TASK PLANNING WITH UNCERTAINTY FOR ROBOTIC SYSTEMS

by

Tiehua Cao

Rensselaer Polytechnic Institute
Electrical, Computer, and Systems Engineering
Troy, New York 12180-3590

March 1993

CIRSSE REPORT #137

To my parents

CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

I am deeply indebted to my thesis advisor, Professor Arthur Sanderson, for his guidance, support, encouragement, and perspective throughout the course of this research. Many of the key ideas in this thesis were stimulated in discussions with him, and this work would have been impossible without his help.

I wish to express my gratitude to Professors Alan Desrochers, Frank DiCesare, and Robert McNaughton for their valuable suggestions and advice, and for serving on my doctoral committee.

The support from the NASA Center for Intelligent Robotic Systems for Space Exploration (CIRSSE), Defense Logistics Agency, the Center for Manufacturing Productivity and Technology Transfer (CMPTT), and the Electrical, Computer, and Systems Engineering (ECSE) Department at Rensselaer is greatly appreciated, and particularly, CIRSSE and CMPTT for providing experimental robotic systems. Computational facilities for this research have been provided by CIRSSE, the Image Processing Laboratory (IPL), and the ECSE Department at Rensselaer. Many thanks to all the friends and colleagues in CIRSSE, CMPTT, IPL, and the ECSE Department for their help. In particular, I would like to thank CIRSSE for providing the unique research environment.

I am also thankful to the members of the research groups I have been with over the past years for their collaboration. Thanks to all the people who provided useful comments on various portions of this work.

It is difficult to describe in words the constant love, support, and encouragement from my parents, Huifeng Cao and Yafang Wu, and my brothers, Jianhua and Qinghua, throughout the years. I am extremely grateful for their giving so much. My special thanks go to my wife, Qing Chang, for her patience and support during this research.

ABSTRACT

In a practical robotic system, it is important to represent and plan sequences of operations and to be able to choose an efficient sequence from them for a specific task. During the generation and execution of task plans, different kinds of uncertainty may occur and erroneous states need to be handled to ensure the efficiency and reliability of the system. In this thesis, we demonstrate a novel approach to task representation, planning, and error recovery for robotic systems. Our approach to task planning is based on an AND/OR net representation, which is then mapped to a Petri net representation of all feasible geometric states and associated feasibility criteria for net transitions. Task decomposition of robotic assembly plans based on this representation is performed on the Petri net for robotic assembly tasks, and the inheritance of properties of liveness, safeness, and reversibility at all levels of decomposition are explored. This approach provides a framework for robust execution of tasks through the properties of traceability and viability. Uncertainty in robotic systems are modeled by local fuzzy variables, fuzzy marking variables, and global fuzzy variables which are incorporated in fuzzy Petri nets. Analysis of properties and reasoning about uncertainty are investigated using fuzzy reasoning structures built into the net. Two applications of fuzzy Petri nets, robot task sequence planning and sensor-based error recovery, are explored. In the first application, the search space for feasible and complete task sequences with correct precedence relationships is reduced via the use of global fuzzy variables in reasoning about subgoals. In the second application, sensory verification operations are modeled by mutually exclusive transitions to reason about local and global fuzzy variables on-line and automatically select a retry or an alternative error recovery sequence when errors occur. Task sequencing and task execution with error recovery capability for one and multiple soft components in robotic systems are investigated.

CHAPTER 1
INTRODUCTION

1.1 Motivation

In a practical robotic system, it is important to represent and plan sequences of operations and to be able to choose an efficient sequence from them for a specific task. The earliest planning methodologies emerged in the area of artificial intelligence where domain-independent planning techniques were developed. In a robotic system, a planning strategy oriented to the characteristics of the system is often more effective than techniques derived from domain independent methods. Conventional representation of a system model without constraints may result in a huge search space for system states and task sequences. During the execution of a planned task sequence, because of uncertainty associated with the robotic system, exceptional or erroneous states are often met and thus the sequence may fail. Many factors may lead to uncertainty, and different kinds of devices in the system, such as manipulators, sensors, task-oriented mechanisms, human-robot interfaces, and the working environment may bring incomplete, approximate, or random information. Because planning is based on the assumption of expected system states, the model of a system should also carry the capability to represent the fuzzy information and provide a robust mechanism to detect and recover from an erroneous state.

Robots are used in many industrial applications including manufacturing, assembly, hazardous environments, undersea or space exploration [31][59]. Feasibility, efficiency, and reliability requirements are necessary for these robotic systems. In assembly, material handling systems, or other manufacturing environments, we should know the following information:

- The geometric descriptions of all components in the system and all feasible

combinations of components which form groups during the execution of a task by the system — A *system state* is defined to be the set of all current geometric configurations of components or component groups at a given time point. A *system substate* is defined to be a subset of a system state. Any component or component group is an example of a system substate.

- Each feasible operation which functions on a corresponding substate of this system — These feasible operations are the feasible geometric relationships among components and component groups in the system. For the current system state, several operations may be enabled. After a feasible operation is performed on the current state, a new state will be created and a new set of enabled operations will be available.

- An initial system state and a final system state — Sometimes, a set of important intermediate system states, i.e., *subgoals*, are indicated to search all feasible sequences more efficiently. These states may be given by users so that a smaller number of feasible states are generated during the search process. Some constraints may be set by relating algorithms so that these subgoals are followed automatically.

- The feasibility assumptions and the descriptions of the working environment — Some operations may be recoverable, i.e., these operations are reversible. Some operations are not recoverable. To plan all intermediate points for a feasible collision-avoidance path for a robot arm, we need to know the geometric descriptions of obstacles in the working environment.

There are two methods to describe a robot task sequence. One method directly uses an operations sequence, which is either a symbolic description or a formal language description. In a symbolic description, a task sequence is equivalent to a string of symbols and each symbol represents a corresponding operation, while in a

formal language description, the set of all feasible task sequences is a specific *task language*. Another method is to use a sequence of partially or completely ordered system states to represent a task sequence. Using this method, we may conceive the state changes in a task sequence as well as monitor the completeness of an assigned task. In this research, we combine these two methods together. Before we search feasible sequences for a task, we first give an efficient and compact representation for the system and all feasible operations. We propose a novel representation for the system states as well as the transition criteria from system substates to substates. This representation can be mapped to and directly use the theory of Petri nets[84, 88, 89] and its applications in modeling and control of manufacturing systems[2]. A simulation tool[73, 74] is available to verify and simulate a sequence chosen from all feasible sequences, before this sequence is practically implemented.

To handle the uncertainty in a robotic system and to represent the subgoals for a global task, we apply the knowledge of fuzzy sets to the representation of the system. To generate a complete representation for the system and to efficiently plan all feasible sequences, different kinds of uncertainty should be analyzed and classified. An approach to fuzzy reasoning embedded into this upgraded representation is then used to compute enabled operations and reason about system states. This fuzzy representation can also reduce the search space for feasible task sequences by defining subgoals for crucial operations.

1.2 Objective of the Research

The objective of this research is to develop an approach to representation and planning with uncertainty for a general robot assembly or material handling system. In this research, error recovery with minimum effort in replanning and changing a system model is also investigated with the representation of fuzzy information. Representation and planning are based on the geometric descriptions of a robotic

system and its environment. The resulting correct sequences are the output of the task planner. This planner will be connected to the path planner, trajectory generator, grasp planner, vision servo system, and other lower level planning systems. The connections and the coordination among these planners are controlled and supervised on-line by a coordination level. The Petri net representation provides a good interface between the planning level and the coordination level.

The ability to represent and automatically find an alternative subsequence or a recovery sequence is very important for the efficiency and robustness of a robotic system. A robust planner should minimize the probability for replanning in case an error occurs. We propose a generalized representation which incorporates the geometric relationships and feasibility among objects or object groups, the subgoals which are necessary for a correct operations sequence, different error recovery strategies, and an efficient reasoning mechanism for uncertainty. Our fuzzy representation offers the following advantages when error recovery is needed.

- It may be unnecessary for the system to re-analyze the system states such as the current state and the initial state or final state, and it may be unnecessary to replan a recovery sequence as well .

- Choosing an alternative sequence can maximize the use of task sequences already generated. A local alternative recovery strategy makes the best use of identical machines. A global error recovery strategy reduces the probability for a system to return to the initial state, and increases the probability for the system to recover from errors and reach the final state.

- For an object containing multiple components of which the properties may change, when an operation on some of these components causes errors, we need not discard other components while recovering to a previous state.

In this research, we also want to analyze the properties of a system using our

representation. This becomes more important when we incorporate more lower level devices and operations into the higher level representation. The case is similar when fuzziness is introduced into the system model. Some of the important properties such as liveness, safeness, and reversibility are discussed.

1.3 Approach

During the course of this research, we generate a high level representation and then decompose it to lower levels. Task sequence planning and property analysis are done on different levels of decomposition. Fuzzy sets are then introduced into this representation, and fuzzy reasoning about uncertainty, planning for subgoals, and sensor-based error recovery are investigated.

1.3.1 A High Level Representation

The scenarios we use in this research may also be applied to other kinds of robotic systems. For example, in the scenario in which a robot moves a book on table A to table B, all operations involved in this scenario can be described as *assembly*, *disassembly*, or *IST*(Internal State Transition) operations, the three basic operations in a generic assembly system which we will discuss. The representation and planning methods for assembly systems may also be used for this example.

We define a *subassembly* in an assembly system as a feasible combination of several components. We define an *assembly* as a special subassembly which is a substate of the final state, and which does not appear in any intermediate state or the initial state. We define an *object* in a robotic system as either a single component, a subassembly, or an assembly. After investigating a generic robot assembly system, we conclude that all possible operations appearing during the time the system is in execution can be classified as three types of basic operations: assembly operations, disassembly operations, and IST operations. Generally, for

an assembly operation, there is more than one object as the precondition, and one object as the postcondition. For a disassembly operation, there is one object as the precondition and more than one object as the postcondition. For an *IST* operation, both precondition and postcondition contain one object.

We first check all components in the system and try to find all feasible combinations of components, and therefore all feasible objects are generated. This checking process may be performed by the computer via a use-machine interacting mode. A *system geometric state representation* will then be generated. Based on this representation and all feasible geometric relations among objects, an *AND/OR net* representation of the corresponding system will be generated. The generation of this net is based on the feasibility assumption for each transition in the net. If some transitions are shown to be infeasible in one direction, a *directed AND/OR net* can be used to model the system. A mapping algorithm is shown to transform an AND/OR net to an ordinary Petri net and a reachability tree of this net from the initial state to the final state may be created based on an existing algorithm[84, 89]. A data structure for searching all possible sequences and the shortest sequence is developed with AND/OR nets. 1-boundedness, safeness, liveness, and reversibility have been proven to be guaranteed for the Petri net mapped from an AND/OR net. The safeness and 1-boundedness of the Petri net mapped from a directed AND/OR net are also proven to be guaranteed. The following research is based on the representation using AND/OR nets and mapped Petri nets for the system.

1.3.2 Hierarchical Planning Decomposition

The Petri net mapped from an AND/OR net is called a *Level 0 Petri Net(PN0)*. Each assembly transition in this net can be decomposed to a *motion* command and a *grasp* command. Each disassembly operation can be decomposed to an *ungrasp* command and a *motion* command. An *IST* operation is more problem-oriented and

we do not decompose it at this level. Therefore, some transitions in $PN0$ may be replaced by a subsequence of transitions and a *Level 1 Petri Net(PN1)* is thus generated. The properties of $PN0$, 1-boundedness, safeness, liveness, and reversibility, are shown to be inherited by $PN1$.

In $PN1$, we still have not included the lower level objects such as sensors. Each transition in the net, which corresponds to a specific operation in the robotic system, requires a plan to control the execution of subtasks. For example, before the robot performs a motion operation, i.e., to move from an initial point to a final point, a collision-avoidance path should be planned. Another example is that a grasp operation needs a grasp plan, i.e., we should choose a collision-free *initial grasp configuration* and a *final grasp configuration*[70]. During the grasp operation, the robot must be *safe* in both the initial grasp configuration and the final grasp configuration. A collision-avoidance path is required for the robot to reach the final configuration. Also, the grasping should be *stable* during the transfer motion. The problem-oriented operation, IST operation, needs a corresponding plan. For motion operations, two different kinds of motions may be classified and the corresponding motions are different in the sense of methodologies of accomplishments. Therefore, in the next level of decomposition, each motion command is decomposed to a *free-motion* command and a *fine-motion* command. The properties of the resultant net are also verified to be 1-bounded, safe, live, and reversible.

At the next step of decomposition, we add plans as preconditions for all transitions in the net and add sensors as preconditions for each sensor-based motion transition. The resulting net still holds the properties which the upper level net has. For each plan in the net, a strategy is investigated to develop some subnet for planning to replace this plan, i.e., how plans are generated, which resources are used for planning, how uncertainties are reduced before planning on-line, and whether there is any resource conflict during the planning process and the execution process.

If we can represent resource conflicts for task planning in a lower level net, a shortest sequence planned from the corresponding Petri net will be of the lowest probability of error occurring.

1.3.3 Generalized Fuzzy Petri Nets

Because of the necessity to represent and reason about uncertainty within a robotic system, we propose a definition of the generalized fuzzy Petri net with three types of fuzzy variables. This definition is used for the following research in planning under uncertainty and sensor-based error recovery for robotic systems. The theory of generalized fuzzy Petri nets can also be applied to other kinds of applications in artificial intelligence, knowledge based systems, and manufacturing. Fuzzy state representation and reasoning rules with fuzzy sets are also incorporated with fuzzy Petri nets. Property analysis for some basic cases of system models are performed.

1.3.4 Planning for Subgoals

A planning Petri net usually incorporates a great number of possible task sequences. When all possible sequences are generated, it still takes a lot of time to choose from them. Normally, these sequences guarantee the properties of feasibility because of the feasibility constraints of the representation. However, some important events or operations, which we will call as *subgoals*, should be included in each *correct sequence* and should satisfy the correct precedence relationship. We define a *complete sequence* as a sequence which contains all subgoals, i.e., all important events. We may require that all possible sequences searched from the Petri net satisfy the properties of feasibility, completeness, and have correct precedence relationships. Based on this requirement, we propose a *prime number marking algorithm* to map the ordinary Petri net we developed to a fuzzy Petri net with global fuzzy variables, so that a strong numerical constraint is satisfied during the search for sequences.

A set of fuzzy reasoning rules are proposed to select an enabled transition and to obtain the fuzzy values of tokens in the output places for this transition. We notice that the properties of some objects may change. Generally, the properties of an object can either be the size, such as radius, length, thickness, etc., or the shape, and other physical properties, or the processing and machining characteristics, of a component this object contains, or the parameters of the structure of this object, such as the distance or the geometric relationships among different components within this object.

Before the process for searching sequences starts, all possible fuzzy values are generated and stored using the prime number marking algorithm. During the search process, if a partial sequence meets a fuzzy value which is not in the possible fuzzy value set, this partial sequence will be discarded and a theorem guarantees this deletion will not lose any correct sequences. If not discarded, all partial sequences continue their development until the complete sequences are generated. Through the implementation of the theorems developed for this research, we found that the set of correct sequences for an assembly system, where one or more components of which the properties may change, is a very small portion of all possible sequences. Therefore, both the storage for saving all these possible sequences and the selection time for choosing the optimal or near-optimal sequence from the set of correct sequences are reduced.

1.3.5 Alternative Error Recovery

During the execution of a selected sequence from the set of correct sequences, some unexpected errors may occur and the sequence is therefore unexecutable. This happens because all operations in a planned task sequence are *expected* to be successful and the execution of any operation in a sequence is dependent on the successes of its previous operations. In particular, in a fuzzy Petri net representation of a

robotic system, all transitions which change the properties of objects are defined to be *key transitions*. The marking for each transition is defined as a *weighting factor*. All weighting factors for key transitions are *expected values*. Therefore, all fuzzy values for the output places of certain key transitions are also expected values. If at some time point, during the implementation of a task sequence, an error occurs, this error will be propagated throughout the net to the final state and it may be difficult to check where the error occurs. An error recovery strategy is especially important for key transitions. If we detect possible errors just after key transitions, an immediate *mutually exclusive* recovery procedure may be followed.

Therefore, for a certain key transition, if we find the output of this transition is beyond a certain range, we may either retry it, or use a local alternative subsequence, or recover back to a previous state or the initial state. All the previous transitions are refired after the corresponding component is replaced, or the parameters of the structure of an object are changed. The selection among these three directions will depend on the local fuzzy value of the object. To retain the original representation of a fuzzy Petri net model for a system, we use a sensor-based error recovery strategy for all key transitions. After a key transition is fired, a sensor verification procedure is called to investigate the current state of an object. Then, depending on the sensed value, the sequence will either retry in the local range, or continue its execution, or automatically choose a local alternative subsequence to execute, or go back to a previous state and follow a global alternative sequence to execute. There is a limit on the number of retries of the local transition. We have developed a method which automatically decreases the fuzzy value of an object when the sensed value continues to stay in the range of retry. After a finite number of retries, if the fuzzy value is still not correct as expected, an alternative local error recovery sequence will be automatically followed.

1.4 Contributions

The main contributions of this research are shown as follows:

- We introduce an AND/OR net representation for robotic task sequence planning. An algorithm is developed to map this AND/OR net representation to a Petri net. Property analysis is performed on the resulting Petri net. Data structures for searching all possible operations sequences and the shortest sequence are proposed and implemented. A directed AND/OR net representation is also developed.

- Algorithms are developed to decompose the high level representation of task sequence planning for generic robotic systems. The inheritance of properties between different levels of decompositions are investigated based on proven theorems. Traceability and viability are shown with robust execution of tasks using this approach.

- We propose a generalized definition of a fuzzy Petri net. This fuzzy Petri net is shown to represent and reason about uncertainty. Fuzzy state representation and fuzzy reasoning rules are defined for the net. Three types of fuzzy variables are discussed. Fuzzy sets are shown to be directly used with fuzzy Petri nets and fuzzy computations are used for reasoning. Basic cases of property analysis with fuzzy Petri nets are given.

- Fuzzy Petri nets with global fuzzy variables are used for task sequence planning. Algorithms for assigning global fuzzy variables to fuzzy Petri nets are proposed. Fuzzy transition rules for global fuzzy variables are given. Fuzzy reasoning for global tasks are shown to reduce the search space for all feasible, complete, and correctly ordered sequences.

- Fuzzy Petri nets with local and global fuzzy variables are used for sensor-based error recovery for task sequences. Mutually exclusive transitions are proposed for on-line selection for alternative enabled transitions. Deterministic and nondeterministic fuzzy Petri nets are discussed. Theorems of error recovery with fuzzy Petri nets and related algorithms are proposed.

1.5 Thesis Outline

In Chapter 2, we present a literature review of recent research in task planning, assembly planning, planning under uncertainty, and Petri nets with fuzzy data. A conclusion of this review relevant to our research is also given.

Chapter 3 and Chapter 4 discuss the methodology of representation for robotic task sequences without the consideration of uncertainty. Section 3.2 deals with the AND/OR representation for robotic systems. An algorithm is shown to generate an AND/OR net from the descriptions of system geometric states. A mapping algorithm for transforming an AND/OR net to a Petri net is then given in Section 3.3. 1-boundedness, safeness, liveness, and reversibility of the resulting Petri net are analyzed. In this section, a directed AND/OR net is also defined and the similar properties are analyzed. In Section 3.4, we present a data structure for searching all possible sequences and the shortest sequence from the AND/OR net. We give an example of task sequence planning using AND/OR nets in Section 3.5.

Chapter 4 generalizes the results given in Chapter 3 and introduces a representation of a generic robotic assembly system by decomposing the higher level nets. Section 4.2 gives the definitions for a generic assembly system. Section 4.3 reviews the AND/OR net representation specially for assembly sequence planning. We then discuss the decomposition algorithms for Level 1 and Level 2 Petri nets and analyze the properties for resulting lower level nets in Sections 4.4 and 4.5. The conditions for the inheritance of properties are discussed. In Section 4.6, a robotic assembly

system modeled using the methods discussed above is simulated. Section 4.7 gives a conclusion about the shortest sequence.

Chapter 5 is independent of the other chapters of the thesis, and develops a novel definition of a fuzzy Petri net(FPN). The results represented in this chapter can be used for other applications in artificial intelligence, knowledge-based systems, and manufacturing. The presentation of the following two chapters are based on Chapter 5. Three types of fuzzy variables are proposed for different kinds of uncertainty modeled with fuzzy Petri nets. State representation and reasoning rules in the FPN are given in Sections 5.3 and 5.4, respectively. Property analysis is then given for different conditions of the FPN in Section 5.5. Two examples of the FPN with different kinds of variables are shown for task planning and robot sensing in the following two sections.

Chapter 6 uses fuzzy Petri nets for robot task sequence planning. A detailed discussion on state representation for task sequences is given in Section 6.2. We show fuzzy sets for modeling system states in Section 6.3. An algorithm for assigning global fuzzy variables is then proposed in Section 6.4. Some theorems about using global fuzzy variables to search feasible sequences are also given. Section 6.5 generalizes the results in Section 6.4 and a fuzzy representation for a system with more general characteristics is discussed. Simulation results, especially the comparison of the complexity for searching sequences with ordinary Petri nets, are given in Section 6.6.

Chapter 7 is an application of FPNs for sensor-based error recovery for robotic task sequences. Some important definitions are given in Section 7.2. Section 7.3 gives the fuzzy transition rules for global fuzzy variables. In Section 7.4, execution of plans on the fuzzy Petri net is discussed with the introduction of mutually exclusive transitions. Deterministic and nondeterministic fuzzy Petri nets are discussed

regarding different characteristics of local fuzzy variables. The issue of error recovery is presented in details in Section 7.5 for different cases. An algorithm for the execution of a task sequence with sensory verification and error recovery is shown. We prove the theorems regarding automatic error recovery. An algorithm for generating an executable FPN is presented in 7.6, followed by the examples using different error recovery strategies in Section 7.7.

Chapter 8 summarizes the presentation of original contributions in this thesis. Directions of future research based on the existing results are discussed.

CHAPTER 2
LITERATURE REVIEW

2.1 Introduction

In this chapter, we briefly review the work relating to our research and investigate the relationships of these published results with our work. Our research was originated from a project on the integration of an automated garment handling system[9, 101, 102], which was supported by the Defense Logistics Agency. During the research on modeling, planning, and software integration on this project, we constructed the AND/OR net modeling tool, and this methodology was then applied to the NASA/CIRSSE space robotic assembly project. The resulting approach to task decomposition, planning under uncertainty, fuzzy descriptions for objects, state identification and verification, and error detection and recovery, is based on Petri nets and fuzzy Petri nets. In this chapter, we review the literature on task planning and assembly planning, and then show some previous results on the representation of uncertainty for task planning. A concise review of the research efforts in Petri nets using fuzzy data is also included.

2.2 Task Planning

Research in robotic planning is closely related to some corresponding areas in artificial intelligence. Domain-independent planning methodologies have been developed which can generate sequences of actions to change the initial world model and make it satisfy the final goal conditions. All possible actions in the sequences belong to a feasible set of operators which cause changes in the state of the system. The ability to reason about actions is a core problem to design a planning system. Domain-independent planners yield planning techniques that are applicable in many

domains with some modifications and provide a general planning capability. A review paper[43] describes the development of classical plan generation systems, the important problems that have arisen in the design of planning systems, and some solutions that have been developed in over 30 years of research in this area. Planning research has identified many issues in the fields of AI including representation, reasoning, search, learning, sensation and perception, and distributed systems. A survey on the AI approach to robot planning is given in [76].

Besides the property of generality for domain-independent planning, a general planner should also provide representations and methods to include domain-specific knowledge and heuristics. A number of planning systems corresponding to this theme have been created, such as EMYCIN[77], NOAH[96], STRIPS[36, 37], MOLGEN[103, 104], DEVISER[114], and SIPE[118]. These planners are designed for a general problem solving environment. Most of the input knowledge takes the form of predicate calculus formulas, and actions are given in the form of transformation rules. Problem solving using these planners requires the capability for representing, retrieving, and manipulating sets of statements. Thus, extensive computing power for searching and inference in order to solve a reasonably complex real-world problem is needed for AI planning. Recent systems have overcome some drawbacks of previous systems, however, current planning systems are still not robust and efficient enough to operate in complex robot workcells. Therefore, in robot working environments, and particularly assembly workcells, domain-dependent planning methods, though possibly lacking generality, are often more effective since they represent and reason about domain-related constraints directly.

A task planner transforms the task-level specifications into manipulator level specifications. To carry out this transformation, the task planner must have a description of the objects being manipulated, the task environment, the robot carrying out the task, the initial state of the environment, and the desired final goal. The

output of the task planner would be a robot program to achieve the desired final state when executed in the specified initial state. Robotic task planning may be divided into three phases: modeling, task specification, and manipulator program synthesis[70]. The modeling phase consists of the following information: geometric description of all objects including robots in the task environment, physical descriptions of all objects, kinematic descriptions of all linkages and descriptions of characteristic of moving objects such as robots. Task specification corresponds to sequences of states of the world model, where state is defined to be the configurations of all objects in the system. This is the goal of domain-independent planning research as well as domain-dependent planning. Our research shown in this thesis will focus on planning the sequences of operations which change the geometric configurations of the system. The manipulation program synthesis stage is to map task plans to the corresponding manipulation program which is composed of motion commands, grasp commands, sensing commands, and commands for grasp planning, motion planning, etc.

Generally, task planning in robotics requires precise models and knowledge about mechanical and geometric specifications. Research in motion planning, grasp planning, assembly planning, robot programming and teleprogramming, and sensor-based manipulation has yielded important results which are closely tied to the planning problem. A portion of the research in task planning takes the form of robot programming languages which allow the descriptions of robot tasks as a high-level language such as AL[83], AML[108], AUTOPASS[66], and MAPLE[28]. Robot programming languages can be classified as joint or actuator level languages, manipulator or end-effector level languages, object-level languages, and task-level languages. Task-level languages, in particular, often require planning capabilities. The progress made in robot programming and task planning systems in the last twenty years and the current research trends are discussed in [33].

2.3 Assembly Planning

One branch of robotics planning research is robot assembly planning. Most published assembly planning contributions focus on the modeling of the assembly process, i.e., describing the geometric configurations of the assembly which is constructed by single parts and the topological relations among the parts. Another direction in the research on assembly takes into account the factor of uncertainty, and therefore sensing operations, the alternative subsequence selection based on sensory information, sensory verification for uncertain states, and checking for error states as well as the generation of recovery sequences are combined into the planning process. We will review the literature in this area in the next section of this chapter.

Bourjault's[6] work on planning was based on an interactive algorithm using questions about the mating of two parts or multiple parts. The information Bourjault used is a list of parts and a network of nodes(parts) and lines(liaisons), where liaisons define the *connection* relationships among parts. Bourjault's graph model is different from Jentsch and Kaden's[55] *connection graph* model, where three types of touches between parts are expressed. All valid assembly sequences are generated algorithmically from a series of rules, which are derived from the answers to the questions about matings among parts. However, Bourjault's method requires $2l^2$(l is the number of liaisons in the network) questions plus a number of subsequent questions whose existence usually depends on answers to part of the former question set. De Fazio and Whitney[29] modified the approach that Bourjault has developed to generate assembly sequences based on the answers to the conditions of liaison establishments, i.e., the precedence relationship for assembly tasks. The question set in their approach contains a smaller number of questions than that in Bourjault's set, and these questions usually obtain more involved answers. Thus, these questions may cause direct relationships equivalent to those in Bourjault's, and valid assembly sequences can be generated algorithmically directly from these equivalent

relationships. Compared with Bourjault's method, the improved method requires $2l$ questions that are answered in a precedence-logical form. Chen[25] transformed the precedence relations for assembling parts into a pattern-matching problem and the problem of generation of all possible assembly sequences is formulated as a *state constrained traveling salesman problem.* The concept of a pattern matching algorithm is to match liaisons or parts with one of the parts so that current last assembly operation is obtained. In Chen's work, a mechanism of precedence knowledge acquisition is proposed which will reduce the time in obtaining such knowledge so that the time for generating sequences is also reduced. This approach results in only l questions to be answered.

All the above methods to generate possible assembly sequences are based on a user-computer interactive mode. The user is requested to answer a lot of questions about the precedence relationships among the liaisons of parts and the correctness and the completeness[49] of the sequences of operations generated are not guaranteed. Homem de Mello and Sanderson[45, 47] used the AND/OR graph to represent the decomposition process of an assembly based on the property of feasibility. Therefore, the problem of finding all possible assembly sequences is converted to find all feasible disassembly operations for an assembly or a subassembly until all subassemblies or parts belong to the initial state. A compact representation of all feasible assembly sequences is then obtained and its correctness and completeness is shown in [49]. A complete comparison of AND/OR graph representation for assembly sequences with other representations and mappings among these representations are discussed in [48]. Based on the work of AND/OR graph representation for assembly sequences, Homem de Mello and Sanderson proposed a heuristic search algorithm for the best sequence, which uses the criteria of maximizing the number of different sequences and minimizing the execution time through parallel execution of assembly tasks[50].

Some researchers concentrated on the constraints among parts inside an assembly to generate all feasible assembly sequences. Morris and Haynes[82] took into account the geometric constraints during the design of a robot programming system. Their understanding of geometric constraints is based on the *degrees of freedom* of the parts being assembled. When parts are assembled, their degrees of freedom are reduced. Besides the degrees of freedom constraints, Thomas and Torras[109] also proposed two other types of constraints, i.e., *shape-matching* constraints and *non-intersection* constraints. These constraints are used to infer assembly configurations for a practical and efficient planner. Vijaykumar and Arbib[115] proposed a strategy to decompose a sequence of operations and also satisfy the constraints arising from task and object characteristics. The assembly operations and object level descriptions are refined to be feature level descriptions using object symmetries, then spatial relationships, and at last commands to path planner and grasp planner. This work was actually not proposing an explicit method for planning but provided a connection between the high level planner, such as an assembly sequence planner, and the lower level planners such as a path planner and a grasp planner. As we will show later in Chapter 4, our decomposition for sequence plans is based on the decomposition of the representation of plans, which incorporates all possible sequences, rather than a specific task sequence. Therefore, during the decomposition process, any conflict or constraints of resources will emerge level by level, so that the feasible sequences searched from the final representation of decomposition are guaranteed.

Another example of using constraints for assembly is by Popplestone, Liu, and Weiss[92]. They used group theory to describe the symmetries of components in a computational form, so that a unified computational treatment of reasoning about how parts with multiple contacting features fit together is provided. Using this approach, when an object is assembled from several parts, the overall symmetry

can be obtained from the parts whose symmetries are already known. Moreover, a condition for features to mate is that they have the same symmetry group, so that the geometric feasibility may be tested by intersecting the constraints corresponding to each symmetry group of mating features.

2.4 Planning Under Uncertainty

Uncertainty may exist during planning generation and execution. Often uncertainty arises from run-time errors in sensing or control. Another cause of uncertainty may be one's lack of knowledge of modeling a system or environment. Different approaches have been proposed to solve the uncertainty problems in different robotics domains. Most efforts have been focused on compliant motion planning to deal with uncertainty[97, 105, 117]. Fine motion strategies are also synthesized in the presence of uncertainty[71]. Other related work is the characterization of manipulation tasks in terms of randomization[34] and entropy[98]. A preimage backchaining approach was used to address the problem of planning motion strategies in robot control and sensing in the presence of uncertainty[60].

When the robot planner generates a task sequence and forwards it to the robot program synthesizer to output an executable program, the robot manipulator will then implement this program to reach a desired final goal and also satisfy some constraints. However, some factors of *uncertainty* might show up during the execution of plans, and if no procedure is existing to verify, compute, and solve these uncertainties, the accumulations of the errors resulting from these uncertainties may cause a failure, and the sequence of tasks will not be finished. Brooks[7] described uncertainty for plan execution from three possible sources. The first source is the positional and repeatability uncertainty of the manipulator, which might be due to either stochastic or long-term drift effects. The latter case could be solved by calibration of the manipulator before each round of execution. The second source is

the objects to be manipulated, since each object or part has its toleranced dimensions during manufacturing. When a grasp planner generates a planning program, the numerical representations of parts play an important role, such as lengths, diameters, and angles. When the system consists of more parts, the problem of the possible errors for the representations may become more serious. Thirdly, if we assume the parts were not in the work environment originally, then the *initial uncertainties* should be taken into account for the introduction of these parts into the environment by human hands or a mechanical device.

In our research, we propose an additional type of uncertainty, the uncertainty of the accomplishment of subgoals. A checking and verification procedure which may use sensing operations for the quality of the fulfillment of subgoals is investigated and an algorithm is proposed in Chapter 7. Using this method, we reduce the propagation of errors for subgoals to a range of tolerance so that the correctness of the final goal is guaranteed.

In Kamel and Kaufmann's[56] work, two more factors are considered for uncertainty, one is inaccurate or error-prone sensing information, and another is the dynamic working environment, which is outside the control of the manipulator. For the latter case, we could also use sensory verification procedures as we use in the verification for the fulfillment of subgoals. For inaccurate sensory data, because we would map it into a fuzzy value during the firing of a fuzzy reasoning rule, the tolerance of this mapping would reduce the possible inaccuracy of sensory operators.

In Brook's work[7], an explicit block of *plan checker* was discussed and this block was separate from the block of *robot planner*. Plan checker was introduced to infer the effects of actions and the propagation of errors. Brook's major effort is to make a program that can automatically determine whether a plan generated by a robot planner is feasible, and when sensing information is obtained by the robot controller, the plan checker must have the ability to check and update plans

for computations which the robot controller will make. Comparing with Brook's approach, our model of error detection and recovery is combined with the task planner. Procedures for dealing with uncertainties and errors do not influence the representation structure for planning, and also the recovery procedure would be automatically called or an alternative sequence followed.

An example of a robot assembly planning system which handles uncertainties is Spar[51]. There is a three-level planning hierarchy for Spar, i.e., the operational level to handle high level operations, the geometric level to couple with geometric constraints, and the uncertainty-reduction level to deal with uncertainties and errors. In the lowest level, Spar uses its knowledge about the uncertainty in the world description to asses the possibility of run-time errors. To achieve this goal, Spar adds sensors to the plan to reduce uncertainties, and if uncertainty is too large, precompiled recovery plans would be added. Compared with Spar, the methodology for our research to decompose a representation of planning considers all possible operations in assembly, i.e., free motion, fine motion, grasp, and ungrasp.

To handle the uncertainties in execution of plans, we should find a strategy to monitor the uncertainties and call an error recovery procedure to remedy any exceptional cases whenever uncertainties are too large to go on the execution of the normal sequences. A number of approaches to error recovery for assembly workcells have been discussed in the literature. A repair sequence generation algorithm was proposed for planning disassembly and repair using the AND/OR graph[46, 99]. In the approach described in [69], the task execution control function of a workcell controller is decomposed into three sub-functions which are performed by three software modules: failure detection, failure diagnosis and failure recovery. Using Petri nets to model the controller in an automated manufacturing system, four basic error recovery planning mechanisms to augment the controller are discussed in [35]: input conditioning, alternate path, backward error recovery, and forward error

recovery. Some important properties of the augmented controller are guaranteed to be preserved[123]. Other recent work on error recovery[32] has focussed on the local physical and geometric constraints related to manipulation tasks.

2.5 Petri Nets with Fuzzy Data

Since C. A. Petri presented his original idea[90], where he formulated the basis for a theory of communication between asynchronous components of a concurrent system, a rich body of knowledge concerning both the theoretical and applied domains of Petri nets has been developed. Petri nets have been widely used in modeling and analyzing flexible manufacturing systems[2, 3, 78, 85, 116, 125], discrete event systems[26, 58], computer systems[44, 75, 80, 94], knowledge-based systems[8][52], robot assembly systems[121], as well as other kinds of engineering applications. This is an efficient abstract and formal information flow model. The Petri net is characterized by its flexibility and efficiency in modeling and analysis of complex discrete-event systems. For an extensive survey and overview on Petri nets, refer to [84, 88, 89]. In a recent article[2], an overview of the Petri net approach to the modeling, analysis, design, and control of automated manufacturing systems is presented.

The formal definition for Petri nets and related properties as well as the mathematical operations on Petri nets are defined in [2, 84, 88, 89].

There are some examples of using Petri nets to model robotic or assembly processes so that a sequence of operations is generated based on the Petri net model. In [120], a plan generating tool for robotic applications using Predicate/Transition nets[39, 40] is described. This tool is based on modeling STRIPS-like rules by Pr/T nets and then the T-invariant method of the Pr/T nets is utilized to generate robot plans. However, this approach still stays in domain-independent formalism and the applications are limited to blocks world. Zhang[121] described an approach

to representing simple specific assembly actions using a Petri net and presented an algorithm for automatic planning of an assembly robot based on the Petri net model of the assembly. There are several shortcomings in this approach and therefore this methodology is difficult to generalize. First, the process of constructing a Petri net model for the assembly is a bottleneck for the algorithm because no systematic method is proposed to derive topological relations among the parts. Second, the creation of places and transitions for the Petri net model is problem-specific so that it could not be generalized to other applications. Third, the algorithm only addresses the problem of finding one feasible sequence. The approach is not shown to be correct or complete.

Some types of Petri nets with fuzzy data have been proposed to handle problems in different applications. Looney[68] modified the usual Petri net to allow fuzzy rule-based reasoning by propositional logic. The resulting net is considered as a new type of neural network where the transitions serve as the *neurons*, and the places serve as the *conditions*, so that fuzzy reasoning for knowledge could be performed. Conditions may be conjuncted and disjuncted in a natural way to allow the firing of the neurons. Aside from the firing methods in usual Petri net models, in Looney's Petri net which implements logic implication, when a neuron is fired, the original token would remain at its precondition and copies would be sent out to all its post-conditions. One difference in this fuzzy net model from the usual fuzzy rule-based reasoning is in the representation of certainty or degree-of-belief values of the fuzzy rules. Chen, Ke, and Chang[24], however, eliminate this difference. In their work, a structured representation of production rules by fuzzy Petri nets, and a systematic procedure for supporting fuzzy reasoning, is proposed. Using this approach, each place represents a proposition. An algorithm is proposed to reason about the degree of truth of proposition d_j, if a degree of truth of another proposition d_i is given in

the net. If no solution is obtained, then these two propositions have no antecedent-consequence relationship. Similar work was done by Garg, Ahson, and Gupta[38] where a fuzzy Petri net was used to represent knowledge and an algorithm was proposed for checking the consistency of a fuzzy knowledge base via a set of reduction rules that preserve the properties of the FPN.

There are also other similar approaches in extending Petri nets in an imprecise or fuzzy sense. Valette, Cardoso, and Dubois[112] introduced uncertainty and imprecision within Petri net based models for application to monitoring of manufacturing systems. This approach is based on the association of a fuzzy value with the time delay for execution of a transition, which results in attaching a fuzzy date to the transition. The marking of Petri nets with *objects* and their interpretation were also introduced in their work[23]. Based on these assumptions, the relations between Petri nets and logic, the necessity to use Petri nets with objects to represent uncertainty, and the implementation of Petri nets as rules, were discussed in [113]. In Tsuji and Matsumoto's work[110], an extended Petri net was proposed to model the vague conditions, and the boundedness, liveness, and reachability for this model of fuzzy inference engines were analyzed. Another kind of approach modeled production systems where a numerical Petri net model was proposed[67] and the correctness, consistency, and completeness of the knowledge base were verified.

2.6 Conclusion of Literature Reviews

In this chapter, we review work on task planning, assembly planning, planning under uncertainty, and Petri nets with fuzzy data. To author's best knowledge, there is no previous work that used fuzzy logic to describe constraints and uncertainty in task planning and no work that applies ordinary or fuzzy Petri nets to task sequence planning, task decomposition, and evaluation and analysis of robotic systems. Prior work has emphasized that uncertainty is very important during both planning and

execution. Using uncertainty for planning, we add sensors to check and verify the uncertain state of the system, while using uncertainty in execution, we are required to call error recovery procedures to remedy exceptional cases should errors occur.

CHAPTER 3
AND/OR NET REPRESENTATION FOR ROBOTIC TASK SEQUENCE PLANNING

This chapter describes an approach to task sequence planning for a generalized robotic workcell. Given the descriptions of the objects in this system and all feasible geometric relationships among these objects, an AND/OR net represents the relationships of all feasible geometric states and associated feasibility criteria for net transitions. This AND/OR net is mapped into a Petri net which incorporates all feasible sequences of operations. The resulting Petri net is shown to be 1-bounded and have guaranteed properties of liveness, safeness, and reversibility. Sequences are found from the AND/OR net or the reachability tree of the Petri net. Feasibility criteria for net transitions may be used to generate an extended Petri net representation of lower level command sequences. The resulting Petri net representation may be used for on-line scheduling and control of the system.

3.1 Introduction

Most applications of robotic systems require the generation of a *task plan* which specifies the sequence of operations which must be carried out in order to achieve a stated goal. The generation of this task plan has been approached from several different perspectives, but in general some set of *operations*, or *actions*, and associated *pre-conditions* and *post-conditions* are defined. The *representation* of actions and conditions(states) defines the universe of the planning task. Exploring the feasible sequences of actions which satisfy pre- and post-conditions defines a *search* problem which must be solved to identify a feasible, and perhaps optimal, sequence.

Research on *domain-independent* planners[36, 37, 77, 96, 118] explored generic

representation and search strategies which yield feasible sequences. Much of this work has focused on state representation by propositional logic(well-formed formulae). Single-level and hierarchical representation and search have played an important role. Strictly sequential search(linear planning) versus non-sequential search(non-linear planning) have also been extensively described.

In robotic systems, a propositional logic state representation often does not capture the geometric relations required to fully describe the system state. Approaches to task planning and task-level languages for robotic systems have therefore concentrated on model-based descriptions of objects, configurations, and geometries of parts and mechanisms to describe systems states. Such a geometric representation results in more complex computational requirements for geometric reasoning about feasibility of operations, and therefore an increased difficulty to search for feasible sequences.

In previous work[47, 48, 49, 50], we have introduced the AND/OR graph representation of assembly plans. The AND/OR graph provides a compact representation of state relations for the specific domain of assembly. In assembly tasks, state relations are governed by a strict recursive decomposition relation described by the AND/OR tree. The resulting data structure is compact and efficient to search. In practice, the complete AND/OR graph may not be generated when the search process is bounded at execution-time.

In this chapter, we describe an extension of the AND/OR graph from a 'tree' to a 'net' structure. The resulting AND/OR net is a more general representation of geometric configurations which lends itself to a compact state description of a more general robotic system. The AND/OR net representation is described in detail in the next section.

The AND/OR net is introduced as a compact representation of feasible system states and state transitions, and incorporates all feasible operations sequences

for any given task. As a means to analyze and evaluate these possible operations sequences, we introduce the AND/OR net to Petri net mapping. Use of this alternate Petri net representation allows us to characterize the resulting system in terms of well-known properties of liveness, 1-boundedness, safeness, and reversibility of discrete-event systems. The resulting Petri net also provides a means to construct a task-level controller for execution of the final operations sequence.

3.2 AND/OR Net Representation

The representation of assembly plans in our previous work is based on an AND/OR graph[47, 48, 49, 50]. AND/OR graphs have *AND-arcs* connecting one initial node to k terminal nodes. The basic definition of an AND/OR graph is:

Definition 3.1 An *AND/OR graph* is a pair of sets *(V, H)* in which V is a finite set, and H is a subset of the Cartesian product $V \times (\prod(V) - \{\emptyset\})$, where $\prod(V)$ is the set of all subsets of V.

The elements of V are called *nodes*, and the elements of H are called AND-arcs. For an AND-arc (λ, Λ), the node λ is the *initial* node, and the nodes in Λ are the *terminal* nodes. The AND-arc (λ, Λ) is *incident from* λ and is *incident to* the nodes in Λ. The AND-arc (λ, Λ) is said to connect node λ to the nodes in Λ which implies that the condition of node λ can simultaneously cause the results of nodes in Λ.

In this section, we introduce the *AND/OR net* representation as a means to represent generic geometric relations and constraints among objects and devices in a robotic system. Given a complete geometric description of objects and object relations, the AND/OR graph described above is extended to represent more general relationships among objects as system substates. The AND/OR net is defined as follows:

Definition 3.2 *Pair-match set* Γ: Given two finite sets $\Sigma_1 = \{a_1, a_2, \ldots, a_m\}$, $\Sigma_2 = \{b_1, b_2, \ldots, b_n\}$,

$$\Gamma(\Sigma_1, \Sigma_2) = \bigcup_{i=1}^{m} \bigcup_{j=1}^{n} \{\{a_i, b_j\}\}$$

For example, if $\Sigma_1 = \{1,2\}$, $\Sigma_2 = \{3, \{4,5\}\}$, $\Gamma(\Sigma_1, \Sigma_2) = \{\{1,3\}, \{1, \{4,5\}\}, \{2,3\}, \{2, \{4,5\}\}\}$.

Definition 3.3 An *AND/OR net* is a three-tuple *(S, A, N)* where S is a finite set of states $\{s_1, s_2, \ldots, s_t\}$, $A \subseteq \Gamma(S, \prod(S) - (\{\emptyset\} \cup \cup_{i=1}^{t} \{\{s_i\}\}))$, $N \subseteq \Gamma(S, S)$, and $A \cap N = \emptyset$, where $\prod(S)$ is the set of all subsets of S.

The elements of S are called *nodes*, the elements of A are called *AND-arcs*, and the elements of N are called *IST-arcs*, where *IST* refers to "Internal State Transition". The AND-arc $\{\lambda, \psi\}$ is said to connect node λ to the nodes in ψ, $\psi \subseteq$ S. The IST-arc $\{\lambda_1, \lambda_2\}$ is said to connect node λ_1 to node λ_2. Both AND-arcs and IST-arcs are undirected. We introduce IST-arcs because some objects or subassemblies may change the shape, size, or have relative motions inside the combination of components. These changes cannot be described by a disassembly or assembly operation because the set of components are the same before and after the operation. Therefore, an AND/OR net incorporates three types of operations: assembly and disassembly operations, which are modeled by AND-arcs, and internal state change operations, which are modeled by IST-arcs. The representation of AND/OR nets can generalize the robotic planning problem from the assembly to moving objects, material handling, or other task-oriented problem.

AND/OR nets and AND/OR graphs both have AND-arcs. At a given time, an operation can be disjunctively chosen from all feasible operations. They can be used for representing parallelism. However, there are several differences between these two representations. First, in an AND/OR graph, there is normally a *start* node and some *concluding* or terminal nodes, while the nodes in an AND/OR net, are not

ordered in this manner. Each node could represent one substate of an initial state or one substate of a final state. Therefore, an AND/OR net is a more generalized representation of a robotic system. Second, the arcs in an AND/OR graph are directed while the arcs in an AND/OR net are bidirectional, which means both directions may be feasible when we show the information flow or transition flow. An AND/OR graph contains no cycles, while an AND/OR net has no assumption of acyclicity. Third, in an AND/OR graph, the initial state can be represented by a single *start* node, while, in contrast, the initial state of an AND/OR net may occupy several nodes in the net. Fourth, there is no arc in an AND/OR graph which corresponds to the IST-arc in an AND/OR net. Because each arc in an AND/OR net can be considered bidirectional, the AND/OR graph representation can be thought of as a special case of the AND/OR net.

3.2.1 AND/OR Net Algorithm

Consider a system which contains M geometric substates, including, objects of one component, i.e., S_1^1, S_2^1, ..., $S_{p_1}^1$, subassemblies of two components, i.e., S_1^2, S_2^2, ..., $S_{p_2}^2$, ..., and assemblies of n components, S_1^n, S_2^n, ..., $S_{p_n}^n$, where $p_1 + p_2 + \ldots + p_n = M$. The algorithm for obtaining the AND/OR net from the geometric state representation is shown below:

Algorithm 3.1 *Obtaining the AND/OR net from a system geometric state representation.*

Input: system geometric state representation.
Output: AND/OR net $N_A = (S, A, N)$.

1. Initialize $S = \emptyset$. For each geometric state S_j^i in the system geometric state representation, $S = S \bigcup \{S_j^i\}$;

2. For $i = n$ to 2

For $j = 1$ to p_i

Consider S_j^i,

2.1 if there is an $S_u^i (u \neq j)$ which contains the same components as S_j^i, and there is no IST between S_j^i and S_u^i, add an arc between S_j^i and $S_u^i (IST)$ in N;

2.2 Find all feasible $S_{b_1}^{k_1}$ and $S_{b_2}^{k_2} (k_1 < i, k_2 < i)$, where $S_{b_1}^{k_1}$ and $S_{b_2}^{k_2}$ have no common components, and $S_{b_1}^{k_1}$ together with $S_{b_2}^{k_2}$ contain the same components as those in S_j^i, add an AND-arc between S_j^i and $S_{b_1}^{k_1}$, $S_{b_2}^{k_2}$ in A.

Using this algorithm, we observe that each subassembly of order n may be further decomposed in $2^{n-1} - 1$ ways. Only those decompositions which are geometrically feasible, i.e., we can find the corresponding set of disassemblies in the geometric state representation, and show that a collision-free path is available, are included in the AND/OR net. In addition, objects with internal state changes are indicated by dark links in the AND/OR net.

Therefore, the nodes in the AND/OR net correspond to all objects, which may be subassemblies and assemblies, appearing in the geometric states representation. The AND-arcs represent the feasible decompositions from subassemblies(assemblies) to a corresponding set of subassemblies. The IST-arcs represent the feasible internal state transitions from a subassembly(assembly) to another subassembly(assembly). These two subassemblies(assemblies) are listed in the same column in the geometric states representation and contain the same number of original components.

An example of an AND/OR net definition is shown in Figures 3.1 to 3.3. Figure 3.1 shows a robot which transfers an object on the floor to the surface of a table. The initial state and final state are presented in Figure 3.1. All feasible geometric relationships among these objects are shown in the geometric state representation in Figure 3.2. For simplicity in this example, the floor is not considered a defined object.

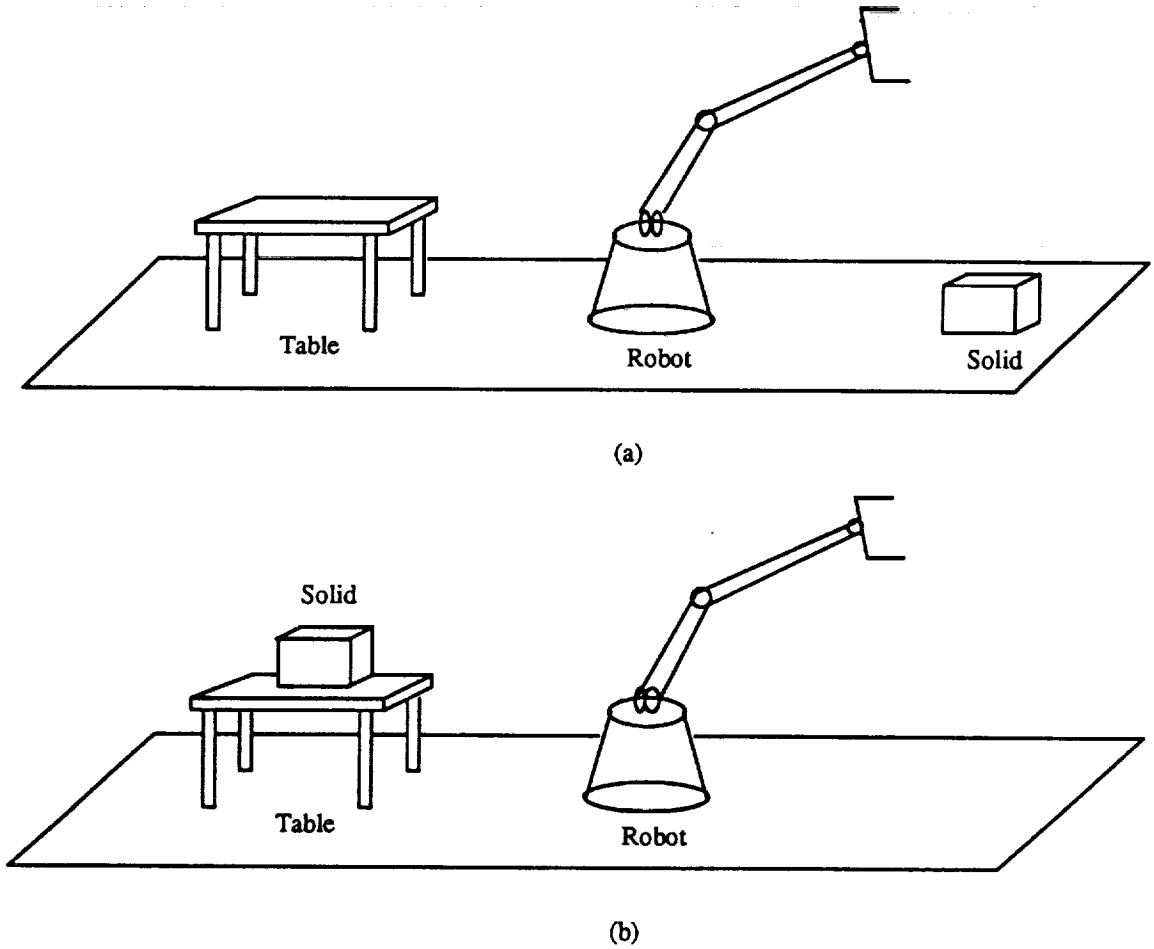


Figure 3.1: Example of a moving task for a robot. (a) Initial state. (b) Final state.

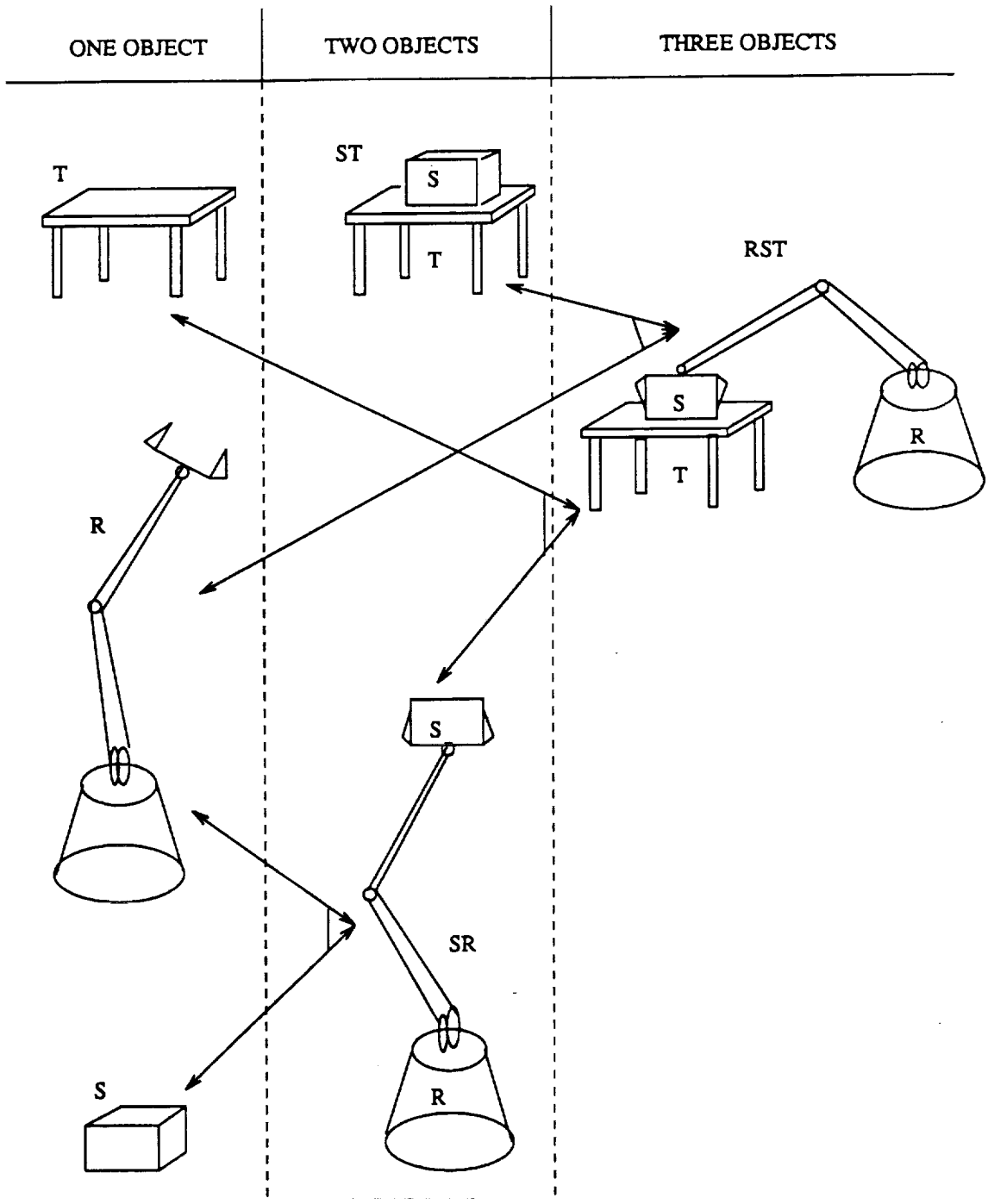


Figure 3.2: System geometric state representation.

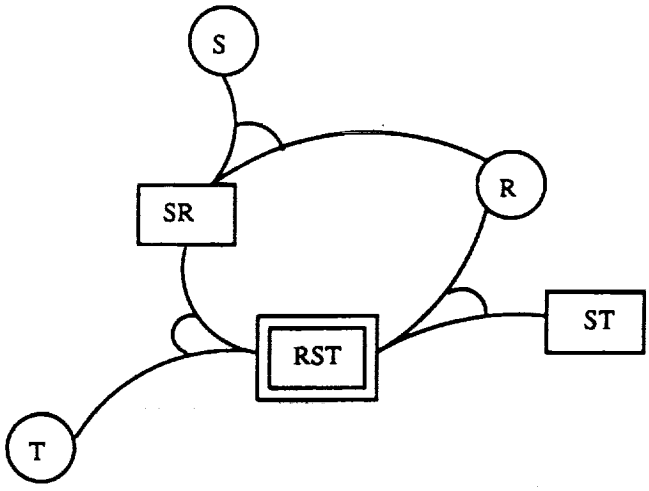


Figure 3.3: The AND/OR net representation for the example.

