgraph that is a complex closed path in a WRG G in state } n \text{ is in an evaluation state.

Definition 5.14. A WRG G is in an evaluation state if all closed paths in G are in an evaluation state.

The system in Example 5.3 is in an evaluation state. This is because part } a_1 \text{ in resource } r_2 \text{ is not committed to the bypass arc } a_6. \text{ The chained closed path } c_1 \cup c_2 \text{ is in an evaluation state per Definition 5.6. The space of the chained closed path } c_1 \cup c_2 \text{ is zero. Clearly, the system is dead. The next theorem proves this concept in general.

Theorem 5.3. Given a complex closed path } c_p \text{ that is in an evaluation state } n_0, \text{ if any chained closed path } c^* \subset c_p \text{ has space}(c^*, n_0) = 0, \text{ then } c_p \text{ is dead.}

Proof. According to Theorem 5.2, closed path } c^* \text{ is dead. Therefore, by Definition 5.7, } c_p \text{ is dead.}
5.4 Main Results

The theorem in this section proves that if any closed path in a WRG $G$ that is in an evaluation state has a space equal to zero, $G$ is dead.

*Theorem 5.4.* Given any closed path $c$ in a WRG $G$ in an evaluation state $n$ if $\text{space}(c, n) = 0$, then $G$ is dead.

*Proof.* Closed path $c$ can be a basic, a chained or a complex closed path. Applying Theorems 5.1, Theorem 5.2 or Theorem 5.3 respectively to these three path types will show that $c$ is dead. Therefore, since $c$ is dead we can conclude by Definition 5.7 that $G$ is dead.

The following Theorem proves the main result of this dissertation.

*Theorem 5.5.* Let $C_G$ be the set of closed paths in a WRG $G$ in an evaluation state $n$, then $G$ is live if, and only if,

$$\text{space}(c, n) > 0 \quad \forall c \in C_G.$$

*Proof.* Using Theorem 4.6 and applying the contrapositive of Theorem 5.4 proves this theorem.

Theorem 5.5 is valid only if the manufacturing system is in an evaluation state. Consider the following example where the system is not in an evaluation state.

*Example 5.5* Suppose that the system in Figure 5.5 has the following parts and process plans as depicted in Table 5.4.
Table 5.4 Process plans for example 5.4

<table>
<thead>
<tr>
<th>Part</th>
<th>Process Plan</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>r_1r_2r_4</td>
</tr>
<tr>
<td>b</td>
<td>r_4r_2r_6</td>
</tr>
<tr>
<td>d</td>
<td>r_3r_6r_1</td>
</tr>
</tbody>
</table>

Assume that the system is in state \( n = [n(a), n(b), n(d_1), n(d_2), n(d_3)] = [1, 1, 3, 2, 1] \).

Let circuit \( c_1 = r_1r_2r_6r_1 \), \( c_2 = r_2r_3r_4r_2 \), and \( c_3 = r_2r_3r_6r_2 \).

![Complex closed path for example 5.5](image)

The system is not in an evaluation state, since part \( d_3 \) is committed to bypass arc \( a_b \). Since the system is not in an evaluation state and \( \text{space}(c_1 \cup c_2 \cup c_3, n) = 0 \), my research cannot determine if the system is live or dead. Therefore, we must declare this state as an undetermined state even though the state is live.
Chapter 6 Results and Discussion

This final chapter will present the main results of Chapter 4 and Chapter 5 and explain issues relating to implementation. It will compare my research with the methods previously presented in the literature review. Finally, it will include a conclusion and recommendations section.

6.1 Results

The goal of this dissertation was to precisely quantify the necessary and sufficient condition for deadlock in a manufacturing system. To accomplish this goal, my research has expanded upon the previously presented research by Judd and Faiz. [16]. The following chapter segments outline my results.

6.1.1 Extending the Methods of Judd and Faiz

My research expands Judd and Faiz’s method by redefining the order of a knot. A knot (as described earlier in Definition 3.27) is a resource that has capacity of one and it is the only resource in the intersection of two closed paths. According to Judd and Faiz, the order of a knot (see Definition 3.28) is defined as one less than the number of closed path that share the knot. However, this definition of order is based only on the structure of the graph. My definition, which appears in Chapter 4, the order of a knot (see Definition 4.3) is based on the structure of the graph and the system state. This new definition of order incorporates part crossings through the knot. This definition is less restrictive; therefore, it allows for more live states to occur. The following example will demonstrate a system
in which Judd and Faiz cannot conclude whether the system is live or dead. The method in my research will claim it is live.

