A RESOURCE-ORIENTED PLANNER FOR INTEGRATED SHOP LEVEL
MATERIAL PROCESSING AND MATERIAL HANDLING OPERATIONS

A Thesis in
Industrial Engineering

by

Jay W. Steele

Copyright 2000 Jay W. Steele

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

August 2000
We approve the thesis of Jay W. Steele.

_________________________    ___________________
Richard A. Wysk
Professor of Industrial Engineering
Thesis Co-Advisor
Co-Chair of Committee

_________________________    ___________________
David J. Cannon
Associate Professor of Industrial Engineering
Thesis Co-Advisor
Co-Chair of Committee

_________________________    ___________________
Soundar R.T. Kumara
Professor of Industrial Engineering

_________________________    ___________________
Sanjay B. Joshi
Professor of Industrial Engineering

_________________________    ___________________
Rajeev Sharma
Assistant Professor of Computer Science and Engineering

_________________________    ___________________
A. Ravindran
Professor of Industrial Engineering
Head of the Harold and Inge Marcus Department of
Industrial and Manufacturing Engineering
ABSTRACT

This dissertation presents a formal definition of a new resource-oriented operations planning architecture and formal models for a set of shop floor activities that integrate material processing and material movement tasks for discrete part manufacturing. Manufacturing planning that incorporates manufacturing processing and material handling tolerance and kinematic constraints has not been reported in the existing literature. Such a system is created in this research. By integrating manufacturing tasks, this planning tool creates executable manufacturing plans. Changes in resources or the part design can be easily incorporated in the engineering process using the software tools developed for this work. The definition of this planner includes a formal specification of what the operations planner does and a description of the algorithms used to accomplish the operations planning.

The architecture, developed for this research, includes an input domain consisting of resource and part models that are object-oriented and designed to facilitate the integration between operations planning and other manufacturing activities such as product design, production cost estimation, scheduling, and production control. This integration is accomplished by defining the information requirements for the resource and part models to satisfy the planning of manufacturing activities. A formal resource model defines how attributes and behaviors of resources may change the part’s location on a shop floor or add new features to the part. New algorithms, which are incorporated in this object-oriented resource model, reason about the capabilities of resources to manufacture parts using tolerance stacking, transformations, standard machining rules, and robot trajectory planning. These algorithms also generate instruction sets for material processing and handling activities. The effects and preconditions of actions by resources on a single part are modeled to facilitate operations planning. Thus, the operations planner is driven by data defining the resources and parts so that it may easily handle changes in either. The operations planner uses these resource and part models as input to a problem-reducing search algorithm that automatically generates sequences of operations that result in the finished part. These sequences of operations are converted into an operations plan that can be directly applied to discrete event simulation, scheduling, manufacturability analysis, and production control.
A formal software specification defines the mapping from the input domain of resources and parts into an output domain consisting of an operations plan. This specification is used to construct the planning algorithms that generate a search graph containing alternative production plans. New algorithms are presented that convert this search graph into an AND/OR operations plan graph that is suitable for production simulation, cost estimation, and control. The manufacturing alternatives in this operations plan enable reallocation of resources, bottleneck avoidance, and job shop scheduling based on the physical capabilities of resources. Illustrations of manufacturing problems that are solved using this planner are presented. The solutions to these problems include planning the tolerance stackups and trajectories associated with robot tasks such as unloading and loading parts in machine tools from fixture setups that are automatically defined by the preconditions of machine tool tasks such as drilling new holes in parts. Other fixture setups are defined by material storage requirements on automatically guided vehicles and buffers. This research has been implemented in software and tested on problems that simulate physical manufacturing tasks that can be performed at the Pennsylvania State University’s Computer Integrated Manufacturing Laboratory using robots, machine tools, and automatically guided vehicles. These results validate the thesis that these resource and part models can be used to generate realistic operations plans using this planner.
# TABLE OF CONTENTS

LIST OF FIGURES .......................................................................................................................... viii
LIST OF TABLES ................................................................................................................................ ix
NOMENCLATURE ............................................................................................................................. x

Symbols ................................................................................................................................................ x
Defined Sets ......................................................................................................................................... x
Vector and Matrix Notation ................................................................................................................ Xi

ACKNOWLEDGEMENTS .................................................................................................................. xli

1. INTRODUCTION ............................................................................................................................. 1
   1.1 Background ................................................................................................................................. 1
   1.2 Research Objectives .................................................................................................................. 5
   1.3 Assumptions and Restrictions ................................................................................................... 6
   1.4 Dissertation Overview .............................................................................................................. 7

2. LITERATURE REVIEW .................................................................................................................... 8
   2.1 Manufacturing Operations planning ......................................................................................... 8
   2.2 Resource Modeling ................................................................................................................... 9
   2.3 Material Movement Planning ................................................................................................... 12
   2.4 Manufacturing Standards ....................................................................................................... 14
   2.5 Manufacturing Cost Estimation and Flexibility ..................................................................... 16

3. PROBLEM FRAMEWORK AND SOLUTION APPROACH .................................................................... 18
   3.1 Introduction ............................................................................................................................... 18
   3.2 Problem Framework .................................................................................................................. 18
      3.2.1 Introduction ....................................................................................................................... 18
      3.2.2 Planning Tasks .................................................................................................................... 19
      3.2.3 Resource and Part Modeling Requirements .................................................................... 19
   3.3 Solution Approach .................................................................................................................... 21
   3.4 Chapter Summary ...................................................................................................................... 24

4. FORMAL SPECIFICATION OF PROBLEM SOLUTION ...................................................................... 25
   4.1 Introduction ............................................................................................................................... 25
   4.2 Part Model Definition ............................................................................................................... 27
   4.3 Resource Model Definitions ..................................................................................................... 29
      4.3.1 Facility Model ...................................................................................................................... 29
      4.3.2 Tool Model .......................................................................................................................... 31
      4.3.3 Fixture Model ...................................................................................................................... 32
      4.3.4 Material Processor Model ................................................................................................. 34
      4.3.5 Material Handler Model .................................................................................................... 39
      4.3.6 Material Transporter Model .............................................................................................. 48
      4.3.7 Buffer Storage Model ....................................................................................................... 49
   4.4 Operations Plan Model and Criteria ......................................................................................... 49
      4.5 Illustration ............................................................................................................................... 49
      4.5.1 Resource Model .................................................................................................................. 53
      4.5.2 Unload and Load Operations ............................................................................................. 55
      4.5.3 Material Processing Operation ......................................................................................... 63
   4.6 Chapter Summary ...................................................................................................................... 66
LIST OF FIGURES

FIGURE 3.1 IDEAL INFORMATION FLOW FOR FLEXIBLE MANUFACTURING WITH RESOURCE MODEL .................................................................21
FIGURE 3.2 PROCESS PLANNING DATA FLOW ..........................................................................................................................22
FIGURE 4.1. FIXTURE LOCATORS, POINTS, AND VECTORS .................................................................................................33
FIGURE 4.2. TOLERANCES FOR CREATING FEATURE ALONG ONE EQUIPMENT AXIS .................................................................36
FIGURE 4.3A. DATUMS FOR MP TASK. FIGURE 4.3B. TOLERANCE LOOP FOR MP TASK ..............................................................................37
FIGURE 4.4. TOLERANCES ASSOCIATED WITH MATERIAL HANDLING UNLOAD OPERATION ALONG ONE FACILITY AXIS .................................................41
FIGURE 4.5. TOLERANCES ASSOCIATED WITH MATERIAL HANDLING LOAD OPERATION ALONG ONE FACILITY AXIS ...............................42
FIGURE 4.6A. DATUMS FOR MH UNLOAD TASK. FIGURE 4.6B. TOLERANCE LOOP FOR MH UNLOAD TASK .................................................43
FIGURE 4.7A. DATUMS FOR MH LOAD TASK. FIGURE 4.7B. TOLERANCE LOOP FOR MH LOAD TASK .................................................................46
FIGURE 4.8. EXAMPLE FACILITY LAYOUT ........................................................................................................................................55
FIGURE 4.9. EXAMPLE PART ...............................................................................................................................................................55
FIGURE 4.10. TOLERANCES ASSOCIATED WITH E1, FIXT1, AND PART1 .............................................................................................57
FIGURE 4.11. TOLERANCES ASSOCIATED WITH E2, FIXT2, AND PART2 FOR OPEN GRIPPER FIXTURE ..................................................58
FIGURE 4.12. TOLERANCE FUNCTIONS ASSOCIATED WITH MAKING FEATURE WITH E3 ALONG XE3 ....................................................63
FIGURE 5.1 ILLUSTRATION OF OR SEARCH GRAPH GENERATION ..................................................................................................69
FIGURE 5.2 ILLUSTRATION OF SEARCH GRAPH GENERATION FOR DRILL OPERATION .................................................................73
FIGURE 5.3 ILLUSTRATION OF SEARCH GRAPH GENERATION FOR MH UNLOAD FROM HOME PORT OPERATOR .................................74
FIGURE 5.4 ILLUSTRATION OF SEARCH GRAPH GENERATION FOR MH LOAD TO AGV PORT .................................................................75
FIGURE 5.5 ILLUSTRATION OF SEARCH GRAPH GENERATION FOR MT MOVE OPERATOR ............................................................................75
FIGURE 5.6 ILLUSTRATION OF OR SEARCH GRAPH WITHOUT ALTERNATIVES ..................................................................................76
FIGURE 5.7 ILLUSTRATION OF SEARCH GRAPH GENERATION FOR RE-FIXTURING OF PART IN A MACHINE ..............................77
FIGURE 5.8 SEARCH GRAPH GENERATION WITH MATERIAL PROCESSING ALTERNATIVE .............................................................79
FIGURE 5.9 ILLUSTRATION OF SEARCH GRAPH GENERATION WITH MATERIAL HANDLING ALTERNATIVE ...........................................80
FIGURE 5.10. EQUIPMENT COORDINATE FRAMES FOR E1 AND E2 ....................................................................................................83
FIGURE 5.11. OR GRAPH TRACE ALGORITHM FLOW CHART .............................................................................................................94
FIGURE 5.12. OPERATIONS PLAN DERIVED FROM SEARCH GRAPH IN FIGURE 5.6 ..............................................................................94
FIGURE 5.13. OPERATIONS PLAN DERIVED FROM SEARCH GRAPH IN FIGURE 5.9 ..............................................................................94
FIGURE 5.14. OPERATIONS PLAN WITHOUT DUPLICATION ...............................................................................................................95
FIGURE 5.15. SEARCH GRAPH WITH AND SEQUENCE OF FEATURES .............................................................................................96
FIGURE 5.16. OPERATIONS PLAN GENERATED FROM SEARCH GRAPH IN FIGURE 5.14 .....................................................................97
FIGURE 5.17. OPERATIONS PLAN AFTER REMOVAL OF DUPLICATE NODES .....................................................................................97
FIGURE 5.18. OPERATIONS PLAN AFTER ADDITION OF AND NODES ...............................................................................................97
FIGURE 7.1 THE RESOURCE MODEL CLASSES USED FOR PROBLEM ILLUSTRATION (UML FORMAT) ..................................................119
FIGURE 7.2 RESOURCE MODEL TOOL CLASSES (UML FORMAT) ......................................................................................................121
FIGURE 7.3 RESOURCE MODEL ROBOT CLASSES (UML FORMAT) ...................................................................................................122
FIGURE 7.4. PART MODEL CLASSES (UML FORMAT) .......................................................................................................................124
FIGURE 7.5 RDBMS RELATIONSHIP DIAGRAM OF RESOURCE MODEL DATABASE ..............................................................................127
FIGURE 7.6 RDBMS RELATIONSHIP DIAGRAM OF OPERATIONS PLAN DATABASE ............................................................................129
FIGURE 7.7 RESULTS FROM RESOURCE MODEL WITH ¼ DRILL AT MACHINE VF-5 AND PORT #6 .........................................................133
FIGURE 7.8 RESULTS FROM RESOURCE MODEL WITH ¼ DRILL, 3/16 DRILL, ¼ REAMER AT MACHINE VF-5 .............................................134
FIGURE 7.9 RESULTS FROM RESOURCE MODEL WITH AUXILIARY BUFFER AT PORT #4 .................................................................135
FIGURE 7.10 RESULTS FROM RESOURCE MODEL WITH ¼ DRILL AT MACHINE VF-0E AND PORT #8 ..........................................................136
FIGURE 7.11 RESULTS FROM PART MODEL WITH GOAL FEATURES GRAPH (*7 8*) ............................................................................137
FIGURE 7.12 RESULTS FROM PART MODEL WITH GOAL FEATURES GRAPH (*7 8 9*) ............................................................................138
FIGURE 7.13 RESULTS FROM PART MODEL WITH GOAL FEATURES GRAPH (#7 8#) ............................................................................138
FIGURE 7.14 RESULTS FROM PART MODEL WITH GOAL FEATURES GRAPH (#7 9#) ............................................................................139
FIGURE 7.15 RESULTS FROM PART MODEL WITH GOAL FEATURES GRAPH (#7 8 &) ............................................................................139
FIGURE A.1. AND/OR OPERATIONS PLAN GRAPH FOR EXAMPLE PART.
LIST OF TABLES

Table 4.1. Operations and Their Effects on the Part State Space ......................................................... 52
Table 4.2. Formal Specification of Resource Model for Illustration ....................................................... 54
Table A.1. Operations for Part1 ................................................................................................................. 152
Table A.2. Sequence #1 ............................................................................................................................ 161
Table A.3. Sequence #2 ............................................................................................................................ 161
Table A.4. Sequence #3 ............................................................................................................................ 162
Table A.5. Sequence #4 ............................................................................................................................ 162
NOMENCLATURE

The following is a partial list of the nomenclature used in this dissertation.

Symbols

∞    Infinity
*    Multiply
●    Vector Dot Product
⊗    Vector Cross Product
∅    Empty set
∀    For all, or the universal quantifier
∃    There exists, or the existential quantifier
∪    Union
∩    Intersection
{a, b, c, . . .}    Set of all individual elements a, b, c, . . .
|    Such that
∈    Element of
⇒    Only if, or material implication
⇔    If and only if, or material equivalence
∧    AND, or conjunction
∨    OR inclusive, or disjunction

Defined Sets

BS    Set of buffer storage equipment
E    Set of equipment available to facility
F    Set of fixtures available to facility
Feat    Set of features manufacturable by facility
FlatSurfaces    Set of flat surface features
Holes    Set of hole features
HorizMills    Set of horizontal mills in facility
Lathes    Set of lathes in facility
MP    Set of material processor equipment
MT    Set of material transportation equipment
MH    Set of material handling equipment
Materials    Set of materials machinable by facility
Parts    Set of parts manufacturable by facility
Ports    Set of ports in facility
Rmh    Set of robotic material handler equipment
Rmt    Set of robotic material transporter equipment
T    Set of tools available to facility
Tbore    Set of boring tools available to facility
Tdrill    Set of twist drill tools available to facility
Treamer    Set of reamer tools available to facility
VertMills    Set of vertical mills in facility
Wmh    Set of human material handlers
Wmt    Set of human material transporters
Wp    Set of processing workstations in facility
Wt    Set of transportation workstations in facility
Ws    Set of storage workstations in facility
# Vector and Matrix Notation

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X$</td>
<td>Unit vector in X direction (1, 0, 0)</td>
</tr>
<tr>
<td>$Y$</td>
<td>Unit vector in Y direction (0, 1, 0)</td>
</tr>
<tr>
<td>$Z$</td>
<td>Unit vector in Z direction (0, 0, 1)</td>
</tr>
<tr>
<td>$^A P$</td>
<td>Point described in coordinate frame A</td>
</tr>
<tr>
<td>$^A X_B, ^A Y_B, ^A Z_B$</td>
<td>Unit vectors giving the principal directions of coordinate frame B described in coordinate frame A</td>
</tr>
<tr>
<td>$^A P_BO$</td>
<td>B's origin described in coordinate frame A</td>
</tr>
<tr>
<td>$^A V$</td>
<td>Vector described in coordinate frame A</td>
</tr>
<tr>
<td>$^A V_B$</td>
<td>Unit Vector described in coordinate frame A</td>
</tr>
<tr>
<td>$^B T$</td>
<td>Homogeneous Transform describing frame B relative to frame A such that $^A P = ^B T * ^B P$ and $^B T = ^A T * ^C T * ^C T$</td>
</tr>
<tr>
<td>$^B T(i,j)$</td>
<td>$(i,j)^{th}$ element of the matrix representing the homogeneous transform, $^B T$</td>
</tr>
<tr>
<td>$[^B T]^{-1}$</td>
<td>Inverse of Matrix such that $[^B T]^{-1} = ^A T$</td>
</tr>
</tbody>
</table>
ACKNOWLEDGEMENTS

First and foremost I would like to thank my sweet wife, Ilze, for all of her patience and support during my pursuit of this degree. Without her love and ability to keep me focused on the important things in life and her great editorial support, none of this would be possible. I would also like to thank my parents, Charles and Marie-Louise, and brothers Eric, Brett, and Ryan for their support.

I would also like to thank my two co-advisors, Dr. Richard Wysk and Dr. David Cannon, for their guidance and mentoring during my graduate studies at Penn State University. I appreciated both Dr. Wysk’s and Dr. Cannon’s attitude that great ideas must be tested with physical implementation. Dr. Cannon has supported my research activities and has always encouraged me to think outside of the box. Dr. Wysk has been a great mentor and friend and his lessons have definitely prepared me for a career in manufacturing research. The hours we spent discussing manufacturing automation, resource models, and process planning were definitely the highlight of my education at Penn State.

I would like to thank Dr. Soundar Kumara and Dr. Sanjay Joshi for their assistance with this research and my education at Penn State. I really enjoyed learning from Dr. Kumara’s philosophical approach to artificial intelligence and industrial engineering. Thanks also go to Dr. Rajeev Sharma from the Computer Engineering and Science department, who has been helpful in providing feedback for my robotics trajectory planning research. Lastly, I would like to thank Dr. João Carlos Ferreira from the Federal University of Santa Catarina, Brazil for his help with improving my software implementation and for extending my operations planning research in the material processing domain.
Chapter 1

INTRODUCTION

Developing an architecture for manufacturing begins with an assessment of the basic tasks that the factory must perform. [Hayes et al, 1988].

1.1 Background

Currently, multi-purpose computer numerical control machining centers (CNCs), motorized robot arms, flexible fixturing, and automatic guided vehicles (AGVs) provide the physical capability to perform multiple automated manufacturing tasks on different parts using the same machine. Thus, heterogeneous parts have the potential to be automatically manufactured in the same factory. Typically, the competitiveness of a manufacturing firm operating such a flexible manufacturing facility is strongly affected by how quickly its machine resources can be programmed to manufacture a new part. To assist engineers who program these machines for new tasks to manufacture new parts, geometric simulation software has been developed that generates machine instructions from simulated machine actions specified by the engineer. Specifically, computer aided manufacturing (CAM) software can be used to generate CNC programs for process machines, and robot kinematics simulation software can be used to generate robot motion programs. These tools have significantly decreased the time needed to develop machine instructions from a new product design specification. Presently, the engineer’s time and ability to specify appropriate machine actions using these tools limit the physical flexibility of the production machines to produce a part. In order to decrease this limitation and improve the programming process further, much research has been directed towards automatically generating processing machine instructions from a geometric part specification [Wang & Wysk, 1987; Kiritsis, 1995]. However, before executing these processing machine instructions, the part must be moved to the appropriate machine and properly fixtured. In this dissertation, these tasks of moving the part between machines are grouped under the term of material movement. There has been significant effort directed towards the automatic generation of robot task level plans for material movement [Latombe, 1995]. Dekleva and Gaberc (1994) suggest planning manufacturing activities using two phases that separately plan out material processing and material movement operations. However, these activities are directly...
dependent on each other because material movement limits the capability of the material processing, and material movement is only necessary because of material processing. In this work, planning of material processing and material movement is directly integrated using a new planning approach.