The maximum number of steps of generating the AND/OR net is $C_1^n + C_2^n + \ldots + C_n^n = 2^n - 1$, where n is the number of components in the system and C_j^n indicates the combinations of n. Within each step, we may obtain more than one configuration of single components or contacting objects. The resulting AND/OR net state representation is shown in Figure 3.3 and is based on the feasible decompositions of subassemblies of order n to subassemblies of order no more than $n-1$. In the next sections, we will describe the mapping of this AND/OR net to a Petri net.

Methods for extracting all possible sequences from the AND/OR graph or AND/OR tree representation of the system will in general not work for the AND/OR net. First, cycles may appear in an AND/OR net, and the methods developed in AND/OR graphs for automatically searching task sequences are no longer valid. However, the AND/OR net often possesses properties which simplify the representation and search process. Under many common assumptions, an AND/OR net possesses the special characteristic of reversibility. In addition, we may consider AND/OR graphs or AND/OR trees as special cases of AND/OR nets. An algorithm for searching in the AND/OR net is described in Section 3.4.

3.3 AND/OR Net to Petri Net Mapping

Some definitions and notations of Petri nets are introduced below:

Definition 3.4 *A Petri net structure, N,* is a four-tuple, $N = (P, T, \alpha, \beta)$. $P = \{p_1, p_2, \ldots, p_n\}$ is a finite set of *places,* $n \geq 0$; $T = \{t_1, t_2, \ldots, t_m\}$ is a finite set of *transitions,* $m \geq 0$; $P \bigcap T = \emptyset$. $\alpha \subseteq \{P \times T\}$ and $\beta \subseteq \{T \times P\}$ are sets of directed arcs.

Definition 3.5 *Marking μ:* Marking μ of Petri net N is a mapping from set P to set $\Lambda = \{0, 1, 2, \ldots, L\}$ which is a finite set, i.e.,

$$\mu : P \rightarrow \Lambda,$$

where μ sets tokens to every place in N. $\mu_i = \mu(p_i) \in \Lambda$ indicates the number of tokens in place p_i. μ can be in the form:

$$\mu = (\mu_1, \mu_2, \ldots, \mu_n)^T; \qquad \mu_i = \mu(p_i), \quad p_i \in P.$$

Definition 3.6 *Marked Petri net M:* A Petri net structure N containing a marking μ is *a marked Petri net* which is the following five-tuple,

$$M = (P, T, \alpha, \beta, \mu).$$

Sometimes, for the sake of simplicity, we refer to a *marked Petri net* as a *Petri net,* as shown later in this chapter.

Definition 3.7 *Petri net graph:* The Petri net graph consists of directed arcs and two kinds of nodes. In the graph, circle nodes and bar nodes represent places and transitions respectively. The directed arc, which links the circle node and the bar node, indicates the relation between place and transition. Marking μ is indicated by solid dots in circle nodes.

One important property of a Petri net is the representation of serial and concurrent events and resource constraints. For this research, we use the Generalized Stochastic Petri Nets(GSPN) software[73, 74] to represent the system and carry out simulations as well as verify the task sequences.

3.3.1 AND/OR Net to Petri Net Mapping Algorithm

For an undirected arc $\{\lambda_1, \lambda_2\} \in N$ in an AND/OR net, the *mapping* to elements in a Petri net is defined as a function $\mathcal{F}_1$, where

$$\mathcal{F}_1(\{\lambda_1, \lambda_2\}) = (\lambda_1, t_1) \bigcup (t_1, \lambda_2) \bigcup (\lambda_2, t_2) \bigcup (t_2, \lambda_1).$$

For an AND-arc $\{\lambda, \psi\} \in A$ in an AND/OR net, the *mapping* to elements in a Petri net is defined as a function $\mathcal{F}_2$, where

$$\mathcal{F}_2(\{\lambda, \psi\}) = (\lambda, t_1) \bigcup \bigcup_{i=1}^{k} (t_1, \lambda_i) \bigcup \bigcup_{i=1}^{k} (\lambda_i, t_2) \bigcup (t_2, \lambda), \qquad \lambda_i \in \psi.$$

The algorithm for converting an AND/OR net to the corresponding Petri net is shown as follows:

Algorithm 3.2 *Mapping from an AND/OR net to a Petri net.*

Input: AND/OR net $N_A = (S, A, N)$.

Output: Petri net $N_P = (P, T, \alpha, \beta)$.

1. *initialize* $P = T = \alpha = \beta = \emptyset$, $n_P = n_T = 0$;
2. *for each set* $n_i \in N, n_i = \{n_{i1}, n_{i2}\}$

   begin

   add 2 transitions t_{n_T+1}, t_{n_T+2},

   $T = T \bigcup \{t_{n_T+1}, t_{n_T+2}\}$;

   $n_T = n_T + 2$;

   check whether n_{i1}, n_{i2} *is in* P,

if n_{i1} *not in* P, $P = P \bigcup \{n_{i1}\}$, $n_P = n_P + 1$;

if n_{i2} *not in* P, $P = P \bigcup \{n_{i2}\}$, $n_P = n_P + 1$;

$\alpha = \alpha \bigcup \{(n_{i1}, t_{n_T+1}), (n_{i2}, t_{n_T+2})\}$;

$\beta = \beta \bigcup \{(t_{n_T+1}, n_{i2}), (t_{n_T+2}, n_{i1})\}$;

end

3. *for each set* $a_i \in A, a_i = \{a_\lambda, \psi_a\}$

begin

add 2 transitions t_{n_T+1}, t_{n_T+2},

$T = T \bigcup \{t_{n_T+1}, t_{n_T+2}\}$;

$n_T = n_T + 2$;

for every $e_j \in \{a_\lambda\} \bigcup \psi_a$ *and* $\{e_j\} \bigcap P = \emptyset$,

$P = P \bigcup \{e_j\}$, $n_P = n_P + 1$;

$\alpha = \alpha \bigcup \bigcup_j \{(a_\lambda, t_{n_T+1}), (e_j, t_{n_T+2})\}$, for all e_j;

$\beta = \beta \bigcup \bigcup_j \{(t_{n_T+1}, e_j), (t_{n_T+2}, a_\lambda)\}$, for all e_j;

end

A Petri net representation for the example in Figure 3.1 is shown in Figure 3.4. The initial marking of one token in places S, R, and T represents the initial state, i.e., there are one robot, one table, and one object available and they are geometrically independent. The important properties of the resulting Petri net may be shown as follows:

Theorem 3.1 *The Petri net mapped from an AND/OR net is safe.*

Proof: Because the Petri net is mapped from an AND/OR net, we clarify the meanings of *AND-arc* and *IST-arc*. The IST-arc in the AND/OR net corresponds to the internal geometric state change inside an object. All possible assemblies of n objects, and all possible subassemblies of m objects are special cases of objects. The AND-arc in the AND/OR net corresponds to combining the geometric states

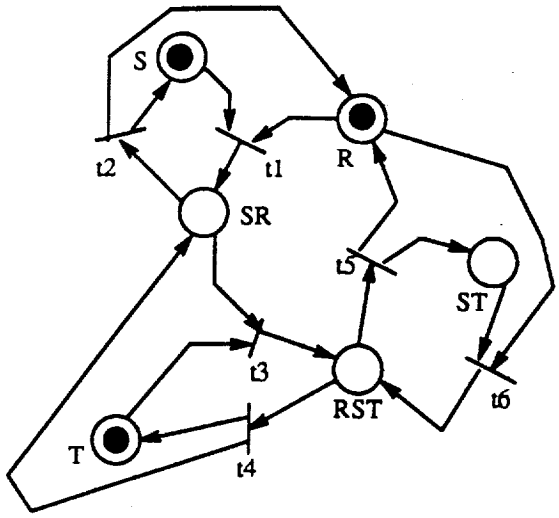


Figure 3.4: The Petri net representation for the example.

of two or more objects.

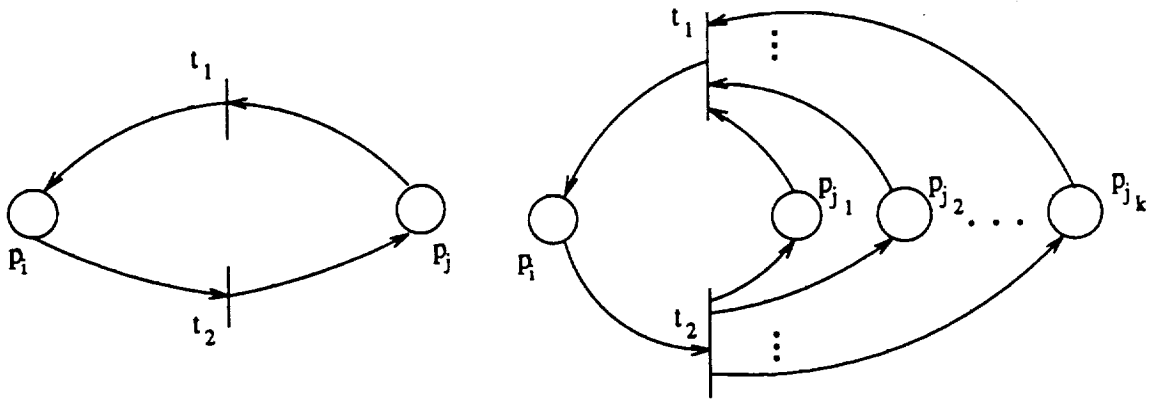
Suppose $N_P = (P, T, \alpha, \beta)$ with initial marking μ. Choose any place $p_i \in P$; $\mu' \in R(N_P, \mu)$ represents a marking which can be reached from μ. If we could verify that $\mu'(p_i) \leq 1$ for all possible μ' and i, then the proof is completed.

p_i might connect with neighboring places in two ways(Figure 3.5).

Case 1: Corresponding to the internal state change of geometric substates of a component or a related set of components in the system(Figure 3.5(a)).

Suppose $\mu'(p_i) \geq 2$. For the sake of simplicity, we assume $\mu'(p_i) = 2$. There should exist a $\mu'' \in R(N_P, \mu)$ and $\mu' \in R(N_P, \mu'')$ such that $\mu'(p_i) = 2$ and $\mu''(p_i) = 1$, and μ' is immediately reached from μ''. Therefore, at the time of marking μ'', $\mu''(p_i) = \mu''(p_j) = 1$, p_j and p_i are neighboring places in the Petri net. Because the Petri net is mapped from the AND/OR net and this case concerns the internal geometric state change, we conclude that at the time of marking μ'', the two feasible internal states of a single component or a related set of components could appear simultaneously. The contradiction is thus obtained.

Case 2: Corresponding to the assembly and disassembly relationship among one subassembly or assembly with a set of other subassemblies or single components



(a) One neighbor (b) Two or more neighbors

Figure 3.5: The connectedness with p_i and its neighboring places.

in the system.

(i) p_i is as shown in Figure 3.5(b).

Also suppose $\mu'(p_i) = 2$. There should exist a μ'' such that $\mu'(p_i) = 2$ and $\mu''(p_i) = 1$. Therefore, at the time of μ'', $\mu''(p_i) = \mu''(p_{j_1}) = \mu''(p_{j_2}) = \cdots = \mu''(p_{j_k}) = 1$, $\mu''(p_{j_1}), \cdots, \mu''(p_{j_u}), \cdots, \mu''(p_{j_k})$ are combining neighboring geometric states in the AND/OR net, $1 \leq u \leq k$. We conclude that at the same time of μ'', the two possible combining geometric states, which include exactly the same objects, could appear simultaneously. A contradiction is thus obtained again.

(ii) p_i is in the place of p_{j_u} as shown in Figure 3.5(b).

Follow the same procedure as in case 1. We could conclude that at the time of μ'', two possible combining geometric states, which contain at least one common object, would appear simultaneously. A contradiction is obtained.

Therefore, the safeness of the Petri net mapped from an AND/OR net is assured.

Q.E.D. □

Corollary 3.1 *The Petri net mapped from an AND/OR net is 1-bounded.*

This corollary is directly derived from Theorem 3.1 because the number of

tokens in any place cannot exceed 1.

Theorem 3.2 *The Petri net mapped from an AND/OR net is live.*

Proof: Based on the properties of generating the AND/OR net from the system geometric state representation, we know that each IST-arc and AND-arc is feasible, and each transition in the Petri net is feasible if we have tokens in the corresponding incident places. If we can verify that no matter what marking μ' has been reached from the initial marking μ, it is possible to ultimately fire any transition of the net through some further firing sequence, then the proof is obtained.

Because at any time, the system contains all components and each component is in a geometric substate. To get tokens for all incident places for *any* transition simultaneously, using the property that no common components are existing in these places at the same time, we can first follow a sequence to get all geometric substates for single components. Then, we follow two or more distinct sequences to obtain the tokens in the incident places of the selected transition, respectively. Therefore, the liveness of the Petri net mapped from an AND/OR net is proven.

Q.E.D. □

Theorem 3.3 *The Petri net mapped from an AND/OR net is reversible.*

Proof: As shown in the proof of Theorem 3.2, the Petri net is a set of loops according to the *mapping* definitions. If we define each pair of transitions in the net as t_i and t'_i, when μ' is reached from μ following $t_1 t_2 \ldots t_p$, we can resume the marking of μ from μ' following $t'_p t'_{p-1} \ldots t'_1$. Therefore, the initial marking is reachable from all reachable markings. The Petri net is thus reversible.

Q.E.D. □

The live Petri net guarantees a deadlock-free system. The boundedness property ensures that the capacity is not exceeded. And the reversibility implies that the

system can re-initialize itself, and is important for the automatic recovery from errors and failures. Therefore, if a robotic assembly or handling system is represented as an AND/OR net, it is not only convenient for the system to generate task plans, but also the controller will supervise and coordinate the system more efficiently.

This mapped Petri net does not satisfy the property of conservation because of the geometric characteristics of the system.

3.3.2 Directed AND/OR Net and the Properties of the Mapped Petri Net

In some practical cases, each operation represented in an AND/OR net is not reversible. For example, the product generated by some physical assembly operation cannot be disassembled following the same strategy as assembly. We define a directed AND/OR net which incorporates both undirected arcs and directed arcs.

Definition 3.8 A *directed AND/OR net* is a five-tuple *(S, A, A', N, N')* where S is a finite set of states $\{s_1, s_2, \ldots, s_t\}$, $A \subseteq \Gamma(S, \prod(S) - (\{\emptyset\} \cup \cup_{i=1}^{t}\{\{S_i\}}))$,

$$A' \subseteq S \times (\prod(S) - \{\emptyset\} - \bigcup_{i=1}^{t}\{\{s_i\}\}) \bigcup (\prod(S) - \{\emptyset\} - \bigcup_{i=1}^{t}\{\{s_i\}\}) \times S,$$

$N \subseteq \Gamma(S, S)$, $N' \subseteq S \times S$, and $A \cap A' = \emptyset$, $A \cap N = \emptyset$, $N \cap N' = \emptyset$, $A' \cap N' = \emptyset$, $A \cap N' = \emptyset$, and $A' \cap N = \emptyset$, where $\prod(S)$ is the set of all subsets of S.

A directed AND/OR net may be mapped to a Petri net, by adding one transition instead of two transitions for directed arcs, with the same direction of that in the directed AND/OR net, for all directed arcs in the net. Therefore, in the mapping algorithm, we add a checking command to see whether the current arc in the AND/OR net is directed or undirected, before adding the corresponding transitions and arcs in the Petri net.

Some properties are not guaranteed in the Petri net mapped from a directed AND/OR net. Liveness and reversibility are not guaranteed because of the existence of some directed arcs. To be more accurate, based on the definition of different levels of liveness[27, 61], the resulting Petri net is *strictly* $L1 - Live$, because each transition in the net can be fired at least once in some firing sequence from the initial marking μ, and some transitions cannot be fired any finite number of times in any firing sequence. The properties of safeness and thus 1-boundedness of the directed Petri net are guaranteed.

Lemma 3.1 *If a Petri net is safe, when one or more transitions are deleted, the remaining net still retains the property of safeness.*

Proof: We assume the original net is N and the modified net is N'. Suppose we can find a sequence of transition $t_1 t_2 \ldots t_p$ in N' such that one place receives more than one token. Therefore, following the same sequence in N, we can also obtain the same result. This leads to the contradiction with the property of safeness of N and thus no such sequence exists. We conclude that N' retains the property of safeness.

Q.E.D. □

Lemma 3.2 *If a Petri net is bounded, when one or more transitions are deleted, the remaining net still retains the property of boundedness.*

Proof: The proof strategy is the same as in Lemma 3.1. If we suppose there exists a sequence to destroy the property of boundedness in N', we will reach a contradiction to the known assumption. Therefore, the property of boundedness is inherited in N' which is generated through deleting some transitions in N.

Q.E.D. □

Theorem 3.4 *The Petri net mapped from a directed AND/OR net is safe.*

Proof: Using Lemma 3.1, we consider the Petri net mapped from a directed AND/OR net as generated through deleting some transitions in a *complete* Petri net which is mapped from an *ordinary* AND/OR net. Because this complete Petri net is safe, the directed Petri net is also safe.

Q.E.D. □

Corollary 3.2 *The Petri net mapped from a directed AND/OR net is 1-bounded.*

This corollary is directly derived from Theorem 3.4 because the number of tokens in any place cannot exceed 1.

3.4 Data Structure for Searching Sequences in the AND/OR Net

To search feasible sequences from the AND/OR net representation of a robotic system, an efficient data structure is required. Two possible requirements may be proposed for practical implementation situations: one is the problem of searching *all possible operations sequences* for the given initial state and final state, and another is the problem of searching *the optimal sequence* under certain evaluation criteria such as cost, number of steps, or flexibility. The complexity of the first problem is much greater than that of the second one.

The search in an AND/OR graph is a recursive procedure which is guaranteed to terminate under the assumption of acyclicity. To search all possible *solution graphs* from the start node to a set of terminal nodes in an AND/OR graph, a breadth-first search algorithm could be defined. To find the optimal solution graph with minimum cost, a heuristic search procedure, AO^* algorithm[87], was used to speed up the search. The AO^* algorithm consists of two major operations, a top-down graph expanding procedure and a bottom-up cost revision procedure. The search algorithm for AND/OR graphs could not be used to search AND/OR nets because of their different topologies and properties.

3.4.1 Searching All Possible Sequences

It may not be possible to represent the state of a robotic system by a single node in the AND/OR net. For example, the initial state S_I may consist of a set of k objects, $\{O_1, O_2, \ldots, O_k\}$, where O_i represents either a component, a subassembly, or an assembly. To find all possible operations sequences, which consist of AND-arcs and IST-arcs, from the initial object set to the final object set, we first transform each distributed node set in an AND/OR net to a distinct node in a corresponding *state graph*. Our algorithm is mainly divided into two steps:

1. Create a directed state graph, where each node in the graph represents a feasible system state, and each directed arc points from one node to another and is marked with a label of a feasible operation.

2. Generate all possible paths from the node which corresponds to the initial state to the node which represents the final state.

Because the complete algorithm is quite complicated, we only informally discuss some important ideas related to the data structure inside the algorithm.

To create the state graph, we have a linear sequential structure $\mathcal{L}$ which may be implemented by an array. Each node in $\mathcal{L}$ represents a system state. Two pointers, Pt_1, which is to indicate the intersecting position of the states which have been processed and not processed, and Pt_2, which is to mark the index of the last inserted state, are used. Initially, $\mathcal{L}$ consists of only S_I. If $Pt_1 \neq Pt_2$, and the current state is not the final state S_f, the set of objects representing this state will be used to search from the feasible-operation-base to find all enabled operations. Correspondingly, each new state reached by an enabled operation will be compared with every existing state in $\mathcal{L}$ to decide whether it should be added to $\mathcal{L}$. In either case, a directed arc marked with the label of this operation will be added to $\mathcal{L}$. For each node in $\mathcal{L}$, the number of nodes it points at will also be recorded. Eventually, because of the finite number of components in the system, $\mathcal{L}$ will stop developing

and all feasible states are incorporated in this graph. The resulting graph is a finite automaton with one final state.

The complexity of the generation of the state graph depends on the practical implementation of a robotic system because the comparison of each existing state with a new state is quite expensive. Also, the same state may be created many times and at each time the comparison should be performed. Searching in the feasible-operation-base also adds to the complexity. A binary search strategy or a heap structure[1] cannot be used here to raise the efficiency of search because otherwise the arcs in the state graph would be frequently modified and thus costly. The complexity is $\Omega(n^2 + mn)$, where n is the number of feasible states in the final graph and m is the number of feasible operations.

The next step is to find all sequences from $\mathcal{L}$. For this task, we use a data structure Δ to store the intermediate states which are being processed, and these states point to the corresponding partial sequences which are being developed. This data structure could be either a queue or a stack, which corresponds to the *breadth-first* and the *depth-first* search, respectively. The states in Δ could also be ordered, which corresponds to a heuristic search to find the smallest cost path. We will discuss it using another more efficient strategy in the next subsection.

Suppose the enabled operations in the state graph for the initial state S_I is t_{I_1}, t_{I_1}, ..., and t_{I_p}. In the first step, we push the corresponding child states of S_I, S_{I_1}, S_{I_2}, ..., S_{I_p}, into Δ, and make them point to the partial sequences, t_{I_1}, t_{I_1}, ..., and t_{I_p}. The following iterations will process and delete a state in one end of Δ. Its child states are compared with the final state and the results control whether to add the corresponding enabled operations to the partial sequences. The procedure for developing a partial sequence continues until (1) a *duplicate* transition is found for a partial sequence, in this case, the partial sequence will be discarded, or (2) the final state is reached, in this case, the complete sequence will be output or stored. This

algorithm is guaranteed to stop(Δ becomes empty) because the feasible operations and the feasible states are finite. All complete sequences are stored in a set and could be evaluated or selected after the searching process stops. The complexity for this type of search and the number of all possible sequences depends on (1) the size and the configuration of the state graph, (2) the cost of comparing a currently enabled operation with each operation existing in the corresponding partial sequence. The storage for the scripts of all possible sequences is also not predictable and therefore may be large. During the procedure of generating all possible sequences, if we add some constraints such as the maximum number of operations in a sequence, the maximum number of sequences we want to generate, the maximum cost for each sequence, and so on, a reduced set of possible sequences is obtained and search time and space required may be reduced.

3.4.2 Searching the Shortest Sequence

In many cases, we wish to search the shortest sequence directly from the AND/OR net representation rather than to choose from a set of all possible sequences. When there is a weight or cost function for each feasible operation defined *a priori*, the shortest operations sequence is defined as the sequence from the initial state to the final state with the lowest cumulative cost. When there are no costs defined for the operations, the shortest sequence is considered as the sequence with the fewest number of steps.

We use a methodology similar to Dijkstra[30] for searching the shortest path from a weighted graph which is based on a greedy strategy. Given two arbitrary nodes, v and w, in the graph, Dijkstra's algorithm is to find the shortest path from v to w in $\Theta(n^2)$ time, where n is the number of vertices in the graph. This algorithm is based on a strategy of generating a search tree which always chooses an edge, such that the cost from the node on one end of the edge to the starting node is the

smallest. Eventually the search tree will reach w. If we directly use this algorithm to search the shortest sequence, the complexity will be $\Omega(n^2 + mn) + \Theta(n^2) = \Omega(n^2 + mn)$, where n is the number of all feasible states and m is the number of all feasible operations.

In our algorithm, we look for a search tree at the same time as the state graph is created. Suppose the partial graph we are developing is G, and the partial search tree is T. Initially, $G = T = \emptyset$. When the search starts, $G = T = \{S_I\}$. The algorithm for searching the shortest path from the AND/OR net is informally shown as follows. The notations are: (1) S_{curr} represents the current state which is being processed and S_{curr_i} is one of the child state of S_{curr}, (2) $c(S_j)$ is the minimum cost of path from S_I to S_j. (3) $arc(S_{j_1}, S_{j_2})$ represents the directed arc connecting from S_{j_1} to S_{j_2}, (4) $w(arc(.))$ is the cost of the corresponding arc.

$S_{curr} = S_I$; $cost(S_{curr}) = 0$;

while $S_{curr} \neq S_F$(final state) *do*

(1) *for* each child state, S_{curr_i}, of S_{curr}

if S_{curr_i} is in G *and* $c(S_{curr_i}) > c(S_{curr}) + w(arc(S_{curr}, S_{curr_i}))$

delete the arc connecting S_{curr_i} and a node in G *and*

add $arc(S_{curr}, S_{curr_i})$ to G;

if S_{curr_i} is not in G

add S_{curr_i} and $arc(S_{curr}, S_{curr_i})$ in G;

(2)find a $g' \in G - T$ which has a cost of $\min_{ij}(cost(t_i) + w(arc(t_i, g_j))$

from S_I to it, $t_i \in T$, $g_j \in G - T$;

Put it into T and mark it as S_{curr}.

This algorithm will stop if the final state is met and the path could be traced from S_F back to S_I. And the shortest cost from the initial state to all other states in T will be found in order of increasing cost. The proof of the correctness of our algorithm is quite straightforward, and we omit the proof here. The complexity is

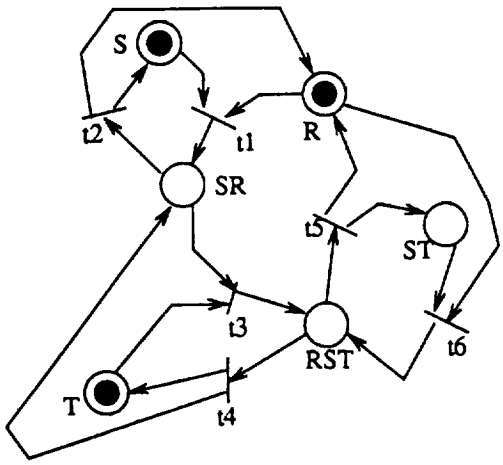
$\Theta(n'^2 + n'm)$ where n' is the number of feasible states which will be developed for G, $n' \leq n$, and m is the number of feasible sequences. This complexity is clearly less than that discussed above.

Using these algorithms for searching, we can generate a state graph for the AND/OR net example shown in Figure 3.3, and obtain the task sequences required. In this case, only one possible sequence is available. This sequence could be mapped to a sequence of transitions in the mapped Petri net. The firings of this sequence of transitions and the corresponding sequence of markings are shown in Figure 3.6. Note that the sequence of places in markings are: R, S, T, SR, ST, and RST, respectively.

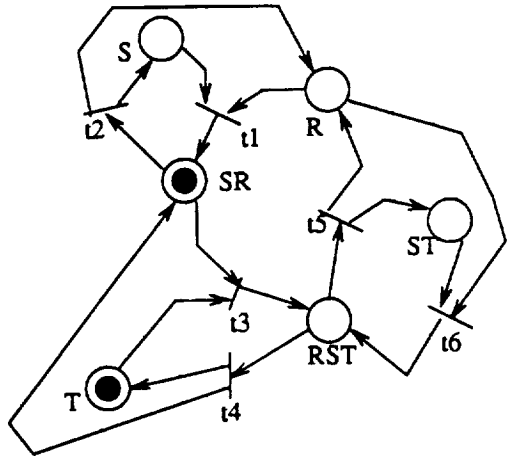
3.4.3 Searching Sequences in Resulting Petri Nets

We could also search sequences from the Petri net which is mapped from the AND/OR net. In this case, the task planning problem maps to a reachability problem in the Petri net, which is a basic Petri net analysis problem[89]. From the task sequence planning point of view, we are not only interested in whether a final state, μ_f, can be reached from the initial state, μ_i, we also require the sequence used to reach the final state. In such a reachability tree generated from a Petri net, the number of leaves in the tree is the number of all possible task sequences. The depth of the tree is the length of the sequence. The length of the shortest path from the root to a leaf is the number of operations in the optimal sequence in the sense of the number of steps. To represent a task sequence from the reachability tree, we can either show a sequence of transitions, or a sequence of system states. The Petri net can be used to simulate and verify the sequences selected.

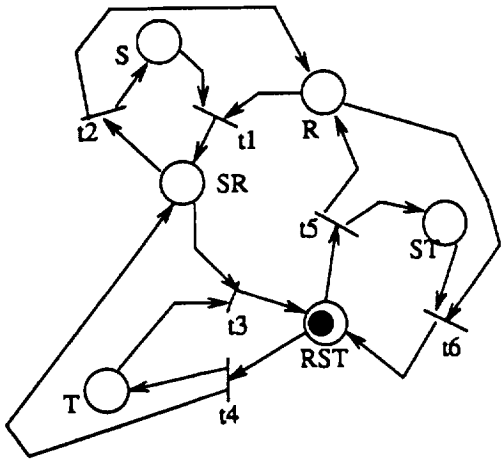
Compared with the search algorithm for all possible operations sequences discussed for AND/OR nets, the time for creating a state graph and a reachability tree is the same, while the representation size of the reachability tree is much greater



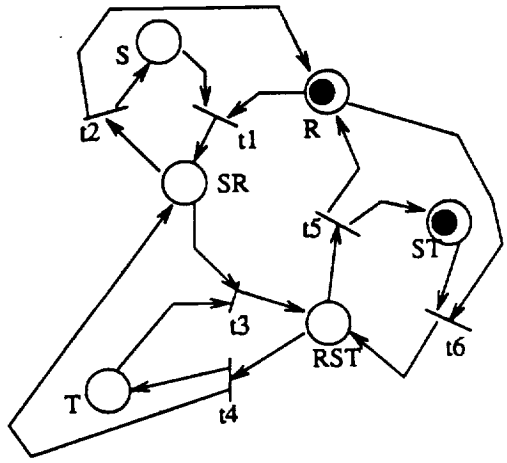
(a) Transition fireable: t1
Marking: 111000
Operation: None



(b) Transition fireable: t2, t3
Marking: 001100
Operation: Robot grasps solid



(c) Transition fireable: t4, t5
Marking: 000001
Operation: Robot reaches table



(d) Transition fireable: t6
Marking: 100010
Operation: Robot leaves table

Figure 3.6: The sequence of markings and corresponding operations.

because of many duplicate nodes. However, the feasible operations sequences can be directly found from the reachability tree. Similar to the discussions on AND/OR nets, it is also not economical to find an optimal sequence using the method of the reachability tree.

3.5 Example of Task Sequence planning Using AND/OR nets

Another example of a task sequence plan is provided by the problem of a robot, two tables, and a book shown in Figure 3.7. An initial state and a desired final state are mapped to their geometric descriptions. All feasible geometric states for one object, two objects and three objects are shown in Figure 3.8. For this problem, we assume the maximum size of the robot gripper is not large enough for the robot to grasp the book when the book is fully lying on the table. It is thus necessary to first move the book to the edge of the table and try to pick up the book from one side. Two cases for the connectedness of the book and the table are considered. For the connectedness of three objects, the cases are more complex because of the relative geometric relations between the book and table. When the book is on the edge of the table and the robot is touching one side of the book, we ignore the place of the robot relative to the table, i.e., $(T_1BR)_4$ and $(T_2BR)_4$ may include two kinds of geometric relations.

From the system geometric states and Algorithm 3.1, we obtain the AND/OR net representation for the task(Figure 3.9). For simplicity of the figure, the net is shown as separate subnets, but because of the common nodes in each subnet, it is really a connected net. We map this AND/OR net to the Petri net(Figure 3.10) following Algorithm 3.2. The description for each operation could be deduced from the Petri net and the corresponding system geometric states descriptions. By searching the AND/OR net directly or the reachability tree of this Petri net, we obtain all possible task sequences. The optimal(shortest) sequence in this case is

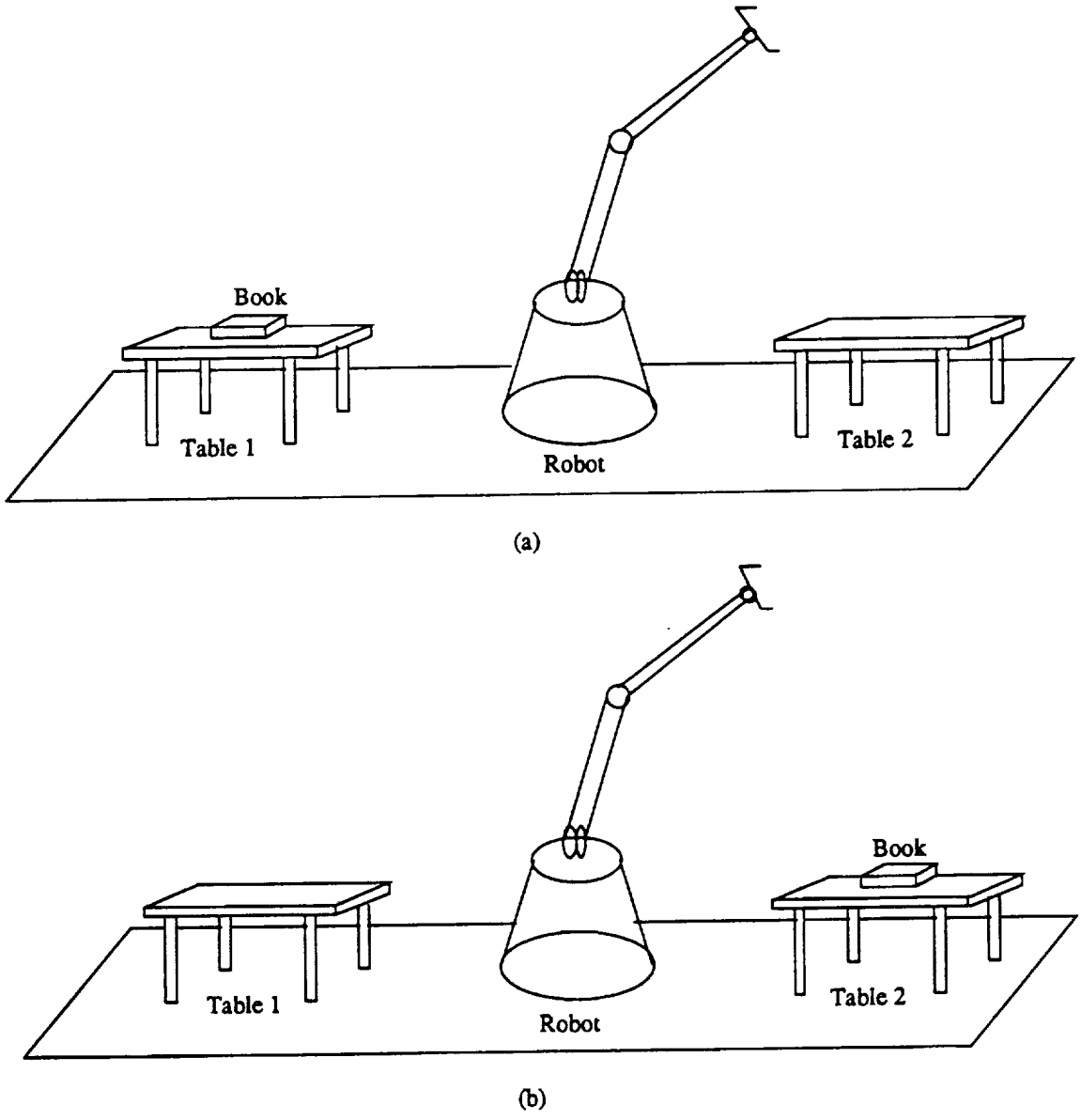


Figure 3.7: A robot moves a book from table 1 to table 2. (a) Initial state. (b) Final state.

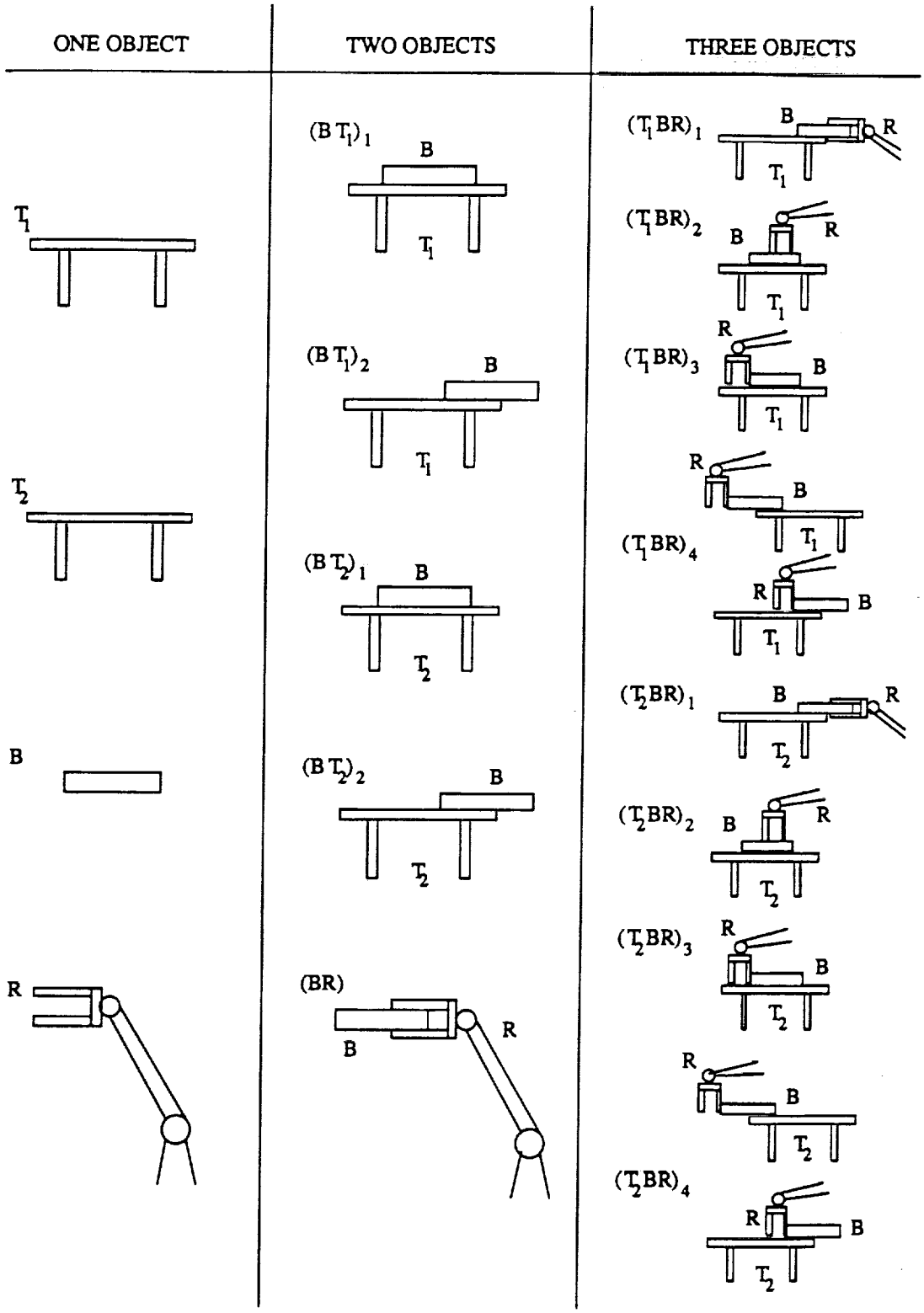


Figure 3.8: System geometric states representation for moving book.

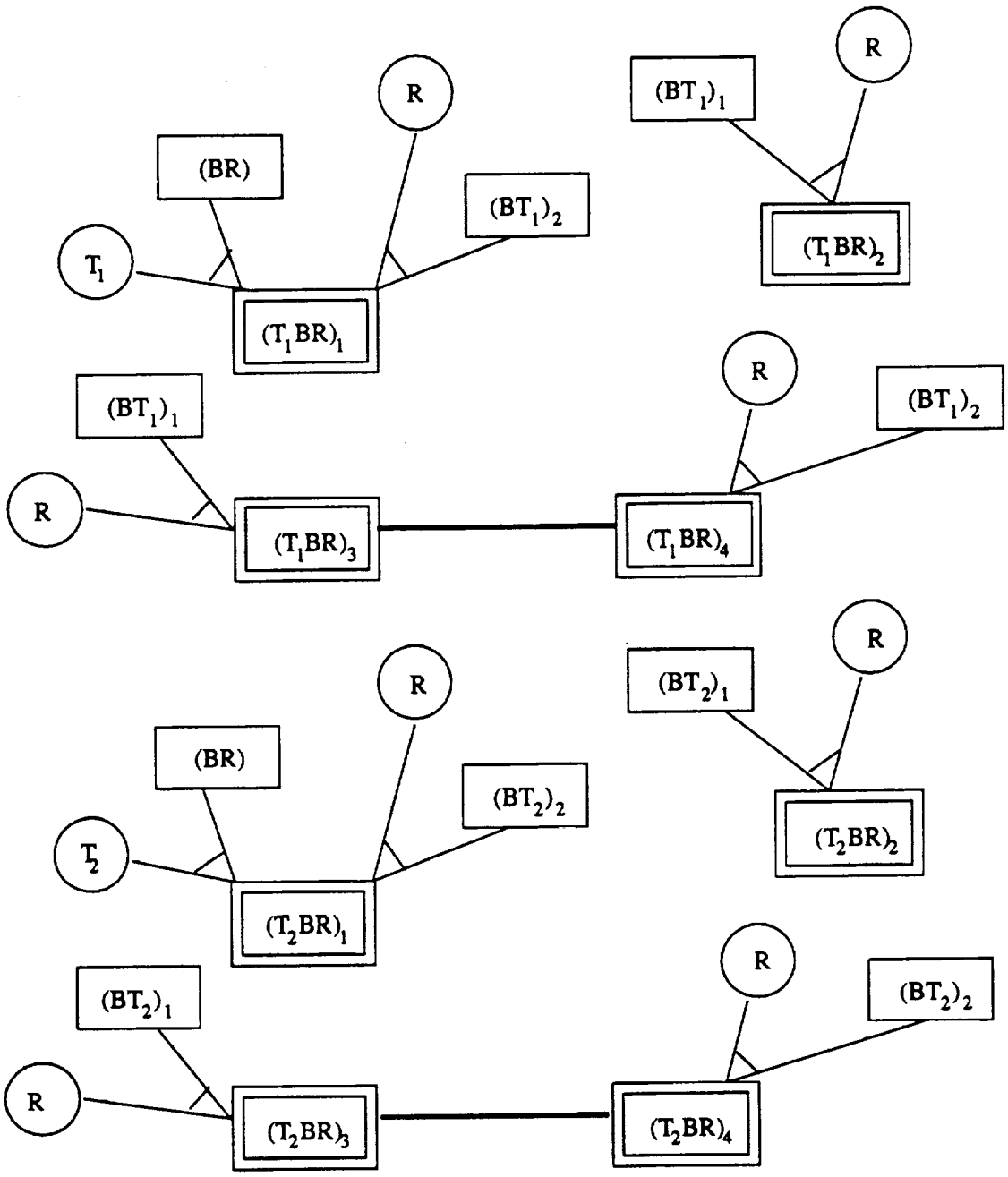


Figure 3.9: The AND/OR net representation for moving book.

shown as follows:

1. t_{10}: the robot moves towards table 1 and touches the book which is lying on the surface of table 1.
2. t_{14}: the robot forces the book on table 1 to move to the edge of the table.
3. t_{11}: the robot leaves the book and table 1.
4. t_6: the robot reaches table 1 again and grasps the book towards the edge of table 1.
5. t_4: the robot which has grasped the book leaves table 1.
6. t_{15}: the robot moves towards table 2 and makes the book touch the surface and lie on the edge of table 2.
7. t_{17}: the robot leaves the book and table 2.
8. t_{24}: the robot touches the book again but the orientation of the gripper has already been changed.
9. t_{25}: the robot forces the book to move to the center of the surface of table 2.
10. t_{21}: the robot leaves the book and table 2 and then goes back to its original place.

This is the only shortest-path solution which could be found for this problem. This optimal task sequence is illustrated in Figure 3.11. This sequence is reversible.

3.6 Conclusions

The AND/OR net is introduced as a tool for representation and reasoning about geometric constraints in a robotic workcell system. A method for mapping the AND/OR net to a Petri net is provided. Some properties of this Petri net are also verified. A directed Petri net is discussed to include more general cases for modeling a system. We could obtain all possible task sequences by searching from

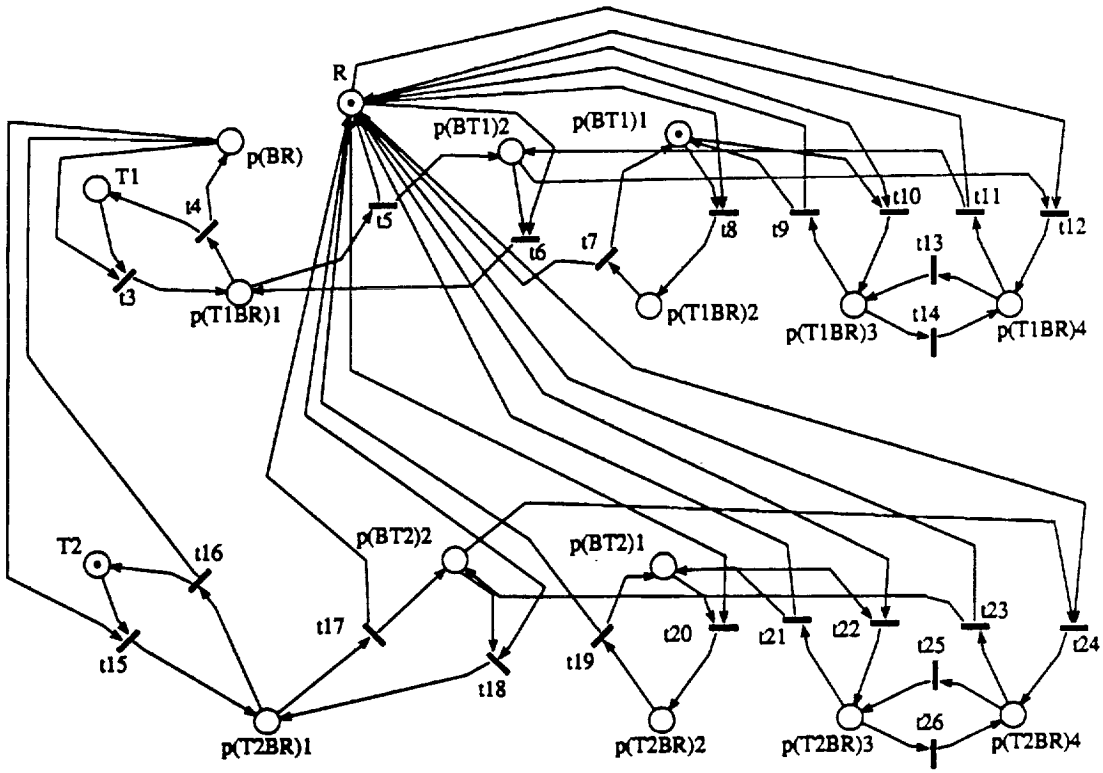


Figure 3.10: The Petri net representation for moving book.

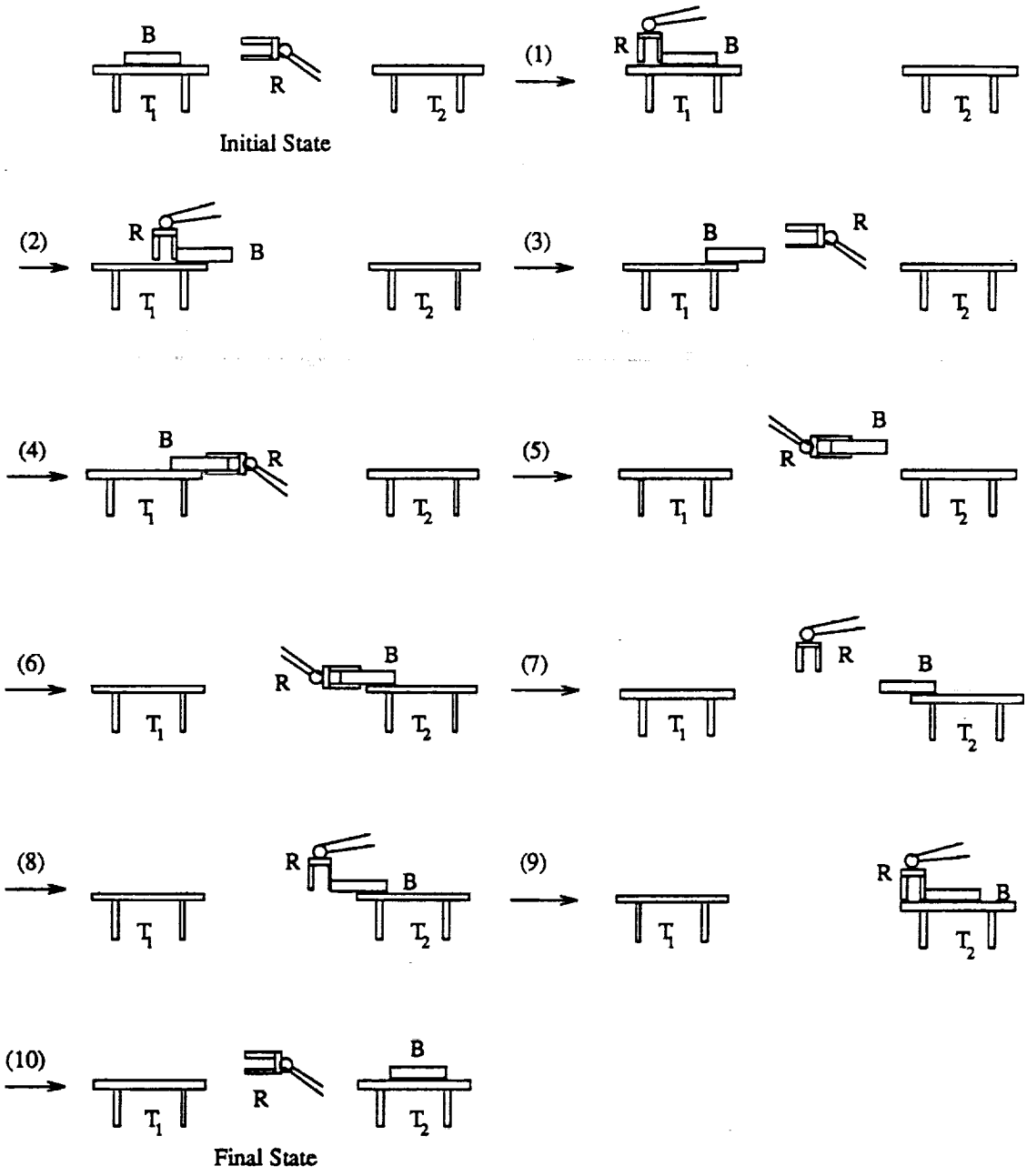


Figure 3.11: A feasible sequence from the initial state to the final state.

the state graph resulting from the AND/OR net, or constructing the reachability tree from the Petri net. The optimal task sequence could be directly searched from the AND/OR net without necessarily creating the complete state space. This off-line planning system has been implemented. The ideas presented here can be applied to robotic planning problems in manufacturing and non-manufacturing domains.

The selection and evaluation of all feasible task sequences, is an important dimension of this work and depends on factors such as time, cost, flexibility, or least-error-possibility. Another extension of this work involves selection and execution of *parallel* operations when a chosen sequence is implemented, so that the time is reduced and resources are used efficiently. The work discussed in this chapter also leads to approaches to error detection and recovery.

CHAPTER 4
TASK DECOMPOSITION AND ANALYSIS OF ROBOTIC ASSEMBLY TASK PLANS USING PETRI NETS

This chapter describes an approach to robotic task sequence planning which decomposes tasks into operations sequences for a generic robotic workcell. The approach provides a framework for *robust* execution of tasks through properties of: *traceability* — implicit mapping of operations to task representation, and *viability* — retaining multiple paths for execution. Given the descriptions of the objects in this system and all feasible geometric configurations and relationships among these objects and combinations of objects, an AND/OR net which describes the relationships of all feasible geometric states and associated feasibility criteria for net transitions is generated. This AND/OR net is mapped into a Petri net which incorporates all feasible sequences of high level operations. The resulting Petri net is then decomposed in a stepwise manner into lower level Petri nets of which each transition can be directly implemented by control commands or command sequences based on devices and objects in the system, or, by lower level planning transitions corresponding to path planning, grasp planning, fine motion planning, etc. All possible task sequences are found using an efficient algorithm which first generates all feasible *system states*. A shortest sequence may be chosen from the lowest level decomposition and is guaranteed to be the shortest sequence output of the hierarchical planning system to efficiently implement the desired tasks. The property analysis for different levels of decomposition is also presented, and the inheritance of properties between levels is defined.

4.1 Introduction

Assembly sequence planning generates sequences of mating operations among objects which will be assembled, including both the original single components and the subassemblies. The AND/OR graph[47] is an efficient way to represent all feasible assembly sequences at this level of task description[48, 49, 100]. For a robotic assembly system, operations which incorporate active devices such as robots or sensors require a more complete description of feasible tasks and a plan for the execution sequences of these devices. The resulting plans facilitate real time implementation and coordination between high level planning and lower level control. In this chapter we describe a decomposition of high level task sequences using a Petri net representation which facilitates the analysis of robustness properties for the resulting plans.

An assembly task may be represented by the feasible geometric states of objects and the transitions among those states, and the assembly sequence planner selects feasible sequences of these states and transitions. When an assembly task sequence is selected, each task must be further decomposed to generate a lower level operations sequence. Such a decomposition raises fundamental issues regarding the properties of the resulting operations sequence. New deadlock situations, conflicts among resources, and error states may arise in the decomposed sequence which were not present at the higher level. In this chapter, we describe an approach to assembly plan decomposition which retains two important properties: (1) *traceability* — each action is traceable to its role in the higher level plan, and (2) *viability* — several paths of actions may be retained and chosen on line. These properties are specifically intended to support *robust execution* of the task, and overcome difficulties in resource conflict and error recovery.

In this chapter, we use an AND/OR net[10, 11, 13, 16] to represent task level sequences and then follow a top-down hierarchical decomposition procedure

based on Petri nets to develop a more comprehensive robotic task representation. We search among feasible sequences from the final net using an efficient search algorithm. This representation and decomposition procedure is described in the following sections. Section 4.2 describes the decomposition of operations for a task. Section 4.3 summarizes the AND/OR net representation for high level tasks, as well as the mapping from an AND/OR net to a Petri net. Section 4.4 describes the decomposition of commands or transitions from different *assembly* or *disassembly* operations. In section 4.5, the net is further decomposed based on types of device motion and sensors. Section 4.6 discusses the simulation results for decompositions and sequences searched from the nets, and Section 4.7 discusses the conclusions of this work and directions for future work.

4.2 Representation of a Robotic Assembly System

Consider a generic assembly system composed of n components, C_1, C_2, ..., C_n. Three possible types of components are defined as follows:

Definition 4.1 *Active components:* An active component has controllable motions in a defined workspace. It may move other components when combined into a *subassembly* or *assembly.*

Definition 4.2 *Passive movable components:* A type of passive component. A passive movable component is defined to be a component which is movable if and only if it is operated on by some active component.

Definition 4.3 *Passive static components:* A type of passive component. A passive static component is fixed in a certain position, and therefore cannot move even if it is combined with some active component.

Usually, the position and orientation of passive static components are known prior to the execution of the system, while the position and orientation of passive

movable components may be unknown before the system starts and thus may contain uncertainties. For example, a block on a table is a passive movable component, but the table is considered as a passive static component. The sets of active, passive movable, and passive static components are independent and their union is the set of components in the system.

A component group, $\{C_{i_1}, C_{i_2}, \ldots, C_{i_m}\}$, in which all components are in contact with each other is called a subassembly, and specifically, a group of components, which is a desired configuration in the final state and only appears in the final state, is called an assembly. We define an *object* in the system as either a component, or a subassembly, or an assembly. At any given time, the system has some objects and we define the *system assembly state* at that time as the set of these objects.

An *operation* may affect this system assembly state by destroying some objects and creating some new objects. We assume when an operation is taking place, at most two objects may be deleted and at most two new objects may be created. In addition, at the time when objects are being deleted or created, exactly one object deleted or created may contain one active component. For example, a robot R is defined to be an active component. After the robot R holding a component A contacts another component B, two objects, RA(R holds A) and B are deleted and one new object, RAB, is created. The resulting object, RAB, is called an *active object*, if it contains one active component and only passive movable components, or contains internal states which may be modified.

The internal state of a component or subassembly is defined in terms of its *properties*. A *property* of a single component or a subassembly may change, or, the parameters corresponding to the interrelationships of some components or subassemblies within this object may change. For example, if an object O consists of two components, C_1 and C_2, and the distance between C_1 and C_2 is extended, we say that the parameter of O, in this case, the distance, is changed and a new or

modified object O' thus takes the place of the old object O.

Based on these characteristics of feasible objects, three types of basic operations are defined: *assembly*, *disassembly*, and *Internal State Transition(IST)* operations. We define the *precondition* of an operation as the objects being destroyed, and the *postcondition* of this operation as the new objects being created. For an *assembly* operation, the precondition is two objects, O_{i_1} and O_{i_2}, and the postcondition is one object, O_j. For a *disassembly* operation, the precondition is one object, O_j, and the postcondition is two objects, O_{i_1} and O_{i_2}. For an *IST* operation, the precondition is one object, O_i, and the postcondition is another object, O_j.

In Section 4.3, we will introduce an AND/OR net[10, 11, 13, 16] which represents these operations and states. In this representation, if the precondition or postcondition of an operation has more than one object, an *AND* arc is used to connect the operation with these objects to show the necessary coexistence of these objects. The *OR* relationship for choosing a feasible operation from all enabled operations is represented by several *AND* arcs or *IST* arcs from the same set of nodes. The decomposition of assembly tasks is achieved using a predefined library of primitive operations. While these operations may vary in detail for different implementations, they capture the fundamental requirements for assembly task execution. Section 4.4 describes this decomposition in more detail.