**Example 6.1.** The manufacturing system depicted in Figure 6.1 is composed of five resources $r_1, r_2, r_3, r_4$ and $r_5$ all with unit capacity. The system consists of three simple circuits $c_1 = r_1r_2r_3$, $c_2 = r_2r_4r_2$, and $c_3 = r_3r_2r_5$. Suppose that the system has the following parts and process plans as shown in Figure 6.1.

Table 6.1 Process plans for example 6.1

<table>
<thead>
<tr>
<th>Part</th>
<th>Process Plan</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>$r_1r_2r_3r_4$</td>
</tr>
<tr>
<td>$b$</td>
<td>$r_4r_2r_5$</td>
</tr>
<tr>
<td>$c$</td>
<td>$r_3r_2r_1$</td>
</tr>
</tbody>
</table>

Assume that the system is in state $n = [n(a), n(b), n(c)] = [1, 1, 1]$. Clearly, this state is live, because part $a$ can move to resource $r_1$; part $c$ can then freely exit the system; and part $b$ can then freely exit the system followed by part $a$. While Judd and Faiz's method cannot claim the system as live, my method presented in this research reveal that the system is live.
Figure 6.1 Manufacturing system for example 6.1

The structure of the graph in Figure 6.1 is a three leaf corolla with leaves $c_1$, $c_2$, and $c_3$ joined at knot $r_2$. The method of Judd and Faiz fails to incorporate the fact that the leaves are cyclically cross-connected through knot $r_2$ in this state. Table 6.2 depicts the slack, order and space computations per Judd and Faiz and my method used in this research.

Table 6.2 Slack, order and space computations

<table>
<thead>
<tr>
<th>Circuit</th>
<th>Judd and Faiz</th>
<th>Chapter 4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Slack</td>
<td>Order</td>
</tr>
<tr>
<td>$c_1$</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$c_2$</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>$c_3$</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>$c_1 \cup c_2$</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>$c_1 \cup c_3$</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$c_2 \cup c_3$</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>$c_1 \cup c_2 \cup c_3$</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>
Recall both methods state, if the space of all closed paths are greater than zero, then the system is live. As shown in Table 6.2, the method of Judd and Faiz cannot claim the system is live because \( \text{space}(c_1 \cup c_3, n) = 0 \) and \( \text{space}(c_1 \cup c_2 \cup c_3, n) = 0 \). Clearly, my method can claim the system is live, since all space values are greater than zero. It is easy to conclude that defining the order based on structure, as well as system state, allows the system to enter more live states.

### 6.1.2 Necessary and Sufficient Conditions for Deadlock

Theorem 4.6 proved sufficient conditions for a manufacturing system to be live. That is, if the space of all closed paths is greater than zero, then this condition is sufficient for a manufacturing system to be live. Unfortunately, the sufficient condition for a manufacturing system to be dead requires the system to be in an evaluation state. There is insufficient information in the WRG to determine these conditions without the system being in an evaluation state. Theorem 5.4 proved sufficient conditions for a manufacturing system in an evaluation state to be dead. The combination of Theorem 4.6 and the contrapositive of Theorem 5.4 resulted in Theorem 5.5. Theorem 5.5 proves necessary and sufficient conditions for a manufacturing system in an evaluation state to be live; that is, a manufacturing system in an evaluation state is live if and only if the space of all closed paths in the manufacturing system is greater than zero.

### 6.2 Implementation

An algorithm that implements the methods presented herein can be applied to any process control systems to detect and to avoid deadlock in actual manufacturing systems.
The methods can be implemented in three different levels. Any algorithm that is written and implemented at any one of these levels will ensure that all deadlock states are avoided. The higher the level of implementation, the more complex the algorithm will be.

A first level implementation might be implemented in the Judd and Faiz model to define the order of all knots to as one. Clearly, this is an improvement on Judd and Faiz's method since order is defined as one less than the number of simple circuits that share a knot. A second level of implementation might include computing the order of each knot per Definition 4.3, in which the system state is incorporated into the order computation. Implementing the algorithm at this second level will add more complexity. A flowchart of these two implementations is depicted in Figure 6.6.
1. Create the System WRG
2. Let $C_G$ be the set of all closed paths
3. Identify all Knot Resources

Figure 6.2 First and second level implementation flowchart
A third level or full implementation might include the addition of determining whether each closed path is in an evaluation state. If any closed path in the system is not in an evaluation state, then we must resort back to the second level implementation. For this reason, my method will still miss some live states as shown in Example 5.5. However, if the space is equal to zero of any closed path and the system is in an evaluation state then it can be absolutely concluded that the system is dead. In any case, determining whether a system is in an evaluation state will add yet another level of complexity to the algorithm. A flowchart of the third level implementation is depicted in Figure 6.3.
1. Create the System WRG
2. Let $C_0$ be the set of all closed paths
3. Identify all Knot Resources
4. Order of Knots state Dependent

For each $c$ in $C_0$

Is $c$ in an evaluation state?