In the past, flexible manufacturing has consisted of automated machining centers that are manually tended by operators. In this scenario, material handling and transportation tasks are performed by humans who are capable of independently planning out their actions to move parts and properly install parts in fixtures for machining. Thus, the focus of manufacturing planning was the planning of machining processes from geometric part specifications. However, given the physical capability of machines to transport parts and place parts in fixtures, the opportunity now exists to completely automate flexible manufacturing. One crucial step to achieve this is to develop a planner that not only plans out machining processes based on part geometry, but also plans out material movement actions based on part location and fixturing requirements resulting from these machining processes.

For planning the automated movement of the new part to material processing machines, the engineer can only utilize the physical flexibility of existing robots, AGVs, and modular fixtures [Lewis, 1983] with great effort by separately programming tasks for each individual machine. Borrowing from definitions from Wysk and colleagues (1995), these material movement tasks are divided into material handling actions that move the part from one fixture to another and material transportation actions that move the part using a fixture. For instance, a material handling robot arm may take the part from a fixture on an AGV and place it into a fixture in a machining center. A material transporting AGV moves the part in a fixture between different locations in a factory. The engineer must manually plan how to link these individual machine actions to move the product throughout the factory. Naturally, coordinating these tasks with the normal machining process planning requires time and delays the production of a new part.

These delays in the production of a new product also create difficulties for the part design process because every iteration of the part design requires a new analysis of the production constraints. According to the principles of concurrent engineering, the design of a new part must be influenced by its manufacturability [Parsaei & Sullivan, 1993]. In many industries, a low production volume to product variety ratio mandates that production
facilities be flexible enough to produce a variety of new products. Furthermore, the new part must be manufacturable below a given cost threshold by a given set of facility resources in order to ensure profitability for the manufacturing firm. One way to determine this cost is to use software to simulate the production of the part in the factory. Because modeling an entire factory with multiple entities is highly complex, discrete event models of production have been developed to reduce the geometric complexities of actions such as drilling a hole or grasping a part down to the amount of time and resources each discrete event requires [Harmonosky & Barrick, 1988]. Thus, this discrete event simulation of a manufacturing facility is based on a limited view of the factory’s resources. While this discrete event simulation model is quite useful for estimating total production cost for parts with existing machine programs and known process times, new machine programs with estimated execution times have to be generated for new part designs under consideration before simulating their production.

These discrete event simulations have also been used to control the actions of, and interactions between physical manufacturing machines at a high level. Thus, the simulation model that was used to evaluate the facility can be reused to control the facility. Such shop floor control systems, described by Smith and Peters (1998), rely on discrete representations of material handling and transportation actions. Wysk and colleagues (1995) define a formal representation schema for this control system to unambiguously describe discrete manufacturing operations and resources for automated manufacturing. However, since this representation is at the discrete event abstraction level, there is ambiguity at the more detailed kinematic level. For instance, a discrete event manufacturing task for a material handler resource is to pick a part from a specific buffer. This discrete event description is ambiguous at the more detailed kinematic level because there may be many different ways for the material handler to pick the part from the buffer.

The advantage of discrete event descriptions of manufacturing plans is that they can be represented in a compact way for resource allocation, scheduling, and execution in addition to discrete event simulation for design analysis. These manufacturing plans with alternatives are easily represented by AND/OR graphs [Cao & Sanderson, 1991]. Smith (1992) developed a useful operations plan description based on an AND/OR graph with nodes representing material handling or processing operations on the part. Smith and colleagues (1993) divided the production function into planning, scheduling, and execution. From the schema developed by Wysk and
colleagues (1995), the planning function ‘de-ORs’ the AND/OR graph for the current set of orders for the facility. Since each OR node represents the use of different resources for a process step, the production planning function allocates resources by de-ORing the graph. Thus, alternate plans enable an automated production system to become more reliable because parts can be routed away from disabled machines to functional machines [Majchrzak, 1988]. According to Mettala (1989), nodes connected with an AND branch indicate that any order of execution of these nodes is permitted. Thus, the production scheduler function ‘de-ANDs’ the graph by determining the order of execution of nodes. Finally, the production execution function causes the execution of the linearized graph in the order determined by the production scheduler.

Thus, before simulating or controlling the production of a new product design, engineers must generate precise plans for manufacturing equipment in the facility that provide the necessary information for these discrete task nodes. This information specifies the resources that are used by the node, the instructions for these resources, and how long the task takes. The tools available to generate these plans are sometimes incomplete due to the lack of models for the constraints of manufacturing resources. An example of such tools in the material handling domain is the robotic agile manufacturing system developed by Merat and colleagues (1997). This system may be reprogrammed for new parts using kinematic simulation software. In the general case, however, engineers must implicitly consider other constraints of the facility resources to handle a new part which are not typically modeled with kinematic simulation software. Kinematic simulation may be used to define a valid robot arm’s path required to move a part from an AGV pallet to a machining fixture, but the engineer must specify how the robot grasps the part. Furthermore, the engineer must determine if the new part is light and small enough to be transported by the AGV and if fixturing can hold the part. These types of constraints on the material transportation of new parts are ignored by AGV modeling literature that assumes that existing AGVs are always capable of transporting new parts [Wilhelm and Evans, 1987]. In the AGV literature, modeling of AGVs deals with such issues as how many AGV units are required, scheduling rules for the control system, and optimal locations of paths [Egbelu, 1992]. Since material movement has been shown to add significantly to the manufacturing cost of products [Mullens & Swart, 1992], either excessive time will be spent studying the constraints for each different iteration of new part design or critical material movement issues will be ignored during the design process. For example, Sands and colleagues (1997) developed an automatic discrete event simulation generator to evaluate new part designs for
manufacturability from a Standard for Exchange of Product Data (STEP) [ISO, 1995] specification and ignored important details of material movement such as geometric constraints.

Automatic generation of manufacturing operation plans that considers the constraints of the part design and manufacturing resources enables a quick determination of critical manufacturing costs for each part design iteration and enables manufacturing resources to handle new parts quickly. Thus, an important research issue is how to design a general software architecture that facilitates the development of software that generates manufacturing operation plans for flexible and automated manufacturing. Another research issue is how this architecture can facilitate the use of these generated plans by discrete event simulators to estimate the manufacturability of a new product design. Production controllers should be also able to use these plans to produce new parts in a facility quickly. Lastly, this operations planning architecture should facilitate the integration of the manufacturing enterprise through the integration of engineering functions beyond the product design and production control functions. This integration is accomplished by formally specifying the elements and functions of this architecture in order to minimize misunderstanding across the enterprise.

1.2 Research Objectives

The objective of this research is to develop a formal specification of a manufacturing operations planner and test it out through implementation. This operations planner generates manufacturing plans for automated production of a part in a given facility, modeled by a resource model, from a part specification. The generated manufacturing plans will contain sufficient information to simulate and control the operations of the production. This planner will generate the manufacturing plans for one instance of a given part in the form of an AND/OR operations plan graph. This operations plan may drive either a simulation or an execution of the plan through the use of information driven shop floor control architectures [Smith, 1992]. This planner uses rules that reflect the capabilities of resources in a manufacturing resource model. This resource model creates a framework for integrating the engineering functions such as part design, production control, scheduling, and simulation. The resource modeling methodology defines the capabilities of manufacturing resources with sufficient fidelity to
enable precise geometric planning constrained by factors such as feature tolerances, robot accuracy, machine tool accuracy, and fixture location tolerances. The material processing planning incorporates rules from standard machining practices while the material movement task planning uses task level path planning from sources in the literature. The operations planner integrates these different types of planners with a part state space description that includes part location and a geometric model.

### 1.3 Assumptions and Restrictions

Manufacturing resources in the resource model are limited to 3-axis milling machines, articulated robot arms, AGVs, and fixturing. The manufacturing planner presented in this work ignores the constraints imposed by the handling and transportation of fixtures and tools. If fixture and tool software objects are owned by equipment software objects, then it is assumed that the actual fixtures and tools are physically available to the equipment, and an operations plan may specify the usage of the fixtures and tools by the equipment. Furthermore, holding requirements of fixtures under the dynamic stresses of robot motion or machining processes are ignored. It is assumed that if a fixture is defined to locate a specific part in a machine, then the fixture will continue locating the part under any stresses that may occur at that machine.

The planner does not incorporate constraints imposed by multiple part interaction. These constraints include bottlenecks due to insufficient buffer space and robot collisions due to intersecting workspaces. Furthermore, it is assumed that the only obstacles in each robot’s workspace are equipment in the resource model. It is also assumed that these obstacles have fixed geometric models without uncertainty in boundaries or locations. These equipment entities have known geometric models with fixed known positions in the facility. Planning of assembly tasks is not included in this research.

For practical implementation of this research to generate valid operations plans for flexible manufacturing facilities, it is assumed that data in the resource model that describes the resources of the facility can be kept valid
with respect to the physical resources. Furthermore, it is assumed that all tolerances modeling inaccuracies in feature location, facility layout, and process capability can be modeled as bilateral (+/-) tolerances.

1.4 Dissertation Overview

The remainder of the document is organized as follows. Chapter 2, Literature Review, discusses efforts in the literature to address the issues raised in this work. Chapter 3, Problem Framework and Solution Approach, discusses the problem definition and provides the solution framework for this thesis. Chapter 4, Formal Specification of Problem Solution, presents a formal specification for the planner that defines the input domain and conditions for the validity of the output. Chapter 5, Operations Plan Generation Methodology, describes the methodology for generating the operations plan at a discrete level and at the geometric computation level. Chapter 6, Material Handling Task Planning Methodology, describes the methodology used to generate material handling tasks using obstacle avoiding trajectory generation. Chapter 7, Implementation Architecture, describes the implementation architecture for the planner and shows the results of the software implementation. This architecture includes the object-oriented information model of the manufacturing resources and the architecture of the storage of the input and output data for the planner. The results include different operation plans that are generated based on different resource and part models. Chapter 8, Research Contribution and Conclusions, presents the contributions of the research and possible extensions to this work. Appendix A, Example Verification of Operations Plan, illustrates and validates a solution to the formal specification of an example manufacturing task presented in Chapter 4. Appendix B, Illustration of Part File Specification, presents an example part specification that matches the part model file specification given in Chapter 7. Appendix C, List of Predicates and Functions, lists all of the predicates and functions used in this dissertation.
Chapter 2

LITERATURE REVIEW

2.1 Manufacturing Operations planning

Efforts to break down the infamous wall between product design and manufacturing [Tuttle, 1983] have resulted in a large body of work concentrating on computer aided process planning (CAPP) from product design specifications. This body of work has mainly emphasized material processing and ignored material movement considerations. For discrete machined parts, a common technique has been to develop expert systems that map part features to machining operations. Examples of such systems include the GARI system developed by Descotte and Latombe (1981) and the Turbo-CAPP system developed by Wang and Wysk (1988). Park and Khoshnevis (1991) built on these process-planning efforts to develop a system that more actively integrates the design process with automatic operations planning. Recognition of the need to standardize the product information model for the connection between diverse computer aided design (CAD) software and CAPP software resulted in the STEP standard, described by Mitchell (1996). These process-planning systems naturally have some model of the processes that can be executed by resources in a given manufacturing facility. Examples in the CAPP literature include the following: Wang and Wysk (1988) develop a Machine Description Module, Subramanyam and Lu (1989) use a manufacturing facility model, and Sands and colleagues (1997) define simulation templates to model resources. Other process planning research for discrete part manufacturing has focused on feature extraction from CAD product models [Joshi & Chang, 1988] and encoding standard machining practice rules into expert systems [Kiritsis, 1995].

These process planning rules map the features corresponding to generalized manufacturing operations to the usage of specific shop resources and manufacturing parameters such as cutting conditions. However, these resource models are all abstracted away from the manufacturing floor. For instance, the Manufacturing Simulation Driver (MSD) system [Sands et al, 1997] uses product models to generate operations plans that can be executed in simulation while actual production is controlled by clearly different plans. The term Design for Manufacturing (DFM) operations plan denotes the plans that can be executed by a simulation of the
manufacturing process for DFM while production operations plan denotes plans for actual production. Such DFM operations plans use temporal precedence to describe a sequence of actions by resources. While a sequence of actions described by temporal precedence integrates nicely with production scheduling, it is hard to execute this sequence to control shop floor production. This is largely due to unavoidable variance in actual process times on the shop floor. Thus, while these generated DFM operations plans are useful for quick design feedback, using these plans for production results in extra labor to convert the CAPP plans into shop floor plans and possibly inaccurate design-for-actual-manufacturing feedback to the product designer. Gu and Zhang (1994) create actual machining plans from an object-oriented part design and resource model, but these plans do not include actions such as material handling and storage and are not sufficient to control the interaction between resources that is necessary for actual production.

2.2 Resource Modeling

Attempts to integrate product design and operations planning with more emphasis on material movement have also resulted in resource modeling that is abstracted away from manufacturing realities. For instance, in Chen (1995), a material movement planner evaluates the manufacturability of a part based on the movement requirements for a part. However, this evaluation is based on discrete event type modeling that ignores issues such as, will the part fit on a fixture on an AGV, and is it light enough for the AGV? Instead, existing AGV modeling research emphasizes issues such as optimal unit load, dispatching rules, and routing rules. Egbelu (1992) notes that the shape of the part and its weight are factors that affect the performance of the material movement system but does not deal with these issues. Such issues are raised in automatic flexible fixturing planning for a given part [Hong et al., 1996] but have not yet been incorporated in material movement research.

Efforts to correct this lack of connection between CAPP plans and actual production have resulted in the recognition that CAPP plans must contain sufficient information to drive the production function to correctly allocate, schedule, and control actual shop floor resources. The proposed Process Specification Language (PSL) standard, currently being developed by the National Institute of Standards and Technology (NIST) and described
by Schlenoff and colleagues (1996), partially reflects this recognition by providing a *lingua franca* to represent the results from the operations planning function in addition to the output from production planners and schedulers. Thus, PSL will represent resources required for each process step for production planning, temporal precedence action sequences for scheduling, and the state transitions for each process step for shop floor control. Along the same lines, the ideal production system must be designed in such a way that it may be driven by a production operations plan for a given product without reprogramming.

In order to develop such reusable production systems, Smith and colleagues (1993) partition the production function into a production planner that allocates resources, a production scheduler that determines the priority of parts, and a production execution system that executes the generated plans and schedules. Smith (1992) developed a formal methodology, called Message based Part State Graph (MPSG), that is based on finite state AND/OR task graphs of part and equipment states. This MPSG methodology generates a production control system (PCS) that is driven by the output of a production planner and scheduler. This PCS is independent of the operations planning function, but it is implicitly integrated with a common resource model. With MPSG, the production operations plan may change without reprogramming the PCS as long as both functions share a common set of resources. Cho and colleagues (1994) began integrating CAPP plans and a production operations plan by mapping features from a CAD specification of a part to work-cell and equipment task graphs for manufacturing entities in a facility. In their work, however, the task graphs, capabilities, and instantiation of the entities in the facility are encoded in the operations planning knowledge base because the system lacked a general resource model. Thus, the operations planning and production functions are not independent in their work. Wysk and colleagues (1995) made further progress in this area by specifying the requirements of a resource model to facilitate production operations planning for material processing as well as handling and transportation. Still, this proposed PCS-oriented resource model only contains sufficient information to execute discrete production actions for resources within a facility, and it ignores modeling of the cost, status, and capabilities of these resources for operations planning, production planning, and production scheduling.

Efforts to remedy this lack of a general manufacturing resource model have resulted in models that do not completely meet the goals of defining what a resource model is and how it integrates the modular engineering
functions. For instance, Jurrens and colleagues (1995) developed standardized object classes with attributes to model machining resources but do not define the engineering functions that may use the model and ignore material movement resources. In another approach, Whiteside and colleagues (1997) attempt to integrate the manufacturing environment with the Common Object Request Broker (CORBA) but place an emphasis on the communication language between objects in different engineering functions without establishing the information requirements for general engineering integration. Similarly, Fox and Grüniger (1997) develop techniques for precisely defining manufacturing enterprise elements with ontologies but do not specify the information requirements for these elements. From an enterprise perspective, software such as SAP has been successful in integrating the finances, human resources, and logistics of manufacturing enterprises using well defined data classes but lack classification of manufacturing capabilities on the shop floor [Hernandez, 1997].

As previously reported by the author [Steele et al, 2000], this new resource modeling methodology for flexible manufacturing integrates the design, operations planning, and production control engineering functions of a manufacturing enterprise while simultaneously keeping these functions modular through the use of common object-oriented resource classes. This approach of using modular resource models of manufacturing machines is inspired by the model of reusable software/hardware components for manufacturing that was originally proposed by Naylor and Volz (1987). Integration of the engineering functions is desirable because it follows the well known principles of concurrent engineering [Miller, 1993]. In this approach, the engineering functions are mutually independent and independent of specific instances of the resource model because their decision-making rules are embedded in different methods in shared resource classes. The object data that instances these classes for a specific facility is located in a single resource model that ideally is stored in a database. By applying this methodology to manufacturing problems, the constraints or process capabilities of actual manufacturing entities that affect production may be modeled in the resource model. Computer-aided material processing and movement planners may be developed that embed reasoning about these resources in class methods to generate operations plans for manufacturing for design feedback or production control. In addition, in order to facilitate the decision making by the production control functions, these resource classes must also specify the status, activity-based cost of use, and position in the facility of the resources. Such information is missing from common models for manufacturing resources. While issues of production planning and scheduling for material movement integrated
with material processing are typically ignored in the literature, Smith and colleagues (1999) demonstrate that these issues need to be considered for optimal performance.

### 2.3 Material Movement Planning

Software planners can generate machine instructions that lead to a desired goal state if it is possible to encapsulate the ability to predict the consequences of a proposed action by the machine [Nilsson, 1998]. Furthermore, it must also be possible to predict what the precondition of this action is. These abilities can be used to search for a sequence of actions that transform the initial state of the part to its goal state in a model of the system, where the part state represents the position and material state of the part. The resource models of specific machining centers can be used to represent the machines’ capabilities to create features on parts. Based on these capabilities, manufacturing operations can be planned out. Preconditions associated with these operations include the part’s location at specific machines and how it must be placed in fixtures. These preconditions lead to new goals for the planner that can be satisfied by material movement sub-plans. A sub-plan that describes the movement of a part by multiple robots and AGVs in a facility consists of a sequence of actions taken by these machines to move the part. To generate this sub-plan, a planner must have the ability to predict if a specific machine in the facility is capable of moving the part from its current state to a limited number of different locations. The resource model of each specific machine can be used to encapsulate the machine’s capability to move a given part. For instance, material handlers may be characterized by their abilities to grasp and lift parts while material transporters are characterized by their abilities to hold and move parts. Given these models, it becomes theoretically possible to search automatically through all of the resource models of possible machines in the facility to discover which machines are capable of moving the part from its initial state to another state. Next, the planner can search from this other state to move the part to another state. This process continues until the goal state is reached.