Each *assembly* operation is decomposed into a sequence of *move* and *combine*. To form an assembly configuration by an active component, a motion operation is necessary to reach the corresponding object. After motion is performed, a mating operation combines the objects by establishing a new contact state. The mating operations include *insert*, *screw*, *grasp*, or *put* operations.

Similarly, a *disassembly* operation can be decomposed into a sequence of *separate* and *move*. The move operation here is in a different direction from that in an assembly operation. The unmate operation separates objects by destroying contact

relations. The mate and unmate operations are both problem-dependent and their implementation depends on the descriptions of tasks.

There are different types of *IST* operations, and many are problem specific. In this chapter, we consider only *move-with-contact* as an example of an *IST* operation. Move-with-contact changes the internal configuration of a subassembly by sliding objects along contacting surfaces. The property which changes is the relative position of parts in the configuration. Sliding an object along a table is an example of this type of *IST* operation.

A *move* operation is further decomposed into a *free-move* suboperation and a *fine-move* suboperation. A free-motion assumes a wide range of workspace, relative high speed, and no tightly constraining obstacles. A fine-motion moves in a small, constrained workspace and moves with a relative slow speed and may often involve compliance or contact motions. A free-motion makes an active object *roughly* reach a goal and a fine-motion makes the active object *exactly* reach a goal. The precedence of free-motion and fine-motion in motion suboperations for an *assembly* and a *disassembly* operation are different. For motion in an *assembly* operation, the free-motion precedes the fine-motion, while in a *disassembly* operation, the fine-motion precedes the free-motion.

Sensors are necessary to assess the current state of a system during execution, and a viable planning strategy must incorporate on-line sensor-based decisions. Sensors are used to restore uncertainties on-line and instantiate lower-level plans. In our work, we construct plans with sensors for (1) state verification, (2) state identification, and (3) sensor-based control. Sensory state verification and identification are used to determine system states and parameters and determine the subsequent task sequence. Sensors are also used in lower-level sensor-based operations which incorporate dynamic sensory feedback to achieve adaptive modes of on-line operation.

In related work[12, 14, 20, 21], we have shown the use of a fuzzy Petri net representation to embed fuzzy reasoning rules and incorporate sensory observations into the on-line task sequence. In all of these cases, the sensor becomes a resource constraint on the sequence of operations in the system, and the resulting Petri net planning tools provide a convenient means to represent and reason about these resources.

For most operations, a motion plan is required before the operation is executed. For sensor-based motion, a plan may specify a mode of sensor-based motion and constraints, but not an explicit path. In our task representation, the motion plan is also viewed as a resource and modeled as such in the Petri net representation. The existence of the plan is therefore an explicit precondition for the execution of motion.

To sequence the generation of plans as well as the task sequences, a *planning for planning* problem occurs, i.e., when to generate a plan for a certain operation and how to schedule the generations of all plans in the system. One approach is to generate the plan right before the execution of the operation so that the uncertainty of the dynamic working environment can be minimized. To reduce the conflict of resources in the system, we search a shortest sequence from the final Petri net. This sequence will guarantee a smallest number of resource conflicts.

4.3 AND/OR Net and Petri Net Representation for High Level Tasks

As discussed in the last section, an *assembly* operation combines several single components or component groups to a new component group. A *disassembly* operation decomposes a component group into a set of single components and component groups. An *IST* operation makes an internal state change for a single component or a component group. The following discussions are based on the representation and sequencing of these operations.

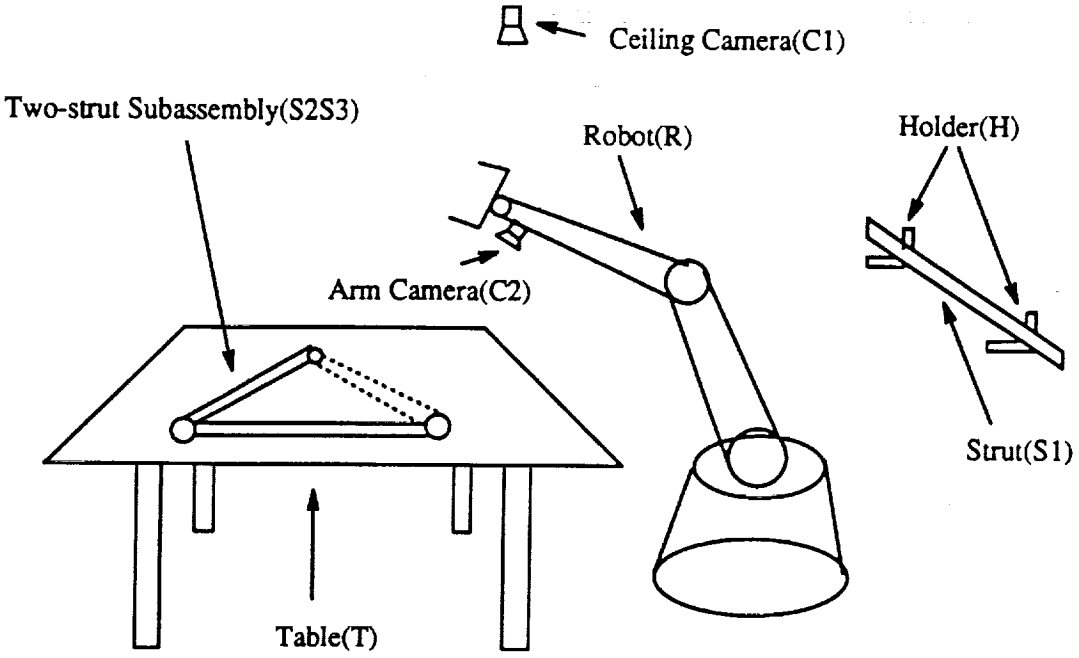


Figure 4.1: A strut-triangle assembly system.

4.3.1 AND/OR Net Representation for Assembly Sequences

Geometric states are used to describe the states for all components and feasible component groups in the system. Each assembly, disassembly, or Internal State Transition operation can be considered as reaching from one system geometric state to another geometric state. The system geometric state was defined in [11, 12]:

Definition 4.4 *System geometric state:* A set of objects which constitute the system including single components, subassemblies, or assemblies. Each object has geometric substate, which represents the corresponding geometric configuration or relations among the components of the object.

For a strut-triangle assembly system example shown in Figure 4.1, the corresponding system geometric states table is shown in Table 1.

Note that each geometric configuration may either represent a contacting relationship of objects with internal state parameters, or a fixed relationship among objects. The fixed relationship is a pre-specified relationship among related objects

TABLE 1

SYSTEM GEOMETRIC STATES

Single Objects	Type	Object External States	Internal States
R(robot)	active	position/orientation	joint positions(kinematics)
H(holder)	passive static	position/orientation	
S1(strut 1)	passive movable	position/orientation	
S2(strut 2)	passive movable	position/orientation	
S3(strut 3)	passive movable	position/orientation	
T(table)	passive static	position/orientation	
C1(ceiling camera)	passive static	position/orientation	
C2(arm camera)	passive static	position/orientation	

Combined Objects	Type	Object External States	Internal States
RS1	active		⟨ joint positions grasp point
S1H	passive static	position/orientation	⟨ holder slot # fine position in holder
RS1H	active		⟨ joint positions grasp point
S2S3T	passive movable		position/orientation of S2S3 on table
S1S2S3T	passive movable		position/orientation of S1S2S3 on table
RS1S2S3T	active	⟨ joint positions grasp point	position/orientation of RS1S2S3 on table

and the non-fixed relationship may be unknown or uncertain. The subassembly of an active component and a passive movable component is rigid, for example, the robot holding a strut, but the grasping position is an internal state variable. The subassembly of a passive movable component and a passive fixed component may or may not have a rigid relationship. For example, in Figure 4.1, the subassembly of $S2$ and $S3$ has a rigid relationship, while T and $S2S3$(or only $S1$) have a non-rigid relationship. A state verification and validation procedure(sensing) will be followed to check and confirm the geometric states for the objects during the execution of the assembly or disassembly task. The current state of a non-rigid group is an internal variable.

Each *assembly* operation may have a reversible decomposition operation. For example, the group of T(table), $S3$(strut 3) and $S2$(strut 2) represents a non-rigid combination of the table and two struts, where the two struts have already formed a subassembly for the triangle configuration. The group R(robot) and $S1$(strut 1) represent a rigid combination of the robot and a strut which shows that the robot is holding a strut. These two groups could be combined or *assembled* to generate a new group of which the robot is touching the assembly of three struts on the table. This new group can either be decomposed to two original groups, or be decomposed to two new groups. For a non-rigid group or object, the internal state change may be defined among all feasible states of this group or object. The system geometric states table incorporates all feasible geometric state relationships for possible states of components and component groups in the system.

The AND/OR net[10, 11, 16] was defined in Chapter 3. It is directly derived from the geometric states table. The nodes in the AND/OR net correspond to all feasible components and component groups, which in turn correspond to all feasible configurations in the geometric states table. The arcs in an AND/OR net have two

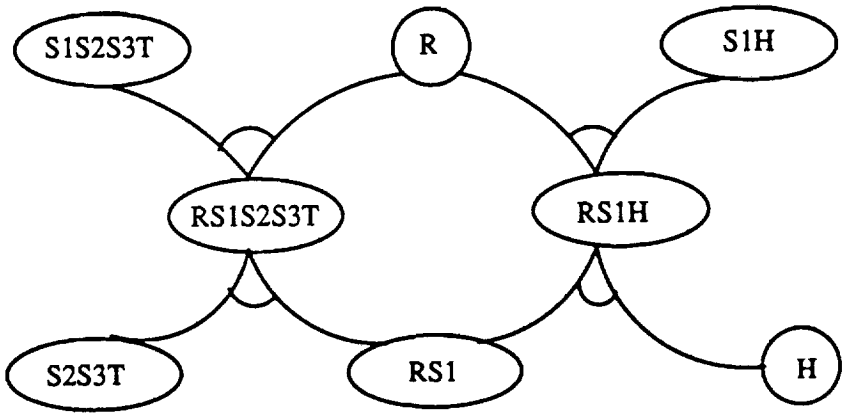


Figure 4.2: The AND/OR net representation.

types, i.e., AND arcs and IST arcs. The AND arcs represent the feasible combinations and decompositions of objects or object groups. The IST arcs represent the internal state changes for component groups or single components. The OR mapping indicates the alternate selections for several different operations, i.e., an component or component group may either follow an IST arc to transfer from one state to another state, or be decomposed to a set of new components and component groups, or be combined with other objects to generate a new component group. Therefore, the AND/OR net not only represents each operation for assembly or disassembly, but also represents the relationships among several alternative operations.

The AND/OR net representation for the strut-triangle representation is shown in Figure 4.2(The internal state variables are not explicitly shown in the figure). Note that the AND/OR net is from the task level point of view. An AND/OR net incorporates all feasible *assembly/disassembly/IST* sequences. We can directly observe an efficient sequence from Figure 4.2, i.e., the robot first reaches $S1$ which is held by the holder. Then the robot goes to pick up $S1$ and leaves the holder. Afterwards, the robot reaches the subassembly of $S2$ and $S3$ which is lying on the table and assembles the triangle structure. Last, the robot leaves the table and

leaves the strut-triangle on the table. The final state is thus reached. This particular *assembly/disassembly/IST* task sequence is listed as follows:

1. R, S1H(initial state) $\Longrightarrow$ RS1H, // *Assembly*
2. RS1H $\Longrightarrow$ RS1, H, // *Disassembly*
3. RS1, S2S3T $\Longrightarrow$ RS1S2S3T, // *Assembly*
4. RS1S2S3T $\Longrightarrow$ S1S2S3T, R(final state). // *Disassembly*

4.3.2 AND/OR Net to Petri Net Mapping

To implement convenient communications between the task planner and the execution controller and to clearly simulate and analyze the task sequence, we map the AND/OR net task level representation to a Petri net[10, 11, 16]. One important property of a Petri net is the representation of serial and concurrent events and resource constraints.[1]

A formal definition of Petri nets and related properties, as well as the mathematical operations on Petri nets, are defined in [84, 89]. Some definitions and notations were introduced in Section 3.3. The mapping of the AND/OR net to the Petri net is described in detail in [10, 11, 16]. For λ_1 and λ_2 discussed in §3.3.1, t_1 and t_2 are two new opposite transitions which are added to the Petri net where the nodes are being directly mapped to places correspondingly.

Each node in the AND/OR net is mapped to a place in the Petri net. All transitions in the system AND/OR net can occur in either direction based on the assumptions of reversibility and feasibility. Using this property, we can map each AND/OR net transition, i.e., an *AND* arc or *IST* arc, to two opposite transitions in the Petri net. Each place in the Petri net represents an individual state even though some states may represent the same component or component group. Thus the

[1] For our current analysis, we use the Generalized Stochastic Petri Nets(GSPN) software[73, 74] to represent the system and carry out some simulations as well as verifying the task sequences. Some properties of the Petri net such as T-invariants, P-invariants and so on, can be obtained using this software.

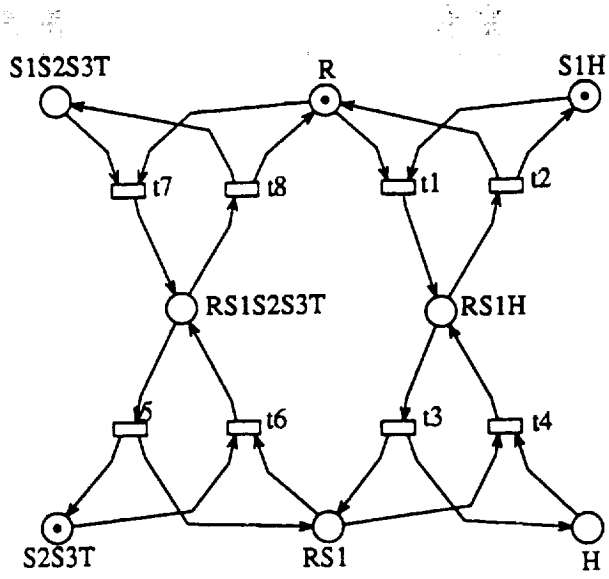


Figure 4.3: The Petri net, *PN0*, mapped from the AND/OR net.

Petri net is a complete representation of the system states, which offers advantages to model both state and operations sequences. We have shown that the following properties of the resulting Petri net, i.e., safeness, 1-boundedness, liveness, and reversibility, are guaranteed[10, 11, 16].

For the sake of simplicity, we call the Petri net mapped from an AND/OR net as a *Level 0 Petri net*(or *PN0*) in the following discussions. *PN0* for the strut-triangle example is shown in Figure 4.3. The initial state is represented by the initial marking of the Petri net. Before the operations of the system, the robot is free, strut 1 is "fixed" on the holder and the subassembly of $S2$ and $S3$ is lying on the table. This initial state is shown with a token in place R, $S1H$, and $S2S3T$, respectively. The final state will be represented with one token in place R, one token in place H, and one token in place $S1S2S3T$. The states of components and component groups existing at the same time are geometrically independent.

4.4 Level 1 Petri Net Decomposition

In this section, we define the high level decomposition of *assembly* operations into *move* → *combine* subsequences, and the decomposition of *disassembly* operations into *separate* → *move* subsequences, where *combine* may refer to grasp or mate operations in assembly. When replacing each assembly and disassembly transition in the AND/OR Petri net by a subsequence of *move*, *combine*, and *separate* operations, we generate a *Level 1 Petri Net(PN1)* in which some common characteristics of *assembly* and *disassembly* operations may be captured. In this decomposition, when we perform the property analysis in the resulting net, we can think of it as an expansion by corresponding subnets, rather than a replacement of transitions.

4.4.1 Decomposition Algorithm: PN0 to PN1

The mapping from *PN0* to *PN1* is the first step of the decomposition for assembly plans. We formally define the decomposition algorithm at this level as follows. It is assumed that the number of input objects of any *assembly* operation, and the number of output objects of any *disassembly* operation, are both 2.

Decomposition Algorithm 1: Decomposition of PN0 to PN1 Decompose a Petri net *PN0* $N = (P, T, \alpha, \beta)$ mapped from an AND/OR net, where $P = \{p_1, p_2, \ldots, p_m\}$, $T = \{t_1, t_2, \ldots, t_n\}$, $\alpha \subseteq \{P \times T\}$, $\beta \subseteq \{T \times P\}$, to a lower level net, *PN1*.

for $i := 1$ **to** n **do**

if $t_i \in \{assembly\_operations\}$, $\{(p_{i_1}, t_i), (p_{i_2}, t_i)\} \subseteq \alpha$, $(t_i, p_{i_3}) \in \beta$ **then**

{Each *assembly* operation is decomposed to a *move* command and a *combine* command.}

$T := T - \{t_i\} + \{t_i^r, t_i^c\}$; $n := n + 1$; $t_i := t_i^r$; $t_n := t_i^c$;

{Add a new state for p_{i_1} after moving, assuming p_{i_1} is or contains an active

component.}

if $p''_{i_1} \notin P$ **then**

$P := P + \{p''_{i_1}\}$; $m := m + 1$; $p_m := p''_{i_1}$

end { if };

$\alpha := \alpha - \{(p_{i_1}, t_i), (p_{i_2}, t_i)\} + \{(p_{i_1}, t^r_i), (p''_{i_1}, t^c_i), (p_{i_2}, t^c_i)\}$;

$\beta := \beta - \{(t_i, p_{i_3})\} + \{(t^r_i, p''_{i_1}), (t^c_i, p_{i_3})\}$

end { if };

elsif $t_i \in \{disassembly\_operations\}$, $(p_{i_1}, t_i) \in \alpha$, $\{(t_i, p_{i_2}), (t_i, p_{i_3})\} \subseteq \beta$ **then**

{Each *disassembly* operation is decomposed to a *separate* command and a *move* command.}

$T := T - \{t_i\} + \{t^s_i, t^l_i\}$; $n := n + 1$; $t_i := t^s_i$; $t_n := t^l_i$;

{Add a new state for p_{i_2} after moving.}

if $p''_{i_2} \notin P$ **then**

$P := P + \{p''_{i_2}\}$; $m := m + 1$; $p_m := p''_{i_2}$

end { if };

$\alpha := \alpha - \{(p_{i_1}, t_i)\} + \{(p_{i_1}, t^s_i), (p''_{i_2}, t^l_i)\}$;

$\beta := \beta - \{(t_i, p_{i_2}), (t_i, p_{i_3})\} + \{(t^s_i, p''_{i_2}), (t^s_i, p_{i_3}), (t^l_i, p_{i_2})\}$

end { elsif }

end { for }.

4.4.2 Analysis

After we obtain *PN1*, we are interested in the analysis of properties such as liveness, boundedness, and reversibility of this net. To avoid directly analyzing this net which is more complicated than the original net, we investigate whether the properties of the original net are inherited after we perform the decomposition. The method of applying reduction rules to analyze a large system[62, 84], which reduces

it to a smaller and simplified system, is helpful to examine the properties of the resulting system when we know the properties of the smaller system before decomposition. As an approach to refine the Petri net, Valette[111] proposed a method which replaces the transitions in the net with corresponding subnets, and guarantees the resulting net to preserve the properties of liveness and safeness. Suzuki and Murata[106] generalized the method for stepwise refinement or abstraction of the Petri net representation, retaining the properties of liveness and boundedness. To refine or simplify a net more efficiently without the loss of properties, Berthelot proposed a set of transformations[4] which preserves the classical properties in nets. A decomposition technique was also discussed to split a system into subsystems which can be analyzed separately[5]. In another approach, reversibility was considered in net decomposition by Zhou *et al.*[122].

To analyze the properties of decomposition for our application in robotic systems, we propose the following theorem which can be used to show that the resultant net retains the properties of liveness, 1-boundedness, safeness, and reversibility. Partial or similar results can be obtained using the results in [106, 111, 122].

Theorem 4.1 If a place in a Petri net shown in Figure 4.4(a) is replaced by a subnet shown in 4.4(b), and the original net is live, bounded, safe, and/or reversible, then the resulting Petri net is also live, bounded, safe, and/or reversible.

Proof: We prove the inheritance of properties of liveness, boundedness, safeness, and reversibility separately as follows.

Liveness: Based on the assumption, the net containing Figure 4.4(a) is live, i.e., no matter what marking is reached from an initial marking, it is possible to ultimately fire any transition of the net by progressing through some further firing sequence. Any transition $t \in T$ can be enabled after a sequence of transition $\mathcal{S} = t_{i_1} t_{i_2} \ldots t_{i_s}$. After p is replaced by the subnet in Figure 4.4(b), we have: (i) If $\mathcal{S}$ passes through p in N, a token will be placed in p via arc (1) or arc (3). Correspondingly, p_1 or

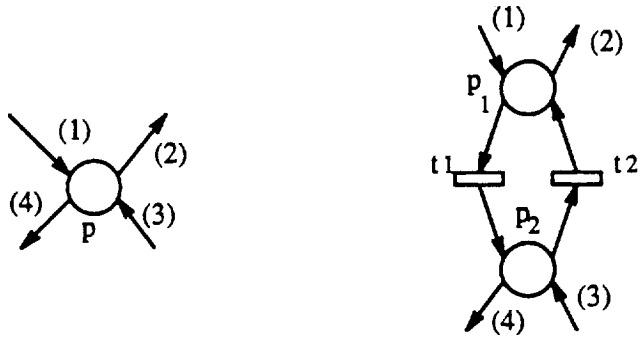


Figure 4.4: Decomposition of a place to a subnet. The net in (a) is $N = (P, T, \alpha, \beta)$, and the net in (b) is $N' = (P', T', \alpha', \beta')$.

p_2 in N' will get a token. Then, $\mathcal{S}$ will pass through arc (2) or (4) in N. Again correspondingly, in N', $\mathcal{S}$ will not change, or, t_1, t_2, t_1t_2, or t_2t_1 is added in the sequence, so that each transition $t \in T' - \{t_1, t_2\}$ can be enabled after a sequence of transitions. (ii) If $\mathcal{S}$ doesn't pass through p in N, the sequence which make any transition in N' enabled except t_1 and t_2, will be the same as in N. Moreover, to enable each transition in N, p should contain a token at least once. Correspondingly, either p_1 or p_2 should contain a token at least once. Therefore, t_1 and t_2 are also enabled when p_1 or p_2 contains a token. Each transition in N' is enabled after firing a certain sequence of transitions. The liveness of N' is guaranteed.

Boundedness: We need to show that for any place in N, the number of tokens does not exceed k. After p is replaced by the subnet in Figure 4.4(b), for any sequence of transitions $\mathcal{S} = t_{i_1} t_{i_2} \ldots t_{i_s}$, we have: (i) If $\mathcal{S}$ passes through p, as in the proof of liveness, for N', the sequence will be the same, or t_1 or t_2 is inserted in the sequence, or a loop of t_1t_2 or t_2t_1 is inserted. All these will not change the capacity of tokens in places of N'. (ii) If $\mathcal{S}$ does not pass through p, the sequence will be the same as in N. Therefore, the property of boundedness is verified.

Safeness: Safeness is a special case of boundedness in which the maximum capacity for each place is 1. Following the same strategy as the proof of boundedness, we can

also verify the property of safeness of N'.

Reversibility: As in the proof of liveness, we assume a reversible sequence from any reachable marking to the initial marking is $\mathcal{S} = t_{i_1} t_{i_2} \ldots t_{i_s}$. (i) If this sequence passes through p, based on the direction of $\mathcal{S}$ following (2) or (4), we get an updated $\mathcal{S}$ which is the same as the original sequence, or with t_1 or t_2 added, or with a loop of $t_1 t_2$ or $t_2 t_1$ added. (ii) If this sequence does not pass through p, $\mathcal{S}$ will not be changed. Moreover, if the reachable marking contains a token in p, then accordingly, there is a token in p_1 or p_2 and the sequence will also be the same or changed with t_1, t_2, $t_1 t_2$ or $t_2 t_1$ added. In any case, the reversibility property in N' is guaranteed.

Q.E.D. □

4.4.3 PN1 for the Example

For the assembly system shown in Figure 4.1, the objects consist of active devices and passive parts. As described in Table 1, the only active device is the robot(R). The parts are designated as fixed(static) and movable parts. The fixed parts are the table(T) and the holder(H). The movable parts are strut 1, strut 2, and strut 3. The *combine* operations for $(R, S1) \rightarrow RS1$ and $(R, S1H) \rightarrow RS1H$ are 'grasp', while the *combine* operations for $(S1, S2S3T) \rightarrow S1S2S3T$ and $(RS1, S2S3T) \rightarrow RS1S2S3T$ are 'mate'.

If we replace each transition in the Petri net shown in Figure 4.3 by associated *move, combine,* and *separate* operations the *PN1* net is generated. The resulting Petri net is shown in Figure 4.5. Using the above theorem, we know that *PN1* maintains the properties of liveness, safeness, and reversibility. Using a search algorithm for feasible sequences[16] in a Petri net, a feasible task command sequence to reach from the initial state to the final state is generated as t1(Move R1 S1, Grasp R1 S1), t3(Move_Comp R1S1 H, Move R1S1 R1(Temp_Pos)), t6(Move R1S1 S2S3T, Move_Comp R1S1 S2S3T), t8(UnGrasp R1 S1S2S3T, Move R1 R1(Init_Pos)). This

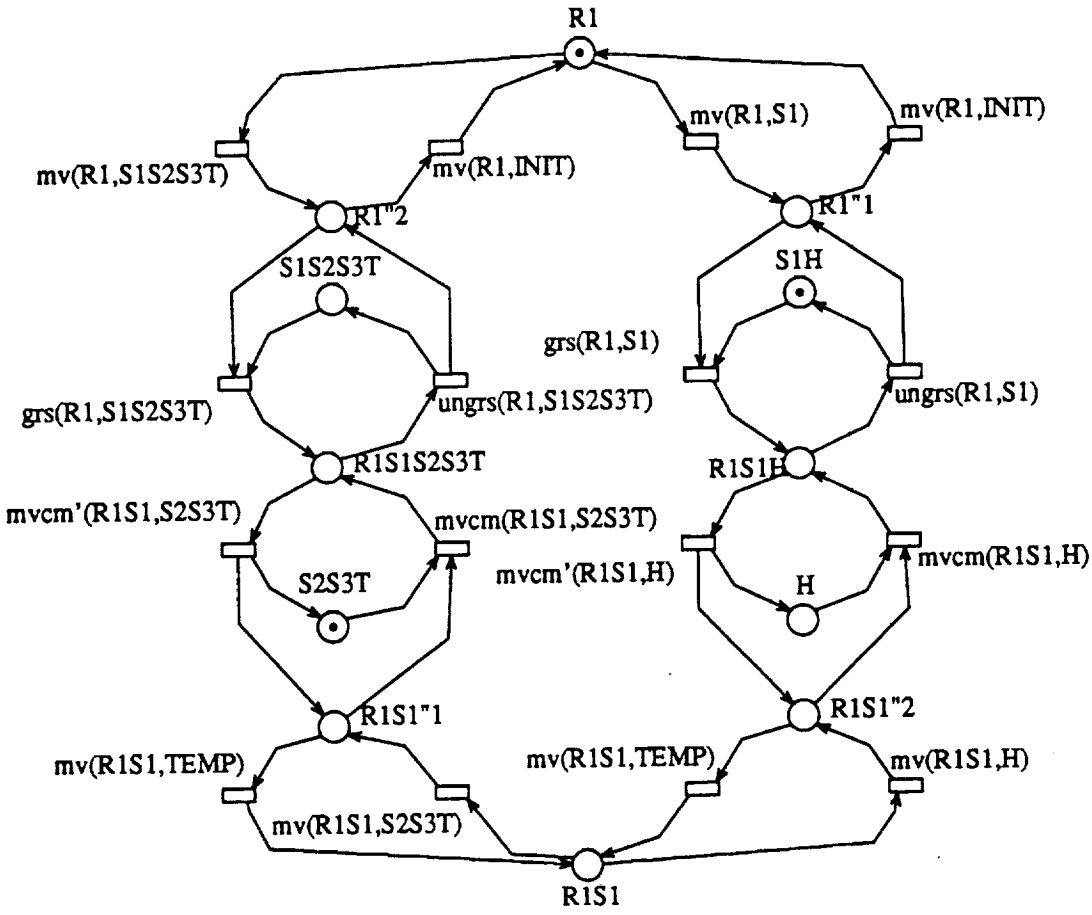


Figure 4.5: Level 1 Petri net, *PN1*, for the example in Figure 4.1. Each transition in the net represents an operation. The label indicates the type of operation: mv(Move), grs(Grasp), ungrs(UnGrasp), mvcm' and mvcm(Compliant Move, in different directions); and the objects involved: R1(robot 1), S1(strut 1), INIT(initial position of robot 1), R1S1(R1&S1 subassembly), H(holder), TEMP(temporary position of R1S1 in the free space), S2S3T(S2&S3 subassembly on the table), S1S2S3T(S1&S2&S3 assembly on the table). The first operand is the movable object.

sequence can be linguistically described as: (1) robot 1 moves to strut 1; (2) robot 1 grasps strut 1; (3) robot 1 holding strut 1 compliantly leaves the holder; (4) robot 1 holding strut 1 reaches the temporary position; (5) robot 1 holding strut 1 moves to subassembly S2S3 on the table; (6) robot 1 holding strut 1 compliantly moves to subassembly S2S3 on the table; (7) robot 1 ungrasps assembly S1S2S3 on the table; (8) robot 1 moves to the initial position.

4.5 Level 2 Petri Net Decomposition

The *PN1* shown in Figure 4.5 can be further decomposed to a set of lower level operations based on the types of motion and the resources required. The decomposition of motion is represented by expansion of a *move* transition into free-motion and fine-motion. The addition of resources required for sensing and planning is achieved by adding places. In the examples discussed here, places are added to represent motion plans and sensors. In the example shown in Figure 4.6, an *assembly* operation is decomposed into motion and mating operations, with motion plans 'P' required as preconditions, and a camera, C2, required to control the fine-motion.

4.5.1 Decomposition of Motion to Free-Motion and Fine-Motion

The following algorithm decomposes each motion operation into free-motion and fine-motion steps. In the resulting Level 2 Petri net, no two places represent the same state for the same component or group. Because this net can be considered as replacing some places by corresponding subnets, and *PN1* has been shown to have properties of liveness, safeness, and reversibility, the resulting *PN2* net in Figure 4.7 also has these properties. The formal algorithm to perform this decomposition is shown as follows:

Decomposition Algorithm 2: Decomposition of Each Motion Command

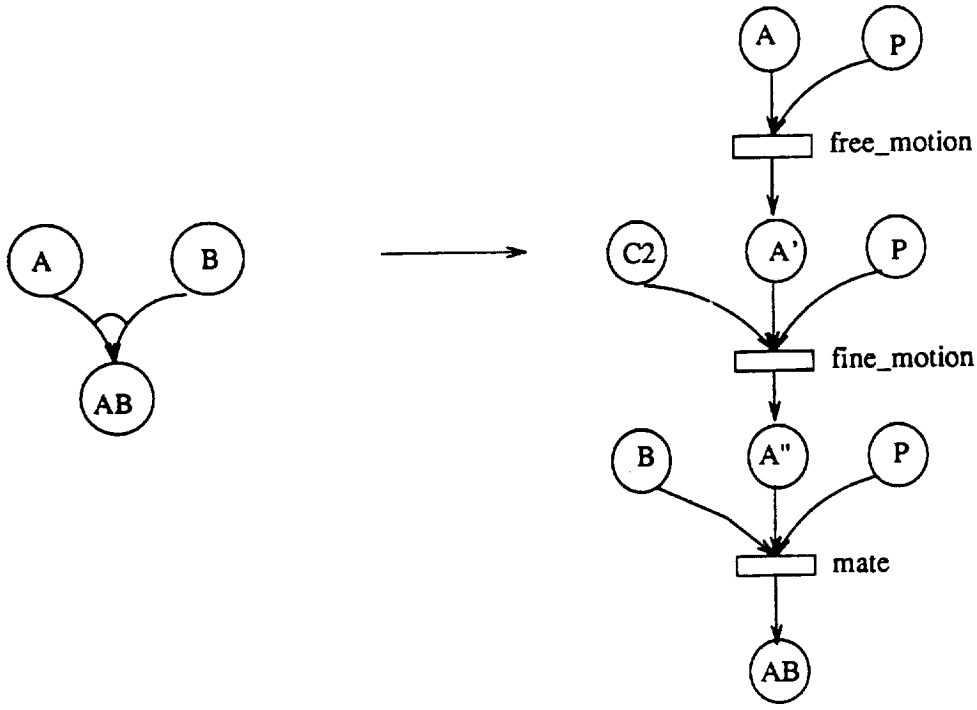


Figure 4.6: Decomposition for the AND/OR net to Level 2 Petri net. In the resulting Petri net, places 'P' are the precondition plans, for the corresponding motion or mating operations. Place 'C2' is used to indicate the arm camera, which is used for sensor-based motion.

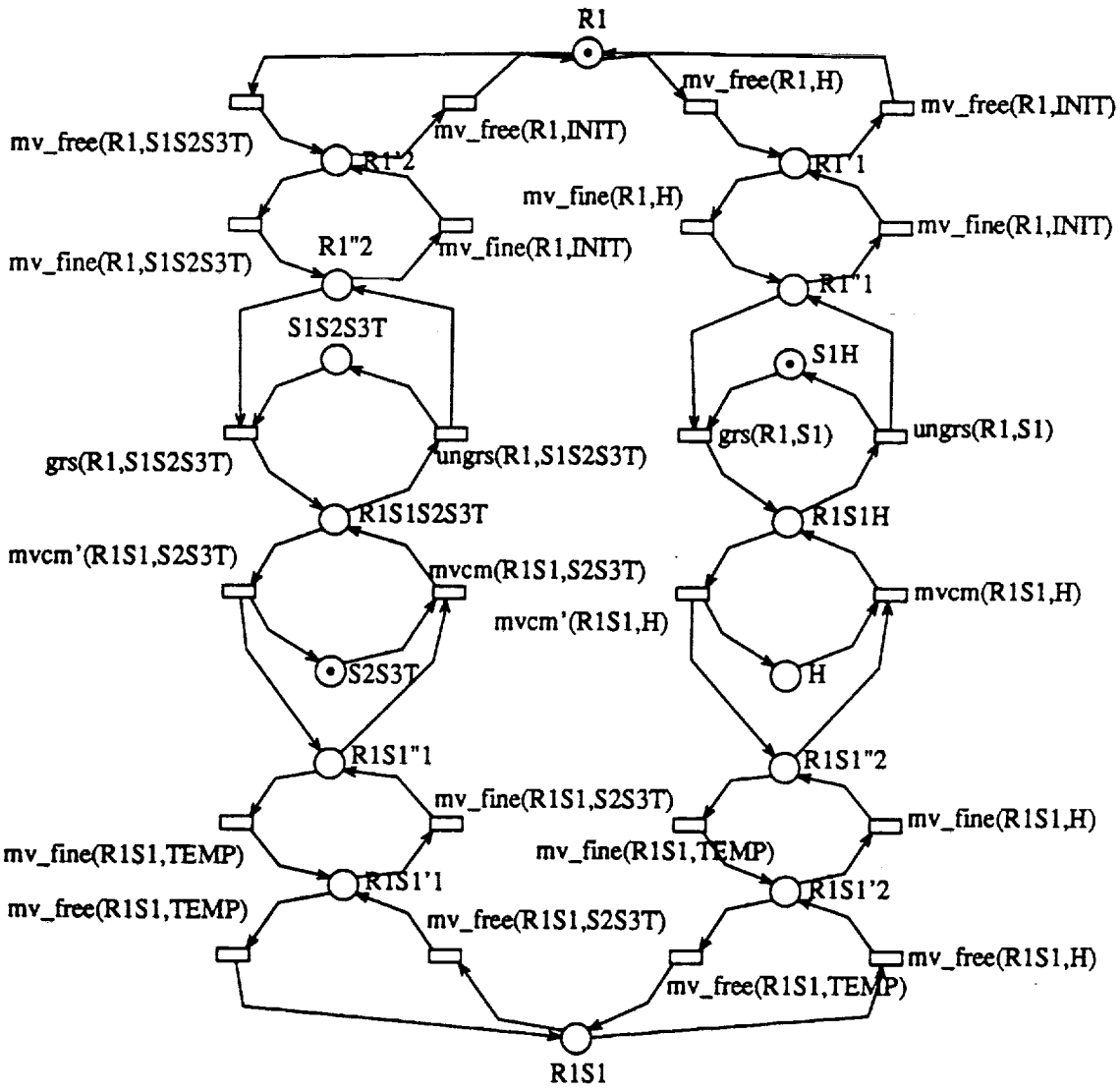


Figure 4.7: Decompositions for Level 2 Petri net, *PN2*, with expanded motion operations.

in PN1 to Free-Motion and Fine-Motion Commands.

for $j := 1$ **to** n **do**
if $t_j = t_i^r$, $(p_t, t_i^r) \in \alpha$, $(t_i^r, p_t'') \in \beta$ **then**
{t_i^{r1} is a free-motion command and t_i^{r2} is a fine-motion command.}
$T := T - \{t_i^r\} + \{t_i^{r1}, t_i^{r2}\}$; $n := n + 1$; $t_j := t_i^{r1}$; $t_n := t_i^{r2}$;
if $p_t' \notin P$ **then**
$P := P + \{p_t'\}$; $m := m + 1$; $p_m := p_t'$
end { if };
$\alpha := \alpha - \{(p_t, t_i^r)\} + \{(p_t, t_i^{r1}), (p_t', t_i^{r2})\}$;
$\beta := \beta - \{(t_i^r, p_t'')\} + \{(t_i^{r1}, p_t'), (t_i^{r2}, p_t'')\}$
end { if };
elsif $t_j = t_i^l$, $(p_t, t_i^l) \in \alpha$, $(t_i^l, p_t'') \in \beta$ **then**
{t_i^{l1} is a fine-motion command and t_i^{l2} is a free-motion command.}
$T := T - \{t_i^l\} + \{t_i^{l1}, t_i^{l2}\}$; $n := n + 1$; $t_j := t_i^{l1}$; $t_n := t_i^{l2}$;
if $p_t' \notin P$ **then**
$P := P + \{p_t'\}$; $m := m + 1$; $p_m := p_t'$
end { if };
$\alpha := \alpha - \{(p_t, t_i^l)\} + \{(p_t, t_i^{l1}), (p_t', t_i^{l2})\}$;
$\beta := \beta - \{(t_i^l, p_t'')\} + \{(t_i^{l1}, p_t'), (t_i^{l2}, p_t'')\}$
end { elsif }
end { for }.

4.5.2 Adding Resource Places to the Net

In a further refinement of the plan, we introduce additional lower level objects which represent required resources for execution. In these examples, sensors and plans are introduced as lower-level objects.

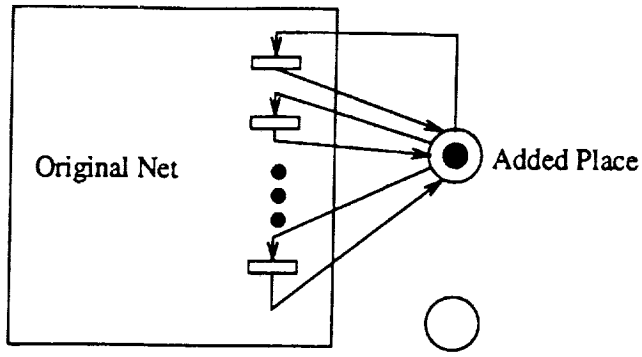


Figure 4.8: Adding a place with loop connections to transitions in a Petri net and/or a place separable with the net.

Many transitions may require a plan as a precondition to execution. There are four kinds of planners existing for this example: free-motion planners, fine-motion planners, grasp planners, and sensor-based motion planners. The plans are often implemented on-line because of the uncertain state and the dynamic environment.

Certain fine motions may require a sensor to perform sensor-based motion. In this example, camera 2 is a shared resource and availability of C2 is represented by a token in a C2 place. The decomposition is based on the following theorem and algorithm.

Theorem 4.2 If we add a place, which (1) contains a token, (2) forms loop connections with some transitions, and/or, a separate place without any connection to a Petri net(Figure 4.8) which has the properties of liveness, boundedness, safeness, and reversibility, the resulting net also preserves these properties.

Proof: (i) The liveness of a Petri net is determined by firing a sequence of transitions and the number of tokens received and produced by transitions. When we fire a sequence of transitions, if this sequence passes through the transitions which have connections with the specified place, these transitions will not produce different numbers of tokens for this place or other places as in the original net. If the sequence does not pass through this place, and/or, a separate place is added to the net, the

sequence firing will be the same as in the original net. In any case, the liveness will be inherited by the resulting net. (ii) The boundedness relates to the capacity of tokens in each place. As we have shown in proof (i), the boundedness of the original net is also preserved in the resulting net. (iii) Because the token in the place we add to connect to the original net always contains 1 token, and the number of tokens in other places will not exceed 1, the resulting net is safe. (iv) When we follow a reversible sequence to go back to the initial state from any reachable marking, because the place we add to connect to the net does not influence any transition in any sequence, the reversibility property is reserved. The case is the same for adding a separate place.

Q.E.D. □

The following algorithm formally proposes the decomposition to add $PLAN$s to all transitions, and a sensor to all sensor-based transitions.

Decomposition Algorithm 3: Add $PLAN$s for All Transitions and Add Sensors to All Sensor-Based Transitions.

{add plans for all transitions}
for $i := 1$ **to** n **do**
  $P := P + \{PLAN_i\}$; $m := m + 1$; $p_m := PLAN_i$; $\mu_m := 1$
end { for };
{add C2 to all sensor-based transitions}
$P := P + \{C2\}$; $m := m + 1$; $p_m := C2$; $\mu_m := 1$;
for $i := 1$ **to** n **do**
  if $sensor\_based(t_i) = TRUE$ **then**
    $\alpha := \alpha + \{(p_m, t_i)\}$;

$\beta := \beta + \{(t_i, p_m)\}$

end { if }

end { for };

Based on this theorem, the complete Level 2 Petri net is generated, and the result is shown in Figure 4.9 for the example. In Figure 4.9, *C2* is a place with loop connections to two fine-motion transitions, and each *PLAN* place(with a token) forms a loop connection with the corresponding transition in the original net. We conclude that the net decomposed is live, 1-bounded, safe, and reversible.

4.5.3 Independence of Plans and Sensors

In the discussion above, the addition of resource places for plans and sensors assured that planning and sensing could occur independently. In practice, a sensing operation may be required to acquire the state of objects before a plan can be executed. The augmentation of the Petri net to represent such dependence is more dependent on specific configurations and devices. The following algorithm defines a decomposition for an assembly task which requires a sensing operation for each motion, and a constraint that sensing for the fine-motion requires a free robot hand. (This situation occurs when a sensor is attached to the hand, and only functions when the hand is empty.) The following algorithm applies to the Petri net in Figure 4.10.

Decomposition Algorithm 4: Decomposition for Plan-Sensor Dependence.

if $C_2 \notin P$ **then**

$P := P + \{C2\}$; $m := m + 1$; $p_m = C2$; $\mu_m := 1$

end {if};

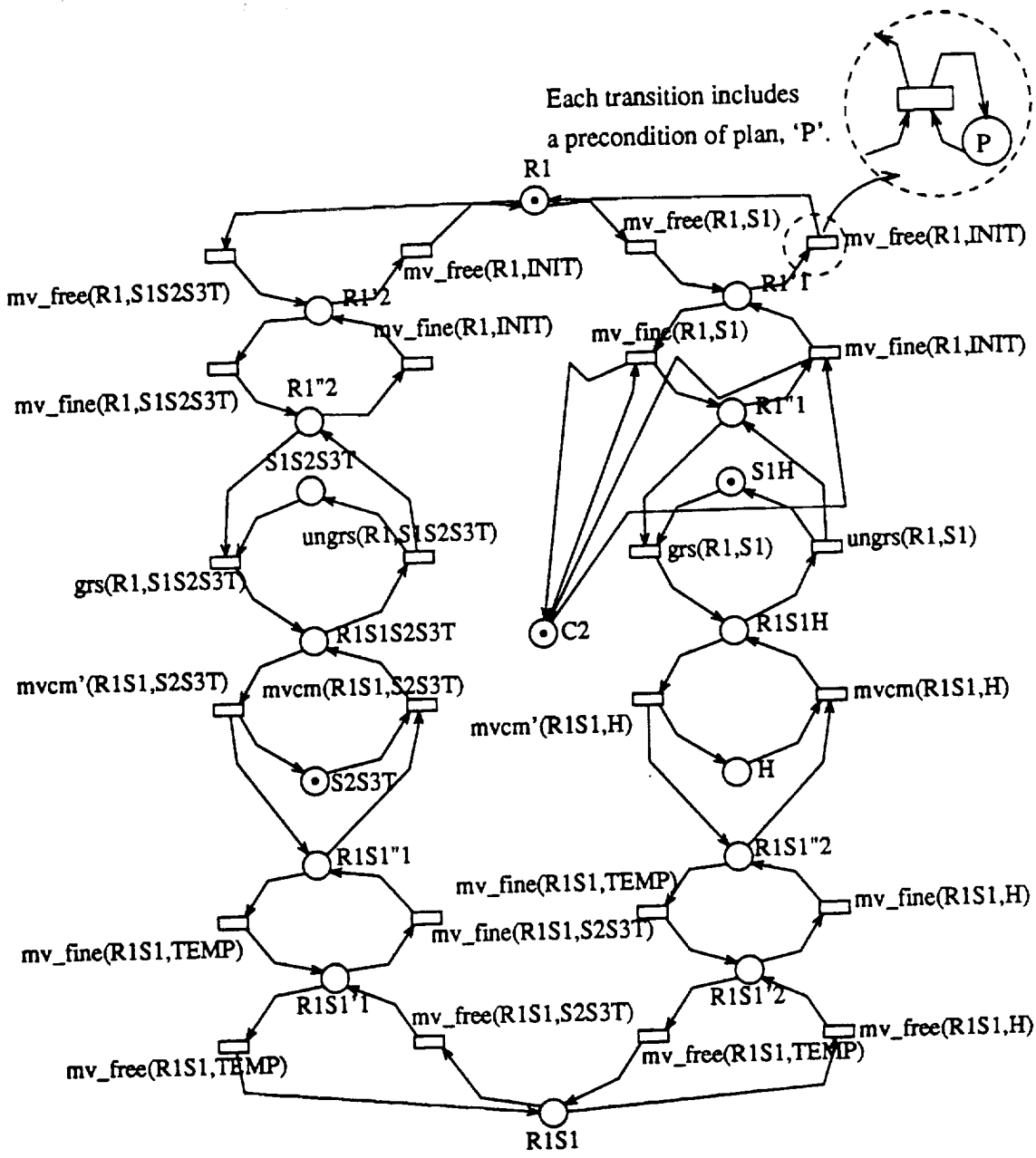


Figure 4.9: Level 2 Petri net, *PN2*, with a resource place, C2, introduced to model camera availability. In addition, each move and grasp operation has a resource place, P, as a precondition.

if $C_1 \notin P$ **then**

$P := P + \{C1\}$; $m := m + 1$; $p_m = C1$; $\mu_m := 1$

end {if};

for $j := 1$ **to** n **do**

if $t_j = t_i^c$, $\{(p''_{i_1}, t_i^c), (p_{i_2}, t_i^c), (p_{i_1}, t_i^{r1}), (p'_{i_1}, t_i^{r2})\} \subseteq \alpha$, $\{(t_i^{r1}, p'_{i_1}), (t_i^{r2}, p''_{i_1})\} \subseteq \beta$ **then**

$n := n + 6$; $t_{n-5} := plan\_path$; $t_{n-4} := plan\_path$; $t_{n-3} := plan\_path$;

$t_{n-2} := find\_view\_pos$; $t_{n-1} := find\_view\_pos$; $t_n := mv\_free(R, p_{i_2})$;

$T := T + \{t_n, t_{n-1}, t_{n-2}, t_{n-3}, t_{n-4}, t_{n-5}\}$;

$m := m + 4$; $p_{m-3} := VIEW\_POS1$; $p_{m-2} := VIEW\_POS2$;

$p_{m-1} := R'$; $p_m := FREE\_PLAN(R)$; $P := P + \{p_m, p_{m-1}, p_{m-2}, p_{m-3}\}$;

$\mu(PLAN(t_i^{r1})) := 0$; $\mu(PLAN(t_i^{r2})) := 0$

$\alpha := \alpha + \{(p_{i_1}, t_{n-5}), (p_{m-3}, t_{n-5}), (C1, t_{n-2}), (p_{i_2}, t_{n-2}), (p_{m-3}, t_{n-3})\}$;

$\alpha := \alpha + \{(p'_{i_1}, t_{n-4}), (p_{m-2}, t_{n-4}), (C2, t_{n-1}), (p_{i_2}, t_{n-1}), (p_{m-1}, t_{n-1})\}$;

$\alpha := \alpha + \{(R, t_n), (p_m, t_n), (R, t_{n-3})\}$;

$\beta := \beta - \{(t_i^{r1}, PLAN(t_i^{r1})), (t_i^{r2}, PLAN(t_i^{r2}))\}$;

$\beta := \beta + \{(t_{n-5}, p_{i_1}), (t_{n-5}, p_{m-3}), (t_{n-5}, PLAN(t_i^{r1})), (t_{n-2}, p_{m-3}), (t_{n-2}, C1)\}$;

$\beta := \beta + \{(t_{n-2}, p_{i_2}), (t_{n-3}, p_{m-3}), (t_{n-4}, p'_{i_1}), (t_{n-4}, PLAN(t_i^{r2})), (t_{n-4}, p_{m-2})\}$;

$\beta := \beta + \{(t_{n-1}, p_{m-2}), (t_{n-1}, C2), (t_{n-1}, p_{i_2}), (t_{n-1}, R), (t_n, p_{m-1}), (t_{n-3}, R), (t_{n-3}, p_m)\}$

end { if }

end { for }

end.

Applying this to our example, we now assume that to obtain the free-motion plan for A, camera 1 is required to find the view position of B and then the path

planning procedure is called to generate a trajectory to move to an approach position. To obtain the fine-motion plan, we must first clear the robot gripper on which the camera is mounted, so that it can freely move to B and use camera 2 which is mounted on the robot arm to detect the precise position of B. Using the resulting view position of B, we generate a free-motion plan for the robot to move near B and view the position of B. Then a fine-motion plan is generated and a fine-motion path can be followed to mate B by A.

If A is an object containing R, a conflict from the resource, R, occurs and then a *disassembly* operation must be performed to the free R from A. After the sensing operation by the camera on R is finished, A must be reassembled again. The Petri net representation in Figure 4.10 shows the conflicts of resources as well as the precedence relationships among the generation of plans and motions. This subnet can be merged with the Level 2 Petri net in Figure 4.9 to obtain a final Level 2 Petri net shown in Figure 4.11. The decomposition for this example is shown for the $R1S1$, $S2S3T$ mating operation.

Figure 4.11 shows the resulting final Petri net. In one possible sequence, the supervising planner could go all the way down to the place $R1S1$ and fire the *plan_path* transition to get a free-motion plan. However, the precise position of $S2S3T$ is not known because of the non-rigid combination of the table and subassembly $S2S3$, and $mv\_free(R, S2S3T)$ cannot be fired because the robot is not free. The planner could generate a sequence to return, get the free robot, and fire $find\_VPos$. After a token has been put into place $VPos$, we store this state until a token is put into $R1S1'1$ to generate the fine-motion plan. At this time, the robot is free. We then go all the way down to the place $R1S1'1$ again and make the transition $mv\_fine(R1S1, S2S3T)$ enabled.

A preferred plan sequences the state identification first, prior to beginning assembly. This plan can be found from the sequences generated by the the search

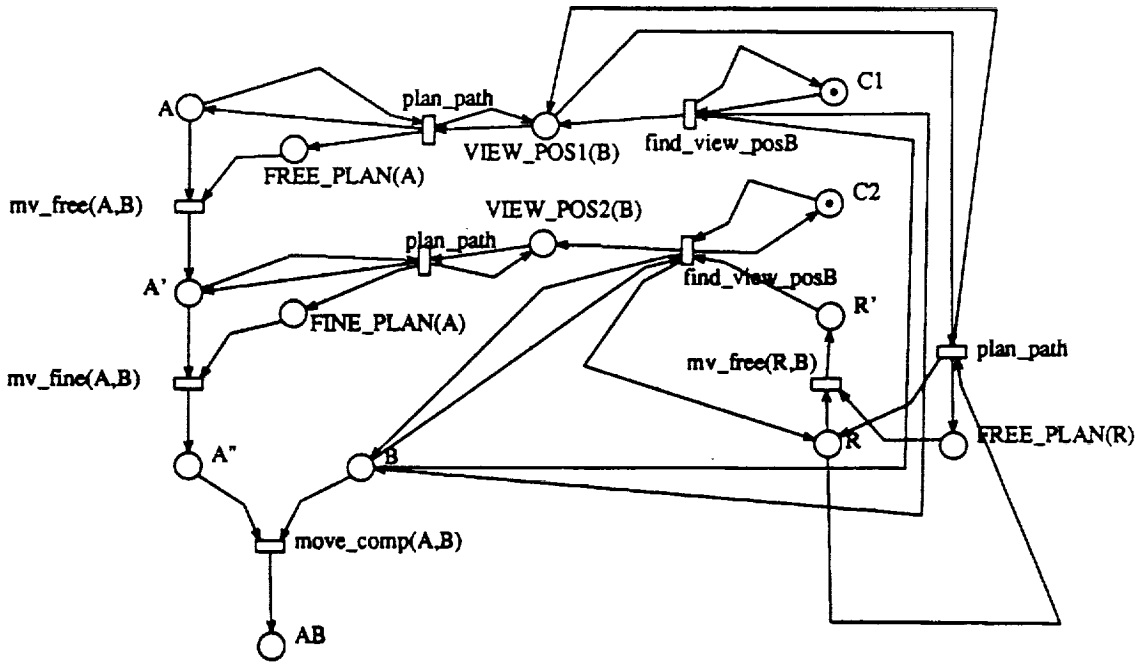


Figure 4.10: Decomposition for plan-sensor dependence.

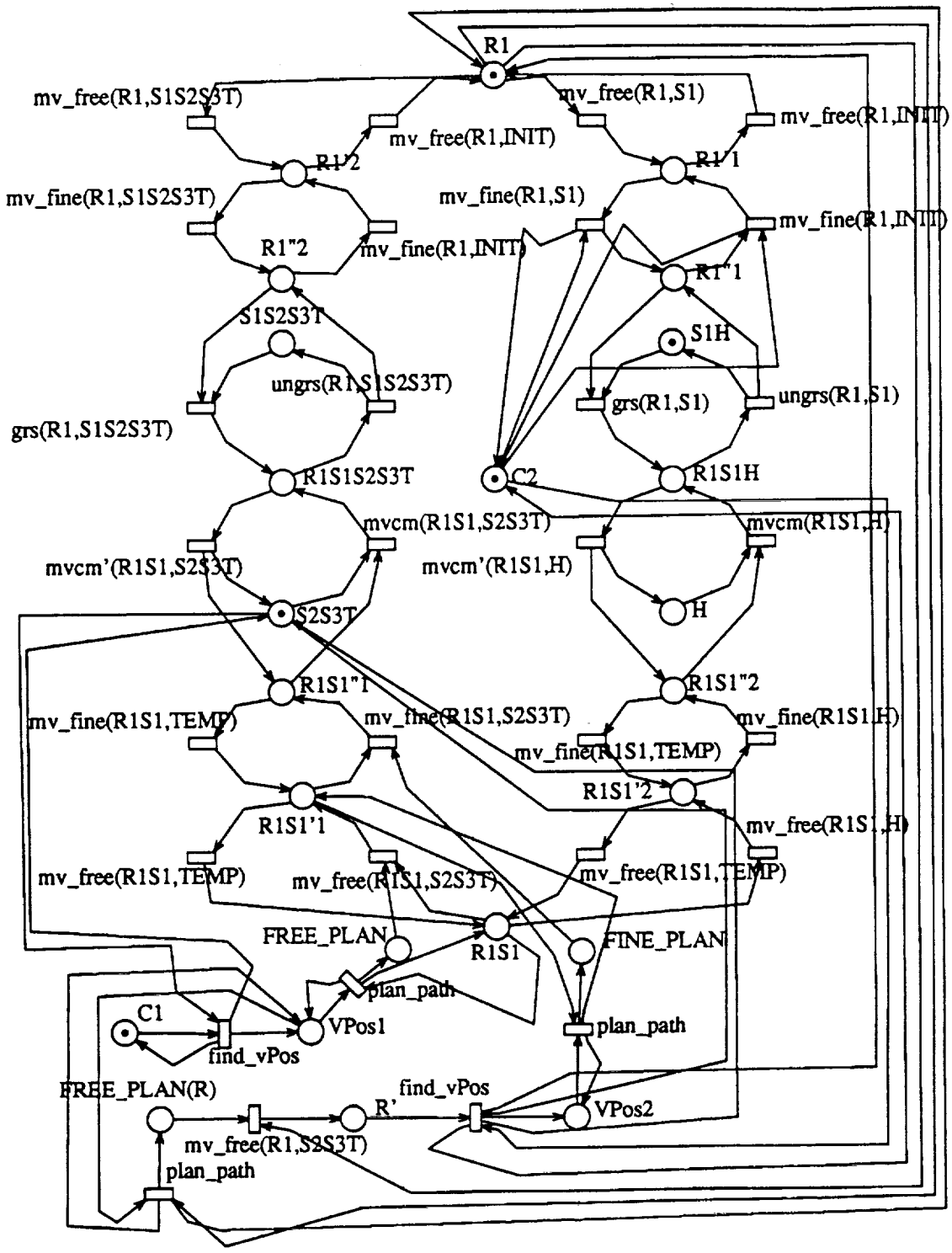


Figure 4.11: The final Petri net.

algorithm discussed before.