Space of $c = 0$

Let $C_0 = C_0 - \{c\}$
Let $c$ be an element of $C_0$

$|C_0| \neq 0$

Second Level Implementation
System is dead
System is live

Figure 6.3 Third level implementation flowchart.
The next section will compare the methods presented in this dissertation with that of the methods in the literature review which fail to utilize resources to their full capacity.

6.3 Comparison with Past Research Methods

Each subsection addresses a method presented in Chapter 2.

6.3.1 Banaszak and Krogh and Hsieh and Chang

Example 6.2 The manufacturing system depicted in Figure 2.1 has been redrawn using the WRG formalism in Figure 6.4. This example will show the restrictive nature of the method presented by Banaszak and Krogh [2] and Hsieh and Chang [18]. This manufacturing system consists of six unit capacity resources $I_1, O_1, I_2, O_2, I_3$ and $O_3$ and one simple circuit $C_1 = I_3 O_3 I_2 O_2 I_1$. The system manufactures one product $p_1$ specified by the process plan $\text{plan}(p_1) = I_3 O_3 I_2 O_2 I_1 O_1$. The current state of the system is $n = [q_1 q_2 q_3] = [2 5 6]$. The restriction policies, DAA1 and DAA2, in [2] and DAC in [18] will only allow two parts on simple circuit $c_1$ at anyone time. Part $q_1$ in Figure 6.4 would not be allowed to enter $c_1$ even though this is a perfectly safe move. Clearly, this is not the case with the method in Chapter 4. Since $c_1$ is a simple circuit, we can conclude that if $\text{slack}(c_1, n) > 0$ then $c_1$ will remain live. $\text{Slack}(c_1, n) = 2$ implies that at least one more part may enter $c_1$. 
6.3.2 Zhou and DiCesare

Example 6.3 This example will show the restrictive nature of Zhou and DiCesare [21]'s method. The manufacturing system depicted in Figure 2.2 has been redrawn using the WRG formalism in Figure 6.5. Suppose that the system has the following parts and process plans as depicted in Table 6.3.

Table 6.3 Process plans for example 6.3

<table>
<thead>
<tr>
<th>Part</th>
<th>Process Plan</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>r_1 r_2</td>
</tr>
<tr>
<td>b</td>
<td>r_3 r_2 r_1</td>
</tr>
</tbody>
</table>

The method in [21] will only allow for a single part to concurrently occupy $r_1$ and $r_2$ in $c_1$. This method eliminates the deadlock state where $a$ and $b$ occupy $r_1$ and $r_2$ respectively, that is $[a,b] = [1,2]$. Although this avoids the deadlock state, it also
eliminates the states where two parts of the same product type occupy \( r_1 \) and \( r_2 \) concurrently, that is states \([a_1, a_2] = [1, 2]\) and \([b_1, b_2] = [2, 3]\). Clearly, this method is not practical for implementation.

Since \( \text{order}(c_1, n) = 0 \) for any state \( n \), then \( c_1 \) is a basic closed path. Since \( c_1 \) is a basic closed path, then \textit{by Theorem 5.1} if \( \text{space}(c_1, n) = 0 \) for any state \( n \), then \( c_1 \) is dead. The only state not allowed is \([a, b] = [1, 2]\). Therefore, \textit{Theorem 5.1} is necessary and sufficient for deadlock.

### 6.3.3 Barkaoui and Addallah

Barkaoui and Addallah in [15] developed a two-phase method to avoid deadlock. Consider the following example.

\textit{Example 6.4} The manufacturing system depicted in Figure 2.3 has been redrawn using the WRG formalism in Figure 6.6. Suppose that the system has the following parts and process plans as depicted in Table 6.4.

<table>
<thead>
<tr>
<th>Table 6.4 Process plans for example 6.4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Part</td>
</tr>
<tr>
<td>------</td>
</tr>
<tr>
<td>( a )</td>
</tr>
<tr>
<td>( b )</td>
</tr>
</tbody>
</table>

The control policies in [15] will only allow two parts of any type in the system at any one time. This is because the method only allows two parts in circuit $c_2$ at anyone time. This method allows several states in which four parts are the system, i.e. $[a_1, a_2, a_1, b_1]=[1,2,3,2]$, $[a_1, a_2, b_1, b_2]=[1,2,1,2]$, and $[a_1, a_2, a_1, b_1]=[6,5,1,2]$ since in these states the order of $r_2$ is zero.