Robot task level planning provides this predictive ability required for automatic material handling planning. Currently, robot simulation software uses precise models of the behavior of individual material handling
machines. However, a major limitation of such software for automatic material handling planning is that it models kinematic behavior rather than the capability to move parts. Typically, the kinematic behavior of a robot arm is modeled so that the arm can be moved in simulation under direct human control to move simulated parts and generate the machine instructions to do the actual motions [Paul & Rosa, 1986]. Virtual tools for specifying these robot actions have been developed for telerobotic control in unstructured environments by Cannon and Thomas (1997) and extended by Small and McDonald (1997). In contrast, task level planning generates instructions for robots to carry out a task that is specified independently of the robot’s kinematics and grasping ability [Lozano-Perez, 1983]. For example, a task level specification gives the command to pick up a part and place it in another location whereas a robot level specification commands a specific robot to move its end effector to a new location. This task level specification is integrated with task level planning by Wang and colleagues (1997). These contributions lead to automatic operations planning if it is possible to extend the concept of virtual tools to virtual objects. The virtual object represents a part that must be moved around the shop floor based on material processing requirements.

Automatic planning to satisfy a task level specification for robots is difficult because the physical flexibility of robots results in multiple degrees of freedom, which generate large search spaces for collision-free paths [Lozano-Perez, 1987]. Relatively simple tasks, such as welding a seam, that require computation of the Cartesian straight line motion of the end effector have been automated in commercial software such as IGRIP (www.deneb.com) or CIMStation (www.silma.com). However, it is difficult to generate automatically a stable grasping strategy for the robot to pick up a given part [Allen, 1990]. Thus, the capability of the arm to grasp and move a part is not typically modeled explicitly. In response, fast computers and research efforts in robotic science are beginning to produce reasonable heuristic solutions to these robot planning problems in structured environments [Barraquand & Latombe, 1990]. An effort to standardize the state of the art in robot task level planning is given by Latombe (1992). Solutions for planning grasping motions for robots are also becoming available. For an example, see Pertin-Troccaz (1988). However, Sharma and colleagues (1996) correctly point out that these solutions are not integrated with material processing requirements. Furthermore, there is no architecture for the simulation and control of production that is designed to integrate these solutions for realistic manufacturing environments with multiple robots. Thus, despite the richness of robotic algorithms in the
literature, these solutions are not incorporated in industrial applications. Since task level planning is specified independently of any specific robot, a material-handling planner can generate a task specification from the part’s material handling requirements and check if any robots in the facility are capable of satisfying it. Thus, task level planners that can predict if a robot in the facility is capable of moving a part enable the development of the proposed material-handling resource model.

Material transportation actions include such tasks as AGVs moving a part or multiple parts by moving their holding fixture to a new location. In automated flexible manufacturing facilities, pallet exchanger tasks also involve moving fixtures and parts but are not included under material transportation because pallet exchangers are typically contained within material processing machines or automated storage buffer systems. Starting and ending locations, tolerance of these locations, and characteristics of the part specify material transportation tasks. The material transporter planner matches these tasks with the constraints of material transporter entities to determine how these entities will transport a given part. These constraints include information that describes the locations and their tolerances in the factory a given AGV may move to, how much space is available to hold a fixture on top of the AGV, and maximum payloads the AGV may move. Optimization of the selection of fixed AGV routes by researchers such as Seo and Egbelu (1996) does not address these constraints. Thus, the planner may incorporate mobile robot path planning techniques such are described by Cherif and Vidal (1998). As with robotics arm research, such algorithms developed by the mobile robotics community need to be combined with resource modeling for enterprise integration to become useful for industrial application. Fixturing resources must also be modeled because design of new fixturing for new parts limits the responsiveness of the manufacturing facility to new part designs [Hong et al, 1996]. Hong and colleagues (1996) provide algorithms similar to robot grasping solutions to determine how to arrange modular fixturing to hold a given part.

2.4 Manufacturing Standards

These algorithms, developed by the research community to plan the correct design or action for material transportation or handling goals, can be implemented as methods for resource model classes to create operations
planners if the object-oriented design of the resource model is sufficiently general. According to the principles of object-oriented design [Booch, 1986], information models that are created by using object classes with inheritance and ownership properties are flexible and facilitate modification. Thus, a properly designed object-oriented model can be expanded to handle many different perspectives. For example, the STEP standard, which uses object-oriented design to specify product data [Mitchell, 1996], has been expanded from its original model to describe many different product types. Similarly, there is a need to create general manufacturing resource classes that are capable of future expansion to facilitate planning solutions. The adoption of such a resource model provides two benefits to the manufacturing research community. First, it provides a level playing field to compare the performance and applicability of different algorithms applied to the same resources. Second, it provides a connection to researchers’ ultimate customers by enabling users of manufacturing resources to compare and use new algorithms. Thus, if there is a general agreement about this common resource model, then the code implementing task level algorithms may be shared by the research community.

The STEP standard provides a successful example of a model that integrates the manufacturing enterprise. STEP accomplishes this by enabling product data to be translated for different applications across the enterprise. Furthermore, STEP describes the information that defines a product type without explicitly describing planning procedures that define the capabilities of the product. The EXPRESS object scripting language [Schenck & Wilson, 1994] defines the information for the STEP classes without any methods or functions. The manufacturing resource model approach differs from STEP since STEP only defines part data whereas the resource model defines resource data as well as behaviors. Our goal for resource modeling is to facilitate integration with engineering functions such as operations planning and scheduling. Thus, algorithms that determine the capability of a manufacturing resource may be replaced with others by simply switching resource class methods in the operations planning code and then recompiling. Vendors are beginning to create software development environments that facilitate such switching of class methods [Schneider et al, 1995]. The data in the resource model classes could be described using a common neutral language such as EXPRESS to avoid platform and language dependence. Software agents, described by Pancerella and colleagues (1995), effectively implement this same flexibility without the necessity of recompilation and limitation of a single software package since certain agents can communicate the object class data to other agents representing the object methods. Using this agent
technology to implement the manufacturing operations planner opens up the possibility of downloading code to implement different manufacturing planning algorithms from the internet for the user’s own laboratory or industrial material movement systems. However, first it is necessary to formally define and test the object-oriented resource model for manufacturing regardless of implementation details such as the issue of whether the operations planner is C++ code or a group of distributed software agents.

2.5 Manufacturing Cost Estimation and Flexibility

In order to integrate the operations planning function with product design, the cost of using manufacturing resources must be modeled. Automatic planning of manufacturing operations first identifies whether the available resources are capable of creating the part and moving it through the manufacturing facility. After determining whether the facility is capable of producing the product, the manufacturing cost of the new part design can be determined by a simulation of the manufacturing operations of the part. Activity-based costing techniques have been developed to estimate the costs of different activities within a manufacturing facility to estimate the true cost of manufacturing a product for the design process [Cooper, 1991]. By assigning these activity-based costs to the manufacturing activities, a cost of manufacturing for the new part design can first be estimated by a simulation that simply sums the cost of using each resource to manufacture the part without normal production queue delays. A more accurate estimation of manufacturing costs uses a discrete event simulation to simulate manufacturing of simultaneous multiple parts in the factory and adds waiting costs to the first estimation. Since a resource model of each manufacturing resource already describes the information about the entity such as its capability to move or manufacture parts, it is natural to include the activity-based cost of using the entity in its resource model. Furthermore, since these resource models of machines can easily be plugged into the information model of the facility, the planning system can also be used to estimate the utility and cost of new manufacturing resources under consideration for purchasing. For example, the utility of material handling robots may be compared to the utility of workers whose capabilities can be modeled with work time standards such as MOST [Zandin, 1980]. Kaplan (1986) suggests that such estimations are useful for determining appropriate investments in manufacturing. From a production viewpoint, having plans with alternatives and associated costs enables the
production planner function to choose the lowest cost production plan from the alternatives with a simple cost estimation that ignores waiting costs.

This proposed manufacturing resource modeling methodology contributes to different kinds of flexibility in manufacturing, which is defined as the range and responsiveness of a manufacturing system with respect to the changes in the product type and production volume [Slack, 1989]. This system flexibility depends on the flexibility of the individual production resources in the facility. Given a manufacturing planner and resource model, the capability to simulate and control quickly the manufacturing operations for a new part becomes limited only by the physical flexibility of the resources. Naturally, this increases the response of the product type flexibility. The production volume flexibility can be achieved through the ability to reallocate resources quickly. A production operations plan describing a sequence of machine actions to produce a part with alternative actions can provide this production volume flexibility by enabling the selection of alternative resources to increase production. Typically, the expense of generating redundant manufacturing operations plans prohibits consideration of alternative plans. However, this becomes possible when manufacturing engineers possess the capability to generate automatically manufacturing plans for factories.
Chapter 3

PROBLEM FRAMEWORK AND SOLUTION APPROACH

3.1 Introduction

There is a significant amount of documented research that solves problems associated with flexible manufacturing planning and integration. This research includes planning of robot tasks, extraction of manufacturing features, and encoding of manufacturing rules into expert systems. There is a problem, however, because the architecture for integrating these algorithms with the resource and part information they require for practical implementation does not exist. The most important problem in flexible manufacturing is how to map a part specification and a facility to manufacturing operation plans because this mapping integrates product design with production constraints that include the equipment layout, tolerances specifications and capabilities, tool capabilities, and robot kinematic flexibility. These operation plans should be usable by simulations and controllers of production activities. Another problem is how to incorporate vendor supplied equipment data and new algorithms into the information model of the facility.

3.2 Problem Framework

3.2.1 Introduction

The present research begins to solve the manufacturing operations planning problem by presenting a formal definition of the problem and one tested solution to the problem. This is the first reported solution to the problem with such detail in the literature. Formal logic is used to specify the problem and its constraints. A planning algorithm is developed that solves the problem. Useful models of parts and resources that support the manufacturing operations planner for a given set of manufacturing activities are also defined and implemented. These models of parts and resources are formally defined and also are sufficiently open to incorporate additional information for the integration of other activities beyond operations planning. For instance, the same model
framework of resources used to plan manufacturing operations enables the storage of the operational and maintenance status of these resources.

### 3.2.2 Planning Tasks

The integration of product design and production requires a planner that considers all of the important constraints that prevent manufacturing resources from creating parts. This research uses constraints arising from machining the part with a limited number of machining operations and constraints arising from moving the part between buffers and machining centers. The result from considering these constraints is an operations plan designed for use by simulation models, production schedulers, design for manufacturability tools, or manufacturing execution systems. The typical simulation model and production scheduler require a sequence of operations, an estimate of the time per operation, and the major resources associated with each operation. The design for manufacturability tool may use an activity-based cost total of all operations that create the part. Thus, the design for manufacturability tool requires a sequence of operations and the cost for each operation.

A manufacturing execution system requires a sequence of operations, the major resources associated with each operation, the tooling and fixturing required for each operation, and a machine instruction set for the programmable equipment entity that performs each operation. In order to satisfy these requirements, the operations planner must plan at different levels of abstraction. At the highest level, the operations planner must plan how the part moves through the manufacturing facility and how features are added at different machining centers. At the machine task level, individual machine tasks are planned. Examples of machine tasks are: (1) a robot unloading a machining center, and (2) a machining center drilling a hole in a part. Lastly, at the kinematic level, machine trajectories are computed. In the model developed for this research, only the kinematic level planning for robotic material handlers is considered, because the trajectory planning for fixed-route material transporters and machine-tool material processor resources is relatively straightforward.

### 3.2.3 Resource and Part Modeling Requirements

To generate these plans for manufacturing equipment, it is necessary to model attributes of resources and their expected capabilities at different levels of abstraction. Also, it is useful to be able to model relationships
between resources. For example, the operations planner can use such information as which tools are available to a machining center or where a robot is placed so that it can unload from an incoming parts buffer. Similarly, it is useful to model the relationship between parts and features for a given part.

Figure 3.1 illustrates how these part and resource models fit in the information flow of an integrated flexible manufacturing enterprise. Using symbology for data flow diagrams from Rumbaugh (1991), this figure shows how information should flow between the resource model representing the existence, capabilities, status, cost, and layout of the resources of a flexible manufacturing facility, and engineering functions, the physical facility, machine vendors, and quality control. The product design engineering function creates a product specification representing the part model. Requirements for the part and resource models will be subsequently developed to match the goal of attaining this information flow. Note that this data flow is independent of the form of the different engineering functions. Examples of different forms include expert systems for machining process planning [Descotte & Latombe, 1981] and intelligent agents for design-for-inspectability [Pancerella et al. 1995].
3.3 Solution Approach

In order to address these research goals, this dissertation presents a new operations planning methodology that is grounded in procedural reasoning contained within behaviors of specific manufacturing resource objects. Figure 3.2 illustrates the information framework of this operations planning methodology. As illustrated in this figure, the operations planner maps a part model and a resource model into an operations plan. This operations plan is first represented by an OR search graph and then converted into a more usable AND/OR operations plan graph. The resource model is represented by a set of data describing specific resources in a facility and a set of behaviors characterizing the capabilities of resource classes. The software specification of the operations planner is formally defined using the approach from Gannon and colleagues (1994). This specification includes a formal
definition of the input and output domains for the operations planner along with the criteria for valid individual operations and sequences of operations. Formal definitions are used to specify these input and output domains using sets and first order predicates and functions [Galton, 1990]. Next, the high level part state space and planning algorithms used to satisfy this specification are described. Then the lower level geometric and tolerance equations are defined. These equations are used to calculate functions and predicates for the specification. An important planning constraint modeled in this work is the robot task planning problem for material handling. Finally, the implementation architecture consisting of object-oriented resource and part models to incorporate planning constraints imposed by resources and parts is specified.

The resource classes take advantage of object-oriented principles such as polymorphism, inheritance, and ownership to efficiently represent the data and capabilities of manufacturing resources. Furthermore, the resource class data and behaviors at different abstraction levels could be provided by different vendors and shared if resource standards such as those proposed by Jurrens and colleagues (1997) were adopted. The author [Steele et al, 2000] proposed a resource model framework that is used in this research. Accordingly, resource classes are defined for material processor MP, material handler MH, material transporter MT, and buffer storage BS resources. Other classes are defined to model tools T and fixtures F used by these equipment resources to make parts. Rules defining the capabilities of these resources are directly linked to models of the resources in class behaviors of these resources. Thus, a specific shop’s capability to produce parts is easily modeled using this approach while a set of rules in a typical machining expert system must be adapted to the specific shop’s resources with a greater level of customization. Furthermore, while following common practice is suitable for

**Figure 3.2 Process Planning Data Flow**

The resource classes take advantage of object-oriented principles such as polymorphism, inheritance, and ownership to efficiently represent the data and capabilities of manufacturing resources. Furthermore, the resource class data and behaviors at different abstraction levels could be provided by different vendors and shared if resource standards such as those proposed by Jurrens and colleagues (1997) were adopted. The author [Steele et al, 2000] proposed a resource model framework that is used in this research. Accordingly, resource classes are defined for material processor MP, material handler MH, material transporter MT, and buffer storage BS resources. Other classes are defined to model tools T and fixtures F used by these equipment resources to make parts. Rules defining the capabilities of these resources are directly linked to models of the resources in class behaviors of these resources. Thus, a specific shop’s capability to produce parts is easily modeled using this approach while a set of rules in a typical machining expert system must be adapted to the specific shop’s resources with a greater level of customization. Furthermore, while following common practice is suitable for
high production flow shops where efficient process strategies for predefined products have been identified, there
is a need to model the physical capability of resources to change the state of the part for flexible manufacturing
where the products are not predefined. There is a need for this type of flexible manufacturing given the trend
towards certain types of products that must be customized to the individual consumer. The state descriptions used
to model these physical capabilities include the location of the part on the shop floor and features added to the
raw material of the part.

This methodology uses object-oriented design to characterize the part model for manufacturing operations
planning. The part is described as an object that may own manufacturing feature objects. The part model is
represented as a grouping of the raw material feature objects with an AND/OR graph of other feature objects to
be added to the raw material. Such a product model can be generated from a CAD model using the algorithm
presented by Lee and Kim (1999). Feature objects may be classified into different types of manufacturing features
such as holes and surfaces. This part object may be considered as an extension of the virtual tool concept, which
was developed by Cannon and Thomas (1997) to specify robot tasks, to a virtual object, because automatic
planning of the part’s motion throughout the shop floor removes the necessity of planning individual tool and
robot gripper tasks.

Given this part description and resource modeling capability, operations plans can be generated that quickly
adapt existing resources in a shop to produce unanticipated products. This is accomplished by presenting a
finished product model as a goal state to a state-based search algorithm that uses key operators from class
behaviors of shop floor resource objects. These key operators eliminate the differences between state descriptions
of the part. The search algorithm uses a backward chaining approach to minimize the differences between an
initial state description of the raw material part located in a raw material buffer and the goal state description of
the finished part located in a finished parts buffer. In contrast to traditional search techniques that stop when one
path to the goal is reached, the search algorithm exhaustively explores the entire search space in order to find all
non-cyclic paths from the initial part state to the final. This search space is limited by use of the problem
reduction technique that was developed by Nilsson (1998). Furthermore, the algorithm checks for repeat nodes
and cycles. Generated nodes that repeat an ancestor node are eliminated from the search graph while a node that
repeats a non-ancestral node becomes an additional parent of that non-ancestral node’s children. When the search
space is entirely explored, another algorithm is used to trace through the resulting search graph in order to
produce an operations plan graph with alternative paths separated by OR branches.

3.4 Chapter Summary

This chapter has pointed to the lack of good models for planning manufacturing operations based on the
information in resource and part specifications. This chapter also has described a methodology for solving this
problem using an object-oriented information model of resources and parts. A planner, developed for this
research, uses artificial intelligence search techniques along with formally defined attributes and behaviors of
resources and parts in these models to plan manufacturing operations. These resource and part models also
contribute to the integration of the manufacturing enterprise by providing a formal information framework to
facilitate the addition of future attributes, behaviors, and subclasses. This resource-oriented planner creates
flexibility in manufacturing by enabling quick generation of manufacturing operations plans based on a direct
consideration of part and resource information that can be easily modified.
Chapter 4

FORMAL SPECIFICATION OF PROBLEM SOLUTION

The equipment, or hardware, by itself is rarely the primary source of a factory’s competitive advantage. What matters is how that hardware is used and how it is integrated with materials, people, and information through software—the systems and procedures that direct and control the factory’s activities [Hayes et al, 1988].