We can find the view position of $S2S3T$ at the very beginning and store this position, at the same time, we can generate the free motion plan for the free robot, and then use the arm camera to find the precise position and store it. In this way, when $R1S1$ state is reached, we can directly generate both free-motion and fine-motion plan without the error recovery loop. Execution of the long loop will be avoided.

It is quite complex to verify the properties of the final net. However, if we *loosen* the definitions of reversibility and add stronger constraints on searching feasible sequences as described below, we still obtain the properties of liveness, safeness, and reversibility for the final net through the decomposition theorems described above.

First, the final net is live. As we see from Figure 4.10, if we have a token in A, B, and R(if there is no token in R, we can follow an additional sequence to obtain a token in R), from the results of simulation on GSPN, we could ultimately get a token in AB. For each $PLAN$ decomposition in the net, we can obtain a similar result. Therefore, the net is live. Secondly, when we search a new system state, any place obtaining more than one token will be considered as containing one token. Therefore, no state will have a marking in any place containing a value other than 0 or 1. Any sequence of transitions will not destroy the property of 1-boundedness and safeness. Thirdly, to investigate the property of reversibility, if we ignore the tokens in the places only for planning such as $VIEW\_POS1(B)$ and $VIEW\_POS2(B)$, we can consider the net as a reversible net, because once $VIEW\_POS1(B)$ or $VIEW\_POS2(B)$ gets a token, it will not lose it. However, as we showed in the proof of liveness above, if A, B, and R have a token initially(Figure 4.10), a token will be finally obtained by AB, and R will retain its token. The case is similar for other transitions. The net is thus reversible, if we assume the values in

some places of the marking are *don't care* conditions for the property of reversibility.

4.6 Simulation Results and Discussions

Successive decomposition of the Level 0 Petri net using the algorithms discussed above yields the *PN2* shown in Figure 4.11. The search algorithm may be used on the Petri net at each level to identify feasible sequences. All feasible sequences may be obtained for each level and the relationships between the sequences at different levels can be studied. At the lowest Level 2, there are greater conflicts and constraints among the shared resources. Normally, we cannot expect a shortest lower level sequence to be obtained via the decomposition of the shortest sequence generated from the higher level representation.

We list the practical feasible sequences generated at each level of the Petri net as follows(we assume no transition is fired more than once):

Level 0. Petri net in Figure 4.3:

The number of feasible states: 5.

The number of feasible sequences: 1.

Feasible sequence: t1 t3 t6 t8.

Level 1. Petri net in Figure 4.5:

The number of feasible states: 10.

The number of feasible sequences: 1.

Feasible sequence:

Move(R1, S1);

Grasp(R1, S1);

Move_Comp(R1S1, H);

Move(R1S1, Temp_Pos(R1S1));

Move(R1S1, S2S3T);

Move_Comp(R1S1, S2S3T);

UnGrasp(R1, S1S2S3T);

Move(R1, Init_Pos(R1)).

Level 2. Petri net in Figure 4.6(Motion decomposition):

The number of feasible states: 15.

The number of feasible sequences: 1.

Feasible sequence:

Move_Free(R1, App_Pos(H));

Move_Fine(R1, Grasp_Pos(S1));

Grasp(R1, S1);

Move_Comp(R1S1, H);

Move_Fine(R1S1, Temp_Pos(R1S1));

Move_Free(R1S1, Temp_Pos(R1S1));

Move_Free(R1S1, App_Pos(S2S3T));

Move_Fine(R1S1, Ins_Pos(S2S3T));

Move_Comp(R1S1, S2S3T);

UnGrasp(R1, S1S2S3T);

Move_Fine(R1, Init_Pos(R1));

Move_Free(R1, Init_Pos(R1)).

Level 2. Petri net in Figure 4.9(Adding C2 and PLANs):

The number of feasible states: 15.

The number of feasible sequences: 1.

Feasible sequence: the same as that in the above decomposition.

Level 2. Petri net in Figure 4.11(Generating PLANs):

The number of feasible states: 142.

The number of feasible sequences: 1.

Feasible sequence:

Find_Pose(S2S3T, C1, Pos(S2S3T));

Plan_Path(R1, View_Pos(N));

Move_Free(R1, View_Pos(N));

Find_Pose(N, C2, Pos(N));

Move_Free(R1, App_Pos(H));

Move_Fine(R1, Grasp_pos(S1));

Grasp(R1, S1);

Move_Comp(R1S1, H);

Move_Fine(R1S1, Temp_Pos(R1S1));

Move_Free(R1S1, Temp_Pos(R1S1));

Plan_Path(R1S1, App_Pos(S2S3T));

Move_Free(R1S1, App_Pos(S2S3T));

Plan_Path(R1S1, Ins_Pos(S2S3T));

Move_Fine(R1S1, Ins_Pos(S2S3T));

Move_Comp(R1S1, S2S3T);

UnGrasp(R1, S1S2S3T);

Move_Fine(R1, Init_Pos(R1));

Move_Free(R1, Init_Pos(R1)).

For most applications, more than one feasible sequence may be generated and an evaluation and selection strategy is used to choose among them. Normally, searching sequences from higher level representations may be performed to verify the correctness of decomposition and the final net is searched to generate a final task sequence.

The assumption of firing any transition at most once is useful to constrain the creation of the shortest sequences. If, under this assumption, we cannot reach the final state, a looser assumption of firing each transition at most twice could be introduced. Alternatively, the shortest sequence which has the least conflict on resources and the least probability of firing any error recovery subsequence could be

used.

As can be seen from the simulation results, the shortest sequence from the lowest level of decomposition of the representation of task sequences could not be obtained via the decomposition of preferred higher level sequences directly. An intuitive observation of this fact is shown where a sensor is used on the active component, the robot. In this case, a resource conflict appears. For example, when a robot is required to pick up a strut on a table and transfer it to a holder, the robot first moves to and near the strut on the table. The grasp planner for the robot might require the exact position of the strut, and if a camera is mounted on the robot arm, it could directly perform sensing operations and no error recovery procedure is involved. If we use the decomposition on the representation rather than on the sequences, a shortest sequence will be guaranteed to be found on the final stage decomposition and the conflicts on resources will be automatically reduced to a minimum.

Another issue we may want to consider is whether our representation includes the possibility of parallel operations or concurrency. Two types of concurrency, parallel and sequential mutual exclusions for Petri net modeling of manufacturing systems with shared resources are discussed in [124]. In Figure 4.11, we can find: (1) the transitions, *find_vPos*(connected with *C1*) and *mv_free(R1,S1)* are two parallel operations without resource conflict; (2) after *find_vPos*(same as above) is fired, *plan_path*(connected with *FREE_PLAN(R)*) and *mv_free(R1,S1)* are two parallel mutually exclusive operations with shared resource *R_1*; (3) if there are two tokens in *S2S3T* and *S1H* with different colors, sequential mutual exclusions with shared resource *R1* appear. In this case, after *R1* moves to one *S2S3T* and uses its camera to sense the insertion position of the subassembly *S2S3*, two choices for the following execution should be selected. The first choice may be continuing to grasp an *S1* on a holder *H* and complete the assembly of *S1S2S3*. Another choice may be to sense

another *S2S3* on the second table. The resource conflict for *R1* also appears at this time. Notice in this discussion the property of 1-boundedness is generalized to k-boundedness, where $k = 2$.

4.7 Conclusions

In the above simulations, it is assumed that no transition will be fired more than one time. In this example, the number of sequences generated is 1 for all the decomposition levels. This is because most transitions in the Petri net for this example are essential. The probability of selecting alternative partial sequences or transitions is therefore very small. If we suppose some transitions can be fired two or more times, the simulations show that the number of feasible sequences will be dramatically increased and it will be very costly to store all sequences and compare different sequences among them. Under any condition, the sequence found for the final net for the example is the shortest and thus most economical to implement.

The verification of property inheritance during decompositions for assembly tasks has been developed. We have shown that the Petri net mapped from the AND/OR net guarantees safeness, 1-boundedness, liveness, and reversibility. Based on these conditions, the properties of all lower levels of net representations are discussed. Because those properties are not lost during specific decompositions, we can guarantee a deadlock-free and fault tolerant system.

An important topic for continued research is how to make the property of reversibility on the final net stronger. If all places in the final net are taken into account, can the initial state be reached when an error occurs? A related topic is to guarantee state reservations for all waiting components or component groups during recovery.

CHAPTER 5
REPRESENTATION AND ANALYSIS OF UNCERTAINTY USING FUZZY PETRI NETS

This chapter proposes a generalized definition of the *fuzzy Petri net(FPN)* and the reasoning structures of transitions in the FPN. Three types of fuzzy variables: local fuzzy variables, fuzzy marking variables, and global fuzzy variables, are used to model uncertainty based on different aspects of fuzzy information. A fuzzy Petri net is used to model the incomplete, uncertain, and approximate information associated with firing of transitions and changing of states in robotics and manufacturing systems. Using FPNs to model a system, a fuzzy reasoning strategy may be used to infer new fuzzy values in output places after the corresponding enabled transition is fired. A global fuzzy variable is used to sequence operations with key precedence relations for a manufacturing system. A local fuzzy variable is used to represent the uncertainty in local configuration variables of the system and may be used to control on-line reasoning about sensor-based execution. Several basic types of fuzzy Petri nets are analyzed, and the necessary and/or sufficient conditions of safeness, liveness, and reversibility are given. An example of modeling sensory transitions in a robotic system is discussed to illustrate reasoning about input local fuzzy variables to obtain mutually exclusive tokens in the output places.

5.1 Introduction

While Petri nets[84, 89] have been widely used to model computer systems[79, 81, 93], manufacturing systems[2, 116, 125], robotic systems[10, 11, 12, 13, 14, 15], knowledge-based systems[8, 52], and other kinds of engineering applications, they may be unable to model incomplete, uncertain, and approximate information or states. An operation and its preconditions and postconditions in a manufacturing

system can be represented by a transition and its input places and output places in a Petri net model. Each token in some place of an ordinary Petri net is used to represent an entity, such as an object or an abstract piece of information. However, a real system may contain objects which require associated variables to fully represent an object state. In addition, the values of those associated variables and the occurrence of the event itself may be uncertain. In such a system the decisions which choose from enabled transitions as well as the generation of a next-step state are based on these approximate descriptions of objects.

Because of the necessity of modeling and representation of lower level operations and objects in a robot system[15, 19], ordinary Petri nets are found not sufficient to represent uncertainty and approximate information. The uncertainty in a robotic system may occur due to many factors during the execution of a planned sequence or program. When an operation such as 'the gripper A grasps the object B', is modeled by a transition t_i in a Petri net N, one kind of uncertainty within this grasping operation is the geometric uncertainty in the coordinate of the grasp position or the contacting position of the gripper with the object. Because this position is important for the succeeding operations of the robot gripper such as motion, force control, and assembly, we need to represent uncertain information in the output place which shows the result of the robotic grasp operation. Another kind of uncertainty within this operation is the uncertainty of the success of this operation, and the uncertainty of degree completion for the whole task of the robotic system, such as assembling a complete set of objects in a certain configuration or reaching a final system state.

Based on the above discussions, a fuzzy Petri net may be used to describe the operations and conditions with uncertainty. Uncertain states are associated with objects, and transitions are used to model fuzzy operations such that the input variables of transitions will be reasoned approximately and efficiently rather than

precisely. This approximation or uncertainty is propagated along the net so that a predefined transition sequence may be followed as desired, or, if errors accumulate, the operations sequence should be stopped so that the error can be detected by the system monitor and the correct recovery sequence may be followed to recover to a correct system state. Based on extensions to the theory of ordinary Petri nets, a definition of the fuzzy Petri net and the associated embedded reasoning structure was presented in [17, 18, 22]. This definition was used to model a sensor-based robotic system which incorporates reactive, uncertain, and dynamic properties.

One example of the utility of using fuzzy concepts to deal with information is found in knowledge based systems with uncertainty. When this type of system is modeled by Petri nets, the assumption of token values in the net to be 0 or 1, is not sufficient to describe the reasoning process or to represent the degree of uncertainty of facts. An example for robotic systems is based on our observations that the properties of some objects in an assembly or material handling system may be changed[12, 14], so that the same place may contain different kinds of tokens during different processing times. The properties of objects may be physical or geometric characteristics which are parameterized to define the objects. In this case, it is hard to use the same crisp value to represent the tokens in different processing states.

Since its emergence in 1965, fuzzy sets[119] have been applied in many aspects of engineering systems, decision systems, medical systems, industry, transportation, and other applications. Numerous papers have been published in all aspects of the theory of fuzzy sets, fuzzy mathematics, and their applications. Our reasoning strategy in fuzzy Petri nets is based on the the theory of fuzzy logic, and the firing of transitions is equivalent to the operations on *membership functions* in a certain universe of discourse. A membership function μ_A for a fuzzy set A is defined by μ_A: $X \rightarrow [0, 1]$, where a distribution of membership grades for each element in a universe of discourse is given. A *fuzzy singleton* is defined as a membership function

in a universe of discourse, of which the membership grades of all elements are 0s, except one and only one element, x, of which the membership grade is 1. We mark the fuzzy value of this membership function as x.

In this chapter, we propose a generalized definition of a fuzzy Petri net and a complete reasoning structure associated with a transition in the net. This definition can be used in many applications of FPNs such as sequencing, planning, reasoning about uncertainty, process control, and knowledge inference. Two kinds of important planning strategies, off-line sequencing for task sequences, and on-line sensing and reactively reasoning about firing sequences, are shown to be effectively solvable by FPNs. We use local fuzzy variables to model the information which locally affects an operation, and we use fuzzy marking variables to represent the state of the system. The main differences between an ordinary Petri net and a fuzzy Petri net are the fuzzy values associated with places and tokens, and the reasoning rules which govern the firing of transitions. For an ordinary Petri net, a transition is fired if all input places contain at least one token. For a fuzzy Petri net, the condition of firing is also based on the local fuzzy variables associated with input places. Using a fuzzy Petri net to represent a robotic or manufacturing system, we are able to handle approximate information or uncertainty in the system. The reasoning about this information is incorporated into the firing rules of transitions.

In the definition of a fuzzy Petri net, local fuzzy variables, fuzzy marking variables, and global fuzzy variables were defined as different fuzzy information carried through the net. In [12, 20], global fuzzy variables were used to model the degree of completion of the robotic task so that an operations sequence could be planned off-line while searching in the fuzzy Petri net model. Compared with the strategy used in ordinary Petri nets, computational time and space are saved because when the order of *key transitions* are given, all planned sequences should imply this order. Therefore, a correct sequence is defined as a feasible, complete,

and correctly ordered sequence. Any sequence which fails in satisfying this definition is discarded during the off-line searching process. In [14, 21], local fuzzy variables and global fuzzy variables are used simultaneously to model sensor-based robot task sequence planning and the execution of operations involving the use of sensory data. During the execution of a robot operations sequence, sensory operations can detect the partial result of some *key operations* on-line and the accumulation of errors will cause a local error recovery sequence or a global error recovery sequence. The local fuzzy variables decide the choice of error recovery strategy.

Previous work on predicate/transition nets has described approaches to handling predicate related expressions[39, 42]. Tokens in predicate/transition nets can be structured objects carrying values, and transition firing can be controlled by imposing conditions on the token values. Predicate/transition nets have been used for the management of expert systems, analysis in database systems, and many other applications. Research on colored Petri nets[54] reports related results though enabling limited reasoning capacity. The major difference between the predicate/transition net or the colored Petri net and the fuzzy Petri net is that a fuzzy Petri net can represent more generalized data using fuzzy numbers and fuzzy reasoning functions. The results of firing on some transition will depend not only on the input values, but also on a reasoning process built in the transitions. The transition firing may depend on the local fuzzy variable or the global marking based on different applications. The strategy of property analysis on predicate/transition nets thus cannot be directly applied to the fuzzy Petri net.

Investigation of the properties of fuzzy Petri nets is very important for performance evaluation of a system being modeled. Reachability is a fundamental problem in the research on ordinary Petri nets. The reachability problem on fuzzy Petri nets is also defined on a feasible reachable set from an initial state. The reachability problem can influence other properties such as liveness, safeness, and reversibility.

Safeness of fuzzy Petri nets is defined under the assumption that no more than one copy of a single object appears in the system. Liveness of a fuzzy Petri net implies that the reasoning process or the execution process can continue when the accumulation of errors is still within a range of tolerance, and when errors go over a threshold, an alternative sequence can be chosen in place of the original sequence. Reversibility implies that at any time if errors are too large, a home state is reachable.

The discussions on fuzzy Petri nets in this chapter are arranged as follows: In Section 5.2, the fuzzy Petri net definition is introduced and some components of this definition are explained. A system state represented by fuzzy information is then discussed in Section 5.3. Section 5.4 shows reasoning rules in the FPN. Section 5.5 gives analysis for some basic cases. FPNs for sequencing and FPNs for sensing are discussed in Sections 5.6 and 5.7, respectively. In Section 5.7, sensing operations are modeled as mutually exclusive transitions in a fuzzy Petri net and therefore reasoning on sensory data can be performed in these transitions. The conclusions are given in the last section.

5.2 Fuzzy Petri Nets

The definition of the generalized fuzzy Petri net is shown as follows[17, 18]:

Definition 5.1 *A fuzzy Petri net* is formally defined as an 8-tuple:

$$FPN = (P,\ T,\ Q_t,\ \alpha,\ \beta,\ m_f,\ m_t,\ \mu_f),$$

where

1) $P = \{p_1,\ p_2,\ \ldots,\ p_n\}$ is a finite set of *places,* $n \geq 0$.

2) $T = \{t_1,\ t_2,\ \ldots,\ t_m\}$ is a finite set of *transitions,* $m \geq 0$. $P \bigcap T = \emptyset$.

3) $Q_t = \{q_1,\ q_2,\ \ldots,\ q_l\}$ is a finite set of *state tokens,* $l \geq 0$.

4) $\alpha \subseteq \{P \times T\}$ is the input function, a set of directed arcs from places to transitions. We call each p_i where $(p_i, t_j) \in \alpha$ as an *input place* of t_j.

5) $\beta \subseteq \{T \times P\}$ is the output function, a set of directed arcs from transitions to places. We call each p_i where $(t_j, p_i) \in \beta$ as an *output place* of t_j.

6) $m_f : P \rightarrow \{(\rho, \varrho)\}$ assigns p_i the value of a 2-tuple, (ρ, ϱ), where ρ represents the *local fuzzy variable* and ϱ represents the *fuzzy marking variable.*

7) $m_t : Q_t \rightarrow \{\bigcup_i(k_i, \sigma_{k_i}),\ C\}$ is a mapping from a token to a union of 2-tuples of k_i and the k_ith *global fuzzy variable,* σ_{k_i}, or, to a constant, C, which indicates no global fuzzy variable is attached to the token. σ_{k_i} is a membership function in a universe of discourse.

8) μ_f: $T \rightarrow \{f_1, f_2, \ldots, f_m\}$ is an association function, a mapping from transitions to corresponding *reasoning functions.* A reasoning function f_i maps variables associated with input places and a set of tokens to variables associated with output places and another set of tokens.

If we denote the reasoning function for a transition t_i as f_i, firing of t_i when it is enabled will map (ρ, ϱ) from all input places to all output places and assign σ to output tokens. f_i has at most three kinds of rules for mapping ρ, ϱ, and σ. They are written as r_ρ, r_ϱ, and r_σ, respectively. These rules may or may not be independent, and after firing t_i, the original (ρ, ϱ) may or may not stay in input places of t_i.

If a place p_i represents an object O_i, then the local fuzzy variable associated with it can be represented as $\rho(p_i)$; the fuzzy marking variable associated with it can be represented as $\varrho(p_i)$; and the k_ith global fuzzy variable within this place can be represented as $\sigma_{k_i}(q_j)$, where q_j occupies p_i in the current state. In this chapter, we assume $\rho(p_i)$, $\varrho(p_i)$, and $\sigma_{k_i}(q_j)$ are independent, i.e., one variable cannot be inferred from any of two others. When the fuzzy Petri net model is generated, we may assign *a priori* local fuzzy variables to all places. Thus, even though a place contains no token, the object it represents still has a local fuzzy variable. If we consider the three types of variables for a set of places, $p_{j_1}, p_{j_2}, \ldots, p_{j_\omega}$, we can write them as

$$\rho(p_{j_1}, p_{j_2}, \ldots, p_{j_\omega}) = (\rho(p_{j_1}), \rho(p_{j_2}), \ldots, \rho(p_{j_\omega})).$$

$$\varrho(p_{j_1}, p_{j_2}, \ldots, p_{j_\omega}) = (\varrho(p_{j_1}), \varrho(p_{j_2}), \ldots, \varrho(p_{j_\omega})).$$

$$\sigma_{k_i}(q_{j_1'}, q_{j_2'}, \ldots, q_{j_\omega'}) = (\sigma_{k_i}(q_{j_1'}), \sigma_{k_i}(q_{j_2'}), \ldots, \sigma_{k_i}(q_{j_\omega'})),$$

where $q_{j_1'}$, $q_{j_2'}$, ..., $q_{j_\omega'}$ occupy p_{j_1}, p_{j_2}, ..., p_{j_ω}, respectively.

Pictorially, each place p_i is represented by a circle with "p_i" and a label which indicates the object p_i represents attached to it, and each transition t_j is represented by a rectangle with "t_j" and "f_j" attached to it. If $(p_i, t_j) \in \alpha$, there is a directed arc from p_i to t_j. If $(t_j, p_i) \in \beta$, there is a directed arc from t_j to p_i.

The interpretation of a 'token' in a place depends on $m_f(p_i) = (\rho(p_i), \varrho(p_i))$. From the above definition, three different types of variables are operated on or carried along through the net. The three types of fuzzy variables have different interpretations:

A *local fuzzy variable* is attached to a place. Its value indicates the uncertainty of the local variable or object which is attached to the place. It is represented by an n dimensional membership distribution function on the assumption that the object is n dimensional. One example is to define $\rho(R)$ for the robot R. Because R has 6 degrees of freedom(3 for position and 3 for orientation), $\rho(R)$ is a 6 dimensional membership function. The membership grades are defined for each possible position and orientation of the gripper in a given universe of discourse.

A *fuzzy marking variable* is attached to a place. It is a 1 dimensional membership distribution function denoting the uncertainty that a token exists in a given place. The universe of discourse for ϱ is the occurrence of the event denoted by the place. In one example, the place may indicate the event that the robot has grasped an object. The fuzzy marking variable indicates the uncertainty that the event has occurred, i.e., whether the robot is actually holding the object.

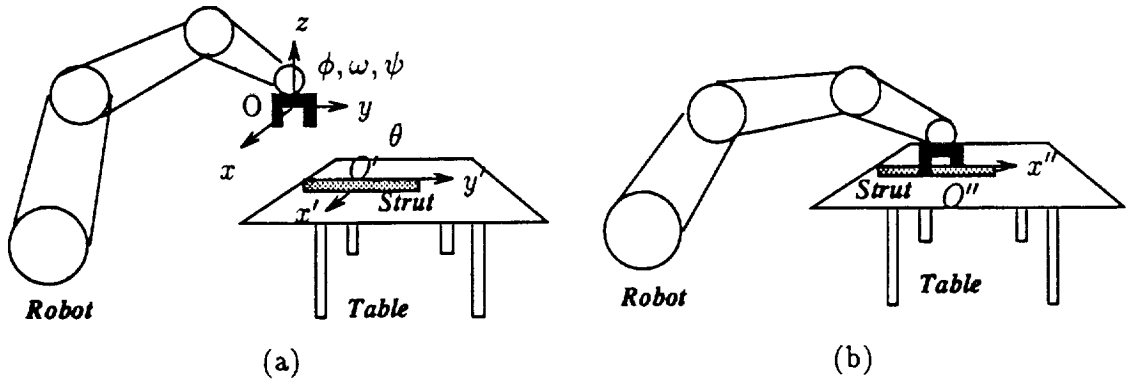


Figure 5.1: A robotic system for a grasp task. The robot(R) moves to the strut(S) on the table and grasps it. (a) shows the initial state for this task. R has 6 degrees of freedom for the position and orientation of the gripper, $(x, y, z, \phi, \omega, \psi)$. S is defined by the position of its center and the angle between S and the x' axis, (x', y', θ). (b) shows the final state of this task. RS is described by x'', the distance between the grasping point and O'', the center point of S, under the assumption that the grasp position will be on the strut.

A *global fuzzy variable* is attached to a token. This is an m dimensional membership distribution function related to a characteristic variable of a global task. The global fuzzy variable may be used to sequence a set of transitions so that a global task can be completed. An example is the use of a global fuzzy variable to represent 'degree of completion' of a task. The fuzzy value should increase during the execution to indicate correct sequencing of operations.

Figure 5.1 shows an example of a robot assembly task which illustrates these three types of fuzzy variables. This robotic system consists of a robot(R) and a strut(T) on a table. The robot gripper(R) moves to and grasps the strut(S) on the table, and this grasping state is described as RS. Figure 5.1(a) and Figure 5.1(b) illustrate the system state before and after the 'grasp' operation.

This assembly task can be represented by a fuzzy Petri net shown in Figure 5.2. p_1, p_2, p_3 and p_4 represent the configurations R_o, S, R_cS, and R_c, respectively. In this representation, R_o and R_c means the robot gripper is open or closed, respectively.

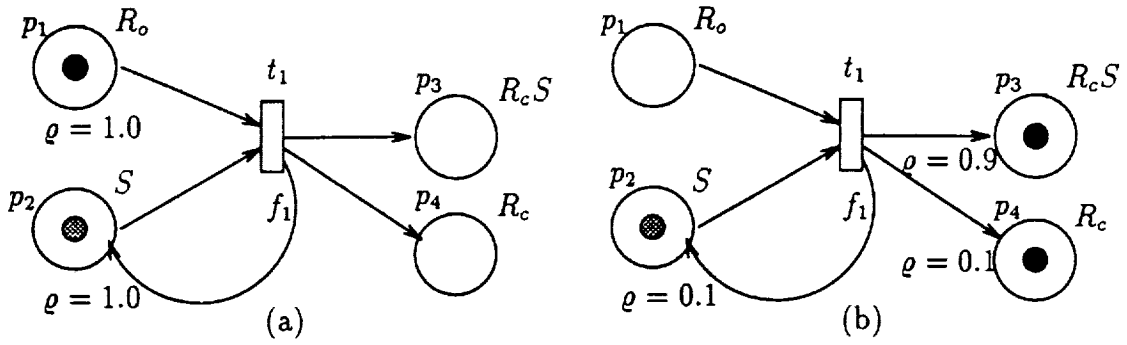


Figure 5.2: The fuzzy Petri net representation for the robotic assembly task shown in Figure 5.1. (a) shows the initial state of the system. (b) shows the final state of the system. R_o and R_c means the robot gripper is open or closed, respectively. ρ, ϱ, and σ of R_o and S in (a) are mapped by f_1 to those of R_cS, R_c, and S in (b). Note that the same color of tokens in p_1 in (a) and in p_3 and p_4 in (b) indicates that a global fuzzy variable is attached to these tokens. Their colors are different from that of the token in p_2 in (a) and (b). The fuzzy marking variable ϱ defines alternative output states R_cS or R_c, S. The global fuzzy values might be $\sigma(R_o) = 0$, $\sigma(R_cS) = 1$, $\sigma(R_c) = 0$. The local fuzzy variable ρ describes the positional uncertainties of the robot and object in terms of their fuzzy membership functions.

$\rho(R_o)$ and $\rho(R_c)$ are 6-D membership functions representing the uncertainty in the robot position. $\rho(S)$ is a 3-D membership function representing the uncertainty in the position of the strut on the table. $\rho(R_cS)$ is a 1-D membership function with parameter x'' which represents the uncertainty in the grasp point along the strut. We assume $\varrho(p_i)$ for this example is a fuzzy singleton, which describes the uncertainty of event completion. For example, in Figure 5.1(a), $\varrho(R_o, S, R_cS, R_c) = (1, 1, 0, 0)$, and in Figure 5.1(b), $\varrho(R_o, S, R_cS, R_c) = (0, 0.1, 0.9, 0.1)$. If the token which represents an entity containing R carries a global fuzzy variable indicating the degree of completion of the task and under the assumption of $m_t(q_j)$ as a fuzzy singleton, then in Figure 5.2(a), $\sigma(R_o) = 0$, and in Figure 5.2(b), $\sigma(R_cS) = 1$, $\sigma(R_c) = 0$.

As with other Petri net models, we would like to use the fuzzy Petri net to analyze properties such as reachability, liveness, safeness, and reversibility. We

define the reachable set of a fuzzy Petri net as the set of markings(consisting of fuzzy marking variables) reachable from the initial marking after all feasible transition sequences are fired. While it is complicated to analyze the properties of a fuzzy Petri net with all three types of fuzzy variables, some basic cases can be treated to yield useful properties. For an ordinary Petri net, all useful properties are defined based upon the markings on the net. The fuzzy Petri net model adds the complexity of local and global variables which must be considered. In this analysis, we will consider first the problem where only local fuzzy variables exist in the fuzzy Petri net model.

There may be three different cases according to the above assumptions. First, firing rules follow from the ordinary Petri net, and local fuzzy variables are unchanged. Second, firing rules follow from the ordinary Petri net, and local fuzzy variables are changed. Third, firing rules are conditional upon input variables, and local fuzzy variables are changed. The analysis for these three cases will be discussed in Section 5.5.

5.3 State Representation of an FPN Model

Because there are three different kinds of variables associated with places and tokens in the FPN, three different system states are defined and given below. In these definitions, places are considered as parameters of system states, and from a place, we can find the corresponding attached local and global variables.

Definition 5.2 *Local fuzzy state* S_l: $S_l(p_1, p_2, \ldots, p_n) = (\rho(p_1), \rho(p_2), \ldots, \rho(p_n))$, an n-tuple of local fuzzy variables in the FPN. $\rho(p_i)$ is either a membership function in a universe of discourse, if this information is available, or, e, if this information is not available.

Definition 5.3 *Fuzzy marking state* S_m:$S_m(p_1, p_2, \ldots, p_n) = (\varrho(p_1), \varrho(p_2), \ldots, \varrho(p_n))$, an n-tuple of fuzzy marking variables in the FPN. $\varrho(p_i)$ is a membership function

in a universe of discourse.

Definition 5.4 *Global fuzzy state* S_g: $S_g(p_1, p_2, \ldots, p_n) = (\sigma(p_1), \sigma(p_2), \ldots, \sigma(p_n))$, an n-tuple of global fuzzy variables in the FPN. $\sigma(p_i)$ is either a value of a token if this token is existing, $m_t(q_j)$, or, e, indicating that no such information is available.

For the system state shown in Figure 5.2(a), $S_l(R_o, S, R_cS, R_c) = (\rho(R_o), \rho(S), e, e)$; $S_m(R_o, S, R_cS, R_c) = (1, 1, 0, 0)$; and $S_g(R_o, S, R_cS, R_c) = (0, C, e, e)$. For the system state shown in Figure 5.2(b), $S_l(R_o, S, R_cS, R_c) = (e, \rho(S), \rho(R_cS), \rho(R_c))$; $S_m(R_o, S, R_cS, R_c) = (0, 0.1, 0.9, 0.1)$; and $S_g(R_o, S, R_cS, R_c) = (e, C, 1, 0)$.

5.4 Reasoning Rules in the FPN

Three types of reasoning rules for f_i of a transition t_i in the FPN model are represented as $r_\rho^{t_i}$, $r_\varrho^{t_i}$, and $r_\sigma^{t_i}$. Specifically, we assume t_i has k input places which correspond to k variables, p_{i_1}, p_{i_2}, ..., p_{i_k}, at time N. And, t_i has t output places which correspond to t variables, $p_{i'_1}$, $p_{i'_2}$, ..., $p_{i'_t}$, at time $N+1$. Therefore,

$$r_\rho^{t_i}(\rho(p_{i_1}^{(N)}), \rho(p_{i_2}^{(N)}), \ldots, \rho(p_{i_k}^{(N)})) = (\rho(p_{i'_1}^{(N+1)}), \rho(p_{i'_2}^{(N+1)}), \ldots, \rho(p_{i'_t}^{(N+1)})). \quad (5.1)$$

$$r_\varrho^{t_i}(\varrho(p_{i_1}^{(N)}), \varrho(p_{i_2}^{(N)}), \ldots, \varrho(p_{i_k}^{(N)})) = (\varrho(p_{i'_1}^{(N+1)}), \varrho(p_{i'_2}^{(N+1)}), \ldots, \varrho(p_{i'_t}^{(N+1)})). \quad (5.2)$$

$$r_\sigma^{t_i}(\sigma(p_{i_1}^{(N)}), \sigma(p_{i_2}^{(N)}), \ldots, \sigma(p_{i_k}^{(N)})) = (\sigma(p_{i'_1}^{(N+1)}), \sigma(p_{i'_2}^{(N+1)}), \ldots, \sigma(p_{i'_t}^{(N+1)})). \quad (5.3)$$

In using r_ρ during firing of t_i, the values of ρ contained in all input places are evaluated and t_i generates the values of ρ for all output places. The other two kinds of reasoning rules are used in a similar way. The selection of reasoning strategies for rules, such as using a rule base or functional relationships, is based on different applications. The choice of whether to destroy the original values may also depend on the specific application. In principle, these rules may be further generalized to model interactions among the three types of fuzzy variables. For purposes of analysis here, we will not consider those cases.

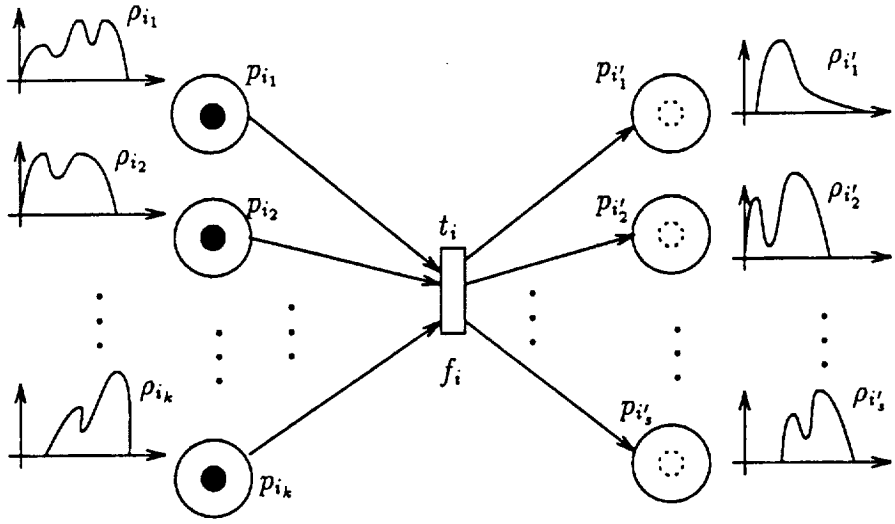


Figure 5.3: The input local variables and output local variables for a transition t_i of Case 1. Before t_i is fired, p_{i_u} contains a token, $1 \leq u \leq k$. After t_i is fired, $p_{i'_v}$ obtains a token, $1 \leq v \leq s$. All local fuzzy variables for this case are fixed.

In the FPN shown in Figure 5.2, the state changes are implemented by t_1 which consists of $r_\rho^{t_1}$, $r_\varrho^{t_1}$, and $r_\sigma^{t_1}$. These rules can be shown as: $r_\rho^{t_1}(\rho(R_o^{(N)}), \rho(S^{(N)})) = (\rho(S^{(N+1)}), \rho(R_c S^{(N+1)}), \rho(R_c^{(N+1)}))$; $r_\varrho^{t_1}(1,1) = (0.1, 0.9, 0.1)$; and $r_\sigma^{t_1}(0, C) = (C, 1, 0)$.

5.5 Property Analysis for Several Basic Cases with Local Fuzzy Variables

Figure 5.3 shows a transition t_i in the net which has k input places corresponding to k variables, $p_{i_1}, p_{i_2}, \ldots, p_{i_k}$. t_i has s output places which correspond to s variables, $p_{i'_1}, p_{i'_2}, \ldots, p_{i'_s}$. The corresponding local fuzzy variables are represented as $\rho_{i_1}, \rho_{i_2}, \ldots, \rho_{i_k}$ and $\rho_{i'_1}, \rho_{i'_2}, \ldots, \rho_{i'_s}$. For the sake of simplicity, we draw one-dimensional membership functions to represent all these variables. The membership functions for $\rho_{i'_v} (1 \leq v \leq s)$ prior to marking, may be thought of either as the function which existed from the previous marking, or as an *a priori* function assigned to the place. Transition t_i has a reasoning function f_i. Two types of reasoning rules

for f_i, $r_\rho^{t_i}$ and $r_\varrho^{t_i}$, are used for the following discussions.

5.5.1 Case 1: Local Fuzzy Variables Unmodified by Transitions

Case 1 can be described by two conditions: (1) $\rho(p_j) = C_j$, $1 \leq j \leq n$ and C_j is a fixed membership function; (2) for some transition t_i, all input places have at least one token at time N, i.e., $\varrho(p_{i_u}^{(N)}) \geq 1$, $1 \leq u \leq k$.

If the above two conditions are satisfied, then t_i is enabled. The resulting rule $r_\varrho^{t_i}$ can be written as

$$r_\varrho^{t_i}(\varrho(p_{i_1}^{(N)}), \varrho(p_{i_2}^{(N)}), \ldots, \varrho(p_{i_k}^{(N)})) = (\Delta(p_{i'_1}), \Delta(p_{i'_2}), \ldots, \Delta(p_{i'_s})),$$

and

$$\Delta(p_{i'_v}) = 1, \qquad 1 \leq v \leq s.$$

Then we can obtain ϱ for each output place as

$$\varrho(p_{i'_v}^{(N+1)}) = \varrho(p_{i'_v}^{(N)}) + \Delta(p_{i'_v}), \quad 1 \leq v \leq s.$$

and

$$\varrho(p_{i_u}^{(N+1)}) = \varrho(p_{i_u}^{(N)}) - 1, \quad 1 \leq u \leq k.$$

The membership functions for all local fuzzy variables are represented by the solid curves in Figure 5.3, and these curves are unchangeable when the system is executed.

Therefore, each time when we reach and fire transition t_i, we expect the same input local variables and output local variables and the output marking only depends on the input marking of t_i. This case can be considered as the same as the ordinary Petri net firing mechanism, with associated fixed local fuzzy variables. The following properties can be seen for this type of FPN.

Theorem 5.1 Assume a fuzzy Petri net of Case 1 is mapped from an ordinary Petri net by assigning to each place a fixed local fuzzy variable, and assume the reasoning function is the same firing rule as in ordinary Petri nets. The resulting fuzzy Petri

net is live, safe, and/or reversible if and only if the original Petri net is live, safe, and/or reversible.

Proof: All these properties are dependent upon the firing mechanisms of transitions in the net. Because the enabling conditions of transitions only depend on the markings of input places, the properties of liveness, safeness, and reversibility are unchanged while the fuzzy Petri net is mapped from an ordinary Petri net or the fuzzy Petri net is mapped back to the ordinary Petri net.

Q.E.D. □

5.5.2 Case 2: Local Fuzzy Variables Modified by Transitions

Case 2 can be described by two conditions: (1) The output local variables of a transition t_i may be changed after t_i is fired, i.e.,

$$r_\rho^{t_i}(\rho(p_{i_1}^{(N)}), \rho(p_{i_2}^{(N)}), \ldots, \rho(p_{i_k}^{(N)})) = (\rho(p_{i'_1}^{(N+1)}), \rho(p_{i'_2}^{(N+1)}), \ldots, \rho(p_{i'_s}^{(N+1)})),$$

where N and $N+1$ represent the different time slots before and after t_i is fired; (2) for some transition t_i, all input places have at least one token at time N, i.e., $\varrho(p_{i_u}^{(N)}) \geq 1$, $1 \leq u \leq k$.

If the above two conditions are satisfied, then t_i is enabled. The resulting rule $r_\varrho^{t_i}$ can be written as

$$r_\varrho^{t_i}(\varrho(p_{i_1}^{(N)}), \varrho(p_{i_2}^{(N)}), \ldots, \varrho(p_{i_k}^{(N)})) = (\Delta(p_{i'_1}), \Delta(p_{i'_2}), \ldots, \Delta(p_{i'_s})).$$

Similarly,

$$\varrho(p_{i'_v}^{(N+1)}) = \varrho(p_{i'_v}^{(N)}) + \Delta(p_{i'_v}), \quad \Delta(p_{i'_v}) = 1, \ \textit{and} \ 1 \leq v \leq s.$$

and

$$\varrho(p_{i_u}^{(N+1)}) = \varrho(p_{i_u}^{(N)}) - 1, \quad 1 \leq u \leq k.$$

The membership functions of all local fuzzy variables are represented by dotted curves in Figure 5.4. These curves may be changed during the execution of the

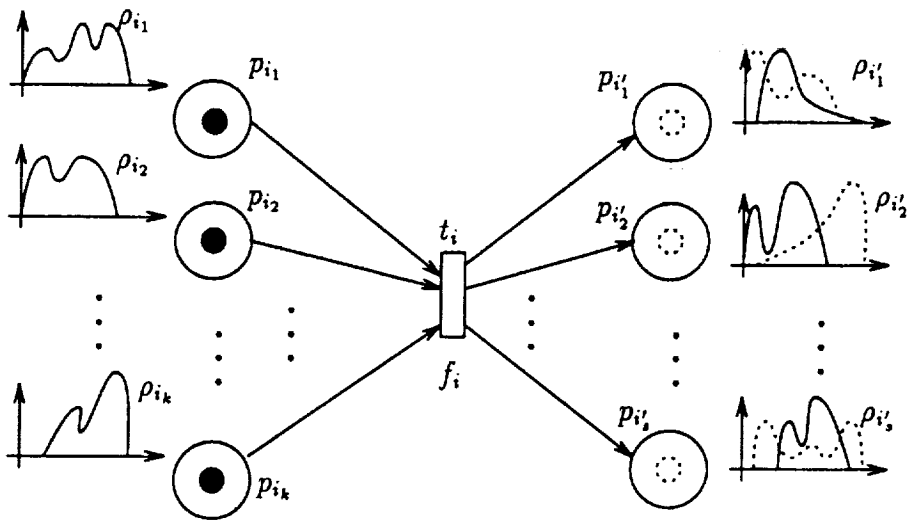


Figure 5.4: The input local variables and output local variables for a transition t_i of Case 2. Before t_i is fired, p_{i_u} contains a token, $1 \leq u \leq k$. After t_i is fired, $p_{i'_v}$ obtains a token, $1 \leq v \leq s$. The output local fuzzy variables for this case are changed.

system. The generation of the output local variables are dependent on the input local variables and the rule $r_\rho^{t_i}$.

Case 2 is more general than Case 1. For the robotic assembly task in Figure 5.1, Case 1 fixes the uncertainty associated with each place, and thus the errors this case can model are very limited. For case 2, the configuration $R_c S$ depends not only on transition t_1 but also the initial configuration $\rho(R_o)$ and $\rho(S)$. The following properties can be found for this type of fuzzy Petri net.

Theorem 5.2 Suppose a fuzzy Petri net is mapped from an ordinary Petri net by assigning to each place a changeable local fuzzy variable and the reasoning functions are the same as the firing strategy as defined in ordinary Petri nets. The resulting fuzzy Petri net is live, safe, and/or reversible if and only if the original Petri net is live, safe, and/or reversible.

Proof: The proof is similar to Theorem 5.1 because the changeable local fuzzy

variables still have no influence on deciding the marking of the net.

Q.E.D. □

5.5.3 Case 3: Transition Firing Depends on Input Local Fuzzy Variables

Case 3 can be described by two conditions: (1) The output local variables of a transition t_i may be changed after t_i is fired, i.e.,

$$r_\rho^{t_i}(\rho(p_{i_1}^{(N)}), \rho(p_{i_2}^{(N)}), \ldots, \rho(p_{i_k}^{(N)})) = (\rho(p_{i'_1}^{(N+1)}), \rho(p_{i'_2}^{(N+1)}), \ldots, \rho(p_{i'_s}^{(N+1)})),$$

where N and $N+1$ represent the different time slots before and after t_i is fired; (2) for some transition t_i, all input places have at least one token at time N, i.e., $\varrho(p_{i_u}^{(N)}) \geq 1$, $1 \leq u \leq k$.

If the above two conditions are satisfied, then t_i is enabled. If we fire t_i, then

$$r_\varrho^{t_i}(\varrho(p_{i_1}^{(N)}), \varrho(p_{i_2}^{(N)}), \ldots, \varrho(p_{i_k}^{(N)}), \rho(p_{i_1}^{(N)}), \rho(p_{i_2}^{(N)}), \ldots, \rho(p_{i_k}^{(N)})) = (\Delta(p_{i'_1}), \Delta(p_{i'_2}), \ldots, \Delta(p_{i'_s})),$$

and

$$\Delta(p_{i'_v}) \in \{0, 1\}, \quad 1 \leq v \leq s.$$

Then we can obtain ϱ for each output place as

$$\varrho(p_{i'_v}^{(N+1)}) = \varrho(p_{i'_v}^{(N)}) + \Delta(p_{i'_v}), \quad 1 \leq v \leq s,$$

and

$$\varrho(p_{i_u}^{(N+1)}) = \varrho(p_{i_u}^{(N)}) - 1, \quad 1 \leq u \leq k.$$

The membership functions of all local fuzzy variables are represented by dotted curves in Figure 5.5. These curves may change during the execution of the system.

For this case, the output local fuzzy variables are dependent on input local fuzzy variables and the fuzzy reasoning rules, and the output markings after t_i is fired are dependent upon the input local fuzzy variables, the input markings, and the firing functions. Therefore, even if all input places have tokens, after t_i is

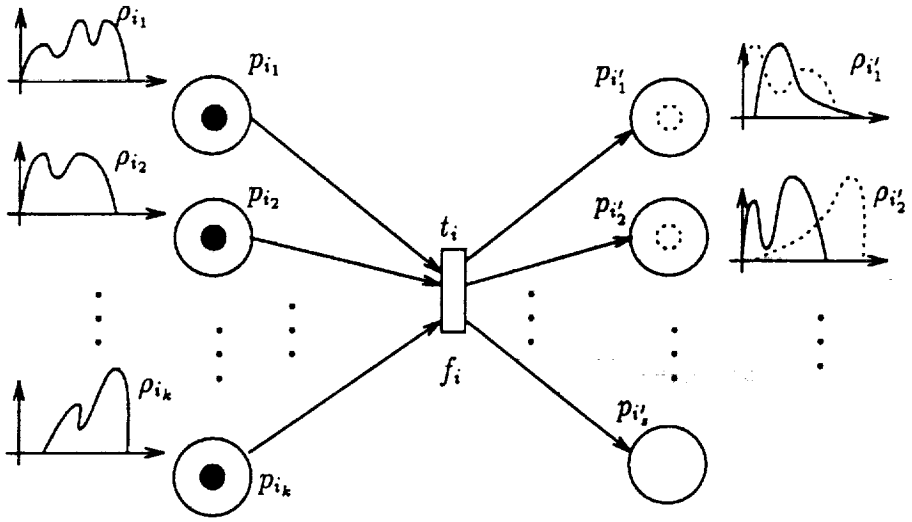


Figure 5.5: The input local variables and output local variables for a transition t_i of Case 3. Before t_i is fired, p_{i_u} contains a token, $1 \leq u \leq k$. After t_i is fired, some $p_{i'_v}$ obtain a token, $1 \leq v < s$. The output local fuzzy variables for this case are changed.

fired, it does not necessarily guarantee that all output places get more tokens. One application of this type of fuzzy Petri net is sensor-based selection for on-line robotic operations. After a sensor is used to verify a system state, the following operation may be local error recovery, global error recovery, or continuation of the execution of the planned task sequence, all of which depend on the token in one output place of a sensing transition.

The properties of this type of fuzzy Petri net are not provable in general because of different reasoning strategies which decide the availability of tokens in output places dependent upon input local fuzzy variables and the rule $r_\varrho^{t_i}$. Some useful subclasses of this case are worthwhile to investigate. We first give the following definition:

Definition 5.5 A mutually exclusive transition t_i: (1) if $p_{i'_1}$, $p_{i'_2}$, ..., and $p_{i'_s}$ are s output places of t_i; (2) After t_i is fired, $\Delta(\varrho(p_{i'_j})) = 1$ or $\Delta(\varrho(p_{i'_j})) = 0$, $1 \leq j \leq s$,

and $\mathcal{D} = \bigcup_j \{p_{i'_j} : \Delta(\varrho(p_{i'_j})) = 1\}$, $\mathcal{D}' = \bigcup_j \{p_{i'_j} : \Delta(\varrho(p_{i'_j})) = 0\}$; (3) $\mathcal{D} \neq \emptyset$, $\mathcal{D}' \neq \emptyset$, $\mathcal{D} \bigcap \mathcal{D}' = \emptyset$, and $\mathcal{D} \bigcup \mathcal{D}' = \bigcup_{j=1}^{s} \{p_{i'_j}\}$.

The following theorem provides the condition for safeness for a class of fuzzy Petri nets.

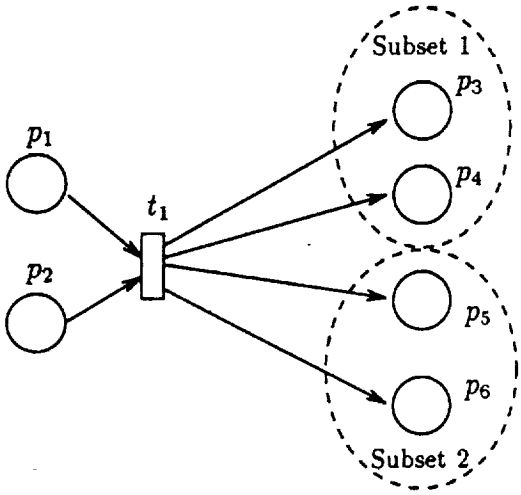
Theorem 5.3 Suppose a fuzzy Petri net is mapped from an ordinary Petri net by assigning each place a changeable local fuzzy variable. The reasoning functions and the input local fuzzy variables decide the output marking and the output local fuzzy variables. If one and only one mutually exclusive transition t_i exists in the net and other transitions satisfy the conditions described in Case 1 or Case 2, and the original Petri net is safe, the resulting fuzzy Petri net is also safe.

Proof: Because after t_i is fired, not all output places will receive tokens, therefore, for any following transition sequences, the places in the net will get fewer or the same number of tokens as in the ordinary Petri net. The resulting fuzzy Petri net will be safe if the original ordinary Petri net is safe.

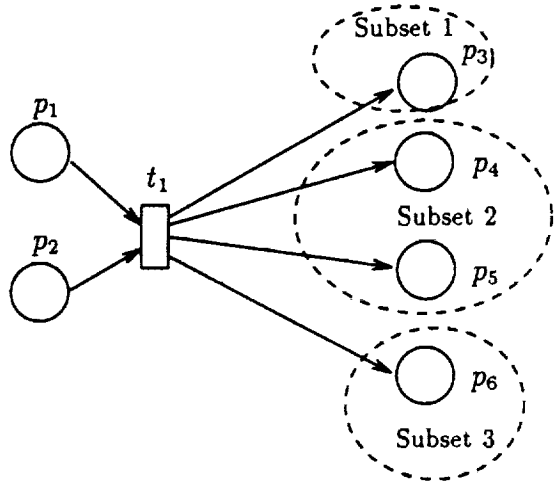
Q.E.D. □

Definition 5.6 *MEO(*mutually exclusive output*)* subsets of a mutually exclusive transition t_i: For all feasible transition sequences from the initial marking and for all possible input local fuzzy variables available for t_i, we assume there are L different partitions of the set of the output places of t_i: $\mathcal{D}_1$, $\mathcal{D}_2$, ..., $\mathcal{D}_L$. If the following conditions are satisfied: (1) $\mathcal{D}_i \bigcap \mathcal{D}_j = \emptyset$, $i \neq j$ and $1 \leq i, j \leq L$; (2) $\bigcup_{l=1}^{L} \mathcal{D}_l = \bigcup_{j=1}^{s} p_{i'_j}$; (3) At any time after t_i is fired, $\Delta(p_{i'_v}) = 1$, for all $p_{i'_v} \in \mathcal{D}_j$ and $\Delta(p_{i'_v}) = 0$ for all $p_{i'_v} \notin \mathcal{D}_j$, $1 \leq v \leq s$ and $1 \leq j \leq L$, then $\mathcal{D}_1$, $\mathcal{D}_2$, ..., $\mathcal{D}_L$ are called *MEO* subsets of t_i.

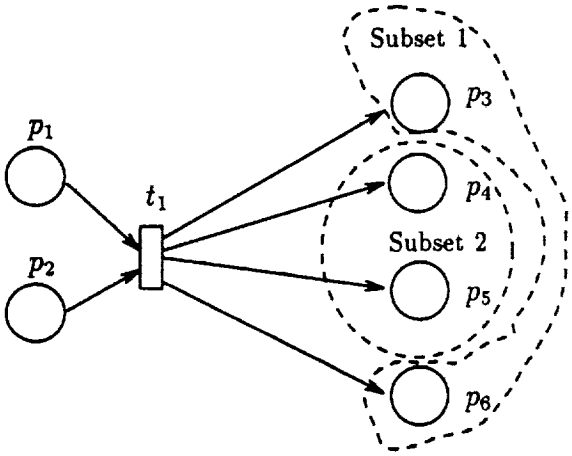
Figure 5.6 shows a mutually exclusive transition with four output places and four examples of possible output markings. Figure 5.6(a), 5.6(b), and 5.6(c) show



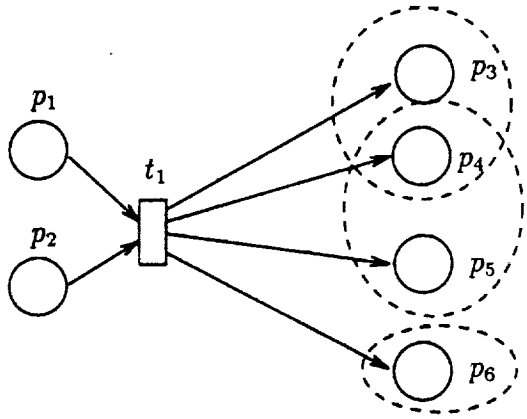
(a) MEO subsets: Subset 1 and 2.



(b) MEO subsets: Subset 1, 2, and 3.



(c) MEO subsets: Subset 1 and 2.



(d) Non-MEO subsets.

Figure 5.6: Some examples of *MEO* subsets for a mutually exclusive transition with four output places.

MEO subsets. In Figure 5.6(a), $L = 2$, $\mathcal{D}_1 = \{p_3, p_4\}$, $\mathcal{D}_2 = \{p_5, p_6\}$, which indicates that sometimes after t_1 is fired, $\varrho(p_3) = \varrho(p_4) = 1$, and $\varrho(p_5) = \varrho(p_6) = 0$, and sometimes $\varrho(p_5) = \varrho(p_6) = 1$ and $\varrho(p_3) = \varrho(p_4) = 0$. No other possibilities may appear for ϱ. In Figure 5.6(b), $L = 3$, $\mathcal{D}_1 = \{p_3\}$, $\mathcal{D}_2 = \{p_4, p_5\}$, and $\mathcal{D}_3 = \{p_6\}$, and in Figure 5.6(c), $L = 2$, $\mathcal{D}_1 = \{p_3, p_6\}$, *and* $\mathcal{D}_2 = \{p_4, p_5\}$. Figure 5.6(d) shows non-*MEO* subsets where $\varrho(p_4) = 1$, $\varrho(p_5) = 1$ and $\varrho(p_3) = 1$, $\varrho(p_4) = 1$ are both possible after firing t_i. The following theorems define a class of fuzzy Petri nets which guarantee the properties of liveness, safeness, and reversibility.

Before we discuss the following theorem, we give definitions of the addition of two Petri nets and subtraction of a subnet from a Petri net. Similar operations on Petri nets are used in synthesis techniques of Petri nets[53].

Definition 5.7 The addition("+") of a net $N_1 = (P_1, T_1, \alpha_1, \beta_1)$ and a net $N_2 = (P_2, T_2, \alpha_2, \beta_2)$: $N_1 + N_2 = (P_1 \bigcup P_2, T_1 \bigcup T_2, \alpha, \beta)$ where $\alpha = \bigcup_{ij}\{(p_i, t_j)\}$, $(p_i, t_j) \in \alpha_1 \bigcup \alpha_2$, and $\beta = \bigcup_{ij}\{(t_j, p_i)\}$, $(t_j, p_i) \in \beta_1 \bigcup \beta_2$.

Definition 5.8 The subtraction("−") of a subnet $N' = (P', T', \alpha', \beta')$ from a Petri net $N = (P, T, \alpha, \beta)$: $N - N' = (P - P', T - T', \alpha'', \beta'')$, where $\alpha'' = \bigcup_{ij}\{(p_i, t_j)\}$, $(p_i, t_j) \in \alpha$, $\beta'' = \bigcup_{ij}\{(t_j, p_i)\}$, $(t_j, p_i) \in \beta$, and $p_i \in P - P'$, $t_j \in T - T'$.

Theorem 5.4 If a fuzzy Petri net is obtained in the same way as described in Theorem 5.3, and t_i has *MEO* subsets $\mathcal{D}_1$, $\mathcal{D}_2$, ..., $\mathcal{D}_L$, and the following conditions are satisfied: (1) there exist L number of subnets $\mathcal{N}_1$, $\mathcal{N}_2$, ..., $\mathcal{N}_L$ and $\mathcal{N}_i$ contains $\mathcal{D}_i$, $1 \leq i \leq L$; (2) $N - (\sum_{i=1}^{L} N_i - N_1)$, $N - (\sum_{i=1}^{L} N_i - N_2)$, ..., $N - (\sum_{i=1}^{L} N_i - N_L)$ are live; (3) $N_i \bigcap N_j = \emptyset$, $i \neq j$ and $1 \leq i, j \leq L$. Then this fuzzy Petri net is also live.