### 6.3.4 Fanti, Maione, Mascolo and Turchiano

Fanti, Maione, Mascolo and Turchiano [23] developed the concepts of the Working Procedure Digraph and Transition Digraph to detect primary deadlock in a manufacturing system. This method introduced five restriction policies to avoid primary deadlock. The following example is taken from [23], using a one-step look-ahead method.

**Example 6.5** Suppose that the following manufacturing system has the following parts and process plans as depicted in Table 6.5.

Table 6.5 Process plans for example 6.5

<table>
<thead>
<tr>
<th>Part</th>
<th>Process Plan</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>$r_1 r_7 r_5 r_6$</td>
</tr>
<tr>
<td>$b$</td>
<td>$r_3 r_8 r_6 r_7$</td>
</tr>
<tr>
<td>$c$</td>
<td>$r_4 r_5 r_1$</td>
</tr>
</tbody>
</table>
The manufacturing system in Figure 6.7 is Example 8 of [23] that has been redrawn using the WRG. Assume that the system is in state
\[ n = [a_1, a_2, a_3, b_1, c_1, c_2] = [3, 2, 1, 1, 2, 1]. \]

Fanti et al. method in [23] is a one-step look-ahead method. The definition of deadlock only includes primary deadlock. An impending deadlock system is not considered as being in a deadlock state until it is placed into primary deadlock. As stated in [23], the state in Figure 6.7 is reachable. The method in [23] will not claim deadlock until part \( b \) or \( c \) is moved to resource \( r_5 \), which will place the system in primary deadlock. According to Theorem 4.4, state \( n \) is live because \( \text{space}(c_1 \cup c_2, n) > 0 \).

### 6.3.5 Cho, Kumaran and Wysk

Example 6.6 This example addresses the heuristic problem of Cho, Kumaran and Wysk in [5]. The manufacturing system depicted in Figure 2.5 has been redrawn using the WRG formalism in Figure 6.8. Suppose that the system has the following parts and process plans as depicted in Table 6.6.
Table 6.6 Process plans for example 6.6

<table>
<thead>
<tr>
<th>Part</th>
<th>Process Plan</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>$r_1r_2$</td>
</tr>
<tr>
<td>b</td>
<td>$r_3r_4r_5$</td>
</tr>
<tr>
<td>c</td>
<td>$r_3r_4r_5r_1$</td>
</tr>
</tbody>
</table>

Assume that the system is in state $n = [a, b, c] = [1, 1, 1]$.

As shown in Example 2.6, the method in [5] cannot determine if a system is live as long as resource $r_2$ (the common node) is occupied. A heuristic is used to clear the common node by moving part $b$ from resource $r_2$ to resource $r_4$, yet this move will cause primary deadlock on circuit $c_3$. Therefore, Cho et al’s method concludes that the system in Figure 6.8 is dead when, in fact, it is not.

According to Table 6.7 the space of all closed paths are greater then zero. Therefore, my method will conclude the state $n$ is live.
Table 6.7 Slack, order and space computations

<table>
<thead>
<tr>
<th>Circuit</th>
<th>Order</th>
<th>Slack</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_1$</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$c_2$</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>$c_3$</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$c_1 \cup c_2$</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>$c_1 \cup c_3$</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>$c_2 \cup c_3$</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>$c_1 \cup c_2 \cup c_3$</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

6.4 Conclusions

My research derives a method that quantifies both necessary and sufficient conditions for a manufacturing system to be live. It has shown that if a given a manufacturing system is in an evaluation state, the system is live if, and only if, the space of all closed paths is greater than zero. This was accomplished by redefining the order of a closed path that incorporates the system state first introduced by Judd and Faiz [16]. My present research leads me to the following conclusions.

1. The order of a knot resource is based on the structure and the system state of the WRG.

2. Space is sufficient to conclude whether a manufacturing system is live.

3. A manufacturing system in an evaluation state space is sufficient to conclude whether a manufacturing system is dead.

4. Insufficient information exists in the WRG formalism to prove necessary and sufficient conditions in the general case.
5. The methods presented allow for the detection of more live states and, thus, enable better resource utilization than any other method published to date.

6.5 Recommendations

In order to extend this research, the following items are recommended.

1. Modify Theorem 4.6 so that it can schedule part movements while avoiding deadlock. The modification will include determining the set of all part movements that will satisfy the space conditions when moving into a particular free resource. From this set, the optimal part movement can be chosen.

2. Extend the method to allow multiple resources per operation.

3. Develop an algorithm that is polynomial in complexity that may be implemented in real-time.

4. Extend the method to allow free and conditional choice.
Chapter 7 References