This chapter presents a formal specification for an algorithm to automatically generate operation plans for flexible manufacturing facilities. This specification is important because it defines the scope of the planning algorithm independently of the implementation. Thus, models are developed to formally define the input data to manufacturing operations planners and the generated output data. The input models define resource and part models while output models define operations plans. Elements of these models are used to develop a set of conditions that must be true in order for the output data to be valid. These models are useful because they define the necessary information for generating operations plans. Section 4.1 introduces software specification and describes the interpretation of the input and output models. The input data model describes the resource model of the manufacturing facility and the parts that can be made in the facility. Section 4.2 defines the part model. Section 4.3 defines the resource model. Section 4.4 defines the model of the output that consists of a set of linear operation plan sequences. Section 4.4 also defines the criteria that must be met if the operation plans are valid. Section 4.5 illustrates the formal specification of the generation of operations plans by proving that a robotic unload operation, a robotic load operation, and a CNC material processing operation are valid for a given set of equipment resources.

4.1 Introduction

Operations planning is the link between the design function and production. Thus, computer-aided operations planning enables faster design for manufacturing feedback and faster production startup for new products. An increasing portion of computer aided operations planning consists of automated decision making by software. According to Galton (1990), “Conventionally, the task of programming is to convert a specification, which says what results are required, into executable code, which details how the results are to be achieved” (p. 3). Thus, in
order to contribute to research focusing on the automated portion of computer-aided operations planning, it is first necessary to define specifications for automated operations planning. From Gannon and colleagues (1994), verification of a program could be achieved by showing the equivalence of the program with a function that maps from an input domain to an output domain.

The input to the operations planner is a resource model of the facility and a part model of the product, and the output is an operations plan for producing the part in the facility. The resource model is based on equipment classifications from Wysk and colleagues (1995). For this dissertation, the resource model is extended to include information that is required for operations planning and provides a framework for integrating other engineering activities. The part model was created to provide the necessary part-based information for planning the critical manufacturing operations to make the part.

A logical framework built from these models defines a first order language and a domain consisting of sets of elements whose interpretation are manufacturing resources and parts that are manufacturable by the facility. Function symbols and predicates are defined within this domain that express properties of these elements. These functions and predicates are defined in this chapter and listed in Appendix C. Logical statements are also defined that reflect the assumptions of the resource model. This first order language is also used to create a separate schema consisting of statements that reflect necessary conditions for the generated operations to be valid. Different interpretations of this domain reflect the modeling of different manufacturing facilities with different resource assets. The truthfulness of these conditions for a given interpretation ensures that the individual operations in the plan are executable by the facility’s resources and satisfy the individual operations’ objectives. These conditions are in the form of predicate calculus statements. Thus, for a given interpretation of the domain that models a specific facility’s resources and a part that is manufacturable by the facility, the generated operations may be proven to be valid by proving that the schema satisfies this interpretation of the domain. In other words, the operations are valid if all statements in the schema are true for this interpretation of the domain. Another test checks if all of the linear, ordered sequences of operations in the operations plan result in the finished part located in the finished parts buffer. This test may be expressed in terms of a state space with operators corresponding to manufacturing operations. If the application of each of the linear sequences of these
operators to the part’s initial state results in a part state that satisfies the goal condition, then the operations plan is valid. In this chapter, it will be demonstrated how the operations planner meets this specification of the operations planning by interpreting the domain to represent an example manufacturing facility and a part that is manufacturable by this facility. Next, an operations plan will be manually generated that manufactures a given part. Lastly, it will be demonstrated that this operations plan meets the specification by proving that the plan is valid according to the necessary conditions for individual operations and by verifying that the results of all linear sequences in the plan satisfy the goal conditions for the part.

4.2 Part Model Definition

The part model for operations planning requires information that is necessary to define goals and part related constraints for the planner. To define material processing goals for the planner, goal features must be defined in the part model. Precedences between goal features must also be defined. Sufficient information describing the individual goal features must be provided in order to determine if specific material processing machines, fixtures, and tools are capable of creating the features and with what machining parameters. This information includes the goal feature type, feature tolerances, and location tolerances relative to specific feature datums. Other critical part information for material processing specifies the material type and hardness of the part. For material handling planning, the geometric model of the part and the location tolerances of existing features are used to determine if automated material handlers are capable of moving parts between different fixtures.

In this operations planning framework, there exists a Parts set consisting of parts that are manufacturable by the facility’s resources. Parts are defined by a set of discrete features. For each element p of the set Parts, the predicate isapart (p) is true. Furthermore, the function hardness (p) returns the hardness of p’s material. There exists a Materials set consisting of materials that are machinable by the facility’s resources. For each element m of the Materials set, the predicate isamaterial (m) is true. The function material (part) returns an element of Materials that represents the material from which the part is made. There exists a set of features (Feat) that are owned by elements of the Parts set. For each element f of the setFeat, isafeature (f) is true. For each element f of
the set $\text{Feat}$, if $f$ represents a goal feature then $\text{isagoalfeature}(f)$ is true. The precedences between goal features are defined by an AND/OR graph of features [Lee & Kim, 1999].

Each part element of the $\text{Parts}$ set owns features that belong to the $\text{Feat}$ set. It is assumed that the features of a part are the same material as the part. These features are either cylindrical holes or flat surfaces that belong to subsets $\text{Holes}$ and $\text{FlatSurfaces}$. The predicates $\text{isahole}$ and $\text{isasurface}$ define membership in these subsets. For each element $f$ of the subset $\text{Holes}$, the function $\text{dia_tol}(f)$ returns the diametric tolerance of the hole. For each element $f$ of the subset $\text{Holes}$ or $\text{FlatSurfaces}$, the function $\text{finish}(f)$ returns the roughest surface finish allowed by the specification of the feature. During the manufacturing process, the part starts out with a set of features that describe the part’s raw material or blank. The goal of the operations planning for the part consists of determining sequences of manufacturing operations that add required features to the part following the correct feature precedences. Ownership between elements in this domain are expressed by the predicate $\text{owns}$. In general, for elements $a$ and $b$, $\text{owns}(a, b)$ is true if element $a$ owns element $b$. The relationship between each part and the features that define the part is expressed using the $\text{owns}$ predicate. The elements in the subset of features owned by a given part may have precedence relations. For example, let it be the case that feature $a$ must be manufactured before feature $b$. Then, the predicate $\text{precondition}(a, b)$ is true. In general, for elements $a$ and $b$, $\text{precondition}(a, b)$ is true if element $a$ must exist before element $b$ can be manufactured. This predicate may be true for pairs of features in the finished part model specification as well as features that must be added to the part as a consequence of precedences for operations. The relation between a feature whose location is specified with respect to another feature is expressed in this domain by the predicate, $\text{specifies}$. In general, for elements datum and feature, $\text{specifies}(\text{datum}, \text{feature})$ is true if element datum specifies the location of element feature. In this domain, only datum features that are elements of the $\text{FlatSurfaces}$ set may specify other features that belong to the $\text{FlatSurfaces}$ or $\text{Holes}$ subsets. Specifications that use flat surfaces as datums represent dimension vectors that are perpendicular to the datum surface. The function $\text{specifies_dist}(\text{datum}, \text{feature})$ returns the nominal distance of this specification. The function $\text{specifies_tol}(\text{datum}, \text{feature})$ returns the tolerance limit of this specification. The ternary functions, $\text{specifies_tolX}(\text{Facility}, \text{datum}, \text{feature})$, $\text{specifies_tolY}(\text{Facility}, \text{datum}, \text{feature})$, and $\text{specifies_tolZ}(\text{Facility}, \text{datum}, \text{feature})$ return the components of the specification tolerance vector along the coordinate axes of the $\text{Facility}$ element. In general, vector components of datum tolerance vectors that are
computed to be 0 are set to infinity (\(\infty\)) because the location of the datum along that vector component is not determined. Based on these definitions, the following formal statements are true.

\[
\text{Parts} = \{\text{Part}_1, \text{Part}_2, \ldots\} \\
\text{Materials} = \{\forall \text{mat} | \exists \text{part} \in \text{Parts} \ (\text{mat} = \text{material} (\text{part}))\} \\
\text{Feat} = \{\text{Feat}_1, \text{Feat}_2, \ldots\} = \{\text{Feat}, | \text{Feat}, \in \text{FlatSurfaces} \cup \text{Holes}\} \\
\forall \text{feat (isafeature (feat) } \Rightarrow \text{isasurface (feat) } \lor \text{isahole (feat))} \\
\forall \text{part, } \forall \text{feat (isapart (part) } \land \text{isafeature (feat)} \land \text{owns (part, feat)} \Rightarrow \text{material (part) } = \text{material (feat))} \\
\forall \text{datum, } \forall \text{feat (isafeature (datum) } \land \text{isafeature (feat) } \land \text{specifies (datum, feat)} \Rightarrow \text{isasurface (datum))} \\
\forall \text{feat, } \exists \text{ part (isafeature (feat) } \land \text{isapart (part) } \Rightarrow \text{owns (part, feat))}
\]

### 4.3 Resource Model Definitions

#### 4.3.1 Facility Model

A resource model of a facility for operations planning must specify the capability of a facility to manufacture parts. Thus, the resource model specifies what resources belong to a facility and where they are located in the layout. These resources are grouped into work centers and are classified according to function. Each class of resources is described using different information that is specific to its function. The work centers are defined by their ownership of individual resources.

A manufacturing facility may be described as a list of assets owned by a facility element. For a given element Facility, if the predicate isafacility (Facility) is true then Facility represents a manufacturing facility. This facility element also owns a set of parts Parts that the facility is capable of manufacturing. Each facility element owns a set of material processing work centers \(W_p\), a set of material transportation work centers \(W_t\), and a set of material storage work centers \(W_s\) in addition to the set of tools \(T\) and set of fixtures \(F\) available to the facility. For a given element \(W\), if isaworkstation \((W)\) is true then \(W\) represents a workstation. The predicates isaprocessorworkstation, isastorageworkstation, and isatransporteworkstation define membership in the work center subsets. The parts that the facility can manufacture are elements of the Parts set. The Facility element also owns a set of buffer ports called Ports. For a given element \(P\), if isaport (P) is true then \(P\) represents a port. Part movement within the facility can be specified with ports. A port represents a unique internal buffer belonging to a machine that owns one or more fixed physical locations for part storage. Each of these ports may belong to one or
more equipment assets. An equipment asset that is fixed on the facility floor owns one port, and the function \textit{port} (a) returns the port element owned by the equipment element a.

Individual equipment resources that are in the facility belong to the equipment set \( E \). These equipment resources represent entities in the facility that are independently capable of some action with the part. For a given element \( E \), if \( \text{isanequipment} \ (E) \) is true then \( E \) represents an equipment entity. Thus, equipment resources belong to a diverse set of entities that may include robots, CNC machines, human workers, and passive part buffers. Some of these equipment assets are defined to have fixed locations on the factory floor with tolerances relative to the facility’s coordinate frame. Other equipment assets such as AGVs move around between stations. Each of the fixed equipment assets owns a port \( \text{Port} \) where parts may be located. The following functions return the tolerances of the locations of these ports for each coordinate axis of the \( \text{Facility} \) element: \( \text{tol}\_\text{posX} \ (\text{Facility}, \text{Port}), \text{tol}\_\text{posY} \ (\text{Facility}, \text{Port}), \text{tol}\_\text{posZ} \ (\text{Facility}, \text{Port}) \). Each equipment asset also has a coordinate reference frame that is fixed to the equipment. Since these entities can be categorized by function, the equipment set includes the subsets \( MP \), \( MH \), \( MT \), and \( BS \) that subdivide the equipment resources of the facility. The \( MP \) set represents material processors in the facility, the \( MH \) set represents material handlers, the \( MT \) set represents material transporters, and the \( BS \) set represents passive or active buffer storage devices. The predicates \( \text{isaprocessor}, \text{isahandler}, \text{isatransporter}, \text{isabuffer} \) define membership in these sets. All of these equipment entities belong to work center subsets. Entities in the \( MP \), \( MH \), and \( BS \) subsets may be owned by elements of the \( Wp \) set, entities in the \( MT \) subsets may be owned by elements of the \( Wt \) sets, and entities in the \( MH \) and \( BS \) subsets may be owned by elements of the \( Ws \) sets. Tool assets in the \( T \) set may be available for usage by equipment assets in the \( MP \) set while fixture assets in the \( F \) set may be available for any equipment asset. This availability is represented by the \( \text{owns} \) predicate. Formally, sets and relationships describing resources are defined as follows.

\[
\exists \text{Facility}, \forall \text{Part} \ (\text{isafacility} \ (\text{Facility}) \land \text{isapart} \ (\text{Part}) \Rightarrow \text{owns} \ (\text{Facility}, \text{Part}))
\]
\[
W = \{Wp \cup Wt \cup Ws\} = \{W_1, W_2, W_3, \ldots\}
\]
\[
\exists \text{Facility}, \forall W \ (\text{isafacility} \ (\text{Facility}) \land \text{isaworkstation} \ (W) \Rightarrow \text{owns} \ (\text{Facility}, W))
\]
\[
\forall W, \forall E \ (\text{isaprocessorworkstation} \ (W) \land \text{isanequipment} \ (E) \land \text{owns} \ (W, E) \Rightarrow \text{isaprocessor} \ (E) \lor \text{isahandler} \ (E) \lor \text{isabuffer} \ (E))
\]
\[
\forall W, \forall E \ (\text{isatransporterworkstation} \ (W) \land \text{isanequipment} \ (E) \land \text{owns} \ (W, E) \Rightarrow \text{isatransporter} \ (E))
\]
\[
\forall W, \forall E \ (\text{isastorageworkstation} \ (W) \land \text{isanequipment} \ (E) \land \text{owns} \ (W, E) \Rightarrow \text{isahandler} \ (E) \lor \text{isabuffer} \ (E))
\]
\[
E = \{MP \cup MH \cup MT \cup BS\} = \{E_1, E_2, E_3, E_4, \ldots\}
\]
\[
T = \{T_1, T_2, \ldots\}
\]
∃Facility, ∀T (isafacility (Facility) ∧ isatool (T) ⇒ owns (Facility, T))
F = {F₁, F₂, . . .}
∃Facility, ∀F (isafacility (Facility) ∧ isafixture (F) ⇒ owns (Facility, F))
Ports = {Port₁, Port₂, . . .}
∃Facility, ∀Port (isafacility (Facility) ∧ isaport (Port) ⇒ owns (Facility, Port))
∀Port, ∃E (isaprocessor (E) ∨ isabuffer (E) ∨ isahandler (E) ⇒ owns (E, Port))

4.3.2 Tool Model

A model of the tools in a facility is required to determine if material processing tasks are possible. These tools may be owned by material processing equipment. These tools are defined with information that specifies what features they can create and with what accuracy on specific material types and hardesses.

The tool assets available to a manufacturing facility belong to the T set. A given element T represents a tool if the predicate isatool (T) is true. These assets are used to enable material processors to perform operations on parts. This set includes three subsets of processing machine tooling: Tdrill, Treamer, Tbore. The partial capability of individual tooling to create different goal features is modeled by the predicate, makes. In general, given elements a and b, the truthfulness of makes (a, b) expresses that element a is designed to make element b. In the interpretation that a ∈ T and b ∈ Feat, makes (a, b) represents the knowledge that the tool is designed to make the feature. Since the elements of the set of manufacturable features Feat are known to be manufacturable by tooling available to the facility, at least one tool exists that is capable of making each feature. Furthermore, the functions, accuracyX (E, t, feat), accuracyY (E, t, feat), and accuracyZ (E, t, feat) return the intrinsic bilateral accuracy of the location of goal feature feat created by the tool t for each x, y, and z coordinate axis of equipment E. Thus, if a tool were perfectly placed with respect to the part’s reference frame, it would place the feature feat within +/- accuracyX (E, tool, feat) along the x axis of the equipment deploying the tool.

For a given tool t and goal feature feat, the predicate makes (t, feat) is only true if specific conditions are met for the hole making tools. The ability for hole making tools to create holes is dependent on the required diameter of the hole, the base material, the hardness of the base material and the desired surface finish of the hole. The function makes_hole (tool, diameter, tolerance of diameter, surface finish, material type, material hardness) is true.
if a tool is able to make a hole with the specified diameter, tolerance of diameter, and surface finish on material with the specified type and hardness. Formally, these concepts are defined as follows.

\[ T = \{ \text{Tdrill} \cup \text{Treamer} \cup \text{Thore} \} \]

\[ \forall \text{tool} \ (\text{isatool}(\text{tool}) \Rightarrow \text{isadrill}(\text{tool}) \lor \text{isareamer}(\text{tool}) \lor \text{isabore}(\text{tool})) \]

\[ \forall \text{f}, \exists \text{tool} \ (\text{isagoalfeature}(\text{f}) \land \text{isatool}(\text{tool}) \Rightarrow \text{makes}(\text{tool}, \text{f})) \]

\[ \forall \text{tool}, \forall \text{f}, \exists \text{part} \ (\text{isatool}(\text{tool}) \land \text{isahole}(\text{f}) \land \text{isapart}(\text{part}) \land \text{owns}(\text{part}, \text{f}) \land \text{makes_hole}(\text{tool}, \text{diameter}(\text{f}), \text{dia_tol}(\text{f}), \text{finish}(\text{f}), \text{material}(\text{part}), \text{hardness}(\text{part})) \Rightarrow \text{makes}(\text{tool}, \text{f})) \]

4.3.3 Fixture Model

In this planning architecture, parts are always located in fixtures located somewhere in the facility. Thus, a model of part holding fixtures is required for planning of all manufacturing tasks in this architecture. This model defines the information that is required from fixture designers and vendors in order to characterize fixtures.

Fixtures in a manufacturing facility are members of the \( F \) set that locate and hold parts to a position and an orientation relative to the coordinate frame of some element of the \( \text{MP}, \text{MH}, \text{MT}, \) or \( \text{BS} \) sets. For a given element \( \text{F} \), if \( \text{isafixture}(\text{F}) \) is true then \( \text{F} \) represents a fixture. For instance, a fixture might be a pneumatic clamp that locates datums of the workpiece and holds it rigidly to a position and orientation in the work area of a milling machine. Another fixture is an electro-mechanical robotic gripper that holds a part fixed relative to the coordinate frame of the robot arm’s end effector coordinate frame. Other fixtures include part bins with a lesser constraint on position and slots on a conveyor belt that are built-in constraints of position and orientation. Fixture entities are defined by their capabilities to locate features of specific part types with respect to the fixture’s reference frame. These capabilities may be expressed as the fixture designer’s intention to locate part datums with respect to the fixture’s reference frame. The designer may design the fixture to hold the same part in different orientations. These different orientations are defined as different fixture intentions. For instance, given a symmetrical part, one fixture intention may be defined to locate features 1, 2, 3 while another fixture intention may be defined to locate features 4, 5, 6. An element \( \text{I} \) represents a fixture intention if \( \text{isafixtureintent}(\text{I}) \) is true. The locators for the primary, secondary, and tertiary datums remain the same on the fixture for different fixture intentions while the fixture intention defines which datums are located by these locators. A single point on the fixture and a vector from that point to the datum defines each locator. For a given element \( \text{L} \), if \( \text{isafixturelocator}(\text{L}) \) is true then \( \text{L} \) represents such a locator. This model is an approximation of the traditional fixture model that uses three locator
pins for the primary datum, two locator pins for the secondary datum, and one locator pin for the tertiary datum [Chang et al., 1998]. Since the fixture may be controlled to move from an open state to a closed state and vice versa, the locators may also move. Thus, the points and vectors that define the locators have one value for the opened fixture state and another for the closed fixture state. Each of these vectors is stored as a unit vector, a nominal vector length and a tolerance. Figure 4.1 illustrates these geometrical concepts for the primary and secondary locators of a fixture and a part.