Proof: Suppose at any time when t_i is fired, the places in $\mathcal{D}_1$ always obtain tokens, then $N - (\sum_{i=1}^{L} N_i - N_1)$ is live. In other words, the transitions contained in $N - (\sum_{i=1}^{L} N_i - N_1)$ are live. Similarly, if we assume $\mathcal{D}_j$ is always guaranteed to obtain

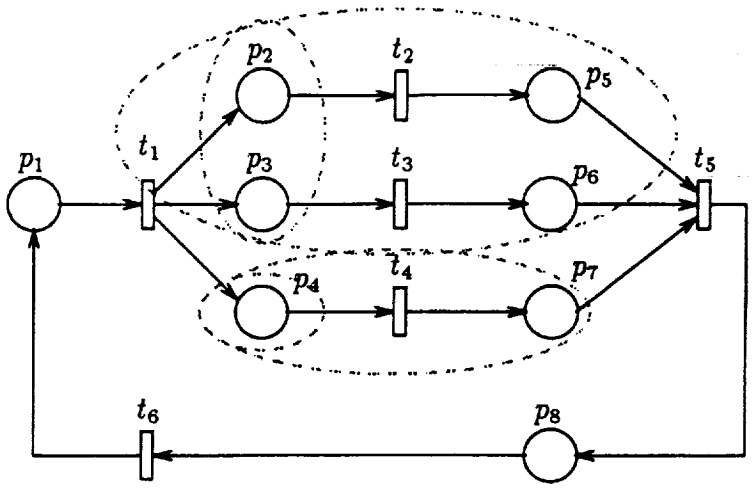


Figure 5.7: A fuzzy Petri net with one mutually exclusive transition t_1. After t_1 is fired, p_2 and p_3 or p_4 will receive the tokens based on the local variable available in p_1 and $r_e^{t_1}$.

tokens after t_i is fired, then the transitions in $N - (\sum_{i=1}^{L} N_i - N_j)$ are live. Because after t_i is fired, all $\mathcal{D}_j$ may contain tokens, therefore, all transitions in the net are live. The fuzzy Petri net satisfying the above conditions is thus live.

Q.E.D. □

An example for a fuzzy Petri net with one mutually exclusive transition t_1 and *MEO* subsets $\mathcal{D}_1 = \{p_2, p_3\}$ and $\mathcal{D}_2 = \{p_4\}$ is shown in Figure 5.7. The subnet N_1 has the following structure: $P_1 = \{p_2, p_3, p_5, p_6\}$, $T_1 = \{t_2, t_3\}$, $\alpha_1 = \{(p_2, t_2), (p_3, t_3)\}$, $\beta_1 = \{(t_2, p_5), (t_3, p_6)\}$. The subnet N_2 has the following structure: $P_2 = \{p_4, p_7\}$, $T_2 = \{t_4\}$, $\alpha_2 = \{(p_4, t_4)\}$, and $\beta_2 = \{(t_4, p_7)\}$. N_1 contains $\mathcal{D}_1$ and N_2 contains $\mathcal{D}_2$. From Theorem 5.4, we can prove that this fuzzy Petri net is live.

The property of reversibility is important for modeling error recovery strategy using Petri nets. When a fuzzy Petri net is used, reversibility implies that the marking is restored for the initial state, and some local fuzzy variables may be

changed. The following theorem proposes the condition for a fuzzy Petri net to be reversible.

Theorem 5.5 If a fuzzy Petri net is obtained in the same way as described in Theorem 5.4 except that subnets $N - (\sum_{i=1}^{L} N_i - N_l)$, $1 \leq l \leq L$, may or may not be live, $N - (\sum_{i=1}^{L} N_i - N_1)$, $N - (\sum_{i=1}^{L} N_i - N_2)$, ..., $N - (\sum_{i=1}^{L} N_i - N_L)$ are reversible, and N_1, N_2, ..., N_L contain no token in the initial marking, then this fuzzy Petri net is also reversible.

Proof: The proof is straightforward following the same strategy as discussed in Theorem 5.4.

Q.E.D. □

If there are more than one mutually exclusive transition, $t_{i_1}, t_{i_2}, \ldots, t_{i_r}$, existing in the fuzzy Petri net, we can generalize the above discussions to the following corollaries:

Corollary 5.1 Suppose a fuzzy Petri net is mapped from an ordinary Petri net by assigning each place a changeable local fuzzy variable. The reasoning functions and the input local fuzzy variables decide the output marking and the output local fuzzy variables. If more than one mutually exclusive transition, $t_{i_1}, t_{i_2}, \ldots, t_{i_r}$, exists in the net and the original Petri net is safe, the resulting fuzzy Petri net is also safe.

Corollary 5.2 If a fuzzy Petri net is obtained in the same way as described in Corollary 5.1, and $t_{i_u}(1 \leq u \leq r)$ has *MEO* subsets $D_{i_{u_1}}$, $D_{i_{u_2}}$, ..., $D_{i_{u_L}}$. And the following conditions are satisfied: (1) there exist u_L number of subnets $N_{i_{u_1}}$, $N_{i_{u_2}}$, ..., $N_{i_{u_L}}$ and N_{i_j} contain $\mathcal{D}_{i_j}$, $u_1 \leq j \leq u_L$; (2) $N - (\sum_{j=u_1}^{u_L} N_{i_j} - N_{i_{u_1}})$, $N - (\sum_{j=u_1}^{u_L} N_{i_j} - N_{i_{u_2}})$, ..., $N - (\sum_{j=u_1}^{u_L} N_{i_j} - N_{i_{u_L}})$ are live; (3) $N_{i_p} \bigcap N_{i_q} = \emptyset$, $p \neq q$ and $u_1 \leq p, q \leq u_L$, then this fuzzy Petri net is also live.

Corollary 5.3 If a fuzzy Petri net is obtained in the same way as described in Corollary 5.2 except that the subnets $N-(\sum_{j=u_1}^{u_L} N_{i_j}-N_{i_l})$, $u_1 \leq l \leq u_L$ may or may not be live. $N-(\sum_{j=u_1}^{u_L} N_j - N_{i_{u_1}})$, $N-(\sum_{j=u_1}^{u_L} N_j - N_{i_{u_2}})$, ..., $N-(\sum_{j=u_1}^{u_L} - N_{i_{u_L}})$ are reversible, and N_{i_1}, N_{i_2}, ..., N_{L_u} contain no token in the initial marking, then this fuzzy Petri net is also reversible.

5.6 FPNs with Global Fuzzy Variables: Example of Task Sequencing

An application of FPNs has been discussed for the task sequencing problem[12, 20]. When a robotic assembly system is modeled by a fuzzy Petri net, all feasible operations in the system are represented by transitions, and all possible objects, such as components, devices, subassemblies, and assemblies are represented by places. Given an initial marking and an expected final marking, a task sequence planning problem is equivalent to the problem of sequencing transitions, which usually occurs in an off-line mode. In this approach a global fuzzy variable is introduced to represent 'degree of completion' of the task while maintaining the precedence of key operations ('key transitions').

A reachability strategy[16, 89] can be used to search for sequences in an ordinary Petri net. For sequencing transitions, while maintaining precedence among those properties that are changed during the process, a *prime number marking algorithm* was proposed[12, 14, 20, 21] to generate global fuzzy values of a variable $\sigma(p_i)$ such that precedence is preserved. In this case, the fuzziness of the net only shows up in tokens, not places.

In the example shown in Figure 5.2, if we assume this net is an FPN carrying σ, the initial marking using σ values is $S_g = (0, C, e, e)$, and the final marking is $S_g = (e, C, 1, 0)$. Places p_1, p_3, and p_4 contain the same kind of token, which indicates whether the task is completed, but with different values, 0 and 1.

For other examples, there may be k colors of tokens, c_1, c_2, ..., c_k. For

color c_i, an independent subsequence for reaching the global task is $t_{i_1}, t_{i_2}, \ldots, t_{i_{n_i}}$. The transitions in this subsequence may not be a consecutively enabled transition sequence. However, to correctly execute the global task, all subsequences should be completely executed with a correct order. When using this strategy to search or plan a transition sequence from the net, time and space will be saved compared with the methods used in ordinary Petri nets.

5.7 FPNs with Local Fuzzy Variables: Examples of Robotic Sensing

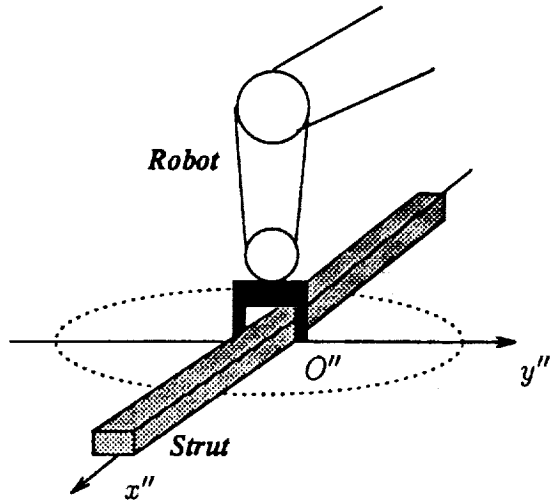
5.7.1 Local Fuzzy Variable for Sensor-Based Error Recovery

Figure 5.8 shows the example of local fuzzy variables for the robotic system from Figure 5.1. The position of the strut on the table is represented by a fuzzy variable $\rho(S)$. The position of the robot gripper is represented by a fuzzy variable $\rho(R)$.

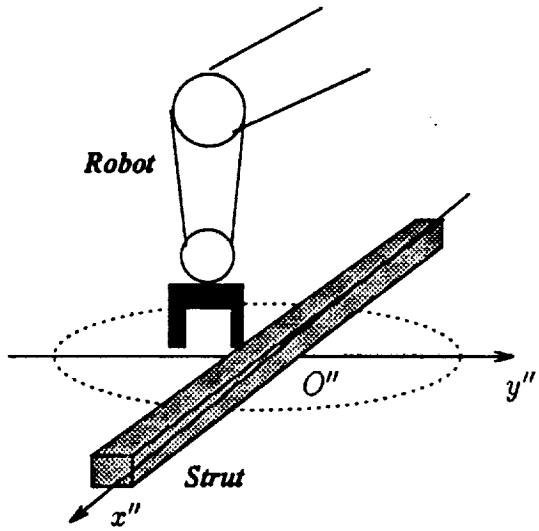
Figure 5.8(a) is a correct positioning of RS, but in Figure 5.8(b), an incorrect configuration is shown. This information of the real state of RS is not known prior to execution and a sensor will be used on-line to verify the state.

We assume the next operation of the robot is t_2, a *move* operation, as shown in Figure 5.9 (this is a simplified extension of Figure 5.2 because we omit the possibility of t_1 resulting in R_c and S, and we don't distinguish R_c and R_o. This simplification can also be implemented by using a sensor after t_1, so that there will be no vagueness between R_cS and S, R_c after t_1 is fired.). This may fail if the robot is not holding anything. An appropriate decision should be made to evoke an error recovery sequence in case the grasp operation fails.

We assume the membership function for $\rho(RS)$ is a 2-D membership function as shown in Figure 5.10. To separate the 2-D membership function into two possible future executions, move or error recovery, we use a sensing operation to verify the state after the grasp operation is done. A sensor separates the 1-D membership



(a)



(b)

Figure 5.8: A scenario of robot-strut assembly in Figure 5.1. (a) shows that the grasp position is above the strut and (b) shows the gripper has missed the strut. The dotted circle displayed on the table plane is a possible range the robot gripper may reach.

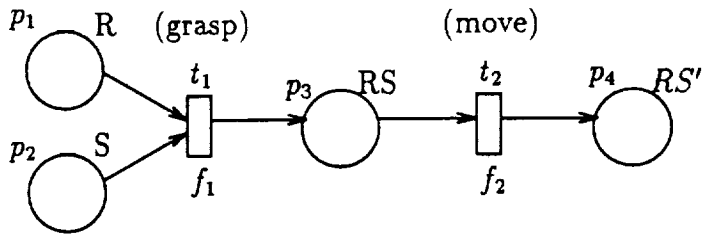


Figure 5.9: A FPN representation for a *grasp* and *move* operation for the robot. RS' is a specified state the move operation is supposed to reach.

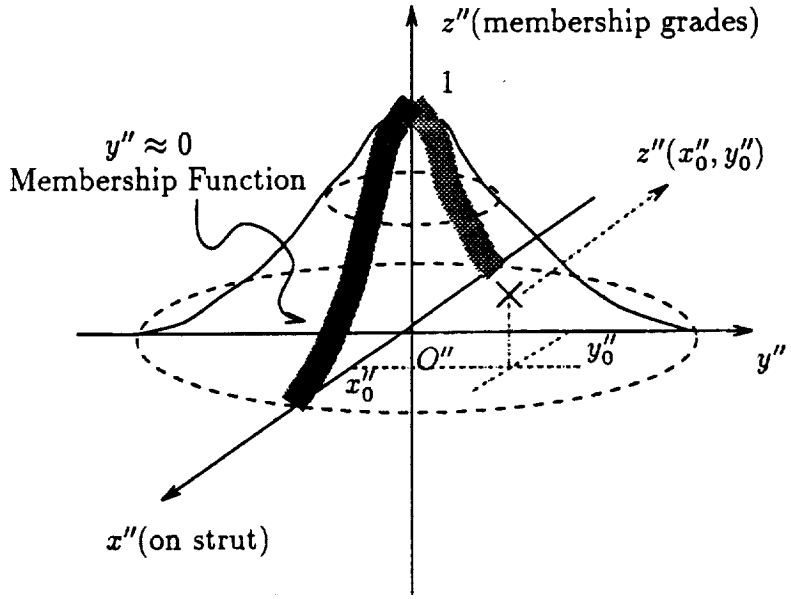


Figure 5.10: A distribution for the membership grades of the position the robot gripper reaches to grasp the strut. The darkened curve is a 1-D membership function where the robot is assumed to reach the strut.

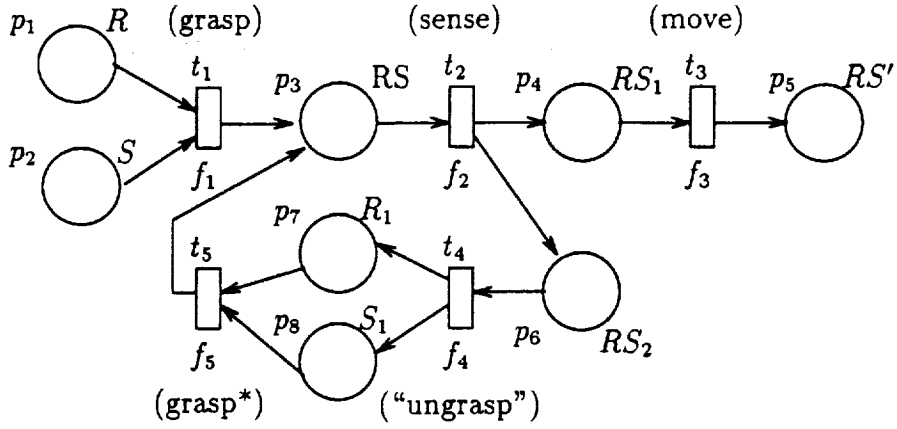


Figure 5.11: A modified FPN which includes an error recovery sequence. If the sensed value does not fall near the x'' axis, an "ungrasp" transition, for a robot to move to another temporary position(not necessarily the original position) and then grasp the strut again, is fired. Note that grasp* may not be the same as grasp. The error recovery sequence is initiated by the fuzzy reasoning rule in f_2 attached to transition t_2.

function from a 2-D membership function as shown in Figure 5.10. If $y'' \approx 0$, the 'darkened' membership function describes the resulting uncertainty in x'' of the grasped strut. If $y'' \not\approx 0$, an error recovery sequence should be followed to disassemble RS. The fuzzy reasoning rules at t_2 models the resulting decisions. On-line execution with an actual sensor value requires a fuzzy control decision rule which executes the appropriate sequence. A modified fuzzy Petri net based on the above discussion is shown in Figure 5.11. The error recovery sequence is initiated by the fuzzy reasoning rule in f_2 attached to transition t_2.

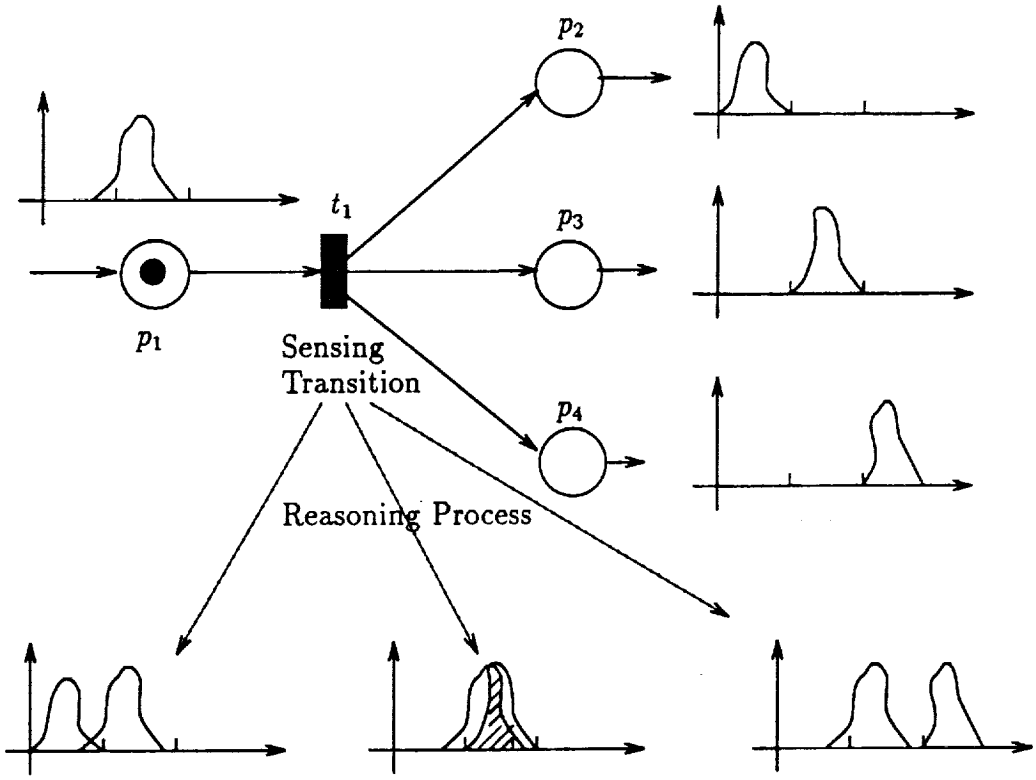
5.7.2 Modeling Sensing Operations as Mutually Exclusive Transitions

During the execution of a robotic system modeled by a fuzzy Petri net, the exact positions of the arm or the state of the object being processed are never known exactly because of the approximation of the controller and the uncertain environment. A sensor or multiple sensors are used to verify and validate on-line

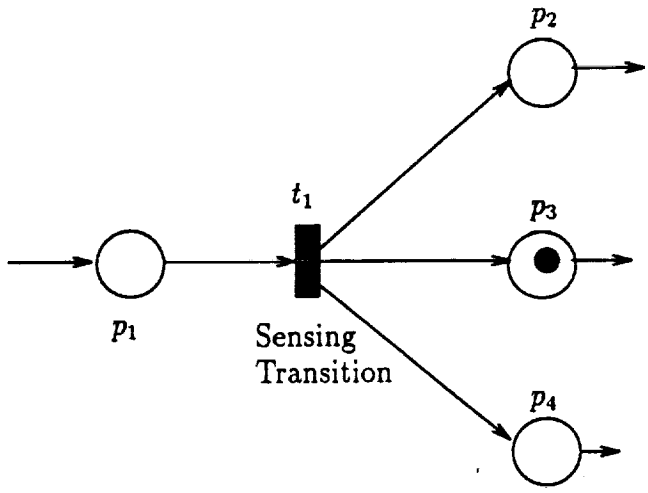
approximate information so that the whole operations space can be divided into several mutually exclusive ranges. Sensors may also have errors and approximation and fuzzy reasoning rules can be used to reduce the fuzziness caused by uncertainty of the partial results. The fuzzy marking variables and the output local fuzzy variables are obtained through reasoning at the sensory transition.

The fuzzy Petri net corresponding to a mutually exclusive transition t_1 with three output places is shown in Figure 5.12. t_1 is modeled as a virtual sensory transition which will verify the state of p_1 and obtain the corresponding output mutually exclusively. From this example, we see that when a local fuzzy variable in p_1 is obtained, it is input into t_1 for reasoning about the token in an output place. The reasoning function $r_e^{t_1}$ consists of three steps: (1) intersect input fuzzy variables with the expected membership function residing in p_2 and obtain the intersection area or the highest membership degree, (2) intersect input fuzzy variables with the expected membership function residing in p_3 and obtain the intersection area or the highest membership degree, (3) intersect input fuzzy variables with the expected membership function residing in p_4 and obtain the intersection area or the highest membership degree. Then, from these three partial results, we can get a maximum value corresponding to a certain place p_i, $2 \leq i \leq 4$. In this example, p_3 has the maximum intersection area as shown by the shaded area in Figure 5.12(a). Therefore, a token is put in p_3 after the reasoning process ends as shown in Figure 5.12(b).

In a real system, a sensing operation may direct the following transition sequences to continue the execution, locally recover from an error if that error is locally recoverable, or globally recover from an error if that error is not locally recoverable. Each sensing operation is a mutually exclusive transition, and its output places constitute *MEO* subsets.



(a) Fuzzy reasoning of a mutually exclusively sensing transition.



(b) The resulting token in place p_3.

Figure 5.12: Fuzzy reasoning in a fuzzy Petri net for obtaining a token in an output place mutually exclusively.

5.8 Conclusions

In this chapter, we have proposed a generalized definition of the fuzzy Petri net using three different types of fuzzy variables: local fuzzy variables, fuzzy marking variables, and global fuzzy variables. Local fuzzy variables are examined in detail, and are used to reason about parameters associated with places. Fuzzy Petri nets are shown to have advantages over ordinary Petri nets to model a system which has vague, random, and approximate information. Sensors can be used to handle uncertainty of occurrence of events and reduce the dimension of membership distribution. Sensor-based verification for states and sensor-based error recovery strategies can be incorporated into the FPN model of the system. FPNs are also shown to be a good model for off-line sequencing and on-line reasoning about execution.

Properties of these nets have been defined for specific cases of interest. An example of the application of these properties is modeling and analysis of a sensor-based robotic system. Uncertain sensory input data can be handled and sensory transitions may be modeled as mutually exclusive transitions. Only a subset of the output places can receive tokens after the transition is fired, the other places will not receive any tokens.

Fuzzy values as defined here are membership functions in a certain universe of uncertainty. Sometimes, we may want to know the crisp values for some tokens or the crisp state of the system. In this case, a fuzzy defuzzifier[63, 64] is necessary to defuzzify the fuzzy values before it is output to the controller. Similarly, the input values to the FPN model may also be crisp values, and a fuzzifier[63, 64] is needed to turn crisp values to fuzzy values. Also, different reasoning strategies may be used for different transitions in the net.

Fuzzy Petri nets are potentially useful to model many different types of discrete event systems with uncertainty. Global fuzzy variables may be used to plan operations sequences for robotic or manufacturing systems. Local fuzzy variables

may be used to reason about sensor-based error recovery sequences. Fuzzy marking variables impose probabilistic conditions on the system marking. The interrelationship among these fuzzy variables is a topic which leads to additional research issues. In practice, other forms of non-fuzzy probabilistic reasoning could be used within this framework.

An important issue related to this research is the choice of rules for transitions. Experiments can be done to learn rules. A fuzzy Petri net will be robust only after many experiments and modifications of the reasoning structures. A neural network may be useful to represent these firing rules. This research also has important implications for knowledge representation, knowledge reasoning, modeling of expert systems, and other AI applications.

CHAPTER 6
TASK SEQUENCE PLANNING USING FUZZY PETRI NETS

This chapter discusses the problem of representation and planning of operations sequences in a robotic system using *fuzzy Petri nets*[12, 17, 18]. In the fuzzy Petri net representation, objects whose internal states are altered during a process are termed 'soft' objects, and the process steps where alterations may occur are labeled 'key' transitions. A correct sequence is defined as a sequence which is feasible, complete, and satisfies precedence relations. In this formulation, the internal state of an object is represented by a global fuzzy variable attached to the token related to the 'degree of completion' of the process. All correct operations sequences must satisfy process sequence constraints imposed by fuzzy transition rules. The correct precedence relationships and the characteristics of completeness for operations in all feasible sequences are guaranteed by the *prime number marking algorithm* which marks the fuzzy Petri net. The use of fuzzy transition rules in this application simplifies the representation and search problems for task planning where correct sequences do not depend on exact knowledge of internal states, but only their precedence relations.

6.1 Introduction

The objective of task sequence planning for a robotic workcell or manufacturing system is to efficiently represent all feasible and complete task sequences with correct precedence relations and to be able to choose among them. A sequence of the shortest length or other optimality criterion may be selected from these correct sequences. In previous work[11, 13, 15], it has been shown that the AND/OR net representation of an assembly system may be mapped to an ordinary Petri net with specific properties such as safeness, 1-boundedness, liveness, and reversibility. In

this chapter, we introduce a fuzzy Petri net mapping instead of the ordinary Petri net mapping to represent a system which includes some *soft objects*, e.g., objects which change their internal states during the task. A state of the system is thus composed of a set of *membership functions* for the completion of the global task on all feasible objects. Fuzzy transition rules implement the sequencing constraints required to direct the process to the final global state of the system.

AND/OR graphs[47, 48] have been used in assembly task planning to represent and search all possible assembly sequences. The AND/OR net[11, 13, 15] extends the AND/OR graph representation to incorporate system mechanisms and devices, and defines an *Internal State Transition*(IST) operation which modifies the internal state of an object. The AND/OR net is generated based on the descriptions of objects and all feasible geometric relationships among them, and it is used to plan operations sequences for geometric manipulations including assembly, disassembly, grasping, and robot motions. In [11, 16], we showed that an AND/OR net could be mapped to a Petri net, and this Petri net can then be decomposed to lower level nets[13, 15, 19] while retaining properties of liveness, 1-boundedness, safeness, and reversibility.

In this chapter, we expand the domain of the Petri net mapping as a representation of a robotic workcell, by defining fuzzy internal states of objects and using fuzzy transition rules in the Petri net to impose precedence constraints on key operations.

An *object* in the system is defined as a single component, a subassembly of several components, or a complete assembly. A *soft object* is defined to be an object which includes at least one internal state variable. Similarly, a *hard object* is defined to be an object which is not a soft object. In this chapter, the internal state variables of soft objects are described by fuzzy values of tokens, and a *prime number marking algorithm* is used to map an ordinary Petri net to a fuzzy Petri net in a manner which

guarantees consistent sequencing of operations. In the resulting fuzzy Petri net, each transition has an associated fuzzy reasoning rule and an associated *weighting factor* which evaluate the resulting values in the output places of this transition based on the fuzzy values of tokens in its input places. Transitions which cause the changes of fuzzy values of tokens for objects, are defined as *key transitions* and must be included into all feasible and complete task sequences.

One difficult problem in choosing feasible task sequences is to order the correct precedence relationships among all important events or transitions. Using prime number marking in modeling the system, the weighting factors as well as the initial tokens and final tokens are chosen for all soft objects and hard objects so that an assigned precedence relationship will be automatically followed, and all sequences which incorporate incorrect precedence relationships will be recognized and discarded. The prime token values of soft objects can be interpreted as the degrees of certainty of completion for these objects.

In our recent work[17], a definition of the generalized fuzzy Petri net was given which incorporated three types of fuzzy variables: local fuzzy variables, fuzzy marking variables, and global fuzzy variables. Property analysis associated with some typical cases of fuzzy Petri nets was given[18]. In this chapter, we use the fuzzy Petri net model which carries only global fuzzy variables for task planning, and use it to represent and reason about all feasible, complete, and correctly ordered task sequences for a robot workcell.

6.2 State Representation for Task Sequences

A robotic or automated manufacturing system consists of many different kinds of components such as robots, sensors, fixtures, handling mechanisms, and parts. Different tasks may be assigned to and accomplished by the system. All devices must be coordinated to insure successful completion of a task goal through a sequence of

feasible operations. This on-line coordination may be managed by a subset of the system Petri net which couples transitions to on-line execution of desired operations.

From the task sequence planning point of view, the system must follow a partially ordered sequence of intermediate states to reach from the initial state to the final state[48], where *state* is defined as the vector of the states for all components in the system. We assume a system consists of n components $C_1, C_2, \ldots, C_n$, where C_i represents the ith component. We use $s_j(C_i)$ to represent the state of component C_i at time j, where we also assume a discrete time representation and j is thus zero or a natural number. For the moment, we assume that each component may occupy a fixed number of feasible states in the range 0 to N_i. The integer vector representation for the state of the system is $\underline{S}_j = (s_j(C_1)\ s_j(C_2)\ \ldots\ s_j(C_n))^T$, $0 \leq j \leq M$, and $M+1$ is the maximum number of all feasible states the system may occupy. In this approach, a partially ordered list of state vectors may be used to represent a task sequence.

An alternative representation for the task sequence is based on the definition of specific types of state transitions. An operator will change the state of the system by making a set of components change from one *substate* to another *substate*, where a substate is defined as $\underline{S'}_j = (s_j(C_{p_1})\ s_j(C_{p_2})\ \ldots\ s_j(C_{p_t}))^T$ and $\{s_j(C_{p_1}),\ s_j(C_{p_2}),\ \ldots,\ s_j(C_{p_t})\} \subseteq \{s_j(C_1),\ s_j(C_2),\ \ldots,\ s_j(C_n)\}$. We introduce the concept of substate because many tasks may be thought of as functioning on several objects, i.e., a subset of the objects in the system. Three kinds of tasks, T_1, T_2, and T_3, are defined as follows:

(1) Assembly: One set of components or subassemblies are combined with or put in geometric contact with one or more other sets of components or subassemblies.

$$T_1(O_{i_1},\ O_{i_2},\ \ldots,\ O_{i_u}) = \{\{C_{j_1},\ C_{j_2},\ \ldots,\ C_{j_t}\}\}, \tag{6.1}$$

where

$$O_{i_k} = \{C_1^{(i_k)},\ C_2^{(i_k)},\ \ldots,\ C_{n_k}^{(i_k)}\},$$

$$O_{i_k} \subseteq \{C_{j_1},\ C_{j_2},\ \ldots,\ C_{j_t}\}, \quad 1 \le k \le u, \quad and,$$

$$O_{i_{k_1}} \bigcap O_{i_{k_2}} = \emptyset, \quad 1 \le k_1, k_2 \le u.$$

(2) Disassembly: A subassembly or assembly is separated into a set of components or subassemblies.

$$T_2(\{C_{i_1},\ C_{i_2},\ \ldots,\ C_{i_t}\}) = \{O_{j_1},\ O_{j_2},\ \ldots,\ O_{j_l}\}, \tag{6.2}$$

where

$$O_{j_k} = \{C_1^{(j_k)},\ C_2^{(j_k)},\ \ldots,\ C_{n_k}^{(j_k)}\},$$

$$O_{j_k} \subseteq \{C_{i_1},\ C_{i_2},\ \ldots,\ C_{i_t}\}, \quad 1 \le k \le l, \quad and,$$

$$O_{j_{k_1}} \bigcap O_{j_{k_2}} = \emptyset, \quad 1 \le k_1, k_2 \le l.$$

(3) Internal State Transition: The internal state of a set of components is modified by changing the internal state of a single component in this set, or by changing the relative geometric positions among the components of an assembly, or by modifying a property of a single component in a compact set of components.

$$T_3(O_p) = O_q, \tag{6.3}$$

where

$$O_p = \{s(C_{i_1}),\ s(C_{i_2}),\ \ldots,\ s(C_{i_s}),\ s(C_{i_{s+1}}),\ \ldots, s(C_{i_{s+d}}),\ \ldots,\ s(C_{i_t})\},$$

$$O_q = \{s(C_{i_1}),\ s(C_{i_2}),\ \ldots,\ s'(C_{i_s}),\ s'(C_{i_{s+1}}),\ \ldots, s'(C_{i_{s+d}}),\ \ldots,\ s(C_{i_t})\},$$

and

$$O_p - O_q = \{s(C_{i_s}),\ s(C_{i_{s+1}}),\ \ldots, s(C_{i_{s+d}})\},$$

$$O_q - O_p = \{s'(C_{i_s}),\ s'(C_{i_{s+1}}),\ \ldots, s'(C_{i_{s+d}})\}.$$

As the number and complexity of the system substates and their interactions increases, the task representation may be further simplified by defining a *geometric*

state vector, which separates geometric state relations from internal state variables of individual objects. The geometric state vector is a binary vector where the elements represent all feasible geometric states which can occur during the process. Using this representation, a set of mathematical functions can be defined for task planning and execution, and transitions from one system state to another system state may be defined based upon the properties of vectors. For each single component C_i, the corresponding *element state* is $e_i = s(C_i)$ and for each feasible set of components $C_{i_1}, C_{i_2}, \ldots, C_{i_m}$, the corresponding element state is $e_i = s(\{C_{i_1}, C_{i_2}, \ldots, C_{i_m}\})$. The geometric state vector is thus $\underline{GS} = (e_1\ e_2\ \ldots\ e_n)^T$ where e_i is either 0 or 1. This vector occupies the same dimension for the same system at any time. We will show in the following discussion how this representation can be generalized to a vector of token values which carries the degrees of completion for the global task.

Figure 6.1 shows an example of a robotic system with two parts, one robot, and two processing machines. In this system, the robot prepares and then inserts the peg cylinder(P) into the hollow cylinder(C) to form a new cylinder assembly. In one feasible, complete, and correctly ordered sequence, the robot first picks up the raw peg cylinder, moves it to the cutting machine for cutting, then transfers it to the lubricating station prior to insertion. The corresponding AND/OR net representation for this system is shown in Figure 6.2.

In this AND/OR net, we use AND arcs to represent all feasible assembly or disassembly operations. For example, RPC is connected with PC and R by an AND arc, which means RPC can be disassembled to PC and R, and PC and R can be assembled and form RPC. An internal state transition operation is represented by a thick line connecting two corresponding nodes in the AND/OR net. The off-line and on-line selection of the assembly, disassembly, or internal state transition operation is based on the system state, i.e., the matching of the precondition of each operation with all element states. AND/OR nets display all feasible objects

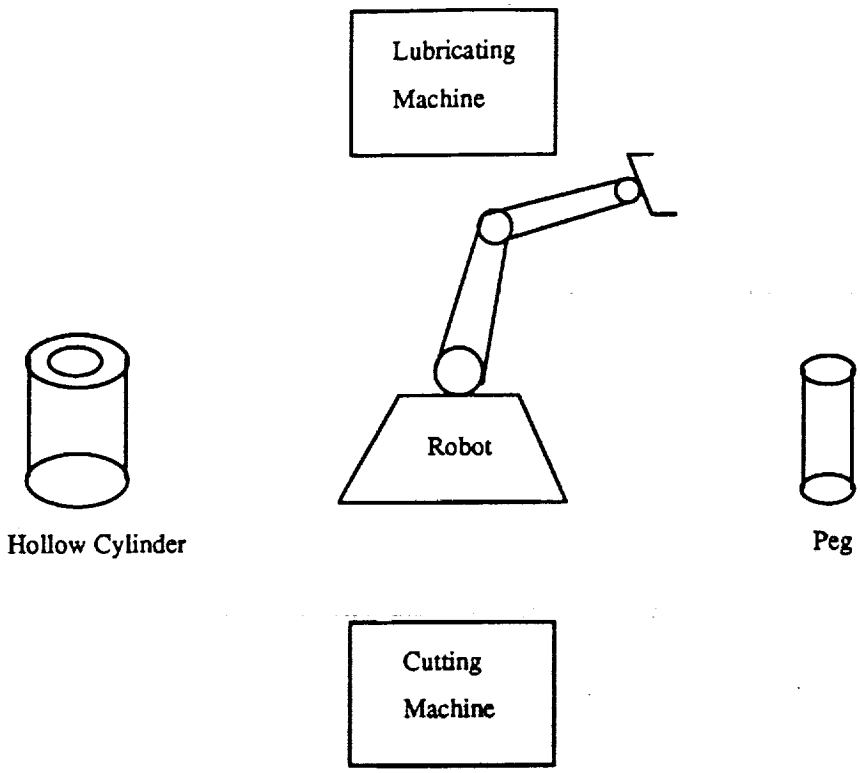


Figure 6.1: A peg-cylinder assembly system.

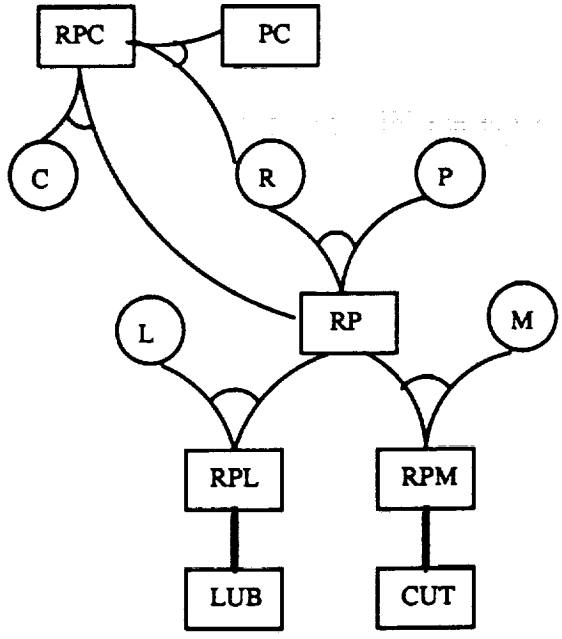


Figure 6.2: The AND/OR net representation for the peg-cylinder assembly system.

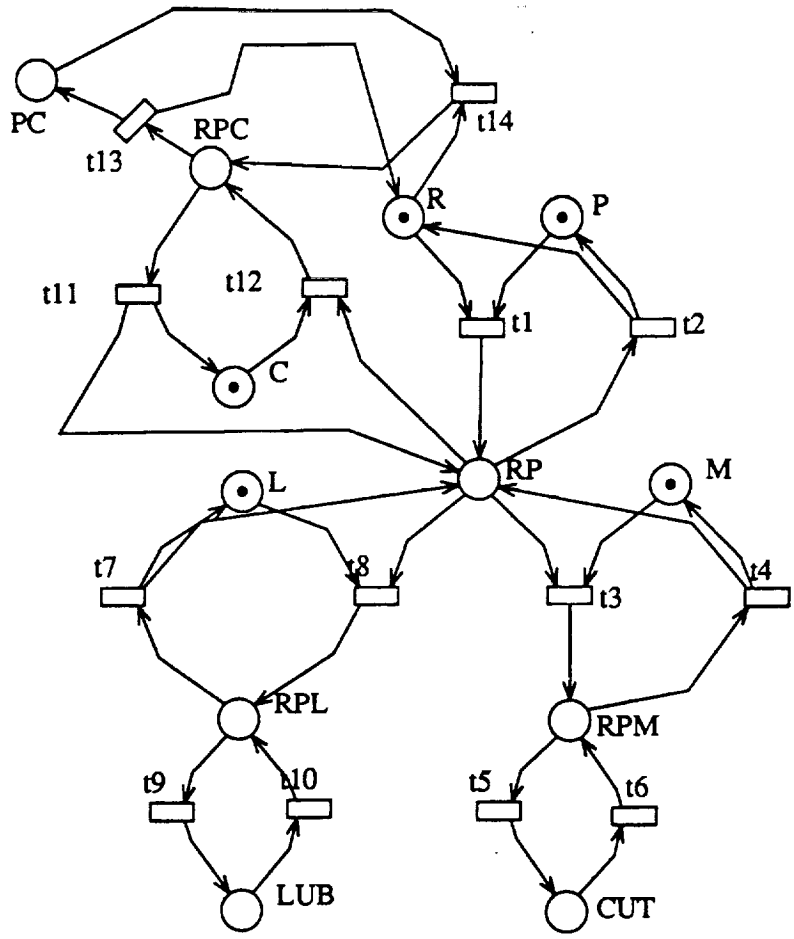


Figure 6.3: The ordinary Petri net mapped from the AND/OR net in Figure 6.2.

and all possible geometric relations among objects, and the geometric constraints among on-line operations. The Petri net mapped from the AND/OR net[11] for this example is illustrated in Figure 6.3. The mapping is based on an algorithm which decomposes each arc in the AND/OR net to two transitions in opposite directions based on feasibility criteria. The resulting Petri nets guarantee the properties of liveness, 1-boundedness, safeness, and reversibility under the assumption of transition feasibility.

The geometric state vector for this assembly scenario is

$$\underline{GS} = (s(R)\ s(P)\ s(C)\ s(L)\ s(M)\ s(RP)\ s(PC)\ s(RPC)\ s(RPL)\ s(RPM)\ s(LUB)\ s(CUT))^T.$$

The value for any element is either 1 or 0 which corresponds to whether a single

component or subassembly is existing in the system at this time. Many different operations sequences could be searched from this Petri net task representation. If we give this Petri net an initial state vector (1 1 1 1 1 0 0 0 0 0 0 0) and a final state vector (1 0 0 1 1 0 1 0 0 0 0 0), a sequence which is $t_1(R, P \rightarrow RP)$, $t_{12}(RP, C \rightarrow RPC)$, $t_{13}(RPC \rightarrow R, PC)$, will be selected as the shortest sequence from all possible sequences. This sequence is actually not a complete sequence because it just picks up the raw peg and inserts it into the hollow cylinder. A new strategy will be necessary to generate only feasible and complete sequences which satisfy the ordered process constraints. Any feasible sequence must include the partial ordering: $CUT \rightarrow LUB \rightarrow INSERT$. The next section introduces an approach to fuzzy marking of the net which implements this process sequencing constraint. In this example, one internal state variable for the peg is its diameter. During the cutting operation, this parameter will be changed. The other internal state variable is the surface lubricating state of the peg. These internal states will be mapped to a global fuzzy variable membership function and carried by the tokens flowing in the net.

6.3 Fuzzy Sets for Modeling System State

Fuzzy set theory[119] has been applied to fuzzy production rules[86], fuzzy control[72], fuzzy expert systems[65], sensor fusion[107], pattern recognition[57], and other interesting areas. Fuzzy logic and its applications provide an effective means of capturing the approximate, inexact nature of the real world. In this chapter, this methodology is used to describe the imprecise characteristics of processing and assembling operations in a robotic assembly or material handling system. The use of fuzzy reasoning rules in this application simplifies the representation and search problems for task planning where correct sequences do not depend on exact knowledge of internal states, but only their precedence relations.

6.3.1 Fuzzy Sets

Fuzzy sets have been used as a broad conceptual framework for dealing with uncertainty and information. For the universe of discourse X, which contains all the possible elements of concern for a particular application, the *crisp set* is defined to dichotomize X to two groups: members(those that belong to a subset A) and nonmembers(those that certainly do not). Because of vagueness in dividing members of the class from nonmembers, a fuzzy set is introduced by assigning to each possible individual in X a value representing its grade of membership in the set. For a garment handling robotic system[9, 101, 102], during the process of turning a piece of cloth into a pair of trousers, the grade of membership of the object, trousers, would gradually increase. Therefore, this grade corresponds to the degree to which the partial product of trousers is similar to the concept of the trousers. Larger values denote higher degrees of membership of the system object. A *global fuzzy state* of the system is thus defined as a mapping from the objects in the system to a set of membership functions defined for each object, or subsets of objects. The formal definition of the global fuzzy state will be discussed in the following subsection.

In a manufacturing system, 'degree of completion' is one such membership function which characterizes the objects in the system which will be discussed in this chapter. Other such global membership functions might be *test validation* which would monitor the mutual functional suitability of a set of components as they move through a process, or *tolerance compatibility* which would track tolerance relations as the process proceeds.

For a robot or manufacturing system, the crisp universal set X of objects that we have defined is $X = \{O_1, O_2, \ldots, O_n\}$. The global fuzzy variable membership value corresponding to each object or set of objects is $V = \{v(O_1), v(O_2), \ldots, v(O_n), \ldots, v(O_i, O_j, \ldots), \ldots\}$, where $v(O_k)$ is e if no token is available for O_k. Following a certain operations sequence, after the system has been working 1 time unit, 2 time

units, and so on, the global fuzzy states will be labeled as V_1, V_2, ..., V_m. The *support* of a fuzzy set A in the universal set X is the crisp set that contains all the elements of X that have a nonzero membership grade in A, which corresponds to a set of all existing objects with token values not equal to 0 in the system. An α-cut of a fuzzy set A is a crisp set A_α that contains all the elements of the universal set X that have a membership grade in A greater than or equal to the specified value of α, i.e., $A_\alpha = \{x \in X | s(x) \geq \alpha\}$, where $s(x)$ is used here to represent the grade of membership. The set of all levels $\alpha \in [0, 1]$ that represent distinct α-cuts of a given fuzzy set A is called a level set of A. Therefore, $\Lambda_A = \{\alpha | s(x) = \alpha$ for some $x \in X\}$, where Λ_A denote the level set of fuzzy set A defined on X. Finally, the scalar cardinality of a fuzzy set A is defined on membership grades of all the elements of X in A. Thus $|A| = \sum_{x \in X} s(x)$. For the set of objects in the systems considered here, the scalar cardinality is changing throughout the processing because the number and values of tokens are changing when the task is executed.

6.3.2 Fuzzy Petri net

In [17, 18], we proposed the definition of the generalized fuzzy Petri net which includes three types of fuzzy variables: local fuzzy variable, fuzzy marking variables, and global fuzzy variables. Local fuzzy variables are used to model the uncertainty of the local variables('internal state') of objects; fuzzy marking variables are used to indicate the uncertainty that events have occurred; and global fuzzy variables are used to represent the characteristic variables of the global task. In this chapter, since we discuss the task sequencing of operations sequences, only global fuzzy variables are used to model the degrees of completion for different global subtasks. Therefore, the following presentation of the definition of the fuzzy Petri net only includes global fuzzy variables. The task sequencing problem is solved by imposing numerical constraints incorporated in the token values in the places. Similarly, the

fuzzy transition functions defined in the net also operate on the values carried by tokens.

Definition 6.1 *A fuzzy Petri net(FPN) with global fuzzy variables* is defined as an 8-tuple:

$$FPN = (P,\ T,\ Q_t,\ I_v,\ \alpha,\ \beta,\ m_t,\ \mu_f),$$

where

1) $P = \{p_1,\ p_2,\ \ldots,\ p_n\}$ is a finite set of *places*, $n \geq 0$.

2) $T = \{t_1,\ t_2,\ \ldots,\ t_m\}$ is a finite set of *transitions*, $m \geq 0$. $P \bigcap T = \emptyset$.

3) $Q_t = \{q_1,\ q_2,\ \ldots,\ q_l\}$ is a finite set of *state tokens*, $l \geq 0$.

4) $I_v = \{I_{v_1}, I_{v_2}, \ldots, I_{v_n}\}$ is a set of internal state variables which are associated with corresponding objects or places. This mapping can be described as $P \rightarrow I_v$. Global fuzzy variables are modified by changing internal state variables when the net is executed.

5) $\alpha \subseteq \{P \times T\}$ is the input function, a set of directed arcs from places to transitions. We call each p_i where $(p_i, t_j) \in \alpha$ as an *input place* of t_j.

6) $\beta \subseteq \{T \times P\}$ is the output function, a set of directed arcs from transitions to places. We call each p_i where $(t_j, p_i) \in \beta$ as an *output place* of t_j.

7) $m_t : Q_t \rightarrow \{\bigcup_i (k_i, \sigma_{k_i}),\ C\}$ is a mapping from a token to a union of 2-tuples of k_i and the k_ith *global fuzzy variable*, σ_{k_i}, or, to a constant, C, which indicates no global fuzzy variable is attached to the token. σ_{k_i} is a membership function in a universe of discourse. In the following discussion we will often refer to the value of the global fuzzy variable as the 'token value'.

8) μ_f: $T \rightarrow \{f_1, f_2, \ldots, f_m\}$ is an association function, a mapping from transitions to corresponding *reasoning functions*. A reasoning function f_i maps a set of tokens in input places to another set of tokens in output places.

In this chapter, global fuzzy state S_g, which was defined in Section 5.3, is used to model a system state. $\sigma(p_i)$ is written as σ^i.

In the following discussion, each token, or group of tokens, will map to a value, $[0, 1]$, of each fuzzy membership function with which it is associated. In this sense, the *universe of discourse* of the fuzzy set includes the prior and current system states(markings and internal states). The *reasoning functions* associated with the transitions, update the fuzzy membership values as the net executes. For the example of peg-cylinder assembly(Figure 6.1), we define key variables associated with cutting, lubricating, and inserting of the peg. A simplified view of the resulting fuzzy membership function is shown schematically in Figure 6.4. The horizontal axes show the partial internal states represented by cutting and lubrication, while the vertical axis indicates the degree of membership in the fuzzy set *task completion.* Clearly the task is not ready for completion without both cutting and lubrication, and cutting to a specific diameter should precede lubrication. Figure 6.4 shows these conceptual relations, while, in practice, these mappings are carried out by the reasoning functions attached to the FPN transitions.

A transition in a fuzzy Petri net may be enabled when the token values of its input places satisfy some specified fuzzy reasoning rule. For example, one such reasoning rule for assembly requires the token values of all objects to be assembled have values not less than a value θ, and therefore be members of a designated level set. If a transition is chosen to fire, all tokens in the input places are removed and fuzzy tokens are added to its output places, which may contain values different from the input values. The values of new tokens will depend on the fuzzy reasoning rule of the transition.

For the three types of transitions, *assembly*, *disassembly*, and IST operations described in Section 6.2, we can define the following fuzzy transition rules. Initially, we assume only one soft component exists in the system. Therefore, we may use only σ_j to represent $\bigcup_i(k_i, \sigma_{k_i})$, as indicated in the FPN definition, and $C = 1$. The case of multiple soft components will be discussed in a later section. Each fuzzy

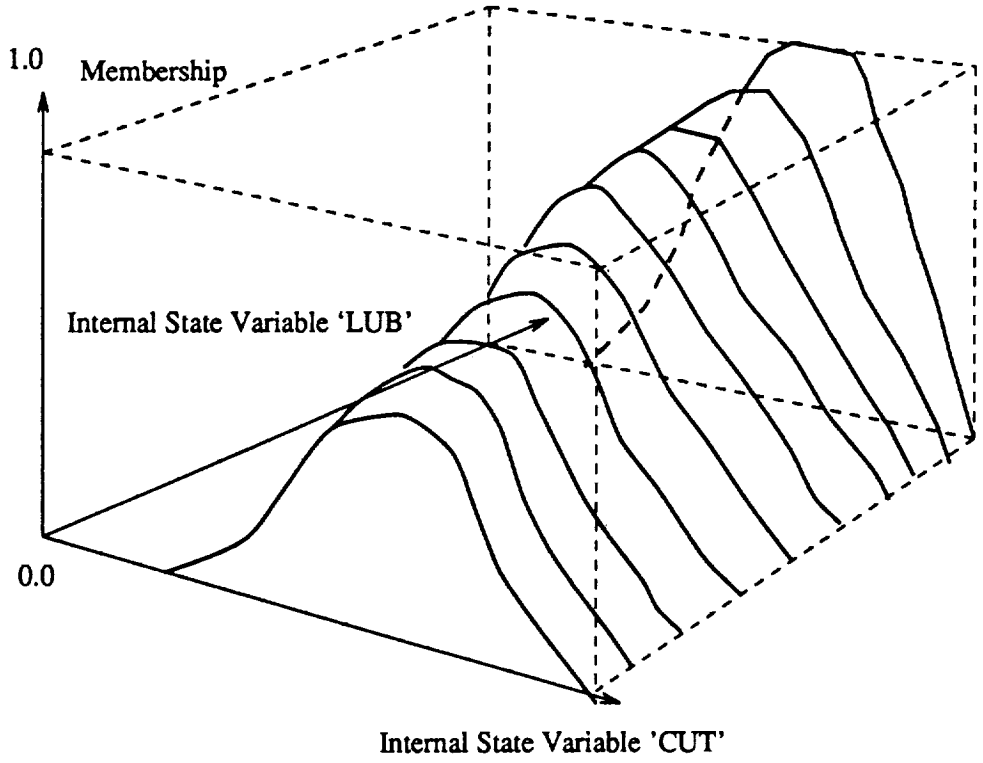


Figure 6.4: Conceptual diagram of the fuzzy membership function for the global fuzzy variable 'task completion' in the peg-cylinder assembly task. The horizontal axes are internal state variables for 'cutting' and 'lubrication', and are not a complete state description. In practice, this membership function is executed using a transition reasoning function.

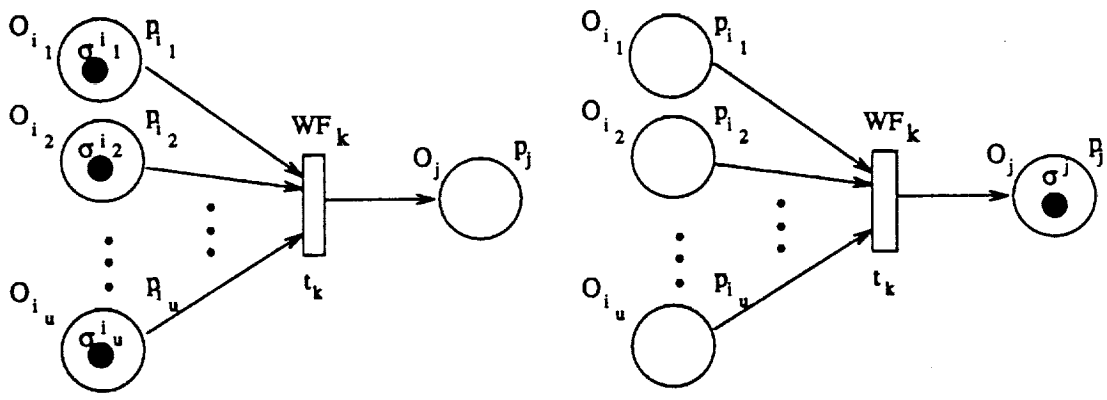


Figure 6.5: Fuzzy Petri net representation for assembly transition.

transition/reasoning function is defined by an integer, which is called a weighting factor, WF. In practice, the weighting factors and the thresholds are adjusted to define appropriate level sets for the transitions sequences. WF is defined as $T \rightarrow \{1, 2, 3, \ldots\}$, a mapping from transitions to integer values.

Assembly operation: $O_{i_1}, O_{i_2}, \ldots, O_{i_u} \rightarrow O_j$. The fuzzy Petri net corresponding to the assembly operation is shown in Figure 6.5.

$$if\ min(\sigma^{i_1}, \sigma^{i_2}, \ldots, \sigma^{i_u}) \geq \theta,\ then\ \sigma^j = min(\sigma^{i_1}, \sigma^{i_2}, \ldots, \sigma^{i_u}) \times WF_k. \qquad (6.4)$$

Disassembly operation: $O_i \rightarrow O_{j_1}, O_{j_2}, \ldots, O_{j_l}$. The fuzzy Petri net corresponding to the disassembly operation is shown in Figure 6.6.

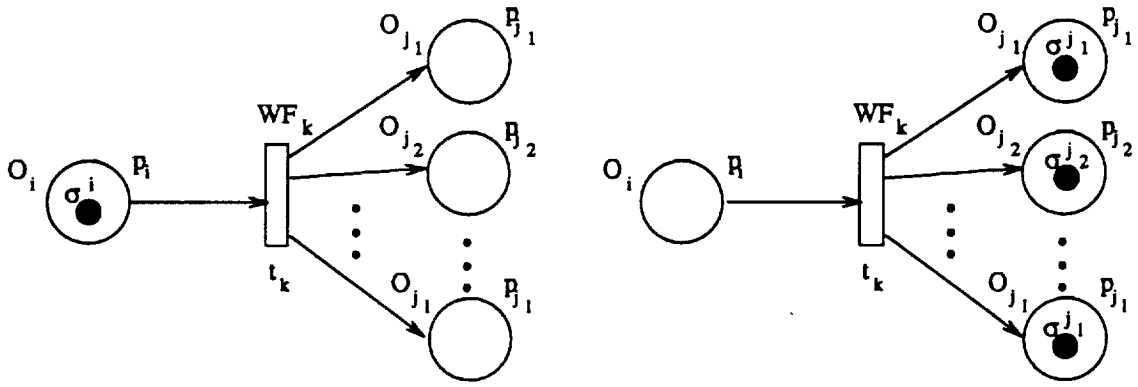
$$if\ \sigma^i \geq \theta,\ then\ \sigma^{j_d} = \sigma^i \times WF_k \times soft(O_{j_d}) + 1 - soft(O_{j_d}), \qquad (6.5)$$

where

$$soft(O_{j_d}) = \begin{cases} 1 & \text{if } O_{j_d} \text{ is a soft object,} \\ 0 & \text{otherwise,} \end{cases} \qquad 1 \leq d \leq l.$$

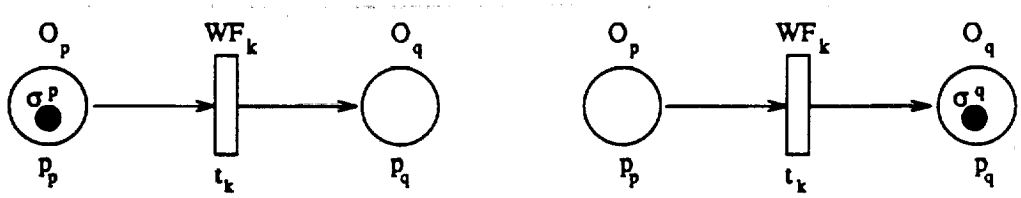
IST operation: $O_p \rightarrow O_q$. The fuzzy Petri net corresponding to the IST operation is shown in Figure 6.7.

$$if\ \sigma^p \geq \theta,\ then\ \sigma^q = \sigma^p \times WF_k. \qquad (6.6)$$



(a) Before t_k is fired. (b) After t_k is fired.

Figure 6.6: Fuzzy Petri net representation for disassembly transition.



(a) Before t_k is fired. (b) After t_k is fired.

Figure 6.7: Fuzzy Petri net representation for IST transition.

6.4 An Algorithm for Assigning Global Fuzzy Variables

In this section, we propose an algorithm which maps an ordinary Petri net to a fuzzy Petri net based on the assumption that a single soft component exists in the system, and the precedence of key transitions which modify internal states of the objects which include this component are known *a priori*.

6.4.1 Prime Number Marking Algorithm

The prime number marking algorithm is based on the *fundamental theorem of arithmetic*. This algorithm assigns weighting factors to all transitions in the ordinary Petri net and initial token values to all corresponding places. The fuzzy Petri net generated using this algorithm assigns prime numbers to transitions, and these may be mapped onto the token values for degrees of completion of the task. We call the token values generated by this algorithm *prime token values*. All feasible and complete sequences which contain all key operations to change the properties of the soft objects will be found in the fuzzy Petri net. These sequences are also guaranteed to have correct precedence relationships among operations.

Prime Number Marking Algorithm

Input: an ordinary Petri net mapped from an AND/OR net, the prime number table, soft component $\tilde{C}$, the number of steps for changing properties of $\tilde{C}$, s.

Output: a fuzzy Petri net.

step 1: Initialization. For all transitions $t_i (1 \leq i \leq m)$, the weighting factor of t_i, $WF_i = 1$.

step 2: Pick the first s prime numbers $P_1, P_2, \ldots, P_s$ in the prime number table.

step 3: For all IST transitions, pick those transitions which change the properties of $\tilde{C}$, and order them according to the required sequence of operations. Suppose

these transitions are $t_{q_1}, t_{q_2}, \ldots, t_{q_s}$.

step 4: Set $WF_{q_1} = P_1, WF_{q_2} = P_2, \ldots, WF_{q_i} = P_i, \ldots, WF_{q_s} = P_s$.

step 5: Map the initial marking from the original Petri net. For each place p_j, $1 \leq j \leq n$, set the corresponding token q_i to the same value as in the ordinary Petri net. If place p_j corresponds to an object which represents or contains $\tilde{C}$, select a positive integer T, such that

$$0.1 < 10^{-T} \times WF_{q_1} \times WF_{q_2} \times \ldots \times WF_{q_s} \leq 1.0. \tag{6.7}$$

step 6: Change the value of σ^j to 10^{-T}.

Proposition 6.1 After the original Petri net is mapped to the fuzzy Petri net, the initial global state for the fuzzy Petri net contains the token values 0 or 1, and the prime token value for the soft object is 10^{-T}. The final global state for the fuzzy Petri net contains the token values of 0 or 1, and the prime token value for the soft object is $10^{-T} \times P_1 \times P_2 \times \ldots \times P_s$.

6.4.2 Interpretation of Prime Token Values

We assumed initially that only one soft component existed and described a global fuzzy values assignment algorithm based on prime number sequences of weighting factors. This assignment guarantees the generation of feasible, complete, and correctly ordered sequences, and defines a method to describe the degrees of completion for soft objects. To understand and interpret the prime token values, we convert them to fuzzy values uniformly distributed between 0 and 1 so that the resulting token value of the soft object in the final global state is 1.

The possible prime token values for the soft objects are:

$$10^{-T}, 10^{-T} \times P_1, 10^{-T} \times P_1 \times P_2, \ldots, 10^{-T} \times P_1 \times P_2 \times \ldots \times P_s,$$

where T satisfies (6.7). In order to map them to the discrete points on a uniformly-distributed unit range $[0,1]$, we use x_0, x_1, x_2, ..., x_s to represent these possible token values, and we use a function $f(x_i)$ to represent the resulting fuzzy values. Therefore, $f(x_0) = 0$, and

$$f(x_i) = f(x_{i-1}) + \frac{1}{s}, \qquad i = 1, 2, \ldots, s.$$

All feasible fuzzy values here can be considered as the proportions that the first i key transitions contribute compared to all s transitions. An interpretation sequence can be generated as follows: 0(not yet processed), $\frac{1}{s}$(the first key transition has been fired and others have not yet been fired), $\frac{2}{s}$(the first two key transitions have been fired and others have not yet been fired), ..., $\frac{s-1}{s}$(the first $s-1$ key transitions have been fired and the last one has not yet been fired), 1(all key transitions have been sequentially fired). With these $s+1$ points in a two-dimensional coordinate frame, we can obtain an equation for a curve which passes through these points. We use *Lagrange's Interpolation Formula*[91] to derive the formula for this equation. Lagrange's Formula is used because $x_{i+1} - x_i$ is not a constant. Lagrange's Interpolation Formula is:

$$f(x) = \frac{(x-x_1)(x-x_2)(x-x_3)\ldots(x-x_n)}{(x_0-x_1)(x_0-x_2)(x_0-x_3)\ldots(x_0-x_n)}y_0 + \frac{(x-x_0)(x-x_2)(x-x_3)\ldots(x-x_n)}{(x_1-x_0)(x_1-x_2)(x_1-x_3)\ldots(x_1-x_n)}y_1$$
$$+\ldots+\frac{(x-x_0)(x-x_1)(x-x_2)\ldots(x-x_{n-1})}{(x_n-x_0)(x_n-x_1)(x_n-x_2)\ldots(x_n-x_{n-1})}y_n, \tag{6.8}$$

where (x_0, y_0), (x_1, y_1), (x_2, y_2), ..., (x_n, y_n) are points already known. For our assumption, the interpolation points are: $(10^{-T}, 0)$, $(10^{-T}P_1, \frac{1}{s})$, $(10^{-T}P_1P_2, \frac{2}{s})$, $(10^{-T}P_1P_2P_3, \frac{3}{s})$, ..., $(10^{-T}P_1P_2\ldots P_s, 1)$. For the fuzzy Petri net representation, this yields:

$$f(x) = \frac{1}{s}\frac{(x-10^{-T})(x-10^{-T}P_1P_2)(x-10^{-T}P_1P_2P_3)\ldots(x-10^{-T}P_1\ldots P_s)}{(10^{-T}P_1-10^{-T})(10^{-T}P_1-10^{-T}P_1P_2)(10^{-T}P_1-10^{-T}P_1P_2P_3)\ldots(10^{-T}P_1-10^{-T}P_1\ldots P_s)}$$
$$+\frac{2}{s}\frac{(x-10^{-T})(x-10^{-T}P_1)(x-10^{-T}P_1P_2P_3)\ldots(x-10^{-T}P_1\ldots P_s)}{(10^{-T}P_1P_2-10^{-T})(10^{-T}P_1P_2-10^{-T}P_1)(10^{-T}P_1P_2-10^{-T}P_1P_2P_3)\ldots(10^{-T}P_1P_2-10^{-T}P_1\ldots P_s)}$$
$$+\ldots$$
$$+1\frac{(x-10^{-T})(x-10^{-T}P_1)\ldots(x-10^{-T}P_1\ldots P_{s-1})}{(10^{-T}P_1\ldots P_s-10^{-T})(10^{-T}P_1\ldots P_s-10^{-T}P_1)\ldots(10^{-T}P_1\ldots P_s-10^{-T}P_1\ldots P_{s-1})}. \tag{6.9}$$

Notice that in the above equation, the first point $(10^{-T}, 0)$ vanishes. We suppose $x = 10^{-T} \times x'$, then $x' = 10^{T} \times x$. Therefore, (6.9) is simplified as

$$f(x) = \frac{1}{s}\frac{(x'-1)(x'-P_1P_2)(x'-P_1P_2P_3)\ldots(x'-P_1P_2\ldots P_s)}{(P_1-1)(P_1-P_1P_2)(P_1-P_1P_2P_3)\ldots(P_1-P_1P_2\ldots P_s)}$$
$$+\frac{2}{s}\frac{(x'-1)(x'-P_1)(x'-P_1P_2P_3)\ldots(x'-P_1P_2\ldots P_s)}{(P_1P_2-1)(P_1P_2-P_1)(P_1P_2-P_1P_2P_3)\ldots(P_1P_2-P_1P_2\ldots P_s)}$$
$$+\ldots+\frac{(x'-1)(x'-P_1)\ldots(x'-P_1P_2\ldots P_{s-1})}{(P_1\ldots P_s-1)(P_1\ldots P_s-P_1)\ldots(P_1\ldots P_s-P_1\ldots P_{s-1})}. \qquad (6.10)$$

When we map the prime token values to fuzzy values for each object in the representation, we also change the weighting factors of all transitions in the fuzzy Petri net so that the sequence of fuzzy values can be obtained using the same fuzzy reasoning rules. Suppose there are s key transitions in the system and the initial prime token value for the soft object is 10^{-T}. The updated weighting factors for all key transitions are: $10^{T} \times \frac{1}{n}$, 2, $\frac{3}{2}$, $\frac{4}{3}$, ..., $\frac{n-1}{n-2}$, $\frac{n}{n-1}$. The weighting factors for other transitions and the initial global fuzzy state are left as the same.

In the example of the peg-cylinder assembly system in Section 6.3, the parameters are: $s = 2$, $P_1 = 2$, $P_2 = 3$, and $T = 1$. In this case, (6.10) becomes

$$f(x) = \frac{1}{2}\frac{(x'-1)(x'-6)}{(2-1)(2-6)} + \frac{2}{2}\frac{(x'-1)(x'-2)}{(6-1)(6-2)} = (x'-1)(-\frac{3}{40}x' + \frac{13}{20}).$$

Therefore,

when $x = 0.1$, $x' = 0.1 \times 10^1 = 1$, $f(x) = 0$,

when $x = 0.2$, $x' = 0.2 \times 10^1 = 2$, $f(x) = 1 \times (-\frac{6}{40} + \frac{13}{20}) = 0.5$,

when $x = 0.6$, $x' = 0.6 \times 10^1 = 6$, $f(x) = 5 \times (-\frac{9}{20} + \frac{13}{20}) = 1.0$.

Using the prime number marking algorithm, the ordinary Petri net shown in Figure 6.3 can be mapped into a fuzzy Petri net as shown in Figure 6.8. The initial global fuzzy state is shown in this net. The soft objects in this example are P, RP, RPM, RPL, RPC, PC. The key transitions are t_5 and t_9. Therefore, $WF_5 = P_1 = 2$ and $WF_9 = P_2 = 3$. Because $10^{-1} \times WF_5 \times WF_9 = 0.6$ satisfies

(6.7), σ^2 of the initial global state is thus 0.1. The token value for place PC in the final global state is 0.6.

6.4.3 Feasible Sequences in the Fuzzy Petri Net

The following theorem provides a method to use the feasible global fuzzy states obtained during the generation of the fuzzy Petri net, to search for feasible, complete, and correctly ordered sequences efficiently.

Theorem 6.1 *The Fundamental Theorem of Arithmetic*
If p_i and q_i are positive primes, and if

$$a = \prod_{i=1}^{n} p_i^{\alpha_i} = \prod_{j=1}^{m} q_j^{\beta_j},$$

where

$$1 < p_1 < p_2 < \ldots < p_{n-1} < p_n,$$

and

$$1 < q_1 < q_2 < \ldots < q_{m-1} < q_m,$$

then $n = m$, $p_i = q_i$ and $\alpha_i = \beta_i$, every positive integer has a composition into positive prime factors, which is unique apart from the order of the factors.

Proof. See [94, p. 263].

Theorem 6.2 Using the prime number marking algorithm, all sequences generated from the fuzzy Petri net of which each transition can only be fired at most once, are feasible, complete, and hold correct precedence relationships among key transitions, *if and only if* the places corresponding to soft objects can only hold the following order of token values: $10^{-T}, 10^{-T} \times P_1$, $10^{-t} \times P_1 \times P_2$, ..., $10^{-T} \times P_1 \times P_2 \times \ldots \times P_s$, where $P_1, P_2, \ldots, P_s$ are the first s primes.

Proof. The values of all weighting factors WF_1, WF_2, ..., WF_m are 1, P_1, P_2, ..., P_s, $1 \leq s \leq m$. If the sequences are feasible, all feasible token values are 10^{-T},

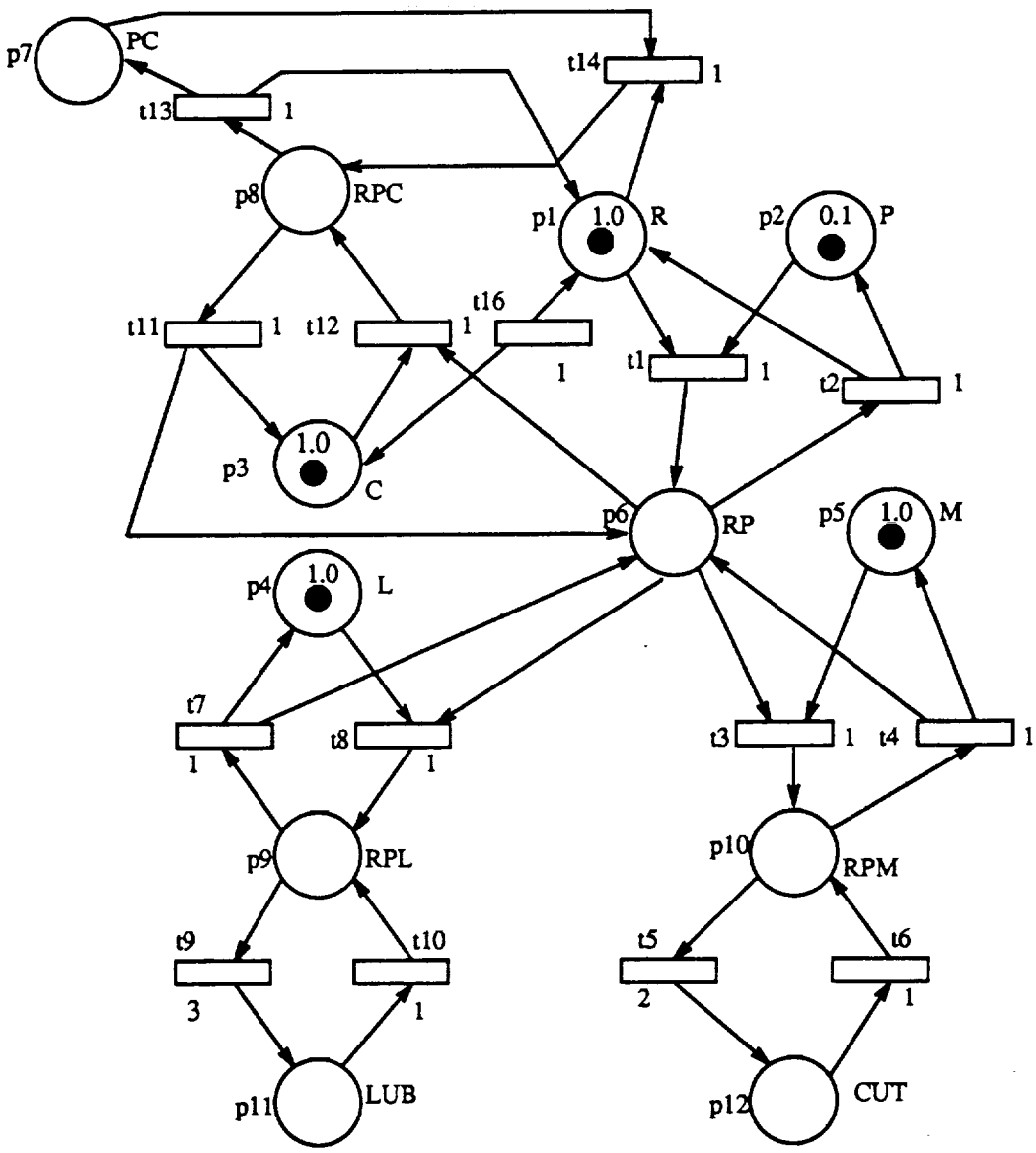


Figure 6.8: The fuzzy Petri net mapped from the previous ordinary Petri net.

$10^{-T} \times P_1$, $10^{-T} \times P_2$, ..., $10^{-T} \times P_s$, $10^{-T} \times P_1 \times P_2$, ..., $10^{-T} \times P_1 \times \ldots \times P_s$, i.e., $C_0^s + C_1^s + C_2^s + \ldots + C_s^s = 2^s$ possible token values. If the sequences also hold correct precedence relationships on key transitions, then the weighting factor of the first key transition that the *developing* sequences meet should be WF_1. All possible token values are now reduced to 10^{-T}, $10^{-T} \times P_1$, $10^{-T} \times P_1 \times \Theta$ where Θ is a product of at most $s-1$ elements which do not contain P_1. Therefore, if the developing sequences meet the second weighting factor which is not equal to 1, all possible token values are now reduced to 10^{-T}, $10^{-T} \times P_1$, $10^{-T} \times P_1 \times P_2$, $10^{-T} \times P_1 \times P_2 \times \Theta'$, where Θ' is a product of at most $s-2$ elements which do not contain P_1 and P_2. Continuing this procedure, all possible token values that the developing sequences will meet are: 10^{-T}, $10^{-T} \times P_1$, $10^{-T} \times P_1 \times P_2$, ..., $10^{-T} \times P_1 \times P_2 \times \ldots \times P_s$, i.e., $s+1$ possible states. If the sequences are also complete, the sequence should follow all possible s weighting factors which are not equal to 1. Therefore, the sequences will meet all possible token values. The necessary part of the theorem is thus proved.

Suppose the soft objects hold the following sequence of token values: 10^{-T}, $10^{-T} \times P_1$, $10^{-T} \times P_1 \times P_2$, ..., $10^{-T} \times P_1 \times P_2 \times \ldots \times P_s$. Notice that at this time, we are given all results of products. Using Theorem 6.1, we will get unique compositions into positive prime factors and 10^{-T}, and then order the prime factors inside the form of products. A unique ordered sequence for each token value will be obtained. The first value corresponds to firing any number of transitions(with no duplication) which do not contain a weighting factor not equal to 1. The second value shows that besides firing any number of transitions which do not contain a weighting factor of a key transition, t_{q_1} is also fired, and then any number of transitions of a weighting factor equal to 1 can be fired. We continue this enumeration and find all values in the above sequence are feasible. The order of the transitions in the sequence obviously holds the correct precedence relationships, i.e., we should fire t_{q_1} first, and then fire t_{q_2}, and so on, and at last fire t_{q_s}. Moreover, the sequences contain all

possible copies of key transitions and are therefore complete.

Q.E.D. □

The corollaries listed below directly follow from Theorem 6.2.

Corollary 6.1 For a fuzzy Petri net marked using the prime number marking algorithm, $m_t(Q_t) = \{1,\ 10^{-T}, 10^{-T} \times P_1,\ 10^{-T} \times P_1 \times P_2,\ \ldots,\ 10^{-T} \times P_1 \times P_2 \times \ldots \times P_s\}$.

Corollary 6.2 The search in the fuzzy Petri net is halted at a token value $\sigma^i \notin m_t(Q_t)$, but will not exclude any feasible, complete sequence which has correct precedence relationships among operations.

Corollary 6.2 is used to search all possible correct sequences which satisfy the three properties from the fuzzy Petri net off-line. When a set of enabled transitions are found for the development of partial sequences, those transitions which lead to undesirable token values in the corresponding output places are discarded.

6.4.4 Multiple Assigned Key Transition Sequences

In the previous sections, it was assumed that the order of the key transitions is assigned in advance and only one order is feasible. However, in practice, it may often occur that more than one partial ordering of key transitions are possible. For example, suppose t_a and t_b are key transitions, both sequences $\ldots t_a \ldots t_b \ldots$ and $\ldots t_b \ldots t_a \ldots$ may be feasible. The plan representation should include feasible and complete sequences which satisfy either ordering of key transitions.

The problem of multiple partial orderings may be solved in a straightforward manner by enumerating all possible orderings of process steps and constructing the union of the plans from each set.

In general, if we have k feasible assigned key transition sequences, $S_1, S_2, \ldots, S_k$. For each S_i, we use the planning strategy described above and obtain all feasible, complete, and correctly ordered sequences represented as $\{plan(S_i)\}$ which is a set of sequences. Then the final complete sequence set can be obtained as $\bigcup_{i=1}^{k} \{plan(S_i)\}$.

6.5 Fuzzy Representation for Multiple Soft Components

Often several soft components may be present in a complex system. For example, more than one type of part might need to be processed in a given system, and then be assembled. The properties for all these soft components may change. In this case, a more general marking and sequencing algorithm is required to permit correct reasoning when multiple soft components take part in a transition. This section describes an algorithm which provides a consistent strategy and reduces to the previous reasoning mechanism when a single soft component is present.

6.5.1 Fuzzy Reasoning for Multiple Soft Components

We assume we have r soft components labeled as $\tilde{C}_1, \tilde{C}_2, \ldots, \tilde{C}_r$, respectively. For any soft object O:

$$soft_j(O) = \begin{cases} 1 & \text{if } O \text{ is the j type soft object,} \\ 0 & \text{otherwise,} \end{cases} \quad 1 \leq j \leq r.$$

An assembly or subassembly may contain both $\tilde{C}_j$ and other soft components:

$$soft_{ij\ldots k}(O) = \begin{cases} 1 & \text{if } O \text{ is the } i\text{-}j\text{-}\ldots\text{-}k \text{ type soft object,} \\ 0 & \text{otherwise,} \end{cases} \quad 1 \leq i, j, \ldots, k \leq r.$$

The i-j-...-k type soft object is defined as an assembly or subassembly containing all the soft components labeled as $\tilde{C}_i$, $\tilde{C}_j$ ..., and $\tilde{C}_k$, and no other soft components.

Based on these definitions, a strategy which uses a prime number sequence to represent key transitions with multiple soft components was presented in [12]. In that approach, P_1, P_{r+1}, P_{2r+1}, ..., $P_{(s_1-1)r+1}$ was used to represent the 1st subsequence of prime numbers for marking the key transitions corresponding to the 1st soft component. P_2, P_{r+2}, P_{2r+2}, ..., $P_{(s_2-1)r+2}$ was used to represent the 2nd subsequence of prime numbers marking those corresponding to the 2nd soft component. Continuing this procedure, P_r, P_{2r}, P_{3r}, ..., $P_{(s_r-1)r+r(=s_r r)}$ was used

to represent the rth subsequence of prime numbers for the rth soft component. In this chapter, we will use a related, but simpler, approach which assigns the same sequence of prime numbers for the key transitions corresponding to different kinds of labeled, or 'colored', soft components. We can generalize the representations in (6.4), (6.5) and (6.6) to obtain the corresponding fuzzy reasoning rules for output places for three types of transitions, as follows:

Assembly operation: $O_{i_1}, O_{i_2}, \ldots, O_{i_u} \rightarrow O_j$. The objects $O_{i_1}, O_{i_2}, \ldots, O_{i_u}, O_j$ may contain more than one soft component. Therefore, the representation for those kinds of soft objects should be distinguished from other objects and at any given time, we should be able to reason about the characteristics of these objects. Before we give the function for the assembly operation, we first give the definition of *min* and *trs* for each object. We also assume t_v will function on the jth soft component.

$$\textit{if } soft_j(O_{i_s}) = 1, \textit{ then } min(O_{i_s}) = \sigma_j \textit{ and } trs(O_{i_s}) = (j, \sigma_j \times WF_v),$$

$$\textit{if } soft_{ij\ldots k}(O_{i_s}) = 1, \textit{ then } min(O_{i_s}) = min(\sigma_i, \sigma_j, \ldots, \sigma_k) \textit{ and}$$

$$trs(O_{i_s}) = (i, \sigma_i) \bigcup (j, \sigma_j \times WF_v) \bigcup \ldots \bigcup (k, \sigma_k),$$

$$\textit{otherwise}, \; min(O_{i_s}) = 1 \textit{ and } trs(O_{i_s}) = \emptyset,$$

$$1 \leq s \leq u.$$

The generalization of formula (6.4) is:

$$\textit{if } min(min(O_{i_1}), min(O_{i_2}), \ldots, min(O_{i_u})) \geq \theta, \textit{ then}$$

$$\sigma^j = soft(O_j) \times \bigcup_{s=1}^{u} trs(O_{i_s}) + 1 - soft(O_j),$$

where

$$soft(O_j) = \sum_i soft_i(O_j) + \sum_{ik} soft_{ik}(O_j) + \ldots + \sum_{12\ldots r} soft_{12\ldots r}(O_j). \qquad (6.11)$$

Disassembly operation: $O_i \rightarrow O_{j_1}, O_{j_2}, \ldots, O_{j_l}$. The generalization of formula (6.5) is:

$$if\ min(O_i) \geq \theta,\ then\ \sigma^s = soft(O_s) \times trs(O_s) + 1 - soft(O_s),$$

$$j_1 \leq s \leq j_l. \tag{6.12}$$

IST operation: $O_p \rightarrow O_q$.

$$if\ min(O_p) \geq \theta,\ then\ \sigma^q = soft(O_q) \times trs(O_q) + 1 - soft(O_q). \tag{6.13}$$

6.5.2 Generalized Prime Number Marking Algorithm

For a system with more than one soft component, we should know the status of the current objects at each step of the process, i.e., which soft components the objects contain, and the degree of completion for this soft object as well as every soft component it contains. For the sake of simplicity, we assume all soft components in the system are independent of each other, i.e., there are no relations between the orders of key transitions for any two soft components. Moreover, we still want to guarantee the precedence relationships among operations for each soft component, so that the combinations of several soft components will still hold this property. The prime number marking algorithm is generalized as follows:

Generalized Prime Number Marking Algorithm

Input: the ordinary Petri net mapped from an AND/OR net, the prime number table, soft components $\tilde{C}_1, \tilde{C}_2, \ldots, \tilde{C}_r$, the numbers of steps for changing properties for each soft component $S_1, S_2, \ldots, S_r$.

Output: a fuzzy Petri net.

step 1: Initialization. For all transitions $t_j (1 \leq j \leq m)$, the weighting factor of t_j, $WF_j = 1$.

step 2: For all IST transitions, pick those transitions which make the changes of properties for $\tilde{C}_i$, $1 \leq i \leq r$, and order them according to the required sequence of changes. Suppose these transitions are $t_1^i, t_2^i, \ldots, t_{S_i}^i$.

step 3: For $1 \leq i \leq r$, set $WF_1^i = P_1, WF_2^i = P_2, \ldots, WF_{s_i}^i = P_{S_i}$.

step 4: Map the initial marking from the original Petri net. For each place p_j, $1 \leq j \leq n$, set the corresponding token value σ^i to have the same token value as in the ordinary Petri net.

step 5: For each $\tilde{C}_i$, $1 \leq i \leq r$, select a positive integer T_i, such that

$$0.1 < 10^{-T_i} \times WF_1^i \times WF_2^i \times \ldots \times WF_{S_i}^i \leq 1.0. \tag{6.14}$$

step 6: Mark σ^j the value of $(i, 10^{-T_i})$, if O_j contains only $\tilde{C}_i$; Mark σ^j the value of $(i_1, 10^{-T_{i_1}}) \bigcup (i_2, 10^{-T_{i_2}}) \bigcup \ldots (i_u, 10^{-T_{i_u}})$, if O_j contains soft components $\tilde{C}_{i_1}$, $\tilde{C}_{i_2}, \ldots, \tilde{C}_{i_u}$, $1 \leq u \leq r$.

An example of this case is obtained by adding to Figure 6.1 a visual sensor mounted near the gripper of the robot arm. Before the peg is inserted into the hollow cylinder, the robot should move near the cylinder to sense the exact insertion position. The sensing operation refines the internal state(size and position) of the cylinder, and for this example the cylinder C becomes a soft object. This sensing operation can be done anytime the robot is not holding anything. We model the combination of the two interacting components R and C as a subassembly marked RC. The sensing process is represented as SEN. The corresponding updated AND/OR net and the ordinary Petri net is shown in Figure 6.9 and Figure 6.10, respectively.

For this example, the resulting updated fuzzy Petri net is shown in Figure 6.11. There are two soft components, peg P and hollow cylinder C in the system. The parameters are $r = 2$, $S_1 = 2$ and $S_2 = 1$. t_5 and t_9 are key transitions for P

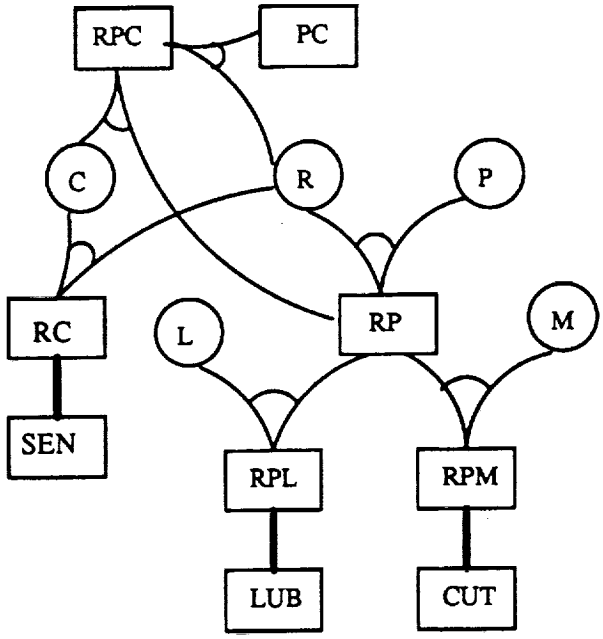


Figure 6.9: The updated AND/OR net.

and t_{17} is the key transition for C. Therefore, $WF_5 = P_1 = 2$, $WF_9 = P_2 = 3$, and $WF_{17} = P_1 = 2$. Because $10^{-1} \times WF_5 \times WF_9 = 0.6$ and $10^{-1} \times WF_{17} = 0.2$ satisfy (6.14), $\sigma^2 = (1, 0.1)$ and $\sigma^3 = (2, 0.1)$. The token value for place PC in the final marking is $(1, 0.6) \bigcup (2, 0.2)$.

6.5.3 Interpretation of Fuzzy Values for Multiple Soft Components

We may interpret the case of multiple soft components in analogy to 'colored' Petri net models[54]. In this context, we say different soft components have different *colors*. A soft object may contain different kinds of soft components, and we call this object a *composite object*. However, we assume that the colors of soft components in one object will not be mixed up, i.e., the colors are independent. Knowing the number of soft components in the system as well as the sequence of key operations for each soft component, we can map the the weighting factors of these key transitions sequences of prime numbers depending on the color of the soft object. Therefore, at any given time, we can get a unique decomposition of primes corresponding to

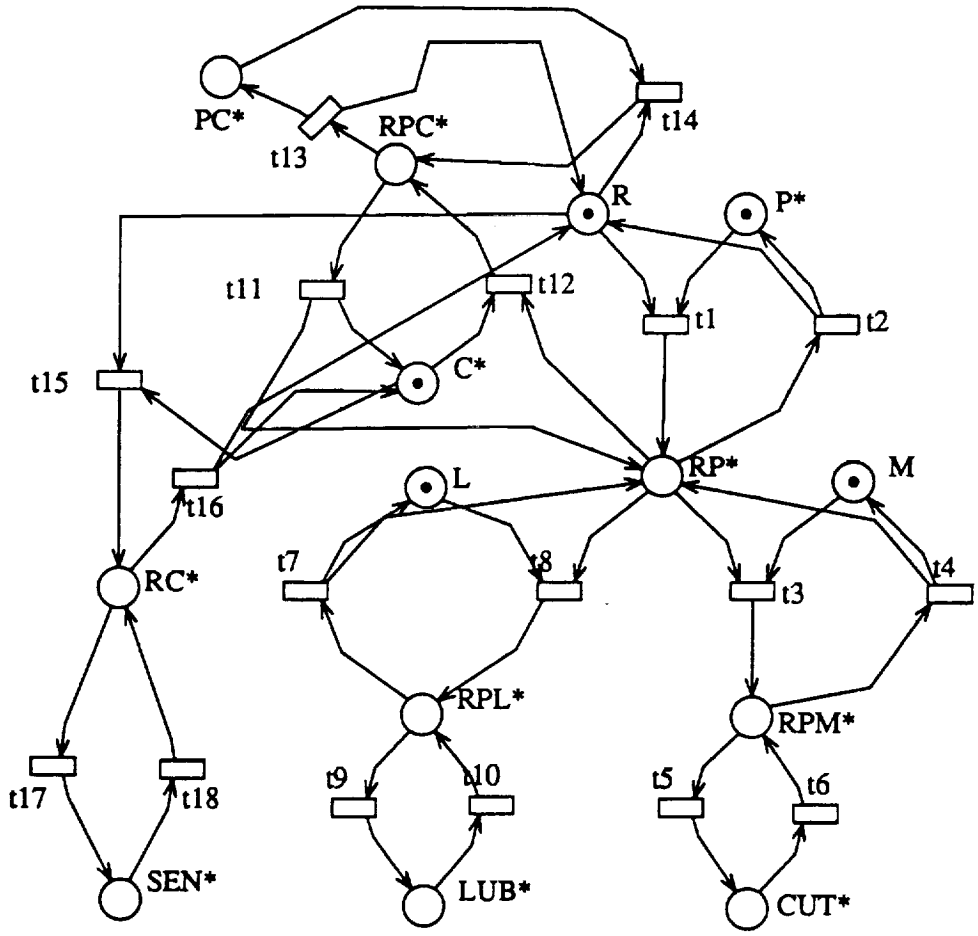


Figure 6.10: The updated ordinary Petri net. The names of soft objects are followed by a symbol: "*".

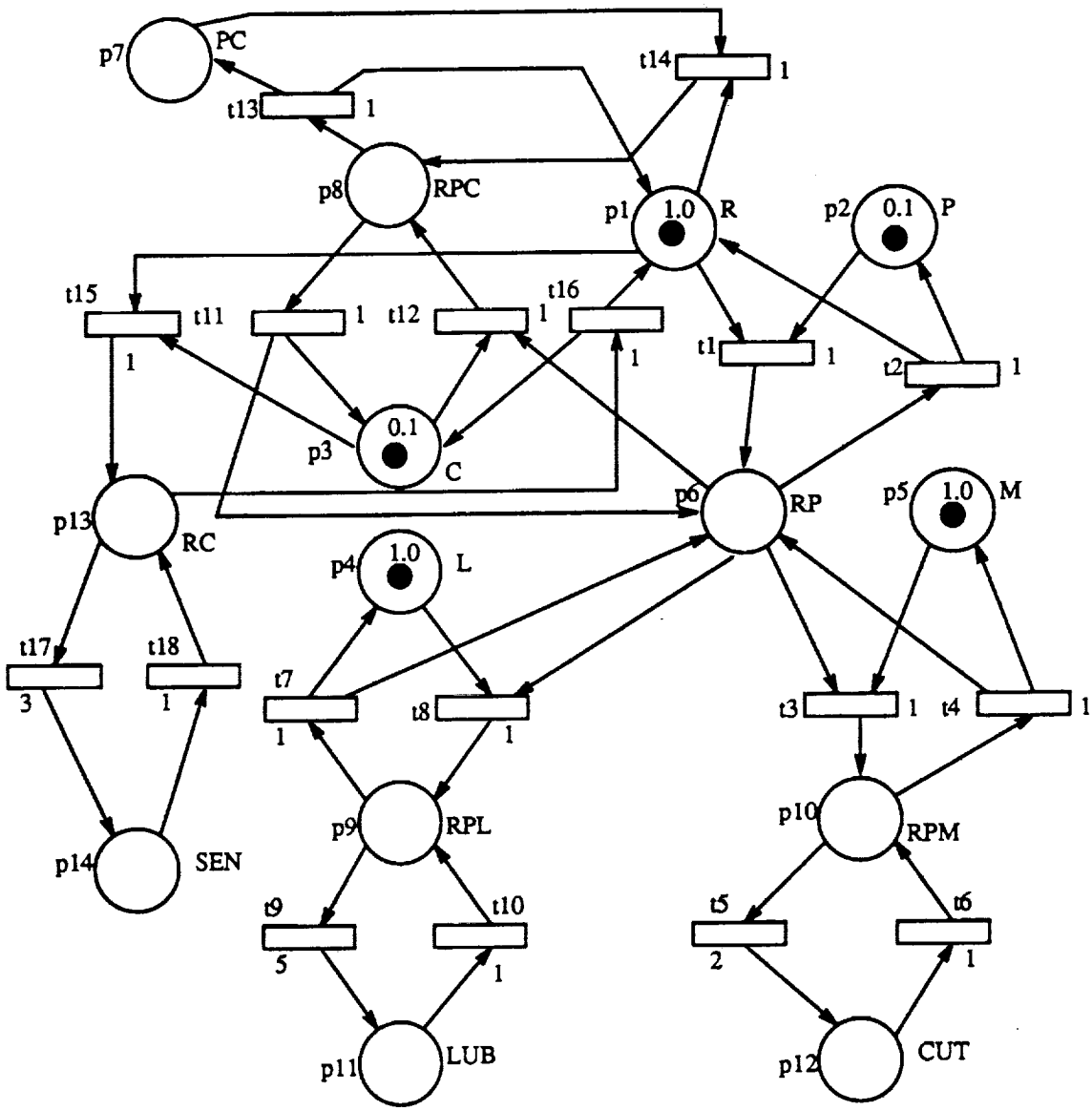


Figure 6.11: The fuzzy Petri net mapped from the updated ordinary Petri net.

one soft component such that a correct precedence relationship can be verified to be followed.

As in the case of a single soft component, we can map the prime token values to fuzzy values uniformly distributed in $[0, 1]$ for each different color of the object. For a soft object which contains more than one soft component, the degree of completion of this soft object can be understood by looking at the degrees of completion of the soft components independently. For the object PC shown in Figure 6.11, if the cutting job has been done on P, and neither sensing on C nor lubricating on P starts, the interpreted fuzzy value of P is 0.5 and the value of C is 0, as discussed in the last section. After we finish sensing on C and lubricating P still doesn't start, the value of P is 0.5 and the value of C is 1. After all jobs are finished on soft components P and C, the value of P and C are both 1.

Because the prime number representation and the marking for different colors keep token values of soft components inside a soft object independent, it is convenient to reason about the degree of completion for any soft component in an object. If we map the prime token values for multiple soft component case to fuzzy values, we are also able to obtain straightforward modification for weighting factors of transitions to keep fuzzy reasoning strategy valid for reasoning fuzzy values, as indicated in the last section. This is a principal advantage to using prime number marking for the multiple soft component case.

6.6 Simulation Results and Conclusions

For the example shown in Figures 6.1, 6.2 and 6.3, using the ordinary Petri net directly mapped from the original AND/OR net, we obtain 13 feasible sequences to accomplish the cylinder assembly task. However, when we search the fuzzy Petri net mapped from this ordinary Petri net(see Figure 6.8), we obtain only one sequence which is feasible, complete and maintains correct precedence relationships

for operations, i.e., t1 t3 t5 t6 t4 t8 t9 t10 t7 t12 t13.

For the example with added sensing shown in Figure 6.9, when we search all feasible sequences, we get the following 39 sequences:

*** Sequence 1 ***t15 t17 t18 t16 t1 t12 t13

*** Sequence 2 ***t15 t17 t18 t16 t1 t8 t9 t10 t7 t12 t13

*** Sequence 3 ***t15 t17 t18 t16 t1 t8 t9 t10 t7 t3 t5 t6 t4 t12 t13

*** Sequence 4 ***t15 t17 t18 t16 t1 t8 t9 t10 t7 t3 t4 t12 t13

*** Sequence 5 ***t15 t17 t18 t16 t1 t8 t7 t12 t13

*** Sequence 6 ***t15 t17 t18 t16 t1 t8 t7 t3 t5 t6 t4 t12 t13

*** Sequence 7 ***t15 t17 t18 t16 t1 t8 t7 t3 t4 t12 t13

*** Sequence 8 ***t15 t17 t18 t16 t1 t3 t5 t6 t4 t12 t13

*** Sequence 9 ***t15 t17 t18 t16 t1 t3 t5 t6 t4 t8 t9 t10 t7 t12 t13

*** Sequence 10 ***t15 t17 t18 t16 t1 t3 t5 t6 t4 t8 t7 t12 t13

*** Sequence 11 ***t15 t17 t18 t16 t1 t3 t4 t12 t13

*** Sequence 12 ***t15 t17 t18 t16 t1 t3 t4 t8 t9 t10 t7 t12 t13

*** Sequence 13 ***t15 t17 t18 t16 t1 t3 t4 t8 t7 t12 t13

*** Sequence 14 ***t15 t16 t1 t12 t13

*** Sequence 15 ***t15 t16 t1 t8 t9 t10 t7 t12 t13

*** Sequence 16 ***t15 t16 t1 t8 t9 t10 t7 t3 t5 t6 t4 t12 t13

*** Sequence 17 ***t15 t16 t1 t8 t9 t10 t7 t3 t4 t12 t13

*** Sequence 18 ***t15 t16 t1 t8 t7 t12 t13

*** Sequence 19 ***t15 t16 t1 t8 t7 t3 t5 t6 t4 t12 t13

*** Sequence 20 ***t15 t16 t1 t8 t7 t3 t4 t12 t13

*** Sequence 21 ***t15 t16 t1 t3 t5 t6 t4 t12 t13

*** Sequence 22 ***t15 t16 t1 t3 t5 t6 t4 t8 t9 t10 t7 t12 t13

*** Sequence 23 ***t15 t16 t1 t3 t5 t6 t4 t8 t7 t12 t13

*** Sequence 24 ***t15 t16 t1 t3 t4 t12 t13

*** Sequence 25 ***t15 t16 t1 t3 t4 t8 t9 t10 t7 t12 t13

*** Sequence 26 ***t15 t16 t1 t3 t4 t8 t7 t12 t13

*** Sequence 27 ***t1 t12 t13

*** Sequence 28 ***t1 t8 t9 t10 t7 t12 t13

*** Sequence 29 ***t1 t8 t9 t10 t7 t3 t5 t6 t4 t12 t13

*** Sequence 30 ***t1 t8 t9 t10 t7 t3 t4 t12 t13

*** Sequence 31 ***t1 t8 t7 t12 t13

*** Sequence 32 ***t1 t8 t7 t3 t5 t6 t4 t12 t13

*** Sequence 33 ***t1 t8 t7 t3 t4 t12 t13

*** Sequence 34 ***t1 t3 t5 t6 t4 t12 t13

*** Sequence 35 ***t1 t3 t5 t6 t4 t8 t9 t10 t7 t12 t13

*** Sequence 36 ***t1 t3 t5 t6 t4 t8 t7 t12 t13

*** Sequence 37 ***t1 t3 t4 t12 t13

*** Sequence 38 ***t1 t3 t4 t8 t9 t10 t7 t12 t13

*** Sequence 39 ***t1 t3 t4 t8 t7 t12 t13

When we map this updated ordinary Petri net to a fuzzy Petri net(see Figure 6.11) with two soft components, only one sequence is obtained. This is equal to Sequence 9 in the above sequence set. Therefore, we can conclude that even though the searching effort for transition sequences as well as the number of sequences obtained for an ordinary Petri net will increase exponentially relative to the increased number of transitions and places, the number of sequences searched in the corresponding prime number marked fuzzy Petri net may be strongly restricted by ordering constraints on the process steps. The fuzzy Petri net thus appears to be an efficient tool for modeling and representing all feasible, complete process sequences which maintain the correct precedence relationships.

One of the fuzzy Petri net variable types, the global fuzzy variable, is used in this research to efficiently search for correct operations sequences from a fuzzy

Petri net to reach from an initial state to a final state while satisfying precedence and completeness constraints. When the planned sequence is developing, the global values carried by tokens are subject to change. If more than one soft component is handled, a generalized assignment algorithm can be used so that a colored sequence of prime markings will be used for each different soft component. The prime token values used for searching can also be interpreted as a fuzzy value uniformly distributed between 0 and 1. When we search the sequences in the fuzzy Petri net, all sequences which will be incomplete or having incorrect precedence relationships will be discarded. Computation time and storage is reduced since it is not necessary to store those incorrect sequences.

CHAPTER 7
SENSOR-BASED ERROR RECOVERY FOR ROBOTIC TASK SEQUENCES USING FUZZY PETRI NETS

During the execution of a task based on an off-line planned sequence of operations, a robot workcell may encounter errors or events which cause the expected sequence to be unexecutable. This chapter addresses the problem of representing and automatically invoking error recovery sequences in response to sensed errors during execution. The approach is based on the use of a fuzzy Petri net model in which sensory verification operations determine local fuzzy variables associated with the objects in the net. The outcome of a sensory verification operation may change the local fuzzy variables and leads to an altered firing sequence and resulting error recovery. The fuzzy Petri net itself is systematically derived from an AND/OR net model of the task[10, 11], and carries guaranteed properties of safeness, liveness, and reversibility, while the fuzzy assignment algorithm for global fuzzy variables[12, 20] guarantees precedence of subgoal operations. An algorithm is described for adding sensory verification transitions and associated fuzzy transition rules which implement error recovery through retry or alternative sequence mechanisms. An executable fuzzy Petri net could be obtained using a feasible, complete, and correctly ordered sequence.

7.1 Introduction

In previous work, we have used AND/OR graphs[47] and nets[10, 11] to represent geometric relations and operations in robotic assembly workcells and materials handling systems. Based on the characteristics of components and relations which may change their properties during the execution of plans, we introduced the concept of a fuzzy Petri net representation and a method of reasoning about correct

precedence relationships among task subgoals[12, 20]. In the fuzzy Petri net, a *prime number marking algorithm* guarantees strong numerical constraints on the precedence and reduces the set of feasible sequences. Using fuzzy Petri nets, we can search and obtain all possible sequences which guarantee the properties of feasibility, completeness, and correct precedence relationships for subgoal operations. The resultant sequence set from the fuzzy Petri net requires less computer storage and time to search for good solutions compared with the strategy used in [10, 11]. The token values for all places in the fuzzy Petri net can be interpreted as fuzzy values in $[0, 1]$ which represent the degree of completion for each object in the system. The fuzzy values representing internal state variables which are defined as local fuzzy variables are used in this chapter to derive a sensor-based verification and reasoning strategy for exception handling.

Task sequence planning is normally performed off-line using a high-level representation of goals and constraints. Task sequences generated based on a system model and an initial and final state are *expected* ordered operations sequences, i.e., with the assumption that no abnormal events will happen. However, at execution time some unexpected conditions may occur; for example, some components may be missing, some objects may be put at incorrect positions, or, the orientations of some components may not be the same as stored in the computer. Therefore, practical automated manufacturing systems often incorporate an enormous amount of control code for error handling and recovery[41]. Some approaches to error recovery for assembly workcells have been reviewed in Section 2.4.

In this chapter, we address the problem of automatically invoking error recovery sequences within a set of possible execution sequences described by a fuzzy Petri net. The fuzzy Petri net incorporates many possible feasible sequences, and the resultant firing sequence depends on the fuzzy values in the net. A *sensory verification* operation is used to observe the system state, and the outcome of sensory

verification may change the local fuzzy variables associated with the objects leading to an altered firing sequence. If the fuzzy Petri net is correctly designed, this altered firing sequence will eventually lead to a goal state by correctly executing an error recovery procedure. A main goal of this chapter is to introduce a systematic approach to synthesis of this fuzzy Petri net with sensor verification and error recovery.

7.2 Fuzzy Petri Net Representation of Task Level Operations

The definition for the generalized fuzzy Petri net[12, 14, 17, 18] is shown in Section 5.2. In this definition, ρ may be regarded as a fuzzy internal state variable. In practice, these variables may also be used as conventional internal state variables.

Each place in the FPN represents an object. Therefore, the local fuzzy variables and fuzzy marking variables are associated with the objects. In the discussions in this chapter, we only use global fuzzy variables and local fuzzy variables for the representation of sequence planning and error recovery, respectively. f_i has a rule for reasoning about local fuzzy variables, $r_\rho(t_i)$(written as $r_\rho^{t_i}$), and a rule for reasoning about global fuzzy variables, $r_\sigma(t_i)$(written as $r_\sigma^{t_i}$). The fuzzy marking variable is also useful in error recovery, since it models the uncertain outcomes of operations, but this will not be discussed in this chapter.

Because there are three different kinds of variables associated with places and tokens in the FPN, three different system states may be defined[17, 22]. Two of them, local fuzzy state and global fuzzy state, are used in this chapter and their definitions were given in Section 5.3. Here, $\rho(p_i)$ is written as ρ_i and $\sigma(p_i)$ is written as σ_i. A local fuzzy variable represents the internal state of an object.

In this chapter, the fuzzy Petri net representation is used as follows. Each *place*, p_i, represents a system geometric substate, or subassembly, and O_i corresponds to a node in the AND/OR net. For example, the state "robot holding an object" might be a place, p_i. Each *transition*, t_{ij}, represents a state change signified

by the transfer of tokens from input place set $\bigcup_i\{p_i\}$ to output place set $\bigcup_j\{p_j\}$. The operation "robot places the object on the table" might be a transition. α and β are the mappings which define feasible transitions among system states for a given workcell.

Each token in the fuzzy Petri net may take on some fuzzy membership value, $m_t(q_j) = \bigcup_i(k_i, \sigma_{k_i})$, or, e, which constitutes a global fuzzy state of the net, S_g. In our definition of the net, the global fuzzy variable constitutes a 'degree of completion' of a process. Each transition has a weighting factor WF_i which affects the fuzzy value of the token when the transition is fired. The changing of local fuzzy variables follow specific reasoning rules[17, 18]. The rules governing the mapping of fuzzy values of tokens across transitions will be described in the next section.

In [12], we showed that a set of fuzzy values of tokens and transition weights could be developed which guarantee that a set of designated operations, called *key transitions*, would be executed in a designated precedence order. The prime number marking algorithm[12, 20] used to synthesize this net imposes numerical constraints on the fuzzy values of tokens in order to enforce these precedence relations. In this approach, the key transitions may be viewed as *subgoals* in the task which yield key states and have fixed prior order constraints. Enforcing these constraints implicitly in the net representation leads to efficient search for feasible solutions.

Figure 7.1 shows an example of a robot workcell which is a modified version of Figure 6.1. In this task, the robot picks up the raw peg, transfers it to the cutting machine, then to the lubricating machine to process the peg, and finally to the hollow cylinder to accomplish the assembly. The sensor on the head of the robot arm is used to identify the precise position of the hole inside the hollow cylinder.

In modeling real processes, we do not need to represent all tokens with fuzzy values, since not all objects undergo changes in properties or relations during the process. For convenience, we will define a class of objects which do undergo these

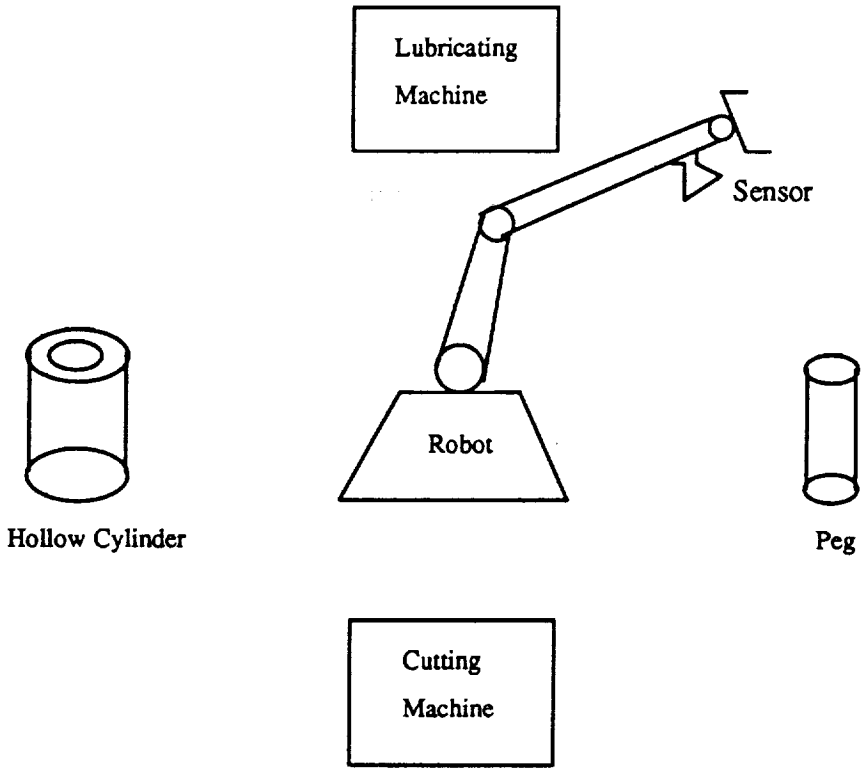


Figure 7.1: A peg-cylinder assembly system.

changes as *soft objects*:

Definition 7.1 *Soft components and soft objects:* A *soft component* is a single part in a system whose "properties" change during execution of plans. For purposes here, these are mostly often geometric properties, such as the change in shape of the peg in example 1.
A *soft object* may be either a soft component, or an assembly or subassembly which includes a soft component, or an object with *soft relation* between parts.

Soft components and objects are characterized by *soft parameters*, which are internal state variables such as geometric parameters of shape or relative positions of parts in an assembly, but may also include mechanical properties such as surface characteristics. In this chapter, some of these soft parameters will be modeled by local fuzzy variables.

In example 1, one soft parameter of the peg is the diameter, which is changed by the cutting operation, and the second is surface lubrication state which is changed by the lubrication operation. A soft parameter of the object RPC is the geometric relation of the peg axis to the cylinder axis which will determine whether insertion is feasible. In Section 7.4, the sensory verification operation will be used to observe the current values of soft parameters during execution, map them to fuzzy membership functions, and control the resulting flow of the sequence using fuzzy reasoning at the transitions.

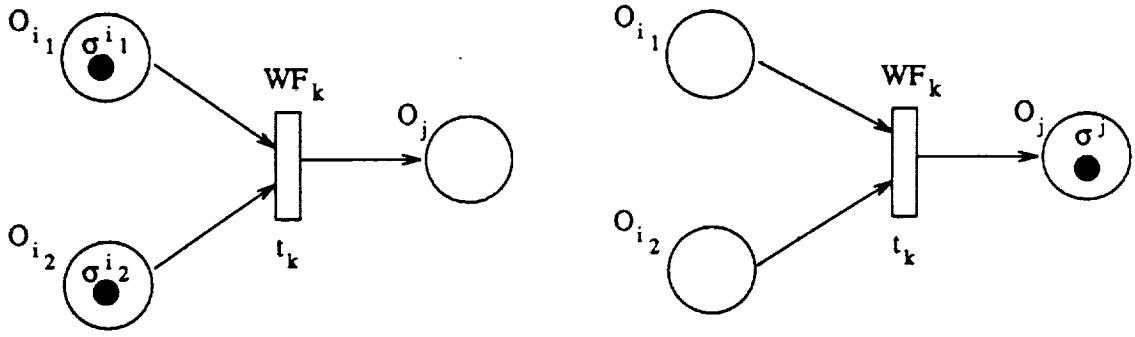
The corresponding fuzzy Petri net for the peg-cylinder assembly example is illustrated in Figure 6.11. The weighting factors assigned to transitions, and the initial global fuzzy state are shown. The fuzzy firing rules for token values are described in Section 7.3. The threshold θ is 0.05. All feasible objects in the system are mapped to places in the net: p_1:R(robot), p_2:P(peg), p_5:M(cutting machine), p_4:L(lubricating machine), p_3:C(hollow cylinder), p_6:RP(robot grasping peg), p_{10}:RPM(robot transferring peg to cutting machine), p_{12}:CUT(cutting job), p_9:RPL(robot transferring

peg to lubricating machine), p_{11}:LUB(lubricating job), p_{13}:RC(robot grasping hollow cylinder), p_{14}:SEN(sensing job), p_8:RPC(robot inserting peg into hollow cylinder) and p_7:PC(peg and cylinder assembly). The global fuzzy state of this fuzzy Petri net is constructed based on a required precedence of subgoal operations: cutting(t_5) precedes lubrication(t_9) precedes insertion(t_{12}), and position sensing(t_{17}) precedes insertion(t_{12}). t_5, t_9 and t_{17} are key transitions in the system, and are defined *a priori*. Based on the prime number marking algorithm[12, 20], the mapping from transitions to integer values is $WF_{17} = 2$, $WF_5 = 2$, and $WF_9 = 3$, and all other weighting factors are equal to 1. From the same algorithm, the initial global fuzzy state of $(p_1,\ p_2,\ \ldots,\ p_{14})^T$ is found to be $(1.0, (1, 0.1), (2, 0.1), 1.0, 1.0, e, e, e, e, e, e, e, e, e)^T$. We have shown in [12] that only one shortest sequence, which is feasible, complete, and maintains correct precedence relationships, is obtained, i.e., $t_{15}t_{17}t_{18}t_{16}t_1t_3t_5t_6t_4t_8t_9t_{10}t_7t_{12}t_{13}$. To find this sequence, we have assumed that each transition can be fired at most once. In this sequence, the robot goes to the hollow cylinder to sense the position of the hole before it carries the peg for machining.

For the purposes of planning this sequence, the sensing operation is assumed to be deterministic and always successful. In Section 7.4, we introduce a nondeterministic fuzzy Petri net in which the outcome of the sensory measurement may affect the sequence at execution time.

7.3 Fuzzy Transition Rules: Global Fuzzy Variables

During the execution of a task sequence, the system reasons about fuzzy values of tokens and the local fuzzy variables of the objects after a transition a fired. In this section, we assume the local fuzzy variables are not affected by transitions, and focus on transition reasoning rules affecting global variables. In the next section, we will discuss the role of local fuzzy variables in transition reasoning rules. A set of fuzzy transition rules for global fuzzy variables in the case of one soft component is



(a) Before t_k is fired. (b) After t_k is fired.