![Figure 4.1. Fixture Locators, Points, and Vectors](image)

Explicit holding requirements for these fixtures are ignored in this model. This model implicitly assumes that the fixture designer incorporated holding requirements for different fixture applications. For instance, the locating ability for a fixture holding a part during machining processes is significantly different from the locating ability for a gripper fixture holding a part during robotic movement.

For flat surface datums, a location is specified using a starting point on the fixture and a vector that is perpendicular to the datum. The relation between a fixture fixt that locates a feature feat with an intent I using its locator locator is expressed in this domain by the predicate locates (fixt, feat, I, L). Thus, locates (fixt, feat, I, L) is true if element fixt locates element feat with intent I using locator L. Certain fixtures are controllable so that they may grasp a part. For a given fixture element fixt, the predicate controllable (fixt) is true if fixt can be controlled in this manner. If a fixture is controllable, the tolerance of the location of features may be different for
the closed state versus the open state of the fixture. The function $close_{-}tol\ (a, b)$ returns the bilateral tolerance limit of this location for feature $b$ if fixture $a$ is closed and if the part was loaded correctly. A part is considered to be loaded correctly into fixtures if the locatable features of the part are placed within tolerance relative to the fixture. For a fixture $a$ and feature $b$, this tolerance is given by the function $open_{-}tol\ (a, b)$. This tolerance is also the tolerance of the part’s datums when the part is ready to be unloaded from the fixture. For fixtures that are not controllable and thus cannot be opened or closed, the $open_{-}tol$ function returns the tolerance of the part’s location to be dropped into the passive fixture. The $close_{-}tol$ function returns the tolerance of the part’s location after it is released by the loading fixture. For a given fixture $Fixt$, and a feature $Feat$, if $controllable\ (Fixt)$ is not true, then $open_{-}tol\ (Fixt, Feat) = close_{-}tol\ (Fixt, Feat)$ unless a passive force such as gravity affects the part’s position after the part is released by the loading gripper-fixture. For instance, a part may be dropped into a passive fixture, and gravity will constrain the part’s position along the vertical axis to the bottom of the passive fixture.

Because fixtures are always used with equipment entities, functions are defined that return the vector components of these tolerances along the coordinate axes of an equipment entity. For an equipment entity $E$, which owns a fixture entity $fixt$ that locates a feature element datum, the following functions are defined that return the open and close tolerance vector components along the coordinate axes of $E$: $close_{-}tolX\ (E, fixt, datum)$, $close_{-}tolY\ (E, fixt, datum)$, $close_{-}tolZ\ (E, fixt, datum)$, $open_{-}tolX\ (E, fixt, datum)$, $open_{-}tolY\ (E, fixt, datum)$, and $open_{-}tolZ\ (E, fixt, datum)$. The open and close tolerances along the coordinate axes of the Facility element are given by the following functions: $close_{-}tolX\ (Facility, E, fixt, datum)$, $close_{-}tolY\ (Facility, E, fixt, datum)$, $close_{-}tolZ\ (Facility, E, fixt, datum)$, $open_{-}tolX\ (Facility, E, fixt, datum)$, $open_{-}tolY\ (Facility, E, fixt, datum)$, and $open_{-}tolZ\ (Facility, E, fixt, datum)$.

4.3.4 Material Processor Model

The material processors are the equipment that add value to a product by transforming its material state closer to the desired final state. These equipment entities are capable of using tools that they own to create features in parts. The model of material processors defines the information that is critical for operations planning that involves material processor equipment. This information includes ownership relationships with tools and fixtures in addition to the accuracy with which the material processor locates these secondary resources.
Instances of different types of material processes in the facility are subdivided into subsets of the MP set. For instance, milling machines in the facility belong to one of these subsets, while turning centers available to the facility belong to another subset. The MP subsets include Lathes, VertMills, and HorizMills. For every feature defined to be manufacturable by the facility, there is at least one equipment asset that can make this feature. Since the elements of the set of goal features Feat are known to be manufacturable by material processors available to the facility, at least one material processor exists that is capable of making each feature within the specified tolerance. Thus, the binary makes predicate is true for at least one material processing entity for each feature in the set of features F that are manufacturable by the facility. The ternary predicate makes (E, feat, fixt) is true if the equipment entity E is capable of making goal feature feat using the fixture fixt. Formally, the MP sets are defined as follows.

\[
\text{MP} = \{\text{Lathes} \cup \text{VertMills} \cup \text{HorizMills}\} = \{E_1, E_2, E_3, \ldots\}
\]

\[
\forall \text{feat}, \exists E (\text{isagoalfeature}(\text{feat}) \land \text{isaprocessor}(E) \Rightarrow \text{makes}(E, \text{feat}))
\]

Figure 4.2 illustrates different tolerances associated with the equipment’s ability to create a feature and the required tolerance of the feature. Each material processor has intrinsic limits to the accuracy with which it is able to place tools in order to create features in parts. For a given material processor element, E, these accuracy limits for the x, y, and z coordinate directions of the equipment’s reference frame are returned by the functions \(\text{accuracyX}(E)\), \(\text{accuracyY}(E)\), and \(\text{accuracyZ}(E)\). This accuracy limit along one equipment coordinate axis is illustrated in Figure 4.2 and labeled “equipment accuracy”. Material processors also own fixtures that are used to hold parts during processing operations. In general, for material processor entity E and fixture entity fixt, this property is described by \(\text{owns}(E, \text{fixt})\). This fixture element fixt is attached to the port Port owned by E. Given this relationship, then the tolerance limits for the position of the fixture along the x, y, and z coordinate directions of the port’s reference frame are returned by the functions \(\text{fixture_pos_tolX}(\text{Port}, \text{fixt})\), \(\text{fixture_pos_tolY}(\text{Port}, \text{fixt})\), and \(\text{fixture_pos_tolZ}(\text{Port}, \text{fixt})\). This position tolerance along one equipment axis is labeled “fixture position tolerance” in Figure 4.2. These same tolerances along the equipment’s coordinate axes are returned by the functions \(\text{fixture_pos_tolX}(E, \text{Port}, \text{fixt})\), \(\text{fixture_pos_tolY}(E, \text{Port}, \text{fixt})\), and \(\text{fixture_pos_tolZ}(E, \text{Port}, \text{fixt})\). Similarly, these same tolerances along the Facility element’s coordinate axes are returned by the functions.
fixture_pos_tolX (Facility, Port, fixt), fixture_pos_tolY (Facility, Port, fixt), and fixture_pos_tolZ (Facility, Port, fixt). For a given part, all datums \( d \) that specify the location of feature \( \text{feat} \) do so with a tolerance given by \( \text{specifies_tol} (d, \text{feat}) \). In Figure 4.2, this tolerance is illustrated along one equipment axis and labeled “feature specification tolerance”.

![Figure 4.2. Tolerances for creating feature along one equipment axis](image)

For a given equipment entity \( E \) that owns a fixture which holds a part with a feature \( \text{feat} \) whose location is specified by another feature element datum \( \text{datum} \), the components of the tolerance of a feature-datum’s specification of another feature along the equipment’s coordinate axes are returned by the following functions: \( \text{specifies_tolX} (E, \text{datum}, \text{feat}) \), \( \text{specifies_tolY} (E, \text{datum}, \text{feat}) \), and \( \text{specifies_tolZ} (E, \text{datum}, \text{feat}) \). These components of tolerance along the coordinate axes of the Facility element are returned by the following functions: \( \text{specifies_tolX} (\text{Facility}, E, \text{datum}, \text{feat}) \), \( \text{specifies_tolY} (\text{Facility}, E, \text{datum}, \text{feat}) \), and \( \text{specifies_tolZ} (\text{Facility}, E, \text{datum}, \text{feat}) \).

Necessary conditions for a material processor to make a feature are that it owns a tool designed to make the feature, and that the material processor owns a fixture designed with the intention to locate the datums used to specify the feature. It is assumed that such a fixture is designed to locate these datums using single point locators that perform even when the part is subjected to machining forces.

For each of the datums that specify a feature on a part, the tolerance of the position of a new feature created in a part must be less than the required tolerance of the feature. Figure 4.2 illustrates the tolerances associated
with a material processor resource, a tool held in the machine’s chuck, and a part held in a fixture mounted on the machine’s bed. In order for this machine to make a feature on the part using the tool, the tolerances associated with creating a new feature with the tool on the part must be less than or equal to the tolerance specifying the tolerance of the feature. These tolerances are specified so that tolerances along the same axis locating the same datum may be stacked. This assumption enables the creation of a tolerance loop that links the sum of tolerances between 2 datums with a specification of tolerances between these datums. Figure 4.3a illustrates the five different datums along one axis for the material processor setup in Figure 4.2. Datum #1 represents the machine tool’s frame. Datum #2 represents the position of the fixture. Datum #3 represents a feature located by the fixture. Datum #4 represents the new feature. Datum #5 represents the position of the hole-making tool. The notation, developed by Fraticelli and colleagues (1997), defines the manufacturing capability to locate datum B from datum A as \( M_{B,A} \). Furthermore, the manufacturing specification of datum A to datum B is defined by \( C_{A,B} \). The tolerances of these specifications are given by the function \( Tol \) such that \( Tol (M_{B,A}) \) is the tolerance of the manufacturing capability to locate datum B from datum A while \( Tol (C_{A,B}) \) is the tolerance of the manufacturing specification of datum A to datum B. In general, a manufacturing process that creates a feature at datum B is valid if \( Tol (C_{A,B}) \geq Tol (M_{B,A}) \). For the material processing task displayed in Figure 4.2, the process is valid if \( Tol (C_{3,4}) \geq Tol (M_{4,3}) = Tol (M_{4,5}) + Tol (M_{5,1}) + Tol (M_{1,2}) + Tol (M_{2,3}) \). Since tolerances along the same axis can be stacked, the tolerances of manufacturing capabilities from datum #4 to datum #3 may be added and compared to the manufacturing specification. Thus, as illustrated in Figure 4.3b, the following equation must be true for the operation to be valid: \( Tol (C_{3,4}) \geq Tol (M_{4,3}) = Tol (M_{4,5}) + Tol (M_{5,1}) + Tol (M_{1,2}) + Tol (M_{2,3}) \).

![Figure 4.3a. Datums for MP Task.](image)

![Figure 4.3b. Tolerance Loop for MP Task.](image)
From the definition of the datums in Figure 4.3a, the following definitions are true:

1. $M_{1,2} = \text{fixture position in equipment}$
2. $M_{2,3} = \text{location of datum by fixture}$
3. $C_{3,4} = \text{specification of new feature by datum}$
4. $M_{4,5} = \text{location of hole created by tool}$
5. $M_{5,1} = \text{equipment placement of tool}$

Using the labels from Figure 4.2, the following condition must be true for each x, y, and z equipment coordinate axis in order to create the feature:

$$\text{feature specification tolerance} \geq \text{equipment accuracy} + \text{tool accuracy} + \text{fixture position tolerance} + \text{fixture-datum location tolerance}$$

The formal functions that model these accuracies and tolerances along the x axis are mapped to the labels in Figure 4.2 as follows for a MP entity $E$, fixture $Fixt$, datum $Datum$ located by this fixture, port $Port$, feature $feat$ that is created, and tool $Tool$ that creates the feature:

1. $\text{specifies}_{tolX}(E, Datum, feat) = \text{feature specification tolerance}$
2. $\text{accuracy}_{X}(E) = \text{equipment accuracy}$
3. $\text{accuracy}_{X}(E, Tool, feat) = \text{tool accuracy}$
4. $\text{fixture-pos}_{tolX}(E, Port, Fixt) = \text{fixture position tolerance}$
5. $\text{close}_{tolX}(E, Fixt, Datum) = \text{fixture-datum location tolerance}$

Thus, the condition for a valid MP operation due to the tolerance loop along the x axis may also be expressed using these functions for the with the following statement.

$$\text{specifies}_{tolX}(E, Datum, feat) \geq \text{accuracy}_{X}(E) + \text{accuracy}_{X}(E, Tool, feat) + \text{fixture-pos}_{tolX}(E, Port, Fixt) + \text{close}_{tolX}(E, Fixt, Datum)$$

The formal conditions for the predicate $\text{makes}(E, feat, fixt)$ are defined using resource elements, predicates, and functions. The manufacturing tolerance capability condition is met if the tolerance loop equation is true for every feature-datum that specifies the new feature’s position and for each x, y, and z axis of the equipment frame. Thus, for a new feature that specified by three feature-datums, there are nine tolerance equations that must be true. Other conditions are that the equipment must own a fixture and this fixture must be designed with the intent to locate the part’s features that specify the new feature. If all of these conditions are met, then $\text{makes}(E, feat, fixt)$ is true. This is formally stated as follows.

Statement #1. $\forall feat, \forall E, \forall Fixt, \forall port, \forall tool, \forall L, \forall datum, \forall L (isagoalfeature(feat) \land isaprocessor(E) \land isafixture(Fixt) \land isaport(port) \land isatool(tool) \land isafixtureintent(I) \land isafeature(datum) \land isafixturelocator(L) \land owns(E, port) \land owns(E, Fixt) \land owns(E, tool) \land makes(tool, feat) \land fixt_intent(Fixt, I) \land specifies(datum, feat) \land locates(Fixt, datum, I, L)$
\[(\text{specifies}_{\text{tol}X}(E, \text{datum, feat}) \geq \text{accuracy}X(E) + \text{accuracy}_{X}(E, \text{tool, feat}) + \text{fixture}_{\text{pos}_{\text{tol}}X}(E, \text{port, fixt}) + \text{close}_{\text{tol}X}(E, \text{fixt, datum})) \wedge \]
\[(\text{specifies}_{\text{tol}Y}(E, \text{datum, feat}) \geq \text{accuracy}Y(E) + \text{accuracy}_{Y}(E, \text{tool, feat}) + \text{fixture}_{\text{pos}_{\text{tol}}Y}(E, \text{port, fixt}) + \text{close}_{\text{tol}Y}(E, \text{fixt, datum})) \wedge \]
\[(\text{specifies}_{\text{tol}Z}(E, \text{datum, feat}) \geq \text{accuracy}Z(E) + \text{accuracy}_{Z}(E, \text{tool, feat}) + \text{fixture}_{\text{pos}_{\text{tol}}Z}(E, \text{port, fixt}) + \text{close}_{\text{tol}Z}(E, \text{fixt, datum})) \Rightarrow \text{makes}(E, \text{feat, fixt}))\]

### 4.3.5 Material Handler Model

The material handlers are equipment that load and unload products to and from equipment for processing, storage, and transportation while material transporters move products between stations. Material handlers have the kinematic flexibility to change the orientation and position of the part to remove parts from equipment fixturing or insert parts into equipment fixturing. The model of material handlers defines the information that is required to plan material handling operations for manufacturing. Thus, material handlers are modeled as assets whose placement in the shop floor layout determines which buffers, machines and ports they may access. Other information specifies the capability of material handlers to use gripper-fixtures to grasp parts and move them.

The material handling assets in a facility belong to the \(\text{MH}\) set. In this model, these material handling assets are grouped into robotic material handlers and human material handlers. Robotic material handler assets available to the facility belong to the subset \(\text{Rmh}\) and human material handler assets belong to the subset \(\text{Wmh}\). Since the planning constraints due to human material handlers are a subset of the constraints due to robotic material handlers, this research focuses on robotic material handlers. For each material handler asset, when it has possession of a part, the part is considered to be in a buffer that is represented by the material handler’s home port element. The coordinate frame of this port coincides with the equipment frame of the material handler entity. Sets of loadable ports \(\text{LoadPorts}\) and unloadable ports \(\text{UnloadPorts}\) describe the intention of the placement of each \(\text{MH}\) element \(E_i\) to load parts from its home port to other equipment resources, and unload parts from other equipment resources to its home port.
In general, the binary predicate `loadable (robot, Port)` is true for elements `robot` and `Port` if element `robot` is intended to load to element `Port`. Furthermore, the binary predicate `unloadable (robot, Port)` is true for elements `robot` and `Port` if element `robot` is intended to unload from element `Port`. Thus, `loadable (E_i, Port)` is true for each material handling element `E_i` and each port `Port` that belongs to `LoadPorts_i`. Similarly, `unloadable (E_i, Port)` is true for each material handling element `E_i` and each port `Port` that belongs to `UnloadPorts_i`. Note that these ports only represent buffers; the binary `loadable` or `unloadable` predicates are only necessary conditions for the physical load or unload operation because other prerequisite conditions exist in order for a given `MH` resource to load or unload specific parts to or from a given port. In order to incorporate these additional conditions, the ternary predicates `unloadable` and `loadable` are defined. The predicate `unloadable (Part, robot, Port, fixt)` is true for elements `Part`, `robot`, `Port`, and `fixt` if element `robot` is capable of unloading element `Part` from element `fixt` located at element `Port`. Similarly, the predicate `loadable (Part, robot, Port, fixt)` is true for elements `Part`, `robot`, `Port`, and `fixt` if element `robot` is capable of loading element `Part` to element `fixt` located at element `Port`. Necessary conditions for `unloadable (Part, E_i, Port, fixt)` are that the `Port` element is an element of `E_i`'s set of unloadable ports `UnloadPorts_i`, and that `E_i` owns a fixture that is capable of locating at least one datum of the part element `Part`. Similarly, necessary conditions for entity `E_i` to load a part element `Part` to a port element `Port` are that `Port` is a member of `E_i`'s set of loadable ports `LoadPorts_i`, and that `E_i` owns a fixture that is capable of locating at least one datum of `Part`. Formally, these assumptions may be expressed as follows.

\[ MH = \{Rmh \cup Wmh\} \]

\[ \forall E_i, \text{UnloadPorts}_i = \{\text{port} \mid \text{isaport (port)} \land \text{isahandler (E_i)} \land \text{unloadable (port, E_i)}\} \]

\[ \forall E_i, \text{LoadPorts}_i = \{\text{port} \mid \text{isaport (port)} \land \text{isahandler (E_i)} \land \text{loadable (port, E_i)}\} \]

\[ \forall E, \forall \text{Part}, \forall \text{port}, \forall \text{fixt} (\text{isahandler (E)} \land \text{isapart (Part)} \land \text{isaport (port)} \land \text{isafixture (fixt)} \land \text{unloadable (Part, E, port, fixt)} \Rightarrow \exists \text{gripper}, \forall \text{feat}, \forall \text{I}_g, \forall \text{L}_g, \exists \text{Buff}, \forall \text{datum}, \forall \text{I}_f, \forall \text{L}_f (\text{isafeature (gripper)} \land \text{isafeature (feat)} \land \text{isafixtureent (I}_g) \land \text{isafixturelocator (L}_g) \land (\text{isaprocessor (Buff)} \lor \text{isabuffer (Buff)} \lor \text{isatransporter (Buff)}) \land \text{owns (Part, feat)} \land \text{locates (gripper, feat, I}_g, \text{L}_g) \land \text{owns (E, gripper)} \land \text{owns (Buff, port)} \land \text{owns (Buff, fixt)} \land \text{locates (fixt, datum, I}_f, \text{L}_f) \land \text{specifies (datum, feat)}) \]

\[ \forall E, \forall \text{Part}, \forall \text{port}, \forall \text{fixt} (\text{isahandler (E)} \land \text{isapart (Part)} \land \text{isaport (port)} \land \text{isafixture (fixt)} \land \text{loadable (Part, E, port, fixt)} \Rightarrow \exists \text{gripper}, \forall \text{feat}, \exists \text{Buff}, \forall \text{datum}, \forall \text{I}_g, \forall \text{L}_g (\text{isafeature (gripper)} \land \text{isafeature (feat)} \land (\text{isaprocessor (Buff)} \lor \text{isabuffer (Buff)} \lor \text{isatransporter (Buff)}) \land \text{isafeature (datum)} \land \text{isafeature (datum)} \land \text{isafeatureent (I}_g) \land \text{isafeaturelocator (L}_g) \land \text{owns (E, gripper)} \land \text{owns (Part, feat)} \land \text{locates (gripper, feat, I}_g, \text{L}_g) \land \text{owns (Buff, port)} \land \text{owns (Buff, fixt)} \land \text{locates (fixt, datum, I}_f, \text{L}_f) \land \text{specifies (datum, feat)}) \]
Figures 4.4 and 4.5 illustrate the tolerances associated with unload and load operations. Material handlers have errors in position control of their end effector mounting surfaces that are illustrated by the robot position accuracy labels in Figures 4.4 and 4.5. The position error bounds for all three of the material handler $E$ element’s coordinate axes are modeled by the following functions: $\text{accuracy}_X(E)$, $\text{accuracy}_Y(E)$, and $\text{accuracy}_Z(E)$. These same position errors along the coordinate axes of the Facility element’s coordinate frame are returned by the following functions: $\text{accuracy}_X(\text{Facility}, E)$, $\text{accuracy}_Y(\text{Facility}, E)$, and $\text{accuracy}_Z(\text{Facility}, E)$.