Figure 7.2: Fuzzy Petri net representation for assembly operation. The local fuzzy variables for O_{i_1}, O_{i_2}, and O_j are ρ_{i_1}, ρ_{i_2}, and ρ_j, respectively. The values of tokens in the places of O_{i_1} and O_{i_2} are σ^{i_1} and σ^{i_2} in (a) and that in the place of O_j is σ^j in (b), respectively. The weighting factor of t_k is WF_k.

given as below. If a system contains more than one soft component, a generalized version of fuzzy transition rules can be used[12, 20]. The symbol θ in the following discussion refers to the threshold of reasoning rules.

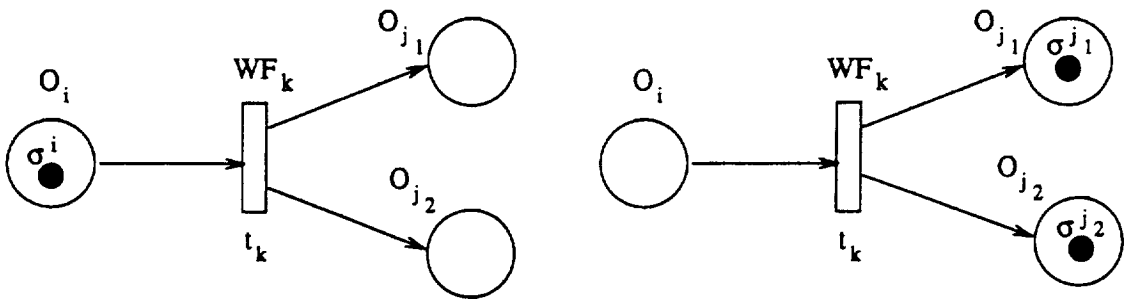
Assembly operation: $O_{i_1}, O_{i_2}, \ldots, O_{i_u} \rightarrow O_j$. The fuzzy Petri net corresponding to the assembly operation with two input places is shown in Figure 7.2, and obeys the following fuzzy transition rule:

$$if\ min(\sigma^{i_1}, \sigma^{i_2}, \ldots, \sigma^{i_u}) \geq \theta,\ then\ \sigma^j = min(\sigma^{i_1}, \sigma^{i_2}, \ldots, \sigma^{i_u}) \times WF_k. \qquad (7.1)$$

Since $WF_k \geq 1$, this rule assigns the same or increased completion value to the output token as that held by the minimum input token.

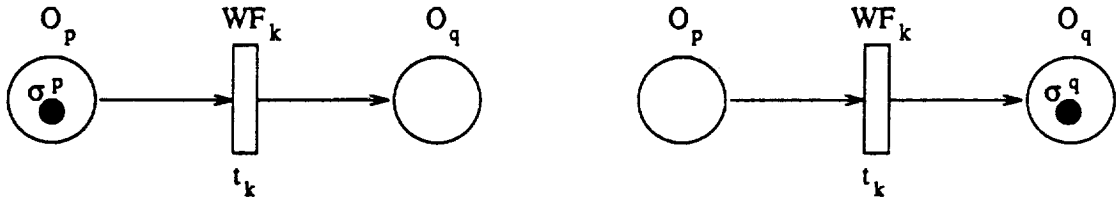
Disassembly operation: $O_i \rightarrow O_{j_1}, O_{j_2}, \ldots, O_{j_l}$. The fuzzy Petri net corresponding to the disassembly operation with two output places is shown in Figure 7.3, and obeys the following fuzzy transition rule:

$$if\ \sigma^i \geq \theta,\ then\ \sigma^{j_d} = \sigma^i \times WF_k \times soft(O_{j_d}) + 1 - soft(O_{j_d}), \qquad (7.2)$$



(a) Before t_k is fired. (b) After t_k is fired.

Figure 7.3: Fuzzy Petri net representation for disassembly operation. The local fuzzy variables for O_i, O_{j_1}, and O_{j_2} are ρ_i, ρ_{j_1}, and ρ_{j_2}, respectively. The values of tokens in the places of O_{j_1} and O_{j_2} are σ^{j_1} and σ^{j_2} in (b) and that in the place of O_i is σ^i in (a), respectively. The weighting factor of t_k is WF_k.



(a) Before t_k is fired. (b) After t_k is fired.

Figure 7.4: Fuzzy Petri net representation for IST operation. The local fuzzy variables for O_p and O_q are ρ_p and ρ_q, respectively. The values of tokens in the places of O_p and O_q are σ^p and σ^q in (a) and (b), respectively. The weighting factor of t_k is WF_k.

where

$$soft(O_{j_d}) = \begin{cases} 1 & \text{if } O_{j_d} \text{ is a soft object,} \\ 0 & \text{otherwise,} \end{cases} \qquad 1 \leq d \leq l.$$

Since $WF_k \geq 1$, this rule assigns the same or increased completion token value to each soft object and leaves other objects at 1.

Internal State Transition(IST) operation: $O_p \rightarrow O_q$. The fuzzy Petri net corresponding to the IST operation is shown in Figure 7.4, and obeys the following fuzzy transition rule:

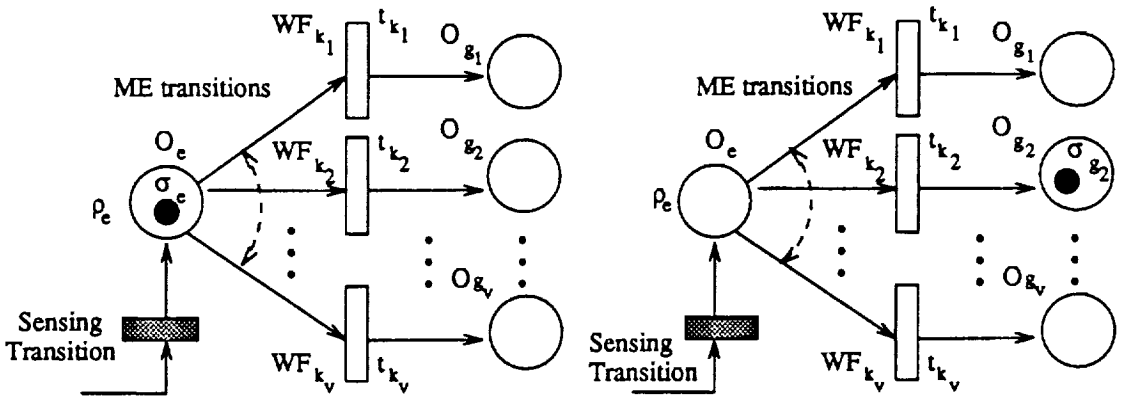
$$\text{if } \sigma^p \geq \theta, \text{ then } \sigma^q = \sigma^p \times WF_k. \tag{7.3}$$

Since $\mu_{f_k} \geq 1$, this rule assigns the same or increased completion value to the output token.

7.4 Execution of Plans on the Fuzzy Petri Net

During the real-time execution of a selected sequence which is feasible, complete, and maintains correct precedence relationships for key transitions, some of these key transitions may not cause the expected results. This uncertainty in local parameters can be represented by local fuzzy variables associated with objects. For example, we assume the input place of a key transition t_i, namely p_j, contains a token with fuzzy value σ^j before t_i is fired. The weighting factor for t_i is WF_i. As mentioned in the previous discussions, the prime token value in the output place p'_j is $\sigma^{j'} = \sigma^j \times WF_i$, assuming $\rho(p_{j'})$ is fixed. In the real-time execution of the planned sequence, it may happen that $\rho(p_{j'})$ is changed because errors may occur during the execution of t_i, while the global token value of $\sigma^{j'}$ does show that t_i has been successfully executed.

To guarantee a correct fuzzy token moving in the Petri net, we introduce a sensor which verifies the states for soft objects, especially after key transitions are fired in the system. In order to incorporate such sensor-based selection of operations, we need to define fuzzy rules governing an additional type of Petri net module corresponding to *mutually exclusive* operations for the execution of plans on-line. This module reasons about the local fuzzy variables based on the input local variables. Therefore, during the execution of a task sequence, the selection of enabled transitions not only depends on token values(global fuzzy variables), but also on local fuzzy variables. In a fuzzy Petri net model used for planning, the alternative choice of mutually exclusive operations is resolved by the off-line search for an expected plan. However, real-time execution of the net requires local resolution of this choice and will depend on the current local variables which occur. The fuzzy transition



(a) Before t_{k_i} is fired. (b) After t_{k_2} is fired.

Figure 7.5: Fuzzy Petri net representation for ME transitions. In (a), The global fuzzy variables and local fuzzy variables of O_e are σ_e and ρ_e. In (b), the global fuzzy variable of O_{g_2} is σ_{g_2}. The weighting factors of $t_{k_1}, t_{k_2}, \ldots, t_{k_v}$ are $WF_{k_1}, WF_{k_2}, \ldots, WF_{k_v}$, respectively.

rules for this important case are given below.

7.4.1 Mutually Exclusive Transitions

The fuzzy Petri net corresponding to the mutually exclusive operation: ($O_e \rightarrow O_{g_1}$ OR $O_e \rightarrow O_{g_2}$ OR ... OR $O_e \rightarrow O_{g_v}$), is shown in Figure 7.5. Here, sensors are introduced to verify the states for soft objects. Therefore, the following operations may be chosen based on fuzzy sensory information. For the sake of simplicity, we assume the local fuzzy variables appearing in the following discussions are fuzzy singletons. The results obtained can be generalized to the case of general fuzzy numbers. The transitions which represent this type of operation are called mutually exclusive, or, *ME*, transitions.

The fuzzy rule governing this mutually exclusive firing strategy is described as follows:

$$if\ 0 \leq \rho_e < \theta^1,\ then\ \sigma^{g_1} = WF_{k_1} \times \sigma^e,\ \sigma^{g_2} = \sigma^{g_3} = \ldots = \sigma^{g_v} = 0;$$

$$if\ \theta^1 \leq \rho_e < \theta^2,\ then\ \sigma^{g_2} = WF_{k_2} \times \sigma^e,\ \sigma^{g_1} = \sigma^{g_3} = \ldots = \sigma^{g_v} = 0;$$

...

$$if\ \theta^{i-1} \leq \rho_e < \theta^i,\ then\ \sigma^{g_i} = WF_{k_i} \times \sigma^e,\ \sigma^{g_1} = \ldots = \sigma^{g_{i-1}} = \sigma^{g_{i+1}} = \ldots = \sigma^{g_v} = 0;$$

...

$$if\ \theta^{v-1} \leq \rho_e \leq \theta^v,\ then\ \sigma^{g_v} = WF_{k_v} \times \sigma^e,\ \sigma^{g_1} = \sigma^{g_2} = \ldots = \sigma^{g_{v-1}} = 0. \quad (7.4)$$

Notice that the range $[0, \theta^v]$ has been divided to v subranges, $[0, \theta^1)$, $[\theta^1, \theta^2)$, ..., $[\theta^{v-1}, \theta^v]$, which are mutually exclusive. Only one transition from $t_{g_1}, t_{g_2}, \ldots, t_{g_v}$ will be fired. The selection of the transition is based on the real-time value of ρ_e. The ME transition acts like a "case" conditional statement in a high-level programming language.

7.4.2 Deterministic and Nondeterministic Fuzzy Petri Nets

In this discussion, we have assumed that each transition t_i in the net has a *constant weighting factor* WF_i, i.e., the fuzzy values of tokens in output places for t_i can be directly determined from the fuzzy values of tokens in corresponding input places. This assumption is valid from the planning point of view where we can guarantee a selected feasible sequence to reach from the initial state to the final state when this sequence is executed. However, this assumption is not always correct from the execution point of view. Some transitions, especially key transitions, may not reach the desired result as expected after the corresponding operation is executed in practice. Therefore, we distinguish *deterministic* and *nondeterministic fuzzy Petri nets* to reflect the possible random properties of the transitions at execution time.

Definition 7.2 *Deterministic fuzzy Petri net(DFPN):* A fuzzy Petri net(FPN) in which each transition t_i has a *deterministic* reasoning rule for local fuzzy variables, $r_p^{t_i}$, and therefore a fixed mapping between input and output local fuzzy variables.

Definition 7.3 *Nondeterministic fuzzy Petri net(NDFPN):* A fuzzy Petri net(FPN) in which there exists at least one transition t_k which has a *random* reasoning rule

for local fuzzy variables. There is no fixed mapping between input and output local fuzzy variables. The *expected mapping* of local fuzzy variables are known corresponding to the mapping of input and output token values.

Definition 7.4 *Random transition:* A transition which has a random mapping between input local fuzzy variables and output local fuzzy variables.

A DFPN contains no random transition. An NDFPN contains at least one random transition. The fuzzy Petri net generated from the AND/OR net is an example of a DFPN; however, during the execution of a planned sequence, some transitions might have random properties. For example, in Figure 7.1, if we assume t_5 cannot always guarantee the cutting to the correct size, then t_5 becomes a random transition. This modified fuzzy Petri net is an example of an NDFPN.

7.4.3 Planning and Execution on the NDFPN with ME Transitions

If we model some key transitions in the DFPN as random transitions, the previous planning strategy may not be suitable for searching sequences in an off-line planning mode because the net has become an NDFPN. To approach this problem, we assume that the reasoning function for local fuzzy variables of all random transitions are their *expected reasoning functions*, and the resulting NDFPN will thus become a DFPN. We can use the planning procedure on the DFPN as before to get the expected correct sequences.

For the sake of simplicity, we first consider the case that t_i^ν is the only random transition in the net, and retry of the operation t_i^ν is chosen as the error recovery strategy. After t_i^ν is executed, a sensing transition defines a new local fuzzy variable ρ_j which determines the next firing through an ME transition. We assume an expected sequence searched from a DFPN, of which all key transitions are assigned the weighting factors for reasoning token values, is $\mathcal{S}_1 t_1^\nu \mathcal{S}_2 t_2^\nu \ldots t_r^\nu \mathcal{S}_{r+1}$, where $\mathcal{S}_1$,

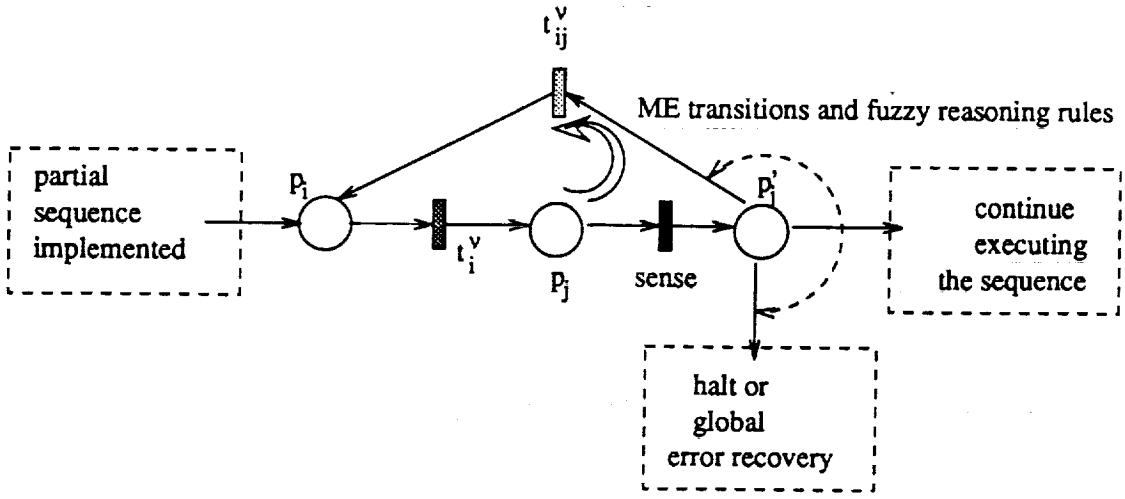


Figure 7.6: Repeated trial error recovery structure.

S_2, ..., S_{r+1} are subsequences which do not contain any key transitions, and t_1^v, t_2^v, ..., t_r^v are r key transitions. We assume that after we execute t_i^v during the implementation of this task sequence, we may meet three possible cases:(Figure 7.6)

(1) *Succeed*: Based on the sensory verification operation, a 'correct' local fuzzy variable is obtained after t_i^v is fired. We then go on executing the remaining part of the sequence. The final actual sequence may be exactly the same as the planned sequence.

(2) *Retry and Succeed*: Based on the sensory verification operation, we do not obtain a correct local fuzzy variable following t_i^v. The refiring of t_i^v is required. This transition may be fired totally $l(1 < l \leq M; M$ is a repair threshold) times. The actual sequence which is executed may be(we consider *sense* and t_{ij}^v as virtual transitions for the sequence representation.)

$$S_1 t_1^v S_2 t_2^v \ldots S_i (t_i^v)^l S_{i+1} \ldots t_r^v S_{r+1}.$$

(The specific mechanism for enabling this retry process using fuzzy values is discussed below.)

(3) *Retry and Fail*: After firing of t_i^v $l'(1 \leq l' \leq M)$ times, an unrecoverable condition is detected for a soft object leading to global error recovery(firing of S_{gr})

or halt. For example, a certain soft object or the included soft component, may need to be discarded and replaced by a new soft object. At this time, we assume the execution of the remaining part of the sequence is stopped. The actual sequence which is executed for this case will be

$$\mathcal{S}_1 t_1^{\nu} \mathcal{S}_2 t_2^{\nu} \ldots \mathcal{S}_i (t_i^{\nu})^{l'} (e \cup \mathcal{S}_{gr}).$$

The selection of M is sometimes an important factor to implement a sequence. And the selection may depend on different criteria for different key transitions. If M is chosen too large, a transition loop has to be executed many times so that the threshold is reached and the sequence can stop. If M is chosen too small, the possibility for *local error recovery* for some key transitions will be lower. Moreover, sometimes the error is due to the hardware mechanisms such as machines, processing units, and so on. We may have several alternative transitions available for an identical logical transition. In the previous example, we may have two cutting machines. Any planned sequence may choose one of these two machines when a cutting transition is fired. When a cutting machine causes an erroneous state on the processed part, we may need to try another cutting machine for local error recovery, rather than continuing using the former one. In other cases, a more extensive *alternative* sequence may be employed, or a more general *global error recovery* sequence may be initiated.

Figure 7.7 shows the global and local alternative error recovery for the peg-cylinder assembly example. Figure 7.7(a) illustrates a feasible sequence to finish the assigned assembly task. Figure 7.7(b) depicts that a global alternative error recovery sequence may be called when a threshold of times for retry could not remedy the insertion error. Figure 7.7(c) shows that local alternative error recovery may be necessary when a cutting error is sensed by a size sensor near the cutting machine. The subnets of the fuzzy Petri net corresponding to Figure 7.7(a), 7.7(b) and 7.7(c) are illustrated in Figure 7.8(a), 7.8(b) and 7.8(c). For the sake of simplicity, hollow

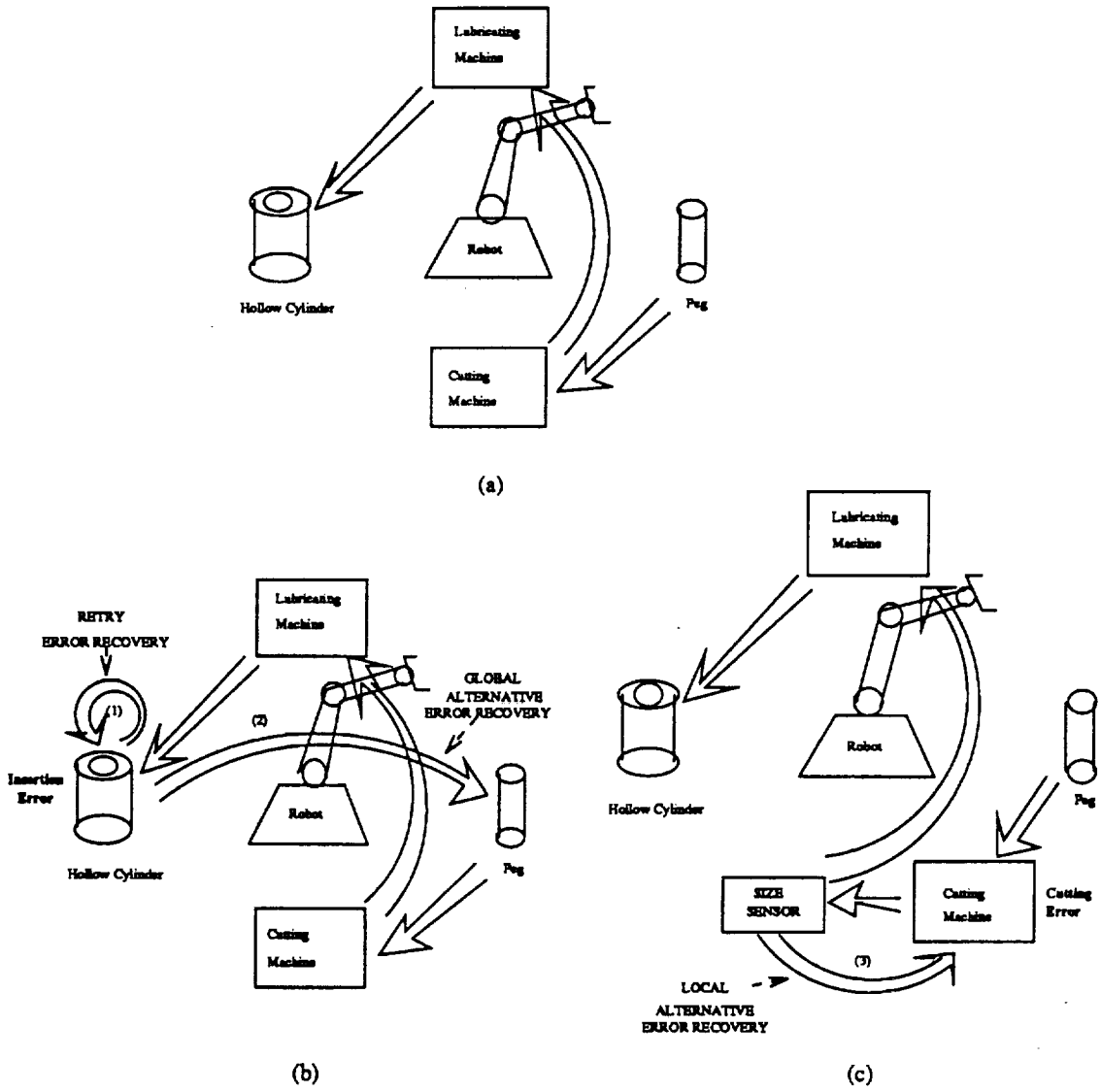


Figure 7.7: Error recovery for the peg-cylinder assembly example. (a) Planned sequence. (b) Insertion error. (c) Size error.

cylinder is not considered as a soft component at this time.

In Figure 7.9, we illustrate a Petri net representation of multiple alternative transitions for a *key operation*. Local error recovery is shown using a retry strategy here which enables alternate transitions through adjustment of local fuzzy variables. t_1^ν, t_2^ν and t_3^ν are considered to be the transitions which accomplish the same subtask. A sensing transition is introduced here to identify the token value in p_j, and the on-line selection of the following subsequences will depend on this sensed value. Suppose originally, p_i contains a token of which the fuzzy value is σ^i. The local fuzzy variable of p_i is $\rho_i^{(0)}$. After t_1^ν is fired, p_j contains a token with the fuzzy value $\sigma^j = \sigma^i \times WF(t_1^\nu)$, and the local fuzzy variable of p_j is ρ_j. Transition *sense* maps the local fuzzy variable of the object to a sensing value, $\rho_{j'}$. The weighting factor of *sense* is 1. Therefore, $\sigma^{j'} = \sigma^j$. If $\theta^1 \leq \rho_{j'} < \theta^2$, we will fire t_{ij}^ν for local error recovery. If we assume the reasoning functions for local fuzzy variables are the same as global fuzzy variables, We may select an α and define

$$WF(t_{ij}^\nu) = \frac{1}{s(t_i) + \alpha}, \quad s(t_i) = WF(t_i^\nu),\ 1 \leq i \leq 3,\ \alpha > 0.$$

The new fuzzy value in p_i will be

$$\rho_i^{(1)} = \frac{1}{WF(t_1^\nu) + \alpha} \cdot WF(t_1^\nu) \cdot \rho_i^{(0)} = \frac{WF(t_1^\nu)\rho_i^{(0)}}{WF(t_1^\nu) + \alpha} < \rho_i^{(0)}.$$

After m times through this loop,

$$\rho_i^{(m)} = \left(\frac{WF(t_1^\nu)}{WF(t_1^\nu) + \alpha}\right)^m \rho_i^{(0)} < \left(\frac{WF(t_1^\nu)}{WF(t_1^\nu) + \alpha}\right)^{m-1} \rho_i^{(0)} < \ldots < \rho_i^{(0)}.$$

Therefore, $\rho_i^{(k)}$, $\rho_i^{(k-1)}$, $\ldots$, $\rho_i^{(0)}$ is a strictly decreasing sequence.

If we divide the ρ range $[\theta^1, \theta^2)$ to three subranges, $[\theta^1, \theta')$, $[\theta', \theta'')$, and $[\theta'', \theta^2)$, initially, $\rho_i^{(0)} \in [\theta'', \theta^2)$. After k_1 iterations, $\rho_i^{(k_1)}$ will fall into $[\theta', \theta'')$. We can therefore fire t_2^ν instead of t_1^ν and at this time, $\rho_i^{(k_1)} = \left(\frac{WF(t_1^\nu)}{WF(t_1^\nu)+\alpha}\right)^{k_1} \rho_i^{(0)}$. If after k_2 iterations for firing t_2^ν we still cannot recover to a desired correct state, $\rho_i^{(k_1+k_2)}$ will

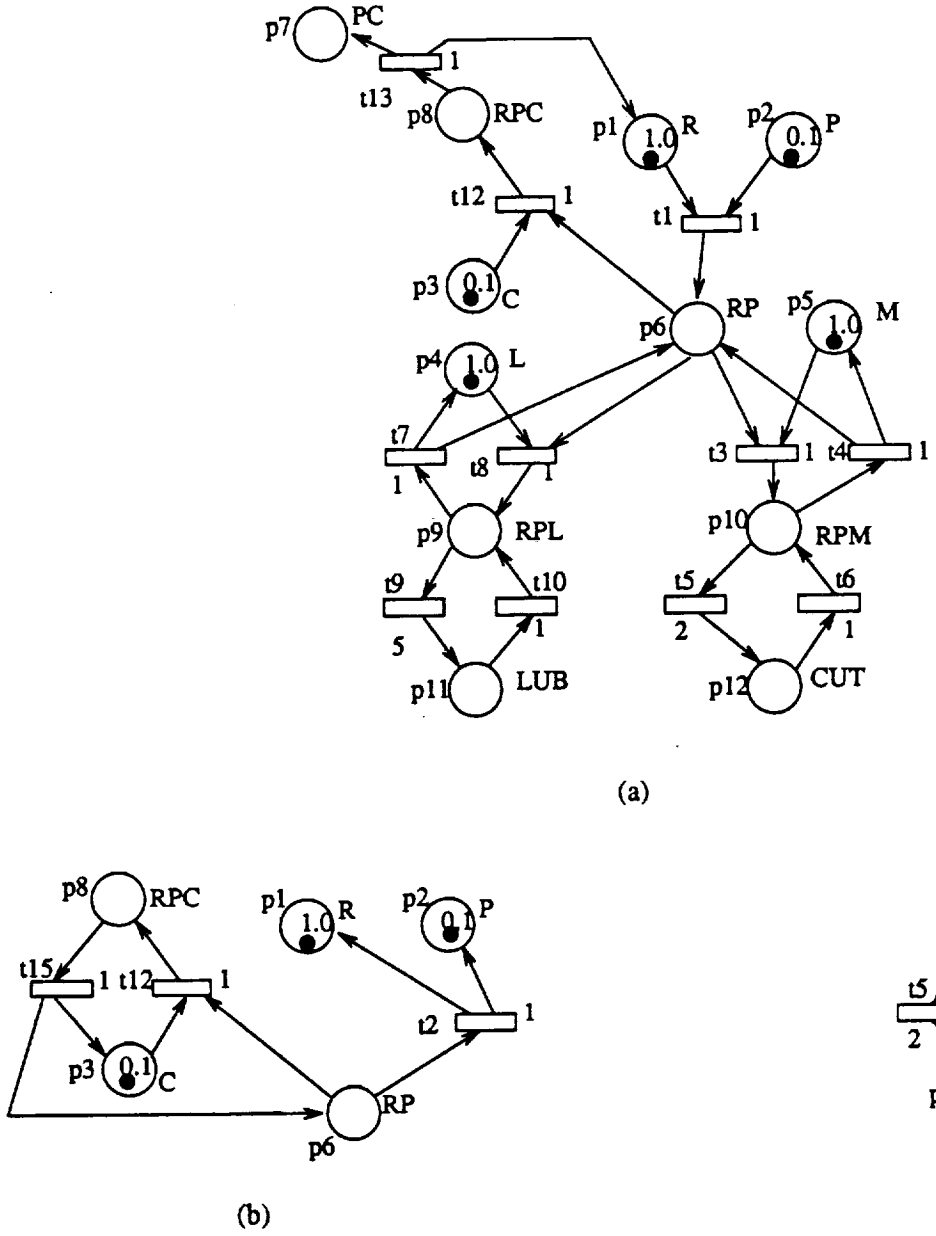


Figure 7.8: The subnets of the fuzzy Petri net of error recovery for the peg-cylinder assembly example. (a) Planned sequence: $t_1t_3t_5t_6t_4t_8t_9t_{10}t_7t_{12}t_{13}$. (b) When an insertion error occurs, and $t_{15}t_{12}$ subsequence cannot remedy the error, a global recovery sequence t_2 will be used. (c) When a size error is found, a subsequence t_6t_5 will be used to remedy the error.

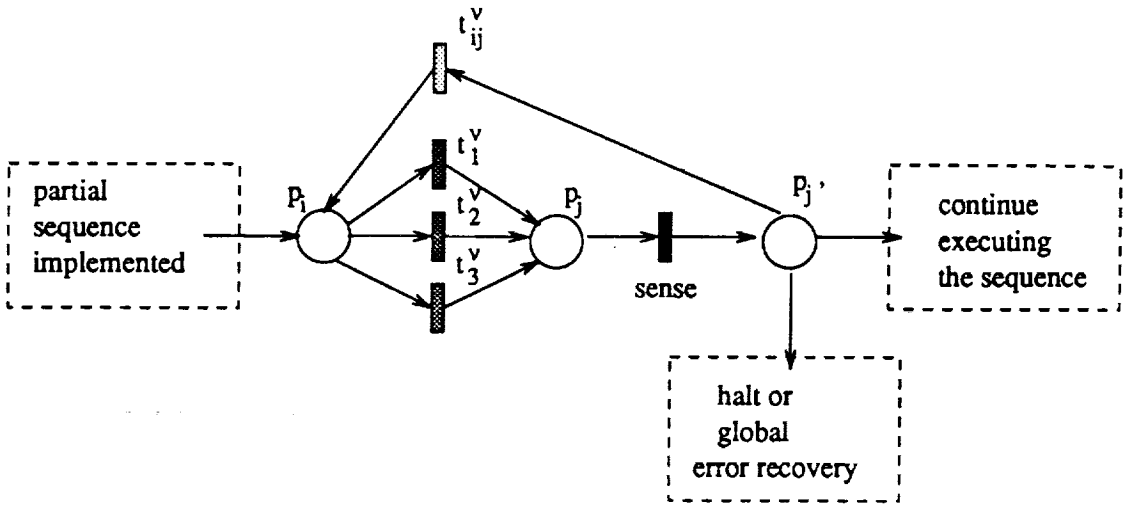


Figure 7.9: Alternative local error recovery.

drop down to $[\theta^1, \theta')$ and at that time,

$$\rho_i^{(k_1+k_2)} = \left(\frac{WF(t_1^\nu)}{WF(t_1^\nu)+\alpha}\right)^{k_1}\left(\frac{WF(t_2^\nu)}{WF(t_2^\nu)+\alpha}\right)^{k_2}\rho_i^{(0)}.$$

Following these iterations, we may succeed in recovery or drop down to the global recovery or halt range, $[0, \theta^1)$. The above strategy, where ρ decreases with iterative tries until it goes through all alternative ME transitions, provides a mechanism to implement alternative error recovery.

More generally, we may define sequences which include transition loops for the execution of certain transitions more than once:

Definition 7.5 *Key-transition-loop sequence:* A sequence which forms a cycle and includes at least one key transition. The number of executions for the loop is unknown prior to the execution.

From the above definition, we know that we cannot decide an actual executable sequence in the off-line planning stage for key-transition-loop sequences. Even though during the off-line planning stage, we can enumerate all possible sequences which reflect all possible directions for the ME transitions, this may require too much computer storage and time to generate all possible sequences. Even

though all sequences are generated, choosing among them also remains a problem because we cannot ensure which sequence will be successful at the implementation stage. Therefore, we will develop a suitable compact representation for feasible task sequences as well as an efficient way to execute them.

Before we map the system DFPN to an NDFPN for execution, we can generate all feasible, complete sequences which have correct precedence relationships for operations. These sequences are expected sequences in the mapped NDFPN. When we add ME transitions, the expected sequences are modified to incorporate alternative error recovery sequences. We assume actual sequences can be modeled as key-transition-loop sequences.

We generalize each key transition t_k to a *key transition subsequence* $\mathcal{T}_k$ because sometimes several operations will be needed to accomplish a *key task*. An expected sequence is generated from the corresponding DFPN as:

$$E(Sequence) = \mathcal{S}_1\mathcal{T}_1^{\nu}\mathcal{S}_2\mathcal{T}_2^{\nu}\ldots\mathcal{S}_i\mathcal{T}_i^{\nu}\mathcal{S}_{i+1}\ldots\mathcal{T}_r^{\nu}\mathcal{S}_{r+1}. \tag{7.5}$$

We further assume all key transition subsequences are transition loop subsequences. Therefore, the executable sequences will become

$$Sequence = \mathcal{S}_1(\mathcal{T}_1^{\nu})^+ \bigcup \mathcal{S}_1(\mathcal{T}_1^{\nu})^+\mathcal{S}_2(\mathcal{T}_2^{\nu})^+ \bigcup \ldots \bigcup$$

$$\mathcal{S}_1(\mathcal{T}_1^{\nu})^+\mathcal{S}_2(\mathcal{T}_2^{\nu})^+\ldots\mathcal{S}_i(\mathcal{T}_i^{\nu})^+\mathcal{S}_{i+1}\ldots(\mathcal{T}_r^{\nu})^+\mathcal{S}_{r+1}. \tag{7.6}$$

where x^+ means executing x one or more than one time. Applying this sequence representation to the real-time implementation, when we meet sign "+" in the sequence, further execution will depend on the current state of the system. The system may go on executing the sequence, or re-execute the operations represented by the transition loop subsequence, or stop executing the remaining part of the sequence.

7.5 Error Recovery

In the previous section, we described an approach to planning and execution on the NDFPN with sensory verification. An unverified sensory state initiates either a retry of the transition, or an alternative procedure, depending on the values of the local fuzzy variables which occur. These procedures involving one transition (perhaps one composite transition) are viewed as local error recovery procedures since they do not involve other key transitions or places in the net.

In the more general case, error detection may occur as part of a long sequence of events within the plan, and the global error recovery may require either backtracking and restart of the sequence, or selection of an alternative error recovery sequence. In this section, we develop the formal definition of these strategies based on an assumption of bidirectional, or brother, transitions which follow from the AND/OR model of the original task. Two cases are considered. In the first case, there is only one soft component in the net, and the retry of that sequence depends only on returning to a known prior state where the process can be reinitiailized. In the second case, there is more than one soft component and the retry becomes more complicated since the intermediate states of other soft components must be considered in the reexecution.

Alternative error recovery in the global case follows also from these definitions, but is more difficult to generalize completely. The backtracking component of alternative error recovery is identical to retry, but the execution of the alternative sequence cannot be predicted since, in general, it depends on the intermediate values of all of the soft components as they were left after the backtracking procedure. While access to the alternative procedure can be guaranteed by the methods defined below, the success of the alternative procedure is not necessarily predictable since the initial state is unknown prior to execution. This situation leads to an emphasis on the design of the alternative sequence to properly account for the occurrence of

a range of potential initial error states at the branch point. In a broad sense, this approach leads to the development of design criteria for the alternative sequences which would emphasize the independence of soft components between the alternatives. This independence constraint would minimize the interaction between error states of soft components in the primary sequence and new states in the alternative sequence.

7.5.1 Error Recovery for One Soft Component

Definition 7.6 *Brother transitions:* When we map an AND/OR net to an ordinary Petri net, each reversible arc will be decomposed into two transitions which are in the opposite directions. We define these as *brother transitions*, t_i and $\overline{t_i}$.

Lemma 7.1 For any two markings m^1 and m^2 in an ordinary Petri net, which is mapped from an AND/OR net, $m^2 \xrightarrow{\overline{t_i}} m^1$ if $m^1 \xrightarrow{t_i} m^2$.

Proof: Suppose $m^1 = (m_1^1, m_2^1, \ldots, m_n^1)$, and $m^2 = (m_1^2, m_2^2, \ldots, m_n^2)$.

(1) Suppose the arc in the AND/OR net is an IST arc, because $m^1 \xrightarrow{t_i} m^2$, we obtain

$$m_j^1 = m_j^2,\ 1 \le j \le n,\ j \ne j_1,\ j \ne j_2,\ \textit{and}\ m_{j_1}^1 = 1,\ m_{j_2}^1 = 0,\ m_{j_1}^2 = 0,\ m_{j_2}^2 = 1.$$

i.e., t_i moves one token in p_{j_1} to p_{j_2}. Therefore $\overline{t_i}$ will move the token in p_{j_2} back to p_{j_1}, which can be represented as $m^2 \xrightarrow{\overline{t_i}} m^1$.

(2) Suppose the arc in the AND/OR net is an AND arc, because $m^1 \xrightarrow{t_i} m^2$, we obtain

$$m_j^1 = m_j^2,\ 1 \le j \le n,\ j \ne j_1,\ j \ne j_2, \ldots, j \ne j_k, j \ne j_{k+1}.$$

and

$$m_{j_1}^1 = m_{j_2}^1 = \ldots = m_{j_k}^1 = 1,\ m_{j_{k+1}}^1 = 0;\ \textit{and}\ m_{j_1}^2 = m_{j_2}^2 = \ldots = m_{j_k}^2 = 0,\ m_{j_{k+1}}^2 = 1,$$

or

$$m^1_{j_1} = m^1_{j_2} = \ldots = m^1_{j_k} = 0,\ m^1_{j_{k+1}} = 1;\ and\ m^2_{j_1} = m^2_{j_2} = \ldots = m^2_{j_k} = 1,\ m^2_{j_{k+1}} = 0.$$

i.e., t_i delete the tokens in p_{j_1}, p_{j_2}, ..., p_{j_k}, and add one token in $p_{j_{k+1}}$, or, delete the token in $p_{j_{k+1}}$ and add one token in p_{j_1}, p_{j_2}, ..., p_{j_k}, respectively. Therefore $\overline{t_i}$ will delete the token in $p_{j_{k+1}}$ and add one token in p_{j_1}, p_{j_2}, ..., p_{j_k}, respectively, or, $\overline{t_i}$ will delete the tokens in p_{j_1}, p_{j_2}, ..., p_{j_k} and add one token in $p_{j_{k+1}}$, which can be represented as $m^2 \xrightarrow{\overline{t_i}} m^1$.

Q.E.D. □

Definition 7.7 *Brother sequence:* If a sequence S is $t_1 t_2 \ldots t_l$, the *brother sequence* of S, which is written as $\overline{S}$, is $\overline{t_l}\ \overline{t_{l-1}} \ldots \overline{t_1}$.

Theorem 7.1 For any two markings m^1 and m^2 in a reversible ordinary Petri net, which is mapped from an AND/OR net, $m^2 \xrightarrow{\overline{S}} m^1$ if $m^1 \xrightarrow{S} m^2$, where S is a transition sequence of $t_1 t_2 \ldots t_l$, $l \geq 1$.

Proof: Using the mathematical deduction method:

1. If $l = 1$, using Lemma 7.1, we conclude that the theorem is correct.

2. Suppose when $l = k$ and if the marking m^2 is reachable from m^1 by firing $\mathcal{S} = t_1 t_2 \ldots t_k$, then m^1 is reachable from m^2 by $\overline{\mathcal{S}} = \overline{t_k}\ \overline{t_{k-1}} \ldots \overline{t_1}$.

3. When $l = k + 1$, the marking m^2 is reachable from m^1 by firing $\mathcal{S}' = t_1 t_2 \ldots t_k t_{k+1}$. We assume the marking m' is reachable when we fire $t_1 t_2 \ldots t_k$ from the marking m^1, and m^2 is reachable from m' by firing t_{k+1}. We know from Lemma 7.1, m' is also reachable by firing $\overline{t_{k+1}}$ from the marking m^2. And again using the above supposition, m^1 is reachable from m' by firing $\overline{t_k}\ \overline{t_{k-1}} \ldots \overline{t_1}$. Therefore, m^1 is reachable from m^2 by firing $\overline{t_{k+1}}\ \overline{t_k} \ldots \overline{t_1} = \overline{\mathcal{S}'}$.

We conclude that the initial marking is reachable by the brother sequence from the final marking, which is reachable by the original sequence from the initial marking.

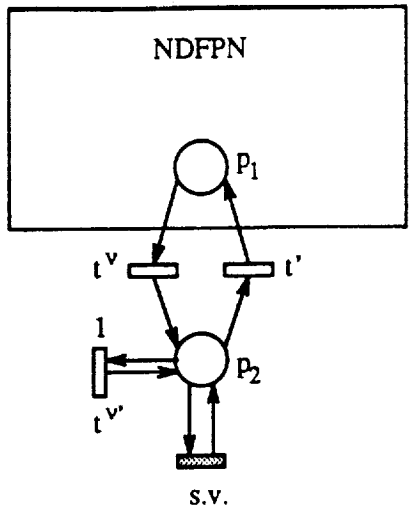


Figure 7.10: An example of NDFPN with ME transitions.

Q.E.D. □

As a mechanism to initiate error recovery, we add a sensing transition, for which the weighting factor of the reasoning function for the local fuzzy variable is a random value. An example of an NDFPN with ME transitions is shown in Figure 7.10. In this example, the key transition t^{ν} remains a *constant transition.* Place p_2 will therefore receive an expected fuzzy token after t^{ν} is fired and the local fuzzy variable in p_2 is fixed. We then fire the *s.v.*(sensor verification) transition, which is assumed to be a random transition, to verify the real local state of p_2. The new local fuzzy value of the object represented by p_2 is thus dependent on the current weighting factor of the transition *s.v.* The following execution of sequences through ME transitions will depend on the current local fuzzy value associated with p_2. If the state is acceptable, the originally planned sequence continues to execute. If the state is not satisfied but may still be reached after some modification is made, we fire $t^{\nu'}$ and fire the *s.v.* transition once more. If an error, which cannot be recovered locally, is detected, we may go back to the initial state and replace the corresponding soft component.

The following theorem reasons about the recovery sequence from the original

task sequence. A definition and a lemma are shown first to introduce some notation.

Definition 7.8 *Token defuzzifying function* $\Omega_p(S_g)$*:*

$$\Omega_p((\sigma^1, \sigma^2, \ldots, \sigma^n)) = (\sigma_d^1, \sigma_d^2, \ldots, \sigma_d^n), \tag{7.7}$$

where

$$\sigma_d^i = \begin{cases} 1 & \text{if } \sigma^i \neq e, \\ e & \text{otherwise,} \end{cases} \quad 1 \leq i \leq n.$$

Definition 7.9 *Transition defuzzifying function* $\Omega_t(r_\sigma)$*:*

$$\Omega_t((WF_1, WF_2, \ldots, WF_m)) = (\underbrace{1, 1, \ldots, 1}_{m}). \tag{7.8}$$

Lemma 7.2 If $S_g^0 \xrightarrow{\mathcal{S}} S_g^k$, then $\Omega_p(S_g^0) \xrightarrow{\mathcal{S}} \Omega_p(S_g^k)$ when $r_\sigma \triangleq \Omega_t(r_\sigma)$, where "$\triangleq$" means "is assigned the value of".

Proof: Suppose $\mathcal{S} = t_1 t_2 \ldots t_k$. Since the weighting factors for all reasoning rules for token values are defuzzified as $r_\sigma \triangleq \Omega_t(r_\sigma)$, from $S_g^0 \xrightarrow{t_1} S_g^1$, $S_g^0 \triangleq \Omega_p(S_g^0)$ and $WF_1 = 1$, we obtain $S_g^1 \triangleq \Omega_p(S_g^1)$. Using the mathematical deduction method, we can obtain $S_g^i \triangleq \Omega_p(S_g^i)$. where $1 \leq i \leq k$. Therefore, $S_g^k \triangleq \Omega_p(S_g^k)$. We conclude that $\Omega_p(S_g^0) \xrightarrow{\mathcal{S}} \Omega_p(S_g^k)$.

Q.E.D. □

Theorem 7.2 If $S_g^0 \xrightarrow{\mathcal{S}^1} S_g^k$, where $\mathcal{S}^1 = \mathcal{S}_1 t_1^\nu \overline{t_1^\nu} \mathcal{S}_2 t_2^\nu \overline{t_2^\nu} \ldots \mathcal{S}_l t_l^\nu$, then $\Omega_p(S_g^k) \xrightarrow{\mathcal{S}^2} \Omega_p(S_g^0)$ when $r_\sigma \triangleq \Omega_t(r_\sigma)$, where $\mathcal{S}^2 = \overline{t_l^\nu}\ \overline{\mathcal{S}_l}\ \overline{\mathcal{S}_{l-1}} \ldots \overline{\mathcal{S}_2}\ \overline{\mathcal{S}_1}$.

Proof: Since $S_g^0 \xrightarrow{\mathcal{S}^1} S_g^k$, $r_\sigma \triangleq \Omega_t(r_\sigma)$ and according to Lemma 7.2, we get $\Omega_p(S_g^0) \xrightarrow{\mathcal{S}^1} \Omega_p(S_g^k)$, where $\mathcal{S}^1 = \mathcal{S}_1 t_1^\nu \overline{t_1^\nu} \mathcal{S}_2 t_2^\nu \overline{t_2^\nu} \ldots \mathcal{S}_l t_l^\nu$. Using Theorem 7.1, we obtain $\Omega_p(S_g^k) \xrightarrow{\overline{\mathcal{S}^1}} \Omega_p(S_g^0)$, where

$$\overline{\mathcal{S}^1} = \overline{\mathcal{S}_1 t_1^\nu \overline{t_1^\nu} \mathcal{S}_2 t_2^\nu \overline{t_2^\nu} \ldots \mathcal{S}_l t_l^\nu} = \overline{t_l^\nu}\ \overline{\mathcal{S}_l} t_{l-1}^\nu \overline{t_{l-1}^\nu} \ldots t_2^\nu \overline{t_2^\nu}\ \overline{\mathcal{S}_2} t_1^\nu \overline{t_1^\nu}\ \overline{\mathcal{S}_1}.$$

Using Lemma 7.1,

$$\Omega_p(S_g^{j+1}) \xrightarrow{\overline{t_i^\nu}} \Omega_p(S_g^j) \Longleftrightarrow \Omega_p(S_g^j) \xrightarrow{t_i^\nu} \Omega_p(S_g^{j+1}),$$

therefore,

$$\Omega_p(S_g^j) \xrightarrow{t_i^\nu \overline{t_i^\nu}} \Omega_p(S_g^j), \quad 1 \leq i \leq l-1.$$

We can delete all pairs of $t_i^\nu \overline{t_i^\nu}$ and the sequences of states the system will follow will not be altered. Therefore, we conclude that $\Omega_p(S_g^k) \xrightarrow{\mathcal{S}^2} \Omega_p(S_g^0)$, where $\mathcal{S}^2 = \overline{t_l^\nu}\ \overline{\mathcal{S}_l}\ \overline{\mathcal{S}_{l-1}} \ldots \overline{\mathcal{S}_2}\ \overline{\mathcal{S}_1}$.

Q.E.D. □

The following algorithm states the procedure discussed so far. We may observe that when we follow the global recovery sequence to replace the incorrect soft component, we need not go back to the initial state. The replacement can be performed just before the first key transition corresponding to this soft component. The sixth step of the following algorithm implements this observation.

Algorithm 7.1 *Execution of a Task Sequence with Sensory Verification and Error Recovery(One Soft Component)*

Input: The correct task sequence set and one element $\mathcal{S} = \mathcal{S}_1 t_1^\nu \overline{t_1^\nu} \mathcal{S}_2 t_2^\nu \overline{t_2^\nu} \ldots t_s^\nu \overline{t_s^\nu} \mathcal{S}_{s+1}$, from this set, where t_1^ν, t_2^ν, ..., t_s^ν are s key transitions on the soft objects, the initial global fuzzy state S_g^0, and the DFPN from which $\mathcal{S}$ is generated.

Output: None.

By-product: The *expected task* is efficiently executed, or recovers to the initial state should an unrecoverable error occur.

1. Add sensory verification transition $s.v._i$ and one or more $t_i^{\nu'}$ to each key transition t_i^ν as shown in Figure 7.10. Set the weighting factor of r_σ for each new $t_i^{\nu'}$ and $s.v._i$ as 1. M = 0.

2. Execute the next transition in the sequence until we meet a key transition t_i^v, fire it, and go to 4.

3. If we reach the end of the sequence, exit.

4. Fire the sensory verification transition $s.v._i$, get the token of new local fuzzy variable ρ_j in output place p_j.

5. If $\rho_j \in [0, \theta_j^1)$ and M reaches an upper bound, fire $\overline{t_i^v}\ \overline{\mathcal{S}_i}\ \overline{\mathcal{S}_{i-1}} \ldots \overline{\mathcal{S}_1}$, notify the high level controller and exit.

6. If $\rho_j \in [0, \theta_j^1)$, and there is no alternative global sequence, or, all alternative global sequences have been tried, $WF(\overline{t_i^v}) = 1/\prod_{t=1}^{i} P_t$, fire $\overline{t_i^v}\ \overline{\mathcal{S}_i}\ \overline{\mathcal{S}_{i-1}} \ldots \overline{\mathcal{S}_2}$, $M = M + 1$, $WF(\overline{t_i^v}) = 1$, go to 2.

7. If $\rho_j \in [0, \theta_j^1)$, and there is a next alternative global sequence, $WF(\overline{t_i^v}) = 1/\prod_{t=1}^{i} P_t$, fire $\overline{t_i^v}\ \overline{\mathcal{S}_i}\ \overline{\mathcal{S}_{i-1}} \ldots \overline{\mathcal{S}_1}$, $M = M+1$, $WF(\overline{t_i^v}) = 1$, choose the alternative sequence and go to 2.

8. If $\rho_j \in [\theta_j^1, \theta_j^2)$, and if for this key operation, there is no alternative transition, fire $t_i^{v'}$ and go to 4, else, go to 10.

9. Otherwise, go to 2.

10. Choose an alternative transition for this key operation to fire, and then go to 4.

7.5.2 Error Recovery for Multiple Soft Components

Multiple soft components are often present in a robotic workcell. A fuzzy marking and representation strategy for multiple soft components during different stages of operations such as assembly, disassembly, and IST, has been discussed in [12, 20]. In this subsection, we show how one of the soft components recovers from

an intermediate fuzzy state, if error conditions are detected for this component or a subassembly which contains this component. The method for execution of the task sequence with sensory verification and error recovery for this case is the same as the single soft component case discussed in the last subsection.

For a system which contains one soft component, when the system recovers from errors back to the initial state and the component is replaced by a new component or repaired, the original task sequence could be re-executed without any modification. For a system which contains multiple soft components, when the system goes back to the initial state and one soft component is replaced, the states of other soft components are in their intermediate fuzzy states. If we re-execute the same task sequence which was planned originally, we are expected to reach another failure state. To solve this problem, we may add a sensory identification procedure before each key transition. In the following theorem, we propose an alternative method to automatically modify the original task sequence after one error component is repaired or replaced.

Definition 7.10 *Sequence masking operation "−":* A set of transitions $\Delta = \{t_1, t_2, \ldots, t_n\}$ are deleted from a sequence of transitions, $\mathcal{S}$, i.e., we assume $\mathcal{S} = \mathcal{S}_1 t_1 \mathcal{S}_2 t_2 \ldots t_k \mathcal{S}_{k+1}$, where $\mathcal{S}_i$ is a subsequence of 0 or more transitions which does not contain t_j, $1 \leq i \leq k+1$, $1 \leq j \leq n$. Therefore,

$$\mathcal{S} - \Delta = \mathcal{S}_1 t_1 \mathcal{S}_2 t_2 \ldots t_k \mathcal{S}_{k+1} - \{t_1, t_2, \ldots, t_n\} = \mathcal{S}_1 \mathcal{S}_2 \ldots \mathcal{S}_{k+1}, \quad 1 \leq k \leq n. \quad (7.9)$$

Definition 7.11 *Sequence concatenation operation "+":* Suppose

$$\mathcal{S}_1 = t_1 t_2 \ldots t_r, \ \mathcal{S}_2 = t_{r+1} t_{r+2} \ldots t_n,$$

then

$$\mathcal{S}_1 + \mathcal{S}_2 = t_1 t_2 \ldots t_r + t_{r+1} t_{r+2} \ldots t_n = t_1 t_2 \ldots t_r t_{r+1} t_{r+2} \ldots t_n. \quad (7.10)$$

Definition 7.12 *Brother transition set:* Given a transition set Δ, $\Delta = \{t_1, t_2, \ldots, t_n\}$, its *brother transition set*, which is denoted by $\overline{\Delta}$, is $\overline{\Delta} = \{\overline{t_1}, \overline{t_2}, \ldots, \overline{t_n}\}$, where $\overline{t_i}$ is the brother transition of t_i, $1 \leq i \leq n$.

Definition 7.13 *Complete transition set* Δ^c*:* Given a transition set $\Delta = \{t_1, t_2, \ldots, t_n\}$, its corresponding complete transition set $\Delta^c = \Delta \bigcup \overline{\Delta} = \{t_1, \overline{t_1}, t_2, \overline{t_2}, \ldots, t_n, \overline{t_n}\}$.

Theorem 7.3 If $\mathcal{S} = t^1 t^2 \ldots t^n$, $\Delta^c = \{t_1, \overline{t_1}, t_2, \overline{t_2}, \ldots, t_s, \overline{t_s}\}$, then

$$\overline{\mathcal{S} - \Delta^c} = \overline{\mathcal{S}} - \Delta^c. \tag{7.11}$$

Proof: (1) If $t^i \notin \Delta^c$, then $\overline{t^i} \notin \Delta^c$, for $1 \leq i \leq n$. Because if we assume $\overline{t^i} \in \Delta^c$, then $t^i = t_j$ or $t^i = \overline{t_j}$, both of which will lead to a contradiction that $t^i \in \Delta^c$. Therefore,

$$\overline{\mathcal{S} - \Delta^c} = \overline{\mathcal{S}} = \overline{\mathcal{S}} - \Delta^c.$$

(2) If there exists at least one transition of t^i which belongs to Δ^c. For the sake of simplicity, we assume $\{t^1, t^2, \ldots, t^n\} \bigcap \Delta^c = \{t^j\}$ and $t^j = t_i$. Then,

$$\overline{\mathcal{S} - \Delta^c} = \overline{t^1 t^2 \ldots t^{j-1} t_i t^{j+1} \ldots t^n - \Delta^c} = \overline{t^1 t^2 \ldots t^{j-1} t^{j+1} \ldots t^n}$$

$$= \overline{t^n}\ \overline{t^{n-1}}\ \ldots \overline{t^{j+1}}\ \overline{t^{j-1}} \ldots \overline{t^1} = \overline{t^n}\ \overline{t^{n-1}} \ldots \overline{t^{j+1}}\ \overline{t^j}\ \overline{t^{j-1}} \ldots \overline{t^1} - \Delta^c$$

$$= \overline{t^1 t^2 \ldots t^{j-1} t^j t^{j+1} \ldots t^n} - \Delta^c = \overline{\mathcal{S}} - \Delta^c.$$

Q.E.D. □

Theorem 7.4 If $S_g^0 \xrightarrow{\mathcal{S}} S_g^k$, $\mathcal{S} = t^1 t^2 \ldots t^k$ and when $\mathcal{S}$ is executed, it stops at t^i, $1 \leq i \leq k$. Then, after error recovery to the initial state and the jth soft component is repaired or replaced, the resumption to the intermediate state where the failure happens, is guaranteed, *if* the task sequence from the initial state after recovery is modified as

$$\mathcal{S}' = t^1 t^2 \ldots t^i - (\Delta_1^c \bigcup \Delta_2^c \bigcup \ldots \bigcup \Delta_{j-1}^c \bigcup \Delta_{j+1}^c \bigcup \ldots \bigcup \Delta_r^c) + t^{i+1} t^{i+2} \ldots t^k, \tag{7.12}$$

where Δ_p represents the set of feasible key transitions for the pth soft component, $1 \leq p \leq r$.

Proof: Since $S_g^0 \xrightarrow{\mathcal{S}} S_g^k$, and $\mathcal{S} = t^1 t^2 \ldots t^i \ldots t^k$, which stops at t^i, according to Theorem 7.2, the recovery sequence is then $\mathcal{S}_r = \overline{t^i}\ \overline{\mathcal{S}_l}\ \overline{\mathcal{S}_{l-1}} \ldots \overline{\mathcal{S}_2}\ \overline{\mathcal{S}_1}$, where

$$\overline{\mathcal{S}_l}\ \overline{\mathcal{S}_{l-1}} \ldots \overline{\mathcal{S}_2}\ \overline{\mathcal{S}_1} = \overline{t^{i-1}}\ \overline{t^{i-2}} \ldots \overline{t^1} - (\Delta_1^c \bigcup \Delta_2^c \bigcup \ldots \bigcup \Delta_{j-1}^c \bigcup \Delta_j^c \bigcup \Delta_{j+1}^c \bigcup \ldots \bigcup \Delta_r^c).$$

When we reach the initial state, we don't fire any key transition, therefore, the fuzzy state for each soft object does not change. After we replace the jth soft component which is not processed, we should process it to its desired intermediate state. Therefore, the subset of the key transition set for jth soft component is refired and the modified sequence is

$$\begin{aligned}
\mathcal{S}' &= \overline{\overline{t^i}\ \overline{t^{i-1}}\ \overline{t^{i-2}} \ldots \overline{t^1} - (\Delta_1^c \bigcup \Delta_2^c \bigcup \ldots \bigcup \Delta_{j-1}^c \bigcup \Delta_{j+1}^c \bigcup \ldots \bigcup \Delta_r^c)} + t^{i+1} t^{i+2} \ldots t^k \\
&= \overline{\overline{t^i}\ \overline{t^{i-1}}\ \overline{t^{i-2}} \ldots \overline{t^1}} - (\Delta_1^c \bigcup \Delta_2^c \bigcup \ldots \bigcup \Delta_{j-1}^c \bigcup \Delta_{j+1}^c \bigcup \ldots \bigcup \Delta_r^c) + t^{i+1} t^{i+2} \ldots t^k \\
&= t^1 t^2 \ldots t^i - (\Delta_1^c \bigcup \Delta_2^c \bigcup \ldots \bigcup \Delta_{j-1}^c \bigcup \Delta_{j+1}^c \bigcup \ldots \bigcup \Delta_r^c) + t^{i+1} t^{i+2} \ldots t^k.
\end{aligned}$$

Q.E.D. □

7.6 An Algorithm for Generating an Executable fuzzy Petri Net

We have already discussed the basic theory of reasoning in a fuzzy Petri net and its application in error detection and recovery. After the planning stage, we may get a 'script' of operations sequences for execution and when an error occurs, we may refer to the net representation and automatically find a recovery sequence to get back to a previous state, and then follow a possibly modified sequence to recover to the interrupting state. Sometimes because of the high probability of errors during the implementation of a sequence, we are required to create an *executable fuzzy Petri net*, so that more flexibility of alternative operations sequences and robust on-line selection of enabling transitions can be incorporated. In this approach, the same

fuzzy reasoning rules for r_ρ can be created with appropriate weighting factors. The first step for creating an executable fuzzy Petri net is to find a feasible, complete, and correctly ordered sequence from the original system fuzzy Petri net. The next step is described by an algorithm shown below.

Algorithm 7.2 *Generation of an executable fuzzy Petri net for implementation of a robotic task*(We assume for this discussion that one soft component exists in the system, but this can be generalized.)

Input: A fuzzy Petri net generated from an AND/OR net for the system, a feasible, complete, and correctly ordered task sequence, $t_1 t_2 \ldots t_n$, searched from the net.
Output: An executable fuzzy Petri net.

1. Suppose (i) all weighting factors of r_ρ for key transitions are WF_1, WF_2, ..., WF_r; (ii) all thresholds(ranges of real numbers) for m transitions in the system are θ_1, θ_2, ..., θ_m; (iii) t_{i_1}, t_{i_2}, ..., t_{i_r} are key transitions in the sequence, $i_1 < i_2 < \ldots < i_r$; (iv) Let L be the initial token value of the soft object(if $r = 0$, then $L = 1.0$). M = 0. Init1 = 1. Init2 = L. $H = 1.0$ if $r = 0$, otherwise, H is assigned an integer.

2. $M = M + 1$. If $M \leq r$, go to 5.

3. For all t_j, $Init1 \leq j \leq n$, $\theta_j = [Init2, H)$.

4. For any transition t_j which has no threshold, if its brother transition $\overline{t_j}$ has threshold θ, assign θ to t_j, otherwise, assign $[L, H)$ to t_j. Exit.

5. For all t_j, $Init1 \leq j \leq i_M$, $\theta_j = [Init2, Init2 \times WF_M)$.

6. $Init1 = i_M + 1$. $Init2 = Init2 \times WF_M$. Go to 2.

In real-time execution of the operations sequence on the executable fuzzy Petri net, we need to model sensor and sensing operations in the net to handle uncertainty.

After sensing transitions are added to the net, the outcome on-line local fuzzy values depend on the detected sensor value mapping, i.e., $r_\rho^{t_i}(S_i)$, where $r_\rho^{t_i}$ is a sensing mapping function, and S_i is a sensed value.

Two types of sensing are used in the fuzzy Petri net. One is called *discrete sensing*, and the other is called *continuous sensing*. Discrete sensing indicates that r_ρ^i and $r_\rho^i(S_i)$ take on discrete values, while continuous sensing means that r_ρ^i and $r_\rho^i(S_i)$ take on continuous values. After we obtain an executable fuzzy Petri net using the above algorithm, we add sensing transitions near some key transitions and after each key transition is fired, we fire the sensing transition, and let the local fuzzy variables of the sensing output decide the following enabled transitions. Then, given the initial state, a sequence will be automatically followed to reach the final state and perform error recovery operations or subsequences if necessary.

A simplified version of the peg-cylinder assembly system(without robot), its executable fuzzy Petri net, and the net with discrete sensing and continuous sensing added, are shown in Figure 7.11. In Figure 7.11(b), cut, lubricate, and insert(or followed by several loops of remove, insert) is the only feasible, complete, and correct sequence found on-line. In Figure 7.11(c), after the 'cut sensing' transition is added, we assume 'cut' transition having a weighting factor of 1 and 'cut sensing' transition having a continuous sensing value $1 \leq WF(S_i) \leq 2$. Depending on this value, a repeated trial error recovery may be necessary if the peg is not cut to the right size. For the 'insertion sensing' transition, if the insertion operation succeeds, a final state will be reached that no transition is enabled anymore. Otherwise, a 'remove' transition may be fired to invoke a global error recovery subsequence.

7.7 Examples

In this section, we show two examples, one of which is oriented to alternative local error recovery, and the other to alternative global error recovery. When a

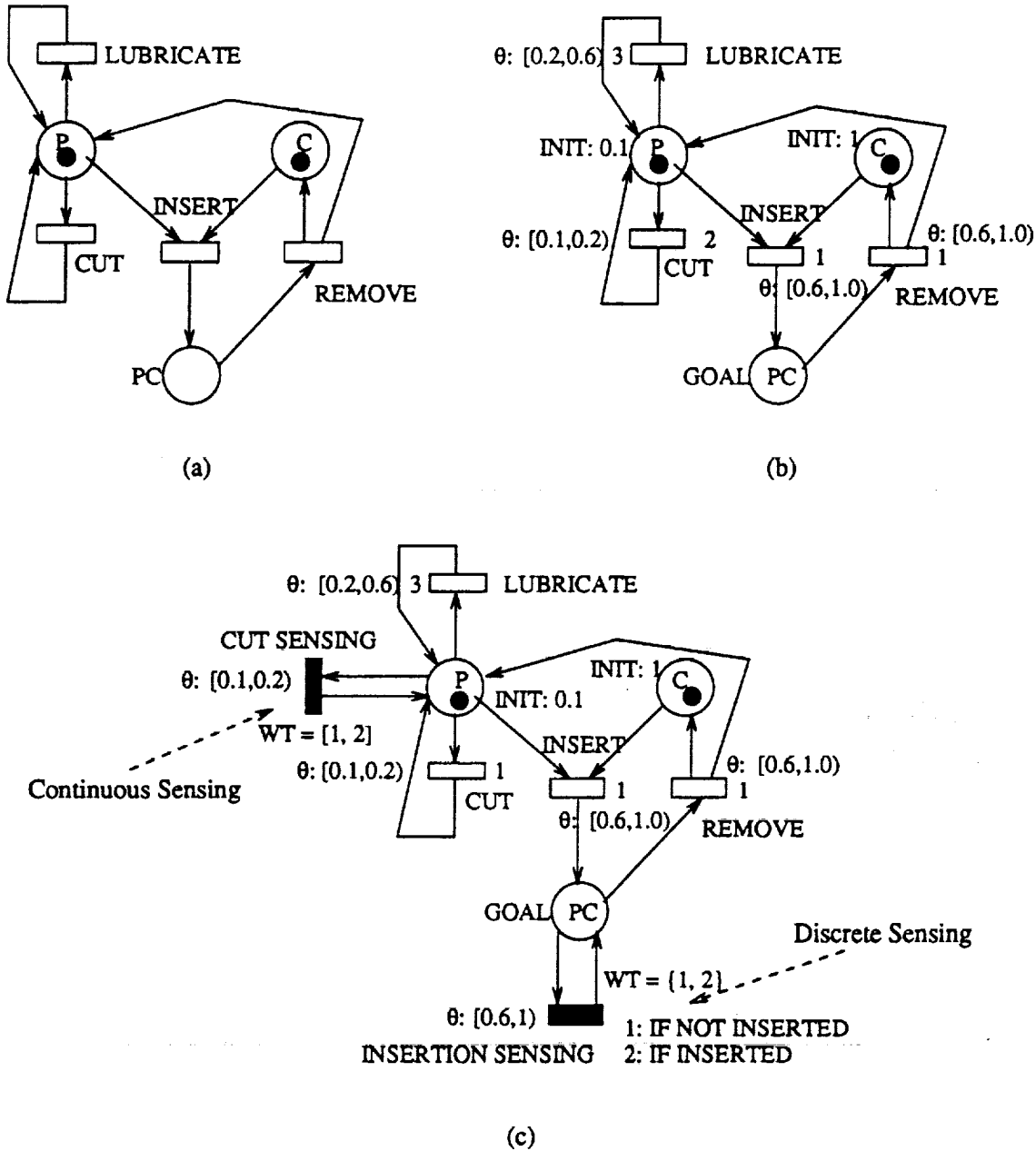


Figure 7.11: An example for the executable fuzzy Petri net. (a) Petri net example for peg-cylinder assembly system(without robot). (b) The executable fuzzy Petri net(no sensors). (c) The net with discrete sensor(global error recovery) and continuous sensors(repeated trial error recovery).

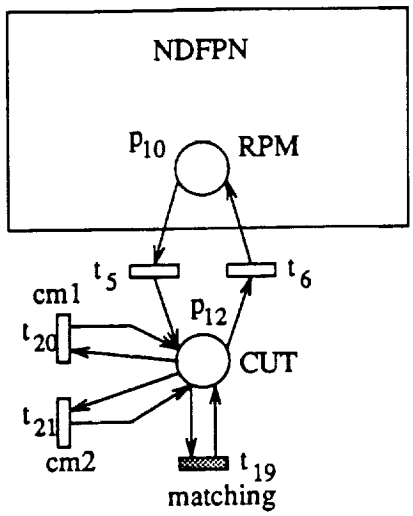


Figure 7.12: An error recovery mechanism for t_5 in Figure 7.1.

local recovery procedure fails to recover the error after a finite number of retries, a backtracking path will be automatically followed to an earlier stage of the task or the initial state, then, the original sequence will be executed again, or, an alternative sequence will be called to replace the original one.

7.7.1 Example of Alternative Local Error Recovery

In this subsection, we show sensor-based alternative error recovery for the example shown in Figure 7.1, where a fuzzy Petri net representation is illustrated. As we see from Figure 7.1, t_5 is a key transition with a weighting factor 2. Based on the algorithm for *execution of task sequence with sensory verification and error recovery*, we add an *s.v. matching* transition, and $cm1$, $cm2$ transitions for local error recovery. $cm1$ corresponds to the first cutting machine and $cm2$ corresponds to the second cutting machine. The modified partial fuzzy Petri net is shown in Figure 7.12. Notice that it's not necessary to add a special transition for decreasing ρ because in this case, each unsuccessful matching will automatically decrease the fuzzy value.

Initially, the prime token value $\sigma^{10} = 0.1$, $WF_5 = 2$, $\sigma^{12} = 2 \times 0.1 = 0.2$, and

the local fuzzy variable $\rho_{12} = \sigma^{12}$. The three ranges $[0, \theta^1)$, $[\theta^1, \theta^2)$ and $(\theta^2, \theta^3]$ are $[0, 0.12)$, $[0.12, 0.18)$, $[0.18, 0.2]$, respectively, for ρ_{12} in p_{12}. $[\theta^1, \theta^2)$ is divided into two subranges, $[0.12, 0.15)$ and $[0.15, 0.18)$, so that the alternative error recovery with two cutting machines can be implemented. For transition t_{19}, $r_\rho^{t_{19}}$ is represented as

$$\rho_{12}^{new} = \rho_{12}^{old} \times WF_{19} \times (1 - match) + match \times WF_{19} \times 0.2,$$

$match = 1$ if $WF_{19} > 0.94$, otherwise, $match = 0$. Moreover, $WF_{20} = WF_{21} = 1$. We list three possible cases of derivations of sequences as follows:

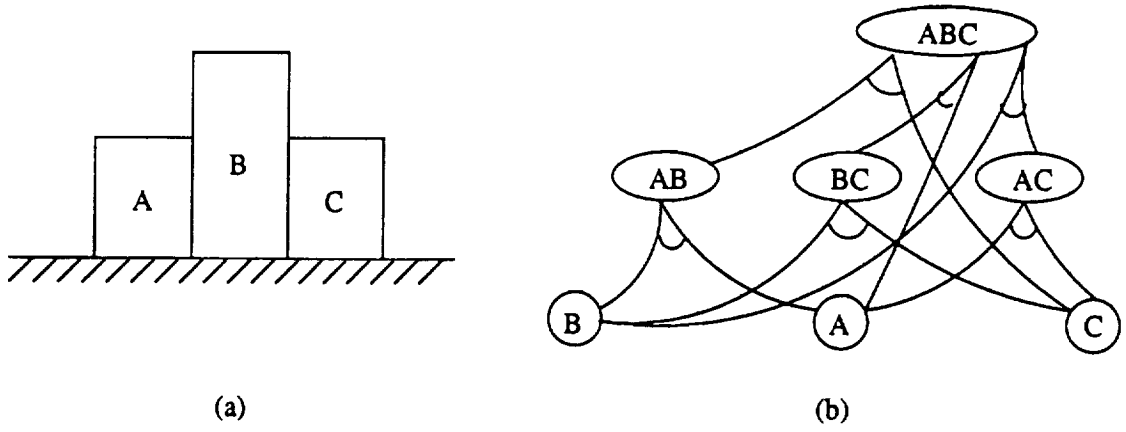
Case 1: The weighting factor of random transition, *matching*, is $WF_{19} = 0.95$. Thus, $\rho_{12} = 0.2 \times 0.95 = 0.19 \in (\theta^2, \theta^3]$ and the sequence succeeds at this point.

Case 2. $WF_{19} = 0.85$, $\rho_{12} = 0.2 \times 0.85 = 0.17 \in [\theta^1, \theta^2)$. $cm1$ will be fired(cutting machine 1 is used again) and $\rho_{12} = 0.17$. If again, $WF_{19} = 0.85$, $\rho_{12} = 0.1445 \in [0.12, 0.15)$. $cm2$ is fired at this time and we suppose WF_{19} now becomes 0.98. Correspondingly, $\rho_{12} = 0 + 0.98 \times 0.2 = 0.196$, therefore, the local error recovery succeeds in this case.

Case 3. At the last step of the above case, if we assume WF_{19} is still 0.85. Correspondingly, $\rho_{12} = 0.85 \times 0.1445 \doteq 0.1226$. $cm2$ is fired again and we suppose WF_{19} is still 0.85, then $\rho_{12} \doteq 0.1042 \in [0, 0.12)$. Therefore, a global error recovery back to the initial state is necessary and the peg should be replaced.

7.7.2 Example of Alternative Global Error Recovery

Suppose we have three blocks, A, B, and C. The robot is required to *assemble* these blocks and the final configuration for this task is shown in Figure 7.13(a). The AND/OR net representation of all feasible components, A, B, and C, all feasible subassemblies, AB, BC, and AC, and the assembly, ABC, and their geometric relations, are illustrated in Figure 7.13(b). For the sake of simplicity and symmetry, we assume the assembly task should start from putting A on the platform first. Of course, more complicated cases could also be considered and the generalities of



(a) (b)

Figure 7.13: (a) The final configuration of the assembly of blocks A, B, and C. (b) The AND/OR net representation of this assembly task.

the following discussion will not be lost. Using the mapping algorithm, we obtain a corresponding Petri net. Figure 7.14 shows a subnet of this Petri net which does not include subassembly BC and the related transitions. In this net, t_6 is a nondeterministic transition in the sense that we don't know whether the assembly operation of AC and B could be accomplished successfully because of the imprecise distance between A and C when AC is obtained.

We may consider AC as a soft object and the distance between A and C in the subassembly AC as a soft parameter. We also consider t_6 as a random transition with a weighting factor $WF(t_6)$. The local fuzzy values for AC and ABC are ρ_{AC} and ρ_{ABC}, respectively, and the firing rule r_ρ for t_6 is

$$\rho_{ABC} = \rho_{AC} \times WF_6 \times (1 - match) + match \times WF_6 \times 1,$$

where $match = 1$ if $WF_6 \geq 0.98$, otherwise, $match = 0$. All other transitions in the net are deterministic and having weighting factors of 1. The firing rules of these transitions are the same as shown in (7.1), (7.2) and (7.3).

There are four possible task sequences for this example, i.e., t_2t_7, t_3t_6, $t_2t_1t_3t_6$ and $t_3t_4t_2t_7$. Suppose we choose t_3t_6 to execute. If the distance between A and C is too small to fit B inside them, t_6t_5 will be fired several times. If the random

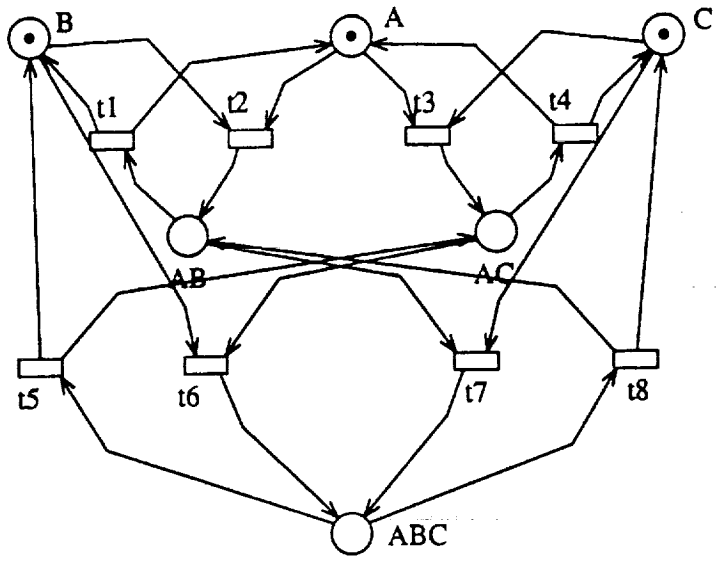


Figure 7.14: The subnet of the Petri net representation of the example of assembling A, B and C.

weighting factor for t_6 is below 0.98 for several consecutive times, the local fuzzy value in AC will strictly decrease and eventually force a global error recovery of this sequence, i.e., t_4 will be fired and AC decomposed. Then, either an alternative planned task sequence, t_2t_7, or, the originally chosen sequence, t_3t_6, is reexecuted. In the latter case, C is reassembled with A and the distance between A and C at this time may be adjusted and the sequence may be successful. Otherwise, if several trials of the sequence are performed and it still fails, an alternative task sequence will be forced to execute in place of the original one by the system supervisor. Figure 7.15 visually shows the error recovery procedures discussed above. A partial executable fuzzy Petri net, with continuous sensing for assembling AC and discrete sensing for assembling ABC, are shown in Figure 7.16. Because there is no soft component existing in the system, we assign $[1, 2)$ to each transition and the initial tokens are all 1 based on the algorithm shown in Section 7.6.

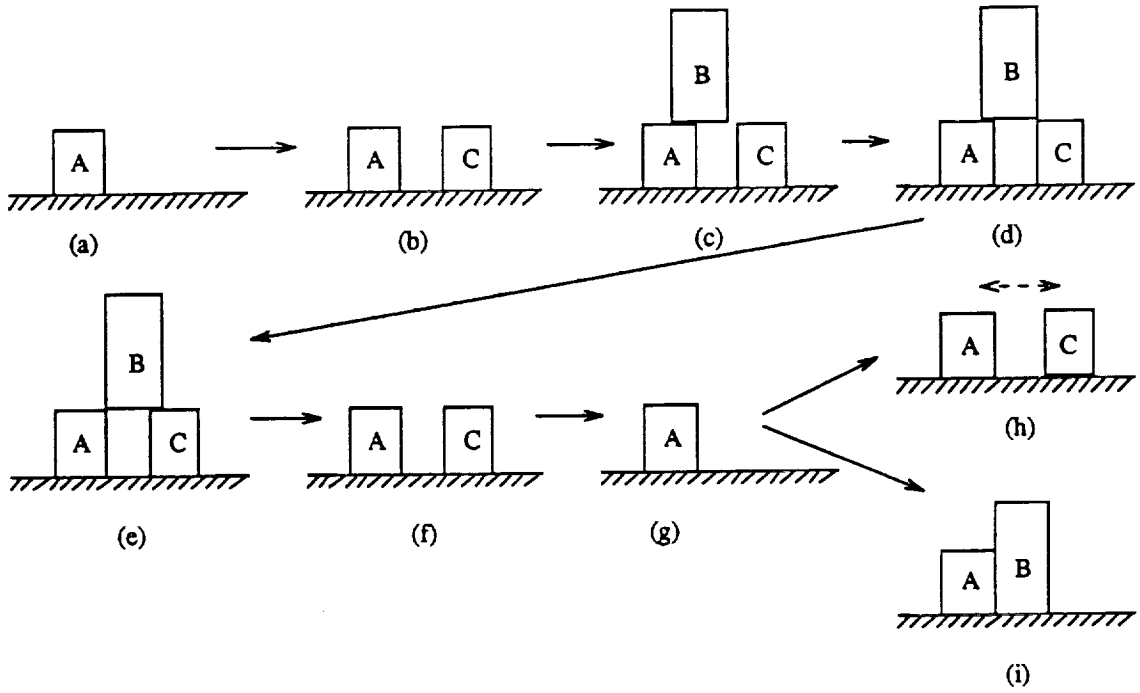


Figure 7.15: The possible errors and the corresponding recovery procedures for the task to assemble A, B, and C. (a) Put A. (b) Put C. (c) Put B between A and C and fails. (d) Same as (c). (e) Same as (c). (f) Remove B. (g) Remove C. (h) Put C, possibly with distance adjusted. (i) Follow an alternative sequence and put B.

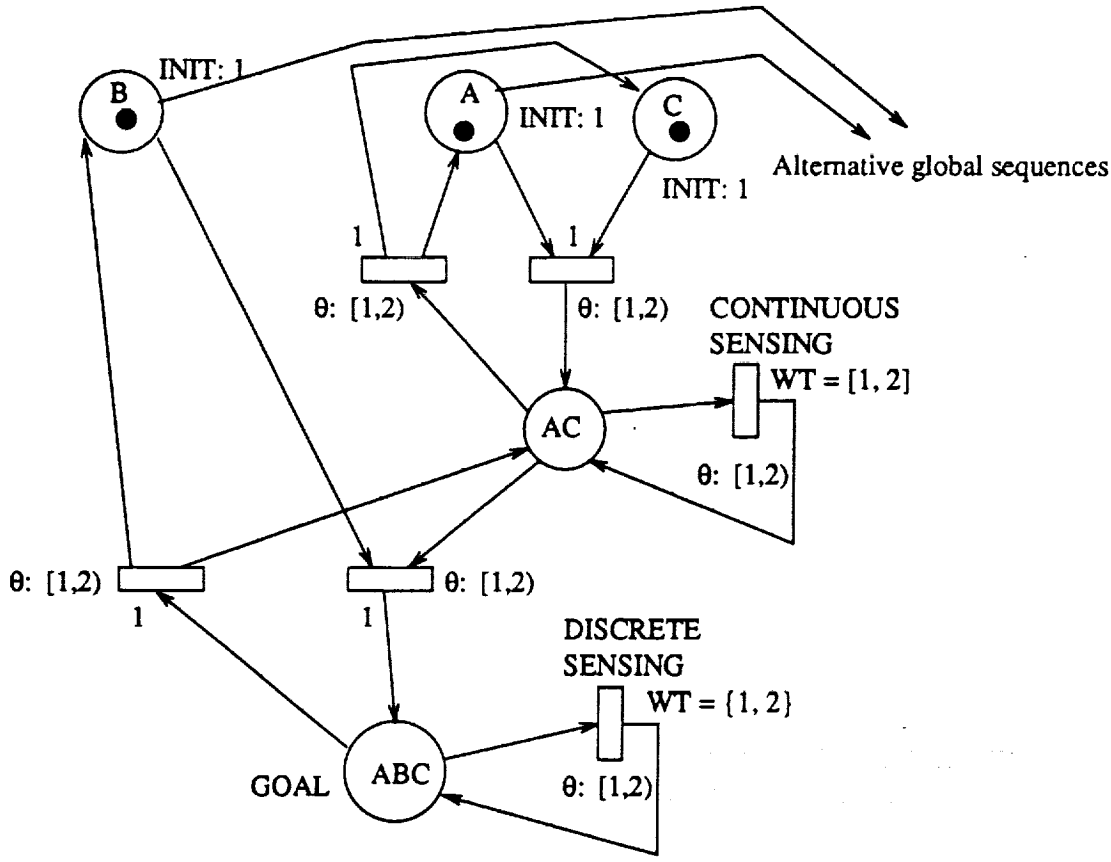


Figure 7.16: The fuzzy Petri net representation for ABC assembly tasks.

7.8 Conclusion

In this chapter, we introduce a novel sensor-based error recovery strategy based on the fuzzy Petri net representation of robotic workcells and the fuzzy property of sensory verification. Fuzzy Petri nets are used to model fuzzy process state, including uncertainty of local parameters and a global fuzzy variable associated with 'degree of completion'. Using fuzzy Petri nets, we can enforce required precedence of operations for a set of 'key' operations, and represent 'soft' objects which have changing properties or internal states.

We have shown in this chapter an approach to implementation of embedded error recovery strategies which change the precedence of operations by sensing fuzzy values. The fuzzy information propagates and resequences operations to accomplish task goals. When the expected information is obtained, the sequence will not be altered. However, when an error is detected, the local recovery loop or the global recovery path will be followed to guarantee that correct fuzzy degrees of completion of soft objects are reached. After the global recovery is finished, different sequences for re-execution are proposed for the case of a single soft component, and that of multiple soft components, respectively. Fuzzy values discussed in this chapter are important for searching in the planning stage as well as for sensing, verification, detection, and reasoning about sensory conditions in the execution stage.

Future directions of this work will concentrate on the applications of the theories proposed in this chapter to more generalized robotic systems and on the decomposition of high level task sequences into lower level motion and execution sequences, so that a hierarchical error detection and recovery mechanism may be incorporated. Sensor-based control and sensor-based verification and error recovery can be integrated, and sensor resources can be coordinated in the whole robotic system. Such a sensor-based robotic control workcell would be more flexible, adaptive, and fault-tolerant and would reduce the effort required for implementation of new tasks.

CHAPTER 8
CONCLUSIONS

8.1 Summary

The principal contribution of this thesis is to task sequence representation, planning, and error recovery for robotic systems. Both plan generation and plan execution are modeled by Petri nets and fuzzy Petri nets, which provide an efficient methodology to simulate and analyze the properties of the systems. The fuzzy Petri net is shown to be a promising tool to incorporate uncertainty into the system model. This treatment of modeling uncertainty using the fuzzy Petri net and its applications to robot task planning and sensor-based error recovery has provided a framework for analysis and design in many problem areas. The work presented in this thesis should also stimulate further interest and research in this direction.

Using the methodology and theory presented in this thesis, we are able to represent, plan, and execute feasible operations sequences for a specific robotic task. The input to our planner is the descriptions of the components and geometric relations among components, feasible operations, initial and final system states, and the working environment. We could use this information to build up a system representation for robotic task sequences. This model is then mapped to a Petri net, which will be decomposed to lower level representations based on the specifications of the execution-level devices. On any level of decomposition, we may search one or more feasible sequences and may want to analyze the properties of the system including safeness, liveness, and reversibility. After fuzzy Petri nets are introduced to represent uncertainty and incompleteness with the system model, we are able to reduce the search space for sequences in the Petri net model by marking subgoals using global fuzzy variables, and we can monitor the degree of completion during

the planning process for verifying and pruning infeasible partial sequences.

One major approach in this work is that three kinds of fuzzy variables are incorporated in the fuzzy Petri net model and the reasoning mechanisms and fuzzy state representation could be built into the fuzzy Petri net. A local fuzzy variable is used to characterize lower level mechanical and geometric parameters of a device or object in the robotic system, a fuzzy marking variable is introduced to denote the uncertainty that an event occurs, and a global fuzzy variable characterizes the degree of completion for a global task. In particular, fuzzy reasoning rules for three kinds of operations, assembly, disassembly, and IST operations derived from the robot assembly system, are developed to reason about the global fuzzy variables. Mutually exclusive transitions are used to select and reason about the execution of robotic actions on-line based on both local and global fuzzy variables. Different error recovery strategies including retrying error recovery, local alternative error recovery, and global alternative error recovery are shown to be efficient and reliable to execute a task sequence under uncertainty.

Some basic cases of fuzzy Petri nets are analyzed based on local fuzzy variables and fuzzy marking variables. In analyzing the case of transition firing depending on input local fuzzy variables, a subclass of the case was found to satisfy the liveness, safeness, and reversibility under a set of conditions. One advantage of using local fuzzy variables to represent the state of execution-level devices including the robot is that the theory of fuzzy sets and fuzzy mathematical operations can be directly used to compute and analyze the uncertainty using fuzzy numbers.

In this research, the efficient, reliable, and robust planning and execution of robotic task sequences are guaranteed through the hierarchical task decomposition and implementation of error recovery strategies incorporated in the Petri net. Our results represented in this thesis provide a novel methodology and applications of robot task planning with uncertainty.

8.2 Future Work

A number of research directions deserve further consideration beyond the results presented in this thesis.

- The selection and evaluation of all feasible task sequences based on the AND/OR net and Petri net representation of task sequence plans is an important extension to the work done here in task sequence planning. In particular, for the selection and execution of parallel operations during the planning process one needs to analyze all feasible enabled transitions in terms of resource conflicts and other factors including timing, cost, flexibility, and reliability. Another possibility is to execute in parallel an error recovery sequence and some operations in the original sequence when the error is independent with these operations. Analysis of uncertainty with parallel execution of operations also needs to be performed.

- The synthesis of planning strategies under uncertainty within a generic hierarchical structure for decomposition may be used to handle errors which appear during the decomposition of a task sequence towards the final net. For example, the working environment of the execution of the sequence may be changed not by sensing and manipulation, but by other random factors. Thus, a sensory verification procedure might be necessary to check some unsafe substate before the substate is processed by the current operation.

- A planning for planning problem may be important to ensure the property of reversibility of the final net during decomposition. Sometimes, we need to recover from an error to the initial state while retaining some lower level plans for future refiring and discarding of others. The coordination and scheduling for processing these plans are incorporated into the recovery procedure. When

a sequence is reinitiated from the initial state, we should guarantee state reservation for those waiting components and execution of unchanged plans which might be verified.

- More general cases of fuzzy Petri nets and their properties should be explored and analyzed when fuzzy Petri nets are used to model more complex systems. It might be useful to look at the interrelationships among the three kinds of fuzzy variables during fuzzy reasoning by rules. More detailed lower level representation and integration of these three variables are needed to expose the necessity to use them in modeling uncertainty. Other forms of non-fuzzy probabilistic reasoning could also be incorporated within this framework.

- The design of reasoning structures within Petri nets is another important issue to design a fuzzy Petri net. Many learning techniques can be explored to generate a rule base. Neural networks provide a good method to represent and train these rules. The mapping from real sensory data to fuzzy numbers for reasoning in the fuzzy Petri net is also an interesting topic.

- Fuzzy Petri nets proposed in this thesis should be applied to modeling, analysis, and simulations of other related areas such as knowledge representation, knowledge reasoning, design of expert systems, discrete event systems, flexible manufacturing systems, scheduling, and other kinds of applications which need to handle uncertainty. The feedback from applications and implementation may bring revisions and enhancements to the theory of fuzzy Petri net initiated in this thesis.

REFERENCES

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms,* Addison-Wesley: Reading, MA, 1974.

[2] R. Y. Al-Jaar and A. Desrochers, "Petri Nets in Automation and Manufacturing," in *Advances in Automation and Robotics,* G. N. Saridis, ed. Vol. 2, JAI Press, pp. 153-218, 1991.

[3] P. Alanche *et al.*, "PSI: A Petri Net Based Simulator for Flexible Manufacturing Systems," in *Advances in Petri Net 1984 (Lecture Notes in Computer Science 188),* G. Rozenberg, Ed. New York: Springer-Verlag, pp. 1-14, 1984.

[4] G. Berthelot, "Checking Properties of Nets Using Transformations," in *Advances in Petri Nets 1985 (Lecture Notes in Computer Science 222),* G. Rozenberg, Ed. New York: Springer-Verlag, pp. 19-40, 1985.

[5] G. Berthelot, "Transformations and Decompositions of Nets," in *Advances in Petri Nets 1986 (Lecture Notes in Computer Science 254),* G. Rozenberg, Ed. New York: Springer-Verlag, pp. 359-376, 1986.

[6] A. Bourjault, *Contribution a une Approche Méthodologique de L'Assemblage Automatisé: Elaboration Automatique des Séquences Opératoires.* Thèse d'État, Université de Franche-Comté, Besançon, France, Nov. 1984.

[7] R. A. Brooks, "Symbolic Error Analysis and Robot Planning," *The International Journal of Robotics Research,* Vol. 1, No. 4, pp. 29-68, Winter 1982.

[8] A. Camurri and P. Franchi, "An Approach to the Design and Implementation of the Hierarchical Control System of FMS, Combining Structured Knowledge Representation Formalisms and High-level Petri Nets," in *Proc. of IEEE International Conference on Robotics Automation,* pp. 520-525, Cincinnati, OH, 1990.

[9] T. Cao *et al.*, *Automated Handling of Garments for Pressing,* Year One Final Report, The Educational Foundation for the Fashion Industries and Center for Manufacturing Productivity and Technology Transfer, Jan. 1991.

[10] T. Cao and A. C. Sanderson, *Task Sequence Planning in a Robot Workcell Using AND/OR Nets,* Technical Report, CIRSSE #94, Rensselaer Polytechnic Institute, June 1991.

[11] T. Cao and A. C. Sanderson, "Task Sequence Planning in a Robot Workcell Using AND/OR Nets," in *Proc. of IEEE International Symposium on Intelligent Control*, pp. 239-244, Arlington, VA, Aug. 1991.

[12] T. Cao and A. C. Sanderson, "Task Sequence Planning Using Fuzzy Petri Nets," in *Proc. of IEEE International Conference on Systems, Man, and Cybernetics*, pp. 349-354, Charlottesville, VA, Oct. 1991.

[13] T. Cao and A. C. Sanderson, "Automatic Decompositions of Assembly Sequence Plans," in *Proc. of Third Annual Conference on Intelligent Robotic Systems for Space Exploration*, pp. 19-28, Troy, NY, Nov. 1991.

[14] T. Cao and A. C. Sanderson, "Sensor-based Error Recovery for Robotic Task Sequences Using Fuzzy Petri Nets," in *Proc. of IEEE International Conference on Robotics and Automation*, pp. 1063-1069, Nice, France, May 1992.

[15] T. Cao and A. C. Sanderson, "Task Decomposition and Analysis of Assembly Sequence Plans Using Petri Nets," in Proc. of *Third International Conference on Computer Integrated Manufacturing*, pp. 138-147, Troy, NY, May 1992.

[16] T. Cao and A. C. Sanderson, "AND/OR Net Representation for Robotic Task Sequence Planning," *IEEE Trans. on Robotics and Automation*, in review.

[17] T. Cao and A. C. Sanderson, "A Fuzzy Petri Net Approach to Reasoning about Uncertainty in Robotic Systems," in *Proc. of IEEE International Conference on Robotics and Automation*, Atlanta, GA, May 1993, in press.

[18] T. Cao and A. C. Sanderson, "Variable Reasoning and Analysis about Uncertainty with Fuzzy Petri Nets," in *Proc. of 14th International Conference on Application and Theory of Petri Nets*, Chicago, IL, June 1993, in press.

[19] T. Cao and A. C. Sanderson, "Task Decomposition and Analysis of Robotic Assembly Task Plans Using Petri Nets," *Journal of Robotic Systems*, in review.

[20] T. Cao and A. C. Sanderson, "Task Sequence Planning Using Fuzzy Petri Nets," *IEEE Trans. on Systems, Man, and Cybernetics*, in review.

[21] T. Cao and A. C. Sanderson, "Sensor-based Error Recovery for Robotic Task Sequences Using Fuzzy Petri Nets," *IEEE Trans. on Systems, Man, and Cybernetics*, in review.

[22] T. Cao and A. C. Sanderson, "Representation and Analysis of Uncertainty Using Fuzzy Petri Nets," *Journal of Intelligent & Fuzzy Systems*, in review.

[23] J. Cardoso, R. Valette, and D. Dubois, "Petri Nets with Uncertainty Markings," in *Advances in Petri Nets 1990 (Lecture Notes in Computer Science)*, G. Rozenberg, Ed. New York: Springer-Verlag, pp. 64-78, 1990.

[24] S. Chen, J. Ke and J. Chang, "Knowledge Representation Using Fuzzy Petri Nets," *IEEE Trans. on Knowledge and Data Engineering*, Vol. 2, No. 3, pp. 311-319, Sep. 1990.

[25] C. L. Chen, "Automatic Assembly Sequences Generation by Pattern Matching," *IEEE Trans. on System, Man, and Cybernetics,* Vol. 21, No. 2, pp.376-389, March/April, 1991.

[26] J. M. Colom, J. Martinez and M. Silva, "Packages for Validating Discrete Production Systems Modeled with Petri Nets," in *Applied Modeling and Simulation of Technological Syst.,* edited by P. Borne and S. G. Tzafestas, North-holland, pp. 529-536, 1987.

[27] F. Commoner, "Deadlocks in Petri Nets," Wakefield, Applied Data Research, Inc., Report # CA-7206-2311, 1972.

[28] J. A. Darringer and M.W. Blasgen, "MAPLE: A High Level Language for Research," in *Mechanical Assembly,* IBM Research Report RC 5606, IBM T. J. Waston Research Center, Yorktown Heights, N.Y., 1975

[29] T. L. De Fazio and D. E. Whitney, "Simplified Generation of All Mechanical Assembly Sequences, " *IEEE Journal of Robotics and Automation,* Vol. RA-3, No. 6, pp. 640-658, Dec. 1987. Also see corrections on the same journal, RA-4, No. 6, pp. 705-708, Dec. 1988.

[30] Edsger W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik,* Vol. 1, pp. 269-271, 1959.

[31] C. J. Divona *et al.,* " A Teleoperation for On-Orbit Manipulation," in *Proc. of Remote Systems and Robotics in Hostile Environments,* pp. 143-149, Pasco, Washington, 1987.

[32] B. R. Donald, *Error Detection and Recovery in Robotics*, Lecture Notes in Computer Science 336, New York: Springer-Verlag, 1989.

[33] H. A. ElMaraghy and J. M. Rondeau, "Automated Planning and Programming Environments for Robots," *Robotica,* Vol. 10, No. 1, pp. 75-82, 1992.

[34] M. Erdmann, "Randomization in Robot Tasks," *The International Journal of Robotics Research,* Vol. 11, No. 5, Oct. 1992, pp. 399-436.

[35] P. J. Fielding, F. DiCesare, and G. Goldbogen, "Error Recovery in Automated Manufacturing through the Augmentation of Programmed Processes," *Journal of Robotic Systems*, Vol. 5, No. 4, pp. 337-362, Aug. 1988.

[36] R. Fikes and N. Nilsson, "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving," *Artificial Intelligence,* 2: 189-208, 1971.

[37] R. Fikes, P. Hart, and N. Nilsson, "Learning and executing generalized robot plans," in *Readings in Artificial Intelligence*, N. Nilsson and B. Webber, Eds. Palo Alto, CA: Tioga, pp. 231-249, 1981.

[38] M. L. Garg, S. I. Ahson, and P. V. Gupta, "A Fuzzy Petri Net for Knowledge Representation and Reasoning," *Information Processing Letters*, Vol. 39, pp. 165-171, 1991.

[39] H. J. Genrich and K. Lautenbach, "Systems Modeling with High-Level Petri Nets," *Theoretical Computer Sciences,* 13, pp. 109-136, 1981.

[40] H. J. Genrich, "Predicate/Transition Nets," in *Lecture Notes in Computer Science,* No. 254, pp. 207-247, New York: Spring-Verlag, 1987.

[41] M. Gini and G. Gini, "Towards Automatic Error Recovery in Robot Programs," in *Proc. of 8th International Joint Conference on Artificial Intelligence*, pp. 821-823, Karlsruhe, Germany, Aug. 1983.

[42] A. Giordana and L. Saitta, "Modeling Production Rules by Means of Predicate Transition Networks," *Information Sciences,* 35, pp. 1-41, 1985.

[43] J. Hendler, A. Tate, and M. Drummond, "AI Planning: Systems and Techniques," *AI Magazine,* pp. 61-77. Summer 1990.

[44] M. A. Holiday and M. K. Vernon, "Performance Estimates for Multiprocessor Memory and Bus Interference," *IEEE Trans. on Computers,* Vol. C-36, pp. 76-85, Jan. 1987.

[45] L. S. Homem de Mello and A. C. Sanderson, "An AND/OR Graph Representation of Assembly Plans," in *AAAI-86 Proc. of the Fifth National Conference on Artificial Intelligence,* pp. 1113-1119, 1986.

[46] L. S. Homem de Mello and A. C. Sanderson, "Planning Repair Sequences Using the AND/OR Graph Representation of Assembly Plans," in *Proc. of IEEE International Conference on Robotics and Automation*, pp. 1861-1862, Philadelphia, PA, Apr. 1988.

[47] L. S. Homem de Mello and A. C. Sanderson, "AND/OR Graph Representation of Assembly Plans," *IEEE Trans. on Robotics and Automation,* Vol. 6, No. 2, pp. 188-199, Apr. 1990.

[48] L. S. Homem de Mello and A. C. Sanderson, "Representations of Mechanical Assembly Sequences," *IEEE Trans. on Robotics and Automation,* Vol. 7, No. 2, pp. 211-227, Apr. 1991.

[49] L. S. Homem de Mello and A. C. Sanderson, "A Correct and Complete Algorithm for the Generation of Mechanical Assembly Sequences," *IEEE Trans. on Robotics and Automation,* Vol. 7, No. 2, pp. 228-240, Apr. 1991.

[50] L. S. Homem de Mello and A. C. Sanderson, "Two Criteria for the Selection of Assembly Plans: Maximizing the Flexibility of Sequencing the Assembly Tasks and Minimizing the Assembly Time Through Parallel Execution of Assembly Tasks," *IEEE Trans. on Robotics and Automation,* Vol. 7, No. 5, pp. 626-633, Oct. 1991.

[51] S. A. Hutchinson and A. C. Kak, "Spar: A Planner That Satisfies Operational and Geometric Goals in Uncertain Environments," *AI Magazine,* pp. 30-61, Spring 1990.

[52] M. Jantzen, "Structured Representation of Knowledge by Petri Nets as an aid for Teaching and Research," in *Net Theory and Applications (Lecture Notes in Computer Science 84),* New York: Springer-Verlag, pp. 507-516, 1980.

[53] M. D. Jeng, *Theory and Applications of Resource Control Petri Nets for Automated Manufacturing Systems,* Ph.D. Thesis, Rensselaer Polytechnic Institute, Aug. 1992.

[54] K. Jensen, "Coloured Petri Nets," in *Advances in Petri Nets 1986*, Vol. 254, New York: Springer-Verlag, pp. 288-299, 1988.

[55] W. Jentsch and F. Kaden, "Automatic Generation of Assembly Sequences," in *Artificial Intelligence and Information-Control Systems of Robots,* I. Plander, Ed. Amsterdam: Elsevier Science Publishers, pp. 197-200, 1984.

[56] M. S. Kamel and P. M. Kaufmann, "Representing Uncertainty in Robot Task Planning," in *Proc. of IEEE International Conference on Robotics and Automation,* pp. 1728-1734, 1988.

[57] A. Kandel, "Fuzzy Techniques in Pattern Recognition," New York: John Wiley & Sons, 1982.

[58] B. H. Krogh, "Controlled Petri Nets and Maximally Permissive Feedback Logic," in *Proc. of 25 Allerton Conference on Communication, Control and Computing,* pp. 317-326, Oct. 1987.

[59] J. U. Korein and J. Ish-shalom, "Robotics," *IBM Systems Journal,* pp. 55-95, Vol. 26, No. 1, 1987.

[60] J.-C. Latombe, A. Lazanas, and S. Shekhar, "Robot Motion Planning with Uncertainty in Control and Sensing," *Artificial Intelligence,* Vol. 52, pp. 1-47, Nov. 1991.

[61] K. Lautenbach, "Liveness in Petri nets," St. Augustin, Gesellschaft fur Mathematik und Datenverarbeitung Bonn, Interner Bericht ISF-75-02.1, 1975.

[62] K. Lee and J. Favrel, "Hierarchical Reduction Method for Analysis and Decomposition of Petri Nets," *IEEE Trans. on Systems, Man, and Cybernetics*, Vol. 15, No. 2, pp. 272-281, 1985.

[63] C. C. Lee, "Fuzzy Logic in Control Systems: Fuzzy Logic Controller — Part I," *IEEE Trans. on Systems, Man, and Cybernetics,* Vol. 20, No. 2, pp. 404-418, 1990.

[64] C. C. Lee, "Fuzzy Logic in Control Systems: Fuzzy Logic Controller, Part II," *IEEE Trans. on Systems, Man, and Cybernetics,* Vol. 20, No. 2, pp. 419-435, 1990.

[65] K. S. Leung and W. Lam, "Fuzzy Concepts in Expert Systems," *IEEE Computer Magazine,* Vol. 21, No. 9, pp. 43-56, 1988.

[66] L. I. Lieberman and M. A. Wesley, "AUTOPASS: An Automatic Programming System for Computer Controlled Mechanical Assembly," *IBM Journal of Research and Development,* Vol. 21, No. 4, pp. 321-333, 1977.

[67] N. K. Liu and T. Dillon, "An Approach Towards the Verification of Expert Systems Using Numerical Petri Nets," *International Journal of Intelligent Systems,* Vol. 6, pp. 255-276, 1991.

[68] C. G. Looney, "Fuzzy Petri Nets for Rule-Based Decisionmaking," *IEEE Trans. on Systems, Man, and Cybernetics*, Vol. 18, No. 1, pp. 178-183, Jan./Feb. 1988.

[69] E. López-Mellado and R. Alami, "A Failure Recovery Scheme for Assembly Workcells," in *Proc. of IEEE International Conference on Robotics and Automation*, Cincinnati, OH, pp. 702-707, May 1990.

[70] T. Lozano-Pérez, "Task planning," in *Robot Motion: Planning and Control*, J. M. Brady *et al.*, Eds. Cambridge, MA: MIT Press, pp. 473-498, 1982.

[71] T. Lozano-Pérez, M. T. Mason, and R. H. Taylor, "Automatic Synthesis of Fine-Motion Strategies for Robots," *The International Journal of Robotics Research,* Vol. 3, No. 1, pp. 3-24, 1984.

[72] E. H. Mamdani and S. Assilian, "An Experiment in Linguistic Synthesis with a Fuzzy Logic Controller," *International Journal of Man-Machine Studies,* Vol.7, No. 1, pp. 1-13, 1975.

[73] M. Ajmone Marson, G. Balbo, and G. Conte, "A Class of Generalized Stochastic Petri Nets for the Performance Analysis of Multiprocessor Systems," *ACM Trans. on Computer Systems* 2(1), May, 1984.

[74] M. Ajmone Marson, G. Balbo, G. Chiola, and G. Conte, "Generalized Stochastic Petri Nets Revisted: Random Switches and Priorities," in *Proc. Int. Workshop on Petri Nets and Performance Models*, IEEE-CS Press, Madison, WI, USA, August, 1987.

[75] M. A. Marson, G. Balbo, and G. Conte, *Performance Models of Multiprocessor Systems,* Cambridge, MA: The MIT Press, 1987.

[76] D. McDermott, "Robot Planning," *AI Magazine*, Summer 1992, pp. 55-79, 1992.

[77] W. van Melle, "A Domain-Independent Production-Rule System for Consultation Programs," in *Proc. of International Joint Conference on Artificial Intelligence*, pp. 923-925, Tokyo, Japan, 1979.

[78] A. Merabet, "Synchronization of Operations in a Flexible Manufacturing Cell: The Petri Net Approach," *Journal of Manufacturing Systems,* Vol. 5, pp. 161-169, 1986.

[79] R. E. Miller, "A Comparison of Some Theoretical Models of Parallel Computations," *IEEE Trans. on Computers*, Vol. C-22, No. 8, pp. 710-717, Aug. 1973.

[80] R. Miller, *Some Relationships Between Various Models of Parallelism and Synchronization,* Report RC-5074, IBM T. J. Watson Research Center, Yorktown Heights, New York, Oct. 1974.

[81] S. Moriguchi and G. S. Shedler, "Diagnosis of Computer Systems By Stochastic Petri Nets," *Ieice Trans. on Fundamentals of Electronics Communications and Computer Sciences,* Vol. E75A, pp. 1369-1377, Oct. 1992.

[82] G. H. Morris and L. S. Haynes, "Robotic Assembly by Constraints," In *Proc. of IEEE International Conference on Robotics and Automation,* pp. 1507-1515, Raleigh, North Carolina, 1987.

[83] M. S. Mujtaba, R. A. Goldman, and T. Binford, "The AL Robot Programming Language," *Computer Engineering,* Vol. 2, pp. 77-86, 1982.

[84] T. Murata, "Petri Nets: Properties, Analysis and Applications," *Proc. of the IEEE,* Vol. 77, No. 4, pp. 541-580, Apr. 1989.

[85] Y. Narahari and N. Viswanadham, "A Petri Net Approach to the Modeling and Analysis of Flexible Manufacturing Systems," *Annals of Operations Research,* Vol. 3, pp. 449-472, 1985.

[86] C. V. Negoita, *Expert Systems and Fuzzy Systems*, Massachusetts: Benjamin/Cummings, 1985.

[87] N. J. Nilsson, *Principles of Artificial Intelligence,* Los Altos, California: Tioga Pub. Co., 1980.

[88] J. L. Peterson, "Petri Nets," *Computer Surveys,* Vol. 9, No. 3, pp. 223-252, Sept. 1977.

[89] J. L. Peterson, *Petri Net Theory and the Modeling of Systems,* Prentice Hall, 1981.

[90] C. A. Petri, *Kommunikation mit Automaten,* Ph.D. dissertation, University of Bonn, Bonn, West Germany, 1962, (in German).

[91] A. J. Pettofrezzo, *Introductory Numerical Analysis.* D. C. Heath and Company 1966.

[92] R. J. Popplestone, Y. Liu, and R. Weiss, "A Group Theoretic Approach to Assembly Planning," *AI Magazine* pp. 82-97, Spring 1990.

[93] C. V. Ramamoorthy and G. S. Ho, "Performance Evaluation of Asynchronous Concurrent Systems Using Petri Nets," *IEEE Trans. on Software Engineering,* Vol. SE-6, No. 5, pp. 440-449, 1980.

[94] W. Riddle, *The Modeling and Analysis of Supervisory Systems,* Ph.D. dissertation, Computer Science Department, Stanford University, Stanford, CA, March 1972.

[95] Hans Riesel, *Prime Numbers and Computer Methods for Factorization.* Birkhäuser Boston, Inc. 1985.

[96] E. Sacerdoti, *A Structure for Plans and Behavior.* New York: North-Holland, 1977.

[97] J. K. Salisbury, "Active Stiffness Control of a Manipulator in Cartesian Coordinates, M. T. Mason and J. K. Salisbury(eds.): *Robot Hands and the Mechanics Manipulation,* Cambridge, MA: MIT Press, pp. 95-108, 1985.

[98] A. C. Sanderson, "Parts Entropy Methods for Robotic Assembly," in *Proc. of IEEE International Conference on Robotics and Automation,* pp. 600-608, 1984.

[99] A. C. Sanderson, M. A. Peshkin and L. S. Homem de Mello, "Task Planning for Robotic Manipulation in Space Applications," *IEEE Trans. on Aerospace and Electronic Systems,* Vol. 24, No. 5, pp. 619-629, 1988.

[100] A. C. Sanderson, L. S. Homem de Mello, and H. Zhang, "Assembly Sequence Planning", *AI Magazine,* pp. 62-81, Spring 1990.

[101] A. C. Sanderson and T. Cao, *Petri Net Based Task Planning for Garment Handling System,* Technical Report, FIT/CMPTT, RPI, Dec. 1990.

[102] A. C. Sanderson and T. Cao, *Object-oriented Integration for Garment Handling System,* Technical Report, FIT/CMPTT, RPI, Dec. 1990.

[103] M. Stefik, "Planning with Constraints(MOLGEN: Part 1)," *Artificial Intelligence,* 16: 111-140, 1981.

[104] M. Stefik, "Planning and Meta-Planning(MOLGEN: Part 2)," *Artificial Intelligence,* 16: 141-170, 1981.

[105] R. H. Sturges, Jr. "A Three-dimensional Assembly Task Quantification with Application to Machine Dexterity," *The International Journal of Robotics Research,* Vol. 7, No. 4, pp 34-78, 1988.

[106] I. Suzuki and T. Murata, "A Method for Stepwise Refinement and Abstraction of Petri Nets," *Journal of Computer and System Sciences,* Vol. 27, pp.51-76, 1983.

[107] H. Tahani and J. M. Keller, "Information Fusion in Computer Vision Using the Fuzzy Integral," *IEEE Trans. on Systems, Man, and Cybernetics,* Vol. 20, No. 3, May/June, 1990.

[108] R. H. Taylor, P. D. Summers, and J. M. Meyer, "AML: A Manufacturing Language," *The International Journal of Robotics Research,* Vol. 1, No. 3, pp. 19-41, 1982.

[109] F. Thomas and C. Torras, "Constraint-Based Inference of Assembly Configurations," in *Proc. of IEEE International Conference on Robotics and Automation,* pp. 1304-1305, 1988.

[110] K. Tsuji and T. Matsumoto, "Extended Petri Net Models for Neural Networks and Fuzzy Inference Engines," in *Proc. of IEEE International Symposium on Circuits and Systems*, pp. 2670-2673, 1990.

[111] R. Valette, "Analysis of Petri Nets by Stepwise Refinements," *Journal of Computer and System Sciences,* Vol. 18, pp. 35-46, 1979.

[112] R. Valette, J. Cardoso and D. Dubois, "Monitoring Manufacturing Systems by Means of Petri Nets with Imprecise Markings," in *Proc. of IEEE International Symposium on Intelligent Control*, pp. 233-237,1989.

[113] R. Valette and M. Courvoisier, "Petri Nets and Artificial Intelligence," *International Workshop on Emerging Technologies for Factory Automation,* North Queensland, Australia, Aug. 17-19, 1992.

[114] S. Vere, *Planning in Time: Window and Durations for Activities and Goals,* Jet Propulsion Lab, Pasadena, CA, 1981.

[115] R. Vijaykumar and M. A. Arbib, "Problem Decomposition for Assembly Planning," in *Proc. of IEEE International Conference on Robotics and Automation,* 1987, pp. 1361-1366.

[116] N. Viswanadham and Y. Narahari, *Performance Modeling of Automated Manufacturing Systems.* Englewood Cliffs, NJ: Prentice Hall, 1992.

[117] D. E. Whitney, "Force Feedback Control of Manipulator Fine Motions," *Journal of Dynamic Systems, Measurement, and Control,* 98: 91-97, 1977.

[118] D. E. Wilkins, "Domain-independent Planning: Representation and Plan Generation," *Artificial Intelligence* 22(1984) 269-301.

[119] L. A. Zadeh, "Fuzzy Sets," *Information and Control,* Vol. 8, pp. 338-353, 1965.

[120] D. Zhang, "Planning with Pr/T Nets," in *Proc. of IEEE International Conference on Robotics and Automation,* pp. 769-775, Sacramento, CA, Apr. 1991.

[121] W. Zhang, "Representation of Assembly and Automatic Robot Planning by Petri Net," *IEEE Trans. on Systems, Man, and Cybernetics,* Vol. 19, No. 2, pp. 418-422, 1989.

[122] M. C. Zhou, F. DiCesare, and A. A. Desrochers, "A Top-down Approach to Systematic Synthesis of Petri Net Models for Manufacturing System," *Proc. of IEEE International Conference on Robotics and Automation,* pp. 534-539, Scottsdale, AZ, May 1989.

[123] M. C. Zhou and F. DiCesare, "Adaptive Design of Petri Net Controllers for Error Recovery in Automated Manufacturing Systems," *IEEE Trans. on Systems, Man, and Cybernetics,* Vol. 19, No. 5, pp. 963-973, Sep./Oct. 1989.

[124] M. C. Zhou and F. DiCesare, "Parallel and Sequential Mutual Exclusions for Petri Net Modeling of Manufacturing Systems with Shared Resources," *IEEE Trans. on Robotics and Automation,* Vol. 7, No. 4, pp. 515-527, Aug. 1991.

[125] M. Zhou and F. DiCesare, *Petri Net Synthesis for Discrete Event Control of Manufacturing Systems,* Boston, MA: Kluwer Academic Publishers, 1993.