Figure 4.4. Tolerances associated with material handling unload operation along one facility axis
Fixtures that are owned by material handler elements represent grippers attached to the end effector mounting surfaces of robotic material handlers. For each robotic material handler $E$, when it is at its home position, its gripper-fixture is located at a transformation returned by the function $home\_trans (E)$. Thus, the gripper-fixture frame is transformed by $home\_trans (E)$ with respect to the port frame of the robot. Just like fixtures for material processors, gripper fixtures locate datums of parts within given tolerances. One significant difference between material processor fixtures and robotic gripper fixtures is that holding requirements are different. In this model, a gripper fixture may locate multiple surfaces with only two parallel points of contact because it is assumed that the gripped part will not rotate during movement. Positional errors in the attachment of these fixtures to the end of robotic material handlers are illustrated along one coordinate axis by the gripper position tolerance labels in Figures 4.4 and 4.5. For a given fixture $\text{fixt}$ and a robot home port $\text{Port}$, the components of these tolerances along the coordinate axes of the robot’s home port frame are modeled by the following functions: $fixture\_pos\_tolX (\text{Port}, \text{fixt})$, $fixture\_pos\_tolY (\text{Port}, \text{fixt})$, and $fixture\_pos\_tolZ (\text{Port}, \text{fixt})$. These same positional errors along the coordinate axes of the Facility element’s frame are modeled by the following functions: $fixture\_pos\_tolX (\text{Facility}, \text{Port}, \text{fixt})$, $fixture\_pos\_tolY (\text{Facility}, \text{Port}, \text{fixt})$, and $fixture\_pos\_tolZ (\text{Facility}, \text{Port}, \text{fixt})$.
For the unload operation, the gripper-fixture owned by the material handler must have sufficient open
tolerance to match the cumulative errors in the placement of the gripper-fixture and fixture holding the part.

Figure 4.4 illustrates the tolerances associated with an unload operation along one axis. Thus, a set of necessary
conditions for an unload operation are that a set of equations derived from a tolerance equation are true for all
datums \( f \) intended to be located by the equipment fixture that gets unloaded, for all datums \( g \) intended
to be located by the gripper fixture, and for each \( x \), \( y \), and \( z \) coordinate axis of the factory floor. Assuming that
tolerances can be stacked along an axis just like in the material processing scenario in Figure 4.2, the sum of all of
the tolerances that represents the placement error of a material handler entity’s placement of its gripper-fixture
must be less than the required tolerance of the part’s position relative to the gripper along each \( x \), \( y \), and \( z \) axis.

The eight datums for a material handling unload operation are illustrated in Figure 4.6a. In this figure, datum #1
represents the Facility element’s floor. Datum #2 represents the port of the equipment that will be unloaded.
Datum #3 represents the fixture on the equipment that gets unloaded. Datum #4 represents one feature that is
located by this fixture. Datum #5 represents one feature that is located by the gripper-fixture. Datum #6 represents
the gripper-fixture. Datum #7 represents the robot that unloads the equipment. The material handling unload task
that is displayed in Figure 4.4 is valid if \( \text{Tol}(C7,6) \geq \text{Tol}(M6,7) \) for each feature located by the gripper-fixture and
along each \( x \), \( y \), and \( z \) axis. Since tolerances along the same axis can be stacked, the tolerances of manufacturing
capabilities from datum #5 to datum #6 may be added and compared to the manufacturing specification. Thus, as
illustrated in Figure 4.6b, the following equation must be true for the operation to be valid: \( \text{Tol}(C7,6) \geq \text{Tol}(M6,7) = \text{Tol}(M6,4) + \text{Tol}(M4,3) + \text{Tol}(M3,2) + \text{Tol}(M2,1) + \text{Tol}(M1,8) + \text{Tol}(M8,5) + \text{Tol}(M5,7) \).
Based on the illustrations in Figures 4.4 and 4.6a, the following definitions for unload operations are true:

1. \( C_{7,6} \) = open fixture specification of feature datum \( g \) located by gripper-fixture
2. \( M_{6,4} \) = feature datum \( g \) (located by gripper fixture) specification of other feature datum \( f \) (located by buffer fixture)
3. \( M_{4,3} \) = closed fixture specification of feature datum \( f \) located by buffer fixture
4. \( M_{3,2} \) = buffer fixture position in equipment port
5. \( M_{2,1} \) = equipment port position in facility
6. \( M_{1,8} \) = robot base port position in facility
7. \( M_{8,5} \) = robot placement of gripper-fixture
8. \( M_{5,7} \) = gripper position relative to end effector mounting plate

Thus, the set of necessary conditions due to accuracy constraints for an unload operation are that a set of equations derived from the following equation are true for all datums \( g \) that are intended to be located by the gripper-fixture and for each \( x \), \( y \), and \( z \) coordinate axis of the factory floor.

\[
\text{open}_{\text{tol}}(\text{Facility}, E, \text{gripper}, \text{datum } g) \geq \text{specifies}_{\text{tol}}(\text{Facility}, \text{datum } g, \text{datum } f) + \text{close}_{\text{tol}}(\text{Facility, Buff, fixture, datum } f) + \text{fixture}_{\text{pos}}_{\text{tol}}(\text{Facility, port } b, \text{fixt}) + \text{tol}_{\text{pos}}(\text{Facility, port } b) + \text{tol}_{\text{pos}}(\text{Facility, port } r) + \text{accuracy}_{\text{X}}(\text{Facility, E}) + \text{fixture}_{\text{pos}}_{\text{tol}}(\text{Facility, port } r, \text{gripper})
\]

The last necessary condition for an unload operation to be valid in this schema is that there must exist a collision free trajectory for the material handler to move the gripper-fixture from the material handler’s home position to a grasping position at the unload fixture and return with the part. This condition is described by the predicate \( \text{unloadTrajExists}(E, \text{gripper, fixture, port, part}) \), which is true if a collision free trajectory exists for material handler \( E \) to move the gripper element to unload the part element from the fixture at the port and bring
the part to the home port of E. The manufacturing tolerance capability condition for an unload operation is met if the tolerance loop equation is true for every all datums $datum_g$ that are intended to be located by the gripper-fixture and for each $x$, $y$, and $z$ coordinate axis of the factory floor. Thus, for a gripper fixture that locates three feature-datums, there are nine tolerance equations that must be true. Finally, all of these necessary conditions for an unload operation may be formally expressed as follows.

Statement #2. $\forall E, \forall Part, \forall port_r, \forall gripper, \forall datum_g, \forall L_g, \forall Buff, \forall port_b, \forall datum_f, \forall L_f$ (isahandler (E) $\land$ isapart (Part) $\land$ isafixture (gripper) $\land$ isafeature (datum_g) $\land$ isafixtureintent (I_g) $\land$ isafixturelocator (L_g) $\land$ (isaprocessor (Buff) $\lor$ isabuffer (Buff) $\lor$ isatransporter (Buff)) $\land$ isaport (port_b) $\land$ owns (E, gripper) $\land$ unloadTrajExists (E, gripper, fixt, port_b, Part) $\land$ owns (Part, datum_g) $\land$ locates (gripper, datum_g, I_g, L_g) $\land$ owns (Buff, port_b) $\land$ owns (Buff, fixt) $\land$ locates (fixt, datum_f, I_f, L_f) $\land$ specifies (datum_g, datum_f) $\land$ (open_tolX (Facility, E, gripper, datum_g) $\geq$ specifies_tolX (Facility, datum_g, datum_f) $+$ close_tolX (Facility, Buff, fixt, datum_g) $+$ fixture_pos_tolX (Facility, port_b, fixt) $+$ tol_posX (Facility, port_b) $+$ tol_posY (Facility, port_b) $+$ tol_posZ (Facility, port_b) $+$ accuracyX (Facility, E) $+$ accuracyY (Facility, E) $+$ accuracyZ (Facility, E) $\land$ (open_tolY (Facility, E, gripper, datum_g) $\geq$ specifies_tolY (Facility, datum_g, datum_f) $+$ close_tolY (Facility, Buff, fixt, datum_g) $+$ fixture_pos_tolY (Facility, port_b, fixt) $+$ tol_posX (Facility, port_b) $+$ tol_posY (Facility, port_b) $+$ tol_posZ (Facility, port_b) $+$ accuracyX (Facility, E) $+$ accuracyY (Facility, E) $+$ accuracyZ (Facility, E) $\land$ (open_tolZ (Facility, E, gripper, datum_g) $\geq$ specifies_tolZ (Facility, datum_g, datum_f) $+$ close_tolZ (Facility, Buff, fixt, datum_g) $+$ fixture_pos_tolZ (Facility, port_b, fixt) $+$ tol_posX (Facility, port_b) $+$ tol_posY (Facility, port_b) $+$ tol_posZ (Facility, port_b) $+$ accuracyX (Facility, E) $+$ accuracyY (Facility, E) $+$ accuracyZ (Facility, E) $\land$ unloadable (Part, E, port_b)) $\Rightarrow$ unloadable (Part, E, port_b, fixt))

Figure 4.5 illustrates the tolerances associated with the load operation. A necessary condition for a load operation is that the tolerance of the material handler’s placement of the part must be less than the tolerance of the placement of the part in the fixture for each $x$, $y$, and $z$ axis. Thus, a set of necessary conditions for a load operation are that a set of equations derived from the tolerance equation are true for all datums $datum_g$ intended to be located by the equipment fixture that gets loaded, for all datums $datum_f$ intended to be located by the gripper fixture, and for each $x$, $y$, and $z$ coordinate axis of the factory floor. The eight datums for a material handling load operation are illustrated in Figure 4.7a. In this figure, datum #1 represents the Facility floor. Datum #2 represents the port of the equipment that will be unloaded. Datum #3 represents the fixture on the equipment that gets loaded. Datum #4 represents one feature that is located by this fixture. Datum #5 represents one feature that is located by the gripper-fixture. Datum #6 represents the gripper-fixture. Datum #7 represents the robot that unloads the equipment. The material handling load task that is displayed in Figure 4.4 is valid if $Tol (C_{3,4}) \geq Tol$
(M_{4,3}) for each feature located by the gripper-fixture and along each x, y, and z axis. Since tolerances along the same axis can be stacked, the tolerances of manufacturing capabilities from datum #5 to datum #6 may be added and compared to the manufacturing specification. Thus, as illustrated in Figure 4.7b, the following equation must be true for the operation to be valid: $Tol(C_{3,4}) \geq Tol(M_{4,3}) = Tol(M_{4,6}) + Tol(M_{6,7}) + Tol(M_{7,5}) + Tol(M_{5,8}) + Tol(M_{8,1}) + Tol(M_{1,2}) + Tol(M_{2,3})$.

**Figure 4.7a. Datums for MH Load Task**  **Figure 4.7b. Tolerance Loop for MH Load Task**

Based on the illustrations in Figures 4.5 and 4.7a, the following definitions for load operations are true:

1. $C_{3,4}$ = open fixture specification of feature datum located by buffer fixture.
2. $M_{4,6}$ = feature datum (located by buffer fixture) specification of other feature datum (located by gripper)
3. $M_{6,7}$ = closed fixture specification of feature datum located by gripper-fixture
4. $M_{7,5}$ = gripper position relative to end effector mounting plate
5. $M_{5,8}$ = robot placement of gripper-fixture
6. $M_{8,1}$ = robot base port position in facility
7. $M_{1,2}$ = equipment port position in facility
8. $M_{2,3}$ = buffer fixture position in equipment port

Thus, the set of necessary conditions due to accuracy considerations for a load operation are that a set of equations derived from the following equation are true for all datums datum that are intended to be located by the buffer fixture and for each x, y, and z coordinate axis of the factory floor.

open fixture datum tolerance $\geq$ specification tolerance from datum to datum + close gripper datum tolerance + gripper position tolerance + robot position accuracy + robot home port position tolerance + buffer port position tolerance + fixture position tolerance
The formal functions that model these accuracies and tolerances along the x axis are mapped to the labels in Figure 4.5 as follows for all datums \( \text{datum}_f \) that are intended to be located by the buffer fixture.

1. \( \text{open_tol}_X(\text{Facility}, \text{Buff}, \text{fixture}, \text{datum}_f) = \text{open fixture datum}_f \) tolerance
2. \( \text{specifies_tol}_X(\text{Facility}, \text{datum}_f, \text{datum}_g) = \text{specification tolerance from datum}_f \) to datum\(_g\)
3. \( \text{close_tol}_X(\text{Facility}, \text{Buff}, \text{gripper}, \text{datum}_g) = \text{closed gripper datum}_g \) tolerance
4. \( \text{fixture_pos_tol}_X(\text{Facility}, \text{port}_r, \text{gripper}) = \text{gripper position tolerance} \)
5. \( \text{accuracy}_X(\text{Facility}, E) = \text{robot position accuracy} \)
6. \( \text{tol_pos}_X(\text{Facility}, \text{port}_r) = \text{robot home port position tolerance} \)
7. \( \text{tol_pos}_X(\text{Facility}, \text{port}_b) = \text{buffer port position tolerance} \)
8. \( \text{fixture_pos_tol}_X(\text{Facility}, \text{port}_b, \text{fixt}) = \text{fixture position tolerance} \)

Using these defined functions, the tolerance equation may be formally expressed as follows for the x axis.

\[
\text{open_tol}_X(\text{Facility}, E, \text{fixture}, \text{datum}_f) \geq \text{specifies_tol}_X(\text{Facility}, \text{datum}_f, \text{datum}_g) + \text{close_tol}_X(\text{Facility}, \text{Buff}, \text{gripper}, \text{datum}_g) + \text{fixture_pos_tol}_X(\text{Facility}, \text{port}_r, \text{gripper}) + \text{accuracy}_X(\text{Facility}, E) + \text{tol_pos}_X(\text{Facility}, \text{port}_r) + \text{tol_pos}_X(\text{Facility}, \text{port}_b) + \text{fixture_pos_tol}_X(\text{Facility}, \text{port}_b, \text{fixt})
\]

The last necessary condition for an load operation to be valid in this schema is that there must exist a collision free trajectory for the material handler to move the gripper-fixture grasping the part from the material handler’s home position to a loading position at the fixture and return without the part. This condition is described by the predicate, \( \text{loadTrajExists}(E, \text{grip}, \text{fixt}, \text{Port}, \text{Part}) \), which is true if a collision free trajectory exists for \( E \) to move the gripper element \( \text{grip} \) to load the element \( \text{Part} \) at the fixture \( \text{fixt} \) at the port \( \text{Port} \) and return without the part to the home port of \( E \). The manufacturing tolerance capability condition for a load operation is met if the tolerance loop equation is true for every all datums \( \text{datum}_f \) that are intended to be located by the buffer’s fixture and for each x, y, and z coordinate axis of the factory floor. Thus, for a buffer fixture that locates three feature-datums, there are nine tolerance equations that must be true. Thus, necessary conditions for a load operation may be formally expressed as follows.

Statement #3. \( \forall E, \forall \text{Part}, \forall \text{port}_r, \forall \text{fixt}, \forall \text{gripper}, \forall \text{datum}_f, \forall I_f, \forall L_f, \forall \text{Buff}, \forall \text{port}_b, \forall \text{datum}_g, \forall I_g, \forall L_g, \forall E, \forall \text{port}_r, \forall \text{gripper}, \forall \text{datum}_g, \forall I_g, \forall L_g \) (isahandler (E) ∧ isapart (Part) ∧ isaport (port) ∧ isafixture (fixt) ∧ isafixture (gripper) ∧ isafixtureint (I) ∧ isafixturelocator (L) ∧ isaprocessor (Buff) ∨ isabuffer (Buff) ∨ isatransporter (Buff)) ∧ isaport (port) ∧ isafixture (datum) ∧ isafixtureint (I) ∧ isafixturelocator (L) ∧ owns (E, gripper) ∧ loadTrajExists (E, gripper, fixt, port, Part) ∧ owns (Part, datum) ∧ locates (fixt, datum, I, L) ∧ owns (Buff, port) ∧ owns (Buff, fixt) ∧ owns (Part, datum) ∧ locates (gripper, datum, I, L) ∧ isafixture (datum) ∧ isafixtureint (I) ∧ isafixturelocator (L) ∧ owns (E, gripper) ∧ loadTrajExists (E, gripper, fixt, port, Part) ∧ owns (Part, datum) ∧ locates (fixt, datum, I, L) ∧ own...
\[
(\text{open\_tolY} (\text{Facility}, \text{E}, \text{fixture}, \text{datum}_f) \geq \text{specifies\_tolY} (\text{Facility}, \text{datum}_g, \text{datum}_f) + \text{close\_tolY} (\text{Facility}, \text{Buff}, \text{gripper}, \text{datum}_g) + \text{fixture\_pos\_tolY} (\text{Facility}, \text{port}_t, \text{gripper}) + \text{accuracyY} (\text{Facility}, \text{E}) + \text{tol\_posY} (\text{Facility}, \text{port}_t) + \text{fixture\_pos\_tolY} (\text{Facility}, \text{port}_b, \text{fixt})) \land
\]
\[
(\text{open\_tolZ} (\text{Facility}, \text{E}, \text{fixture}, \text{datum}_f) \geq \text{specifies\_tolZ} (\text{Facility}, \text{datum}_g, \text{datum}_f) + \text{close\_tolZ} (\text{Facility}, \text{Buff}, \text{gripper}, \text{datum}_g) + \text{fixture\_pos\_tolZ} (\text{Facility}, \text{port}_t, \text{gripper}) + \text{accuracyZ} (\text{Facility}, \text{E}) + \text{tol\_posZ} (\text{Facility}, \text{port}_t) + \text{fixture\_pos\_tolZ} (\text{Facility}, \text{port}_b, \text{fixt}))
\]
\[
\Rightarrow \text{loadable} (\text{Part}, \text{E}, \text{port}_b, \text{fixt}))
\]

### 4.3.6 Material Transporter Model

Material transporters are equipment that lack the ability to manipulate and simply move products from one location to another. Examples of material transporters are AGVs, conveyors, and workers. In this research, it is assumed that the controllers for automated material transporters do not require planning to move parts between ports. The model of material transporters includes information that is required for manufacturing material transport planning for this operations planning architecture. Thus, the material transporter model includes details such as which ports a material transporter may move to and the fixtures that can hold parts during this movement.

The material transporters available to the facility are grouped into the subset of automated material transporters \( \text{Rmt} \) and the subset of human material transporters \( \text{Hmt} \). The capability for an material transporter element to move parts between ports is partially defined by a set of reachable ports that represent accessible locations on the facility floor. In general, the binary predicate \( \text{reachable} (\text{E}, \text{Port}) \) is true for elements \( \text{E} \) and \( \text{Port} \) if material transporter element \( \text{E} \) is capable of moving parts to element \( \text{Port} \). This binary predicate \( \text{reachable} (\text{E}_i, \text{Port}) \) is true for each material transporter \( \text{E}_i \) and each \( \text{Port} \) that belongs to \( \text{E}_i \)’s list of reachable ports \( \text{ReachPorts} \).

Note that these ports only represent buffers; the binary \( \text{reachable} \) predicate is only a necessary condition for the physical operation because other prerequisite conditions exist in order for a given \( \text{MT} \) resource to move a specific part to or from a given port. To incorporate these additional conditions, the quaternary predicate \( \text{reachable} \) is used. The quaternary predicate \( \text{reachable} (\text{Part}, \text{Port}, \text{E}, \text{fixt}) \) is true for elements \( \text{Part}, \text{Port}, \text{E}, \text{and fixt} \) if material transporter element \( \text{E} \) is capable of moving element \( \text{Port} \) to or from element \( \text{port} \) located by element \( \text{fixt} \). In this interpretation, \( \text{reachable} (\text{Part}, \text{Port}, \text{E}, \text{fixt}) \) is true for a given part, a material transporter \( \text{E} \), a port \( \text{Port} \), and a fixture \( \text{fixt} \) if \( \text{Port} \) is reachable by \( \text{E} \) and \( \text{E} \) owns a fixture \( \text{fixt} \) that can locate at least one datum of the part \( \text{Part} \) during movement. Furthermore, a material transporter may move a part between two ports if both ports are in the
set of reachable ports and the material transporter owns a fixture that can locate at least one datum of the part
during movement. The predicate movable (Part, E, p_j, p_k, fixt) is true if material transporter E is capable of
moving part Part between ports p_j and p_k using fixture fixt. These assumptions and predicates are defined formally
as follows.

\[ MT = \{ Rmt \cup Wmt \} \]

∀E_i, ReachPortsi = \{ p | isatransporter (E_i) \land isaport (p) \land reachable (p, E_i) \}

Statement #4. ∀E, ∀Part, ∀p, ∀Fixt, ∃Feat, ∀I, ∀L (isatransporter (E) \land isapart (Part) \land reachable (p, E) \land isafixture (Fixt) \land isafeature (Feat) \land isafixtureintent (I) \land isafixturelocator (L) \land owns (Part, Feat) \land owns (E, Fixt) \land locates (Fixt, Feat, I, L)) ⇒ reachable (Part, p, E, Fixt))

Statement #5. ∀E, ∀Part, ∀p_j, ∀p_k, ∀Fixt (isatransporter (E) \land isapart (Part) \land reachable (Part, E, p_j, Fixt) \land reachable (Part, E, p_k, Fixt) ⇒ movable (Part, E, p_j, p_k, Fixt))

4.3.7 Buffer Storage Model

The model of buffer storage entities in this operations planning architecture is required to specify the
information that is necessary for operations planning. Since the operations planner determines if buffer storage
resources are capable of storing specific parts and if material handlers may load and unload their ports, buffer
storage resources are simply modeled as entities that own ports and fixtures.

Buffer storage entities in the BS set either passively hold parts with a possible mechanical inversion or
actively store the parts with an automated storage and retrieval system (AS/RS). Buffer storage entities may either
be actively controlled or not. Furthermore, a part may be storable by a buffer storage entity if the entity owns a
fixture that locates at least one datum of the part. The predicate storable (Part, E, Fixt) is true if buffer storage
entity E is capable of storing part Part using fixture Fixt. These assumptions are defined formally as follows.

∀E (isabuffer (E) ⇒ active (E) \lor \neg active (E))

Statement #6. ∀E, ∀Part, ∀fixt, ∃feat, ∀I, ∀L (isabuffer (E) \land isapart (Part) \land isafixture (fixt) \land isafeature (feat) \land isafixtureintent (I) \land isafixturelocator (L) \land owns (Part, feat) \land owns (E, fixt) \land locates (fixt, feat, I, L)) ⇒ storable (Part, E, fixt))

4.4 Operations Plan Model and Criteria

The operations plan model represents operations plans that are generated by the planner. This model must
specify the operations plan with sufficient information to drive applications such as design for manufacturing,
planning, scheduling, simulation and execution. Furthermore, to illustrate the assumptions behind the generation of these plans, the criteria for valid operations with respect to the resource and part models must be defined.

The necessary conditions for a set of operations \( \mathbf{OP} \) to be valid for a given facility model can be defined as a set of logic statements. For each element of the \( \mathbf{OP} \) set \( \mathbf{OP}_i \), \( \text{equip} (\mathbf{OP}_i) \) returns the equipment element that performs operation \( \mathbf{OP}_i \), \( \text{port} (\mathbf{OP}_i) \) returns the port element associated with \( \mathbf{OP}_i \), and \( \text{part} (\mathbf{OP}_i) \) returns the part element for which \( \mathbf{OP}_i \) is defined. Other functions that are used to describe an operation include \( \text{time} (\mathbf{OP}_i) \) and \( \text{cost} (\mathbf{OP}_i) \). For material processing operations, \( \text{feat} (\mathbf{OP}_i) \) returns the feature element that is created by \( \mathbf{OP}_i \) at port \( \text{port} (\mathbf{OP}_i) \), \( \text{tool} (\mathbf{OP}_i) \) returns the tool element used by \( \mathbf{OP}_i \), and \( \text{fixture} (\mathbf{OP}_i) \) returns the fixture element used to hold the part during processing. For material handling unload operations, \( \text{port} (\mathbf{OP}_i) \) returns the port element from which the part is unloaded while \( \text{fromfixture} (\mathbf{OP}_i) \) returns the fixture element that is unloaded. For material handling load operations, \( \text{port} (\mathbf{OP}_i) \) returns the port element that the part is loaded to while \( \text{tofixture} (\mathbf{OP}_i) \) returns the fixture element that gets loaded. For material transportation move operations, \( \text{startport} (\mathbf{OP}_i) \) returns the port element where the part is located initially, \( \text{endport} (\mathbf{OP}_i) \) returns the port where the part is located as a result of the operation, and \( \text{fixture} (\mathbf{OP}_i) \) returns the fixture element used to hold the part during the movement. Each element of the \( \mathbf{OP} \) set also may have the following properties that reflect the type of operation: \( \text{make} (\mathbf{OP}_i) \), \( \text{unload} (\mathbf{OP}_i) \), \( \text{load} (\mathbf{OP}_i) \), and \( \text{move} (\mathbf{OP}_i) \). These conditions are formally defined as follows.

\[
\forall x \ (\text{isanoperation} (x) \land \text{process} (x) \Rightarrow \text{makes} (\text{equip} (x), \text{feat} (x), \text{fixture} (x))) \\
\forall x \ (\text{isanoperation} (x) \land \text{unload} (x) \Rightarrow \text{unloadable} (\text{part} (x), \text{equip} (x), \text{port} (x), \text{fromfixture} (x))) \\
\forall x \ (\text{isanoperation} (x) \land \text{load} (x) \Rightarrow \text{loadable} (\text{part} (x), \text{equip} (x), \text{port} (x), \text{tofixture} (x))) \\
\forall x \ (\text{isanoperation} (x) \land \text{move}(x) \Rightarrow \text{moveable} (\text{part} (x), \text{equip} (x), \text{startport} (x), \text{endport} (x), \text{fixture} (x)))
\]

Given that all of the individual operations in a given operations plan are proven to be valid for a given interpretation according to the previous schema, all of the linear sequences of operations in this plan also must be proven to be valid. A given sequence of operations is valid if the operations change the part state from its initial state to a state that meets the goal condition. According to Nilsson (1998), there are three basic components to a representation of a state-space graph:

1. A description with which to label the start node. This description is some data structure modeling the initial state of the environment.
2. Functions that transform a state description representing one state of the environment into one that represents the state resulting after an action. These functions are usually called operators.

3. A goal condition, which can be either a True-False-valued function on the state descriptions or a list of actual instances of state descriptions that correspond to goal states. (p. 130)

According to this representation, for a given part Part the part state space is defined by (Part, p, flist, fixt, fixtIntent), where Part is the part, p is the port where part is located, flist is the set of features that have been added to the part, fixt is the fixture that currently holds the part, and fixtIntent is the intent that defines how the part is placed in the fixture. The port that represents the raw material buffer for the part and the set of features that define the part’s blank defines the start node. The set of all possible node elements in this state space is Nodes. The function node (Part, p, flist, fixt, fixtIntent) returns the node corresponding to Part, p, flist and fixt, and fixtIntent. The four different types of operations in this model correspond to four different operators (make, load, unload, and move) that may change the part’s state within the defined part state space. These operators transform the part state description into another part state description that represents the state resulting after the physical action corresponding to the operator. The process operator describes the operation of machining a new goal feature using a material processor in a part located by a fixture with a fixture intention. This operator is defined as process (f, E, fixt, intent), where the material processor is E, the goal feature is f, the fixture locating the part is fixt, and the fixture intention is intent. The unload operator describes the operation of unloading a part from a port, fixture, and fixture intention to a material handler’s home port using the material handler. This operator is defined as unload (p, E, fixt, intent), where the material handler is E, the port is p, the fixture is fixt, and the fixture intention is intent. The load operator describes the operation of loading a part from a material handler’s home port to another port, fixture, and fixture intention. This operator is defined as load (p, E, fixt, intent), where the material handler is E and the part is loaded to port p at fixture fixt with intention intent. The move operator describes the operation of transporting a part between stations using a material transporter. This operator is defined as move (p1, p2, E, fixt, intent), where the material transporter E moves the part from port p1 to port p2 while holding the part in fixture fixt with fixture intention intent. These operators, their preconditions, and their transformations of the state space are formally defined in Table 4.1.
Table 4.1 defines the high level preconditions and consequences of the four operators on the part state. The preconditions for operator process are that the part must be located at the material processor’s port and fixture and all of \( f \)'s precondition features must be in the part state’s feature list \( flist \). Furthermore, the predicate makes must be true for the material processor, fixture, and goal feature. The consequence of operator process is that the goal feature is added to the part’s feature list \( flist \). The preconditions for operator unload are that the part must be located at a buffer’s port and fixture. Furthermore, the predicate unloadable must be true for the part, material handler, and fixture. The consequence of the operator unload is that the part is located at the material handler’s home port and gripper-fixture. The precondition for the operator load is that the part is located at the material handler’s home port. Furthermore, the predicate loadable must be true for the part, material handler, port, and buffer fixture. The consequence of the operator load is that the part is located at the target buffer’s port and fixture. The precondition for the operator move is that the part is located at one station of the material transporter. Furthermore, the predicate moveable must be true for the part, the material transporter, a fixture owned by the material transporter, and the two ports. The consequence of the move operator is that the part is located at another station of the material transporter.

The goal condition for this state space is defined as a True-False-valued function on the state description. Given a state space node \((\text{Part, p, flist, fixt, intent})\), the goal predicate isaggoal \((\text{node (Part, p, flist, fixt, intent)})\) is true if \( p \) is the goal port representing the finished parts buffer, and if all required goal features are elements of the

<table>
<thead>
<tr>
<th>OPERATIONS</th>
<th>PRECONDITIONS for State ((p, \text{flist, fixt, intent}))</th>
<th>TRANSFORMATION</th>
</tr>
</thead>
</table>
| process       | \( \text{fixt} = \text{fixt}', \text{owns} (E, p), \text{owns} (E, \text{fixt}) \)
\( \forall f_{\text{pre}} \in \text{flist} | \text{precondition} (f_{\text{pre}}, f). \)
|               | \( (p, \text{flist, fixt, intent}) \rightarrow (p, \text{flist}', \text{fixt, intent}) \),
where \( \text{flist}' = (\text{flist} \cup f). \)
| unload        | \( \text{fixt} = \text{fixt}', p = p', \text{intent} = \text{intent}' \)                                                                 | \( (p, \text{flist, fixt, intent}) \rightarrow (p', \text{flist, fixt}', \text{intent}') \) |
| load          | \( p = \text{port} (E), \text{owns} (E, \text{fixt}) \)                                                                  | \( (p, \text{flist, fixt, intent}) \rightarrow (p', \text{flist, fixt}') \) |
| move          | \( \text{fixt} = \text{fixt}', p = p_1, \text{intent} = \text{intent}' \)                                                  | \( (p_1, \text{flist, fixt, intent}) \rightarrow (p_2, \text{flist, fixt, intent}) \) |
The function `goalp (Part)` returns the port representing the finished parts buffer. For a given node returned by the function `node (Part, p, flist, fixt, intent)`, the predicate `goalfeaturespresent (node (Part, p, flist, fixt, intent))` is true if all features in the list of goal features for `Part` are in the list of features represented by `flist`.

The function `goalfixture (Part)` returns the goal fixture for the part `Part`. The function `goalfixtureintent (Part)` returns the fixture intent for the part `Part`. Formally, the goal condition is met if the predicate `goal` is determined to be true as follows.

\[
\forall Part, \forall p, \forall flist, \forall fixt, \forall intent (isapart (Part) \land isaport (p) \land isalistoffeatures (flist) \land isafixture (fixt) \land isafixtureintent (intent) \land p = goalp (Part) \land goalfeaturespresent (node (Part, p, flist, fixt, intent)) \land fixt = goalfixture (Part) \land intent = goalfixtureintent (Part) \Rightarrow goal (node (Part, p, flist, fixt, intent)))
\]

4.5 Illustration

In order to illustrate these formal specifications for the input and output for manufacturing operations planning, an example interpretation of the domain is presented that represents a simple manufacturing facility and a simple part. The information that describes an example facility’s resources and their layout for this research is given in Figure 4.8, while the information describing the part is illustrated in Figure 4.9. The formal specification of the facility’s resource model is given. Next, details of three pieces of equipment in this facility that belong to the BS, MH, and MP sets are given. In order to illustrate the information that defines operations plans, three example operations associated with this equipment are specified and proofs of their validity are presented. These operations are as follows: material handler E2 unloads a part from buffer storage E1, material handler E2 loads a part to buffer storage E1, and material processor E3 makes a feature on the part. Other operations and their sequences that are required to completely manufacture the part are described and validated in Appendix A.

4.5.1 Resource Model

The simple facility that is illustrated by Figure 4.8 owns three work centers which may be classified into storage, processing, and transportation types. Work center #1 is a material storage work center that owns two equipment assets: an AS/RS equipment entity E1 of the set BS and a robot equipment entity E2 of the set MH. Work center #2 is a material processing work center that owns three equipment assets: a vertical mill equipment entity E3 of the set MP, a robot equipment entity E4 of the set MH, and a table equipment entity E5 of the set BS.
Work center #3 is a material transportation work center that owns an AGV equipment asset E₆ of the set MT. The facility owns 7 ports and a twist drill asset that is also owned by the vertical mill. A part element, Part₁, represents the physical part represented by Figure 4.9. This part is made from Aluminum 1108 and has a hardness of 148. The part is also defined to have 6 surface features: Feat₁, Feat₂, Feat₃, Feat₄, Feat₅, and Feat₆ in the part blank and an extra hole feature Feat₇ to be added during the manufacturing process. The location of Feat₇ is specified relative to Feat₂, Feat₃, and Feat₅. Equipment assets also own fixtures whose design intent is to locate this part. The resource E₁ owns fixture F₁, which is designed to locate features Feat₂, Feat₃, and Feat₅ of Part₁ with intention #1. E₂ owns fixture F₂, which is designed to locate features Feat₂, Feat₆, and Feat₅ of Part₁ with intention #1. E₃ owns fixture F₃, which is designed to locate features Feat₂, Feat₃, and Feat₅ of Part₁ with intention #1. In this model, it is assumed that trajectories exist for the robotic material handler resources to load and unload parts of type Part₁ to and from the ports in their load and unload port sets. Formally, this interpretation of the operations planning domain is defined in Table 4.2.

### Table 4.2. Formal Specification of Resource Model for Illustration

<table>
<thead>
<tr>
<th>Wp</th>
<th>{W₂}</th>
<th>W₁</th>
<th>W₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>W₄</td>
<td>{∅}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>{T₁}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>{Fixt₁, Fixt₂, Fixt₃, Fixt₄, Fixt₅, Fixt₆}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tdrill</td>
<td>{∅}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BM</td>
<td>{E₁, E₂, E₃, E₄, E₅, E₆}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BS</td>
<td>{E₁, E₃}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MP</td>
<td>{E₁}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MT</td>
<td>{E₆}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MH</td>
<td>{E₂, E₄}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lathes</td>
<td>{∅}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VertMills</td>
<td>{E₁}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HorizMills</td>
<td>{∅}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>owns(E₁, Port₁), owns(E₁, Port₃), owns(E₂, Port₂), owns(E₂, Port₃), owns(E₃, Port₃), owns(E₄, Port₃), owns(E₅, Port₃), owns(E₆, Port₄)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>controllable(Fixt₁), controllable(Fixt₂), controllable(Fixt₃), controllable(Fixt₄), controllable(Fixt₅), controllable(Fixt₆)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RM₉</td>
<td>{E₂, E₄}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wm₉</td>
<td>{∅}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UnloadPorts₂</td>
<td>{Port₁, Port₃}, UnloadPorts₄</td>
<td>{Port₃, Port₄, Port₅}</td>
<td></td>
</tr>
<tr>
<td>LoadPorts₂</td>
<td>{Port₁, Port₃}, LoadPorts₄</td>
<td>{Port₂, Port₄, Port₅}</td>
<td></td>
</tr>
<tr>
<td>Rmt</td>
<td>{E₁}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wmt</td>
<td>{∅}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ReachPorts₉</td>
<td>{Port₃, Port₄}</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Parts

<table>
<thead>
<tr>
<th>Part</th>
<th>{Part₁, Part₃, Part₅}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feat</td>
<td>{Feat₁, Feat₂, Feat₃, Feat₄, Feat₅, Feat₆, Feat₇}</td>
</tr>
<tr>
<td>material(Part)</td>
<td>Aluminum 1108, hardness(Part) = 48, diameter(Feat₁, Feat₅) = 0.25, dia_tol(Feat₁, Feat₅) = 0.008</td>
</tr>
<tr>
<td>finish(Feat₁, Feat₅)</td>
<td>100, makes hole(T₁, Feat₂, Feat₄, Feat₆, Feat₇) = 0.25, 0.008, 100, Aluminum 1108, 48</td>
</tr>
<tr>
<td>locates(Fixt₁, Feat₁, 1, 1, 1)</td>
<td>locates(Fixt₁, Feat₁, 1, 1, 2, 2)</td>
</tr>
<tr>
<td>locates(Fixt₁, Feat₁, 1, 1, 3, 3)</td>
<td>locates(Fixt₂, Feat₂, 1, 1, 1, 1)</td>
</tr>
<tr>
<td>locates(Fixt₂, Feat₂, 1, 1, 2, 2)</td>
<td>locates(Fixt₂, Feat₂, 1, 1, 3, 3)</td>
</tr>
<tr>
<td>locates(Fixt₃, Feat₃, 1, 1, 1, 1)</td>
<td>locates(Fixt₃, Feat₃, 1, 1, 2, 2)</td>
</tr>
<tr>
<td>locates(Fixt₃, Feat₃, 1, 1, 3, 3)</td>
<td>locates(Fixt₄, Feat₄, 1, 1, 1, 1)</td>
</tr>
<tr>
<td>locates(Fixt₄, Feat₄, 1, 1, 2, 2)</td>
<td>locates(Fixt₄, Feat₄, 1, 1, 3, 3)</td>
</tr>
<tr>
<td>locates(Fixt₅, Feat₅, 1, 1, 1, 1)</td>
<td>locates(Fixt₅, Feat₅, 1, 1, 2, 2)</td>
</tr>
<tr>
<td>locates(Fixt₅, Feat₅, 1, 1, 3, 3)</td>
<td>locates(Fixt₆, Feat₆, 1, 1, 1, 1)</td>
</tr>
<tr>
<td>locates(Fixt₆, Feat₆, 1, 1, 2, 2)</td>
<td>locates(Fixt₆, Feat₆, 1, 1, 3, 3)</td>
</tr>
</tbody>
</table>
unloadTrajExists \( (E_2, \text{Fixt}_2, \text{Fixt}_1, \text{Port}_1, \text{Part}_1) \), loadTrajExists \( (E_2, \text{Fixt}_2, \text{Fixt}_1, \text{Port}_1, \text{Part}_1) \)

unloadTrajExists \( (E_4, \text{Fixt}_2, \text{Fixt}_6, \text{Port}_3, \text{Part}_1) \), loadTrajExists \( (E_4, \text{Fixt}_2, \text{Fixt}_6, \text{Port}_3, \text{Part}_1) \)

unloadTrajExists \( (E_4, \text{Fixt}_2, \text{Fixt}_5, \text{Port}_5, \text{Part}_1) \), loadTrajExists \( (E_4, \text{Fixt}_2, \text{Fixt}_5, \text{Port}_5, \text{Part}_1) \)

unloadTrajExists \( (E_4, \text{Fixt}_2, \text{Fixt}_6, \text{Port}_4, \text{Part}_1) \), loadTrajExists \( (E_4, \text{Fixt}_2, \text{Fixt}_6, \text{Port}_4, \text{Part}_1) \)

4.5.2 Unload and Load Operations

Coordinate frames and tolerance information for some of the equipment and fixtures in the example facility are illustrated in Figures 4.10, 4.11, and 4.12. Figure 4.10 describes the setup for the AS/RS \( E_1 \). Note that
since \( X_{E_1} \) coincides with \( X_{\text{Facility}} \) and \( Y_{E_1} \) coincides with \( Y_{\text{Facility}} \), the tolerances along the Facility coordinate axes are equal to the tolerances along the \( E_1 \) coordinate axes. All coordinate frames in this example have Z axes that are in the vertical direction. Functions that describe \( E_1 \)'s setup have the following values.

\[
\begin{align*}
\text{tol}_{\text{posX}}(\text{Facility, Port}_1) &= 0.01, \quad \text{tol}_{\text{posY}}(\text{Facility, Port}_1) = 0.01, \quad \text{tol}_{\text{posZ}}(\text{Facility, Port}_1) = 0.01 \\
\text{fixture}_{\text{pos tolX}}(\text{Port}_1, \text{Fixt}_1) &= 0.001, \quad \text{fixture}_{\text{pos tolY}}(\text{Port}_1, \text{Fixt}_1) = 0.001, \quad \text{fixture}_{\text{pos tolZ}}(\text{Port}_1, \text{Fixt}_1) = 0.001 \\
\text{open}_{\text{tolX}}(E_1, \text{Fixt}_1, \text{Feat}_3) &= 0.03, \quad \text{open}_{\text{tolY}}(E_1, \text{Fixt}_1, \text{Feat}_3) = \infty, \quad \text{open}_{\text{tolZ}}(E_1, \text{Fixt}_1, \text{Feat}_3) = \infty \\
\text{close}_{\text{tolX}}(E_1, \text{Fixt}_1, \text{Feat}_3) &= 0.03, \quad \text{close}_{\text{tolY}}(E_1, \text{Fixt}_1, \text{Feat}_3) = \infty, \quad \text{close}_{\text{tolZ}}(E_1, \text{Fixt}_1, \text{Feat}_3) = \infty \\
\text{open}_{\text{tolX}}(E_1, \text{Fixt}_1, \text{Feat}_2) &= \infty, \quad \text{open}_{\text{tolY}}(E_1, \text{Fixt}_1, \text{Feat}_2) = 0.03, \quad \text{open}_{\text{tolZ}}(E_1, \text{Fixt}_1, \text{Feat}_2) = \infty \\
\text{close}_{\text{tolX}}(E_1, \text{Fixt}_1, \text{Feat}_2) &= \infty, \quad \text{close}_{\text{tolY}}(E_1, \text{Fixt}_1, \text{Feat}_2) = 0.03, \quad \text{close}_{\text{tolZ}}(E_1, \text{Fixt}_1, \text{Feat}_2) = \infty \\
\text{open}_{\text{tolX}}(E_1, \text{Fixt}_1, \text{Feat}_3) &= \infty, \quad \text{open}_{\text{tolY}}(E_1, \text{Fixt}_1, \text{Feat}_3) = \infty, \quad \text{open}_{\text{tolZ}}(E_1, \text{Fixt}_1, \text{Feat}_3) = \infty \\
\text{close}_{\text{tolX}}(E_1, \text{Fixt}_1, \text{Feat}_3) &= \infty, \quad \text{close}_{\text{tolY}}(E_1, \text{Fixt}_1, \text{Feat}_3) = \infty, \quad \text{close}_{\text{tolZ}}(E_1, \text{Fixt}_1, \text{Feat}_3) = \infty \\
\text{open}_{\text{tolX}}(E_1, \text{Fixt}_1, \text{Feat}_5) &= \infty, \quad \text{open}_{\text{tolY}}(E_1, \text{Fixt}_1, \text{Feat}_5) = \infty, \quad \text{open}_{\text{tolZ}}(E_1, \text{Fixt}_1, \text{Feat}_5) = \infty \\
\text{close}_{\text{tolX}}(E_1, \text{Fixt}_1, \text{Feat}_5) &= \infty, \quad \text{close}_{\text{tolY}}(E_1, \text{Fixt}_1, \text{Feat}_5) = \infty, \quad \text{close}_{\text{tolZ}}(E_1, \text{Fixt}_1, \text{Feat}_5) = \infty \\
\text{open}_{\text{tolX}}(E_1, \text{Fixt}_1, \text{Feat}_5) &= \infty, \quad \text{open}_{\text{tolY}}(E_1, \text{Fixt}_1, \text{Feat}_5) = \infty, \quad \text{open}_{\text{tolZ}}(E_1, \text{Fixt}_1, \text{Feat}_5) = \infty \\
\text{close}_{\text{tolX}}(E_1, \text{Fixt}_1, \text{Feat}_5) &= \infty, \quad \text{close}_{\text{tolY}}(E_1, \text{Fixt}_1, \text{Feat}_5) = \infty, \quad \text{close}_{\text{tolZ}}(E_1, \text{Fixt}_1, \text{Feat}_5) = \infty
\end{align*}
\]
Based on the information describing the robotic material handler E2 in Figure 4.11, the following functions are defined. Note that since \(X_{E2}\) coincides with \(X_{Facility}\) and \(Y_{E2}\) coincides with \(Y_{Facility}\), the tolerances along the Facility’s coordinate axes are equal to the tolerances along the \(E2\) coordinate axes.

### Tolerances

- **\(tol_{posX}(Facility, Port1) = 0.01\)**
- **\(tol_{posY}(Facility, Port1) = 0.01\)**
- **\(tol_{posZ}(Facility, Port1) = 0.01\)**
- **\(accuracy_{X}(Facility, E2) = 0.001\)**
- **\(accuracy_{Y}(Facility, E2) = 0.001\)**
- **\(accuracy_{Z}(Facility, E2) = 0.001\)**
- **\(fixture_{pos}_{tolX}(Port6, Fixt2) = 0.001\)**
- **\(fixture_{pos}_{tolY}(Port6, Fixt2) = 0.001\)**
- **\(fixture_{pos}_{tolZ}(Port6, Fixt2) = 0.001\)**
- **\(open_{tolX}(E2, Fixt2, Feat6) = 0.001\)**
- **\(open_{tolY}(E2, Fixt2, Feat6) = \infty\)**
- **\(open_{tolZ}(E2, Fixt2, Feat6) = \infty\)**
- **\(close_{tolX}(E2, Fixt2, Feat6) = 0.001\)**
- **\(close_{tolY}(E2, Fixt2, Feat6) = \infty\)**
- **\(close_{tolZ}(E2, Fixt2, Feat6) = \infty\)**
- **\(open_{tolX}(E2, Fixt2, Feat2) = \infty\)**
- **\(open_{tolY}(E2, Fixt2, Feat2) = 0.001\)**
- **\(open_{tolZ}(E2, Fixt2, Feat2) = \infty\)**
- **\(close_{tolX}(E2, Fixt2, Feat2) = \infty\)**
- **\(close_{tolY}(E2, Fixt2, Feat2) = \infty\)**
- **\(close_{tolZ}(E2, Fixt2, Feat2) = \infty\)**
- **\(open_{tolX}(E2, Fixt2, Feat5) = \infty\)**
- **\(open_{tolY}(E2, Fixt2, Feat5) = \infty\)**
- **\(open_{tolZ}(E2, Fixt2, Feat5) = 0.001\)**
- **\(close_{tolX}(E2, Fixt2, Feat5) = \infty\)**
- **\(close_{tolY}(E2, Fixt2, Feat5) = \infty\)**
- **\(close_{tolZ}(E2, Fixt2, Feat5) = \infty\)**

### Diagram

![Diagram of tolerances associated with \(E1\), \(Fixt1\), and \(Part1\)](image-url)

Figure 4.10. Tolerances associated with \(E1\), \(Fixt1\), and \(Part1\)
specifies_tolX(Facility, E2, Feat3, Feat6) = 0.001, specifies_tolY(Facility, E2, Feat3, Feat6) = ∞, specifies_tolZ(Facility, E2, Feat3, Feat6) = ∞.

Figure 4.11. Tolerances associated with E2, Fixt2, and Part1 for open gripper fixture

A part of type Part1 may be unloaded from fixture Fixt1 and Port1 by the robot E2 if the predicate unloadable(Part1, Port1, E2, Fixt1) is true. Given the preceding tolerances for the AS/RS E1 and the robot E2, this predicate unloadable(Part1, Port1, E2, Fixt1) can be proven through statement #2. First, from the specification of the resource model it is known that Port1 ∈ UnloadPorts2, Feat2 ∈ Feat, Feat3 ∈ Feat, Feat5 ∈ Feat, owns(Part1, Feat2), owns(Part1, Feat3), owns(Part1, Feat5), owns(Part1, Feat6), owns(E1, Fixt1), locates(Fixt1, Feat2, 1, 1), locates(Fixt1, Feat3, 1, 2), locates(Fixt1, Feat5, 1, 3), owns(E2, Fixt2), locates(Fixt2, Feat2, 1, 1), locates(Fixt2, Feat6, 1, 2), locates(Fixt2, Feat5, 1, 3), specifies(Feat5, Feat6), and unloadTrajExists(E2, Fixt2, Fixt1, Port1, Part1). This leaves nine mathematical expressions to be proven true because the robot must be capable of locating three datums of the part within the gripper’s open tolerances along the Facility’s x, y, and z coordinate axes. These expressions and their equivalent values are true as follows. For each expression, datumg is the feature located by the gripper and datumn is the feature located by the fixture.
For the first expression, the tolerance loop for $\text{datum}_g = \text{Feat}_2$ and $\text{datum}_f = \text{Feat}_2$ along the Facility element’s x axis is checked. Since neither the gripper nor the fixture specify the location of $\text{Feat}_2$ along the x axis, the expression compares equally infinite tolerances to each other and is true.

1. $\text{datum}_g = \text{Feat}_2$, $\text{datum}_f = \text{Feat}_2$ : $\text{open_tolX}(\text{Facility}, \text{E}_2, \text{Fixt}_2, \text{Feat}_2) \geq \text{tol_posX}(\text{Facility}, \text{Port}_6) + \text{accuracyX}(\text{Facility}, \text{E}_2) + \text{fixture_pos_tolX}(\text{Facility, Port}_6, \text{Fixt}_2) + \text{tol_posX}(\text{Facility, Port}_1) + \text{fixture_pos_tolX}(\text{Facility, Port}_1, \text{Fixt}_1) + \text{close_tolX}(\text{Facility, E}_1, \text{Fixt}_1, \text{Feat}_2)$
   $\Leftrightarrow \infty \geq 0.01 + 0.001 + 0.001 + 0.01 + 0.001 + \infty \Leftrightarrow \infty \geq \infty$

For the second expression, the tolerance loop for $\text{datum}_g = \text{Feat}_2$ and $\text{datum}_f = \text{Feat}_2$ along the Facility element’s y axis is checked. In this case, the gripper and fixture do specify the location of $\text{Feat}_2$ along the y axis with a finite value. Furthermore, the fixture specifies the location of $\text{Feat}_2$ sufficiently for the $\text{E}_2$ robot to locate it.

2. $\text{datum}_g = \text{Feat}_2$, $\text{datum}_f = \text{Feat}_2$ : $\text{open_tolY}(\text{Facility, E}_2, \text{Fixt}_2, \text{Feat}_2) \geq \text{tol_posY}(\text{Facility, Port}_6) + \text{accuracyY}(\text{Facility, E}_2) + \text{fixture_pos_tolY}(\text{Facility, Port}_6, \text{Fixt}_2) + \text{tol_posY}(\text{Facility, Port}_1) + \text{fixture_pos_tolY}(\text{Facility, Port}_1, \text{Fixt}_1) + \text{close_tolY}(\text{Facility, E}_1, \text{Fixt}_1, \text{Feat}_2)$
   $\Leftrightarrow 0.08 \geq 0.01 + 0.001 + 0.001 + 0.01 + 0.001 + 0.03 \Leftrightarrow 0.08 \geq 0.053$

For the third expression, the tolerance loop for $\text{datum}_g = \text{Feat}_6$ and $\text{datum}_f = \text{Feat}_2$ along the Facility element’s z axis is checked. Since neither the gripper nor the fixture specify the location of $\text{Feat}_2$ along the z axis, the expression compares equally infinite tolerances to each other and is true.

3. $\text{datum}_g = \text{Feat}_6$, $\text{datum}_f = \text{Feat}_2$ : $\text{open_tolZ}(\text{Facility, E}_2, \text{Fixt}_2, \text{Feat}_2) \geq \text{tol_posZ}(\text{Facility, Port}_6) + \text{accuracyZ}(\text{Facility, E}_2) + \text{fixture_pos_tolZ}(\text{Facility, Port}_6, \text{Fixt}_2) + \text{tol_posZ}(\text{Facility, Port}_1) + \text{fixture_pos_tolZ}(\text{Facility, Port}_1, \text{Fixt}_1) + \text{close_tolZ}(\text{Facility, E}_1, \text{Fixt}_1, \text{Feat}_2)$
   $\Leftrightarrow \infty \geq 0.01 + 0.001 + 0.001 + 0.01 + 0.001 + \infty \Leftrightarrow \infty \geq \infty$

For the fourth expression, the tolerance loop for $\text{datum}_g = \text{Feat}_6$ and $\text{datum}_f = \text{Feat}_3$ along the Facility element’s x axis is checked. In this case, the robot and the fixture do specify the location of $\text{Feat}_3$ and $\text{Feat}_6$ along the x axis with a finite value. Furthermore, the fixture specifies the location of $\text{Feat}_3$ sufficiently for the $\text{E}_2$ robot to locate $\text{Feat}_6$.

4. $\text{datum}_g = \text{Feat}_6$, $\text{datum}_f = \text{Feat}_3$ : $\text{open_tolX}(\text{Facility, E}_2, \text{Fixt}_2, \text{Feat}_2) \geq \text{tol_posX}(\text{Facility, Port}_6) + \text{accuracyX}(\text{Facility, E}_2) + \text{fixture_pos_tolX}(\text{Facility, Port}_6, \text{Fixt}_2) + \text{tol_posX}(\text{Facility, Port}_1) + \text{fixture_pos_tolX}(\text{Facility, Port}_1, \text{Fixt}_1) + \text{close_tolX}(\text{Facility, E}_1, \text{Fixt}_1, \text{Feat}_2) + \text{specifies_tolX}(\text{Facility, Feat}_3, \text{Feat}_6)$
   $\Leftrightarrow 0.08 \geq 0.01 + 0.001 + 0.001 + 0.01 + 0.001 + 0.03 + 0.001 \Leftrightarrow 0.08 \geq 0.054$

For the fifth expression, the tolerance loop for $\text{datum}_g = \text{Feat}_6$ and $\text{datum}_f = \text{Feat}_3$ along the Facility element’s y axis is checked. Since neither the gripper nor the fixture specify the location of $\text{Feat}_3$ and $\text{Feat}_6$ along the y axis, the expression compares equally infinite tolerances to each other and is true.

5. $\text{datum}_g = \text{Feat}_6$, $\text{datum}_f = \text{Feat}_3$ : $\text{open_tolY}(\text{Facility, E}_2, \text{Fixt}_2, \text{Feat}_2) \geq \text{tol_posY}(\text{Facility, Port}_6) + \text{accuracyY}(\text{Facility, E}_2) + \text{fixture_pos_tolY}(\text{Facility, Port}_6, \text{Fixt}_2) + \text{tol_posY}(\text{Facility, Port}_1) + \text{fixture_pos_tolY}(\text{Facility, Port}_1, \text{Fixt}_1) + \text{close_tolY}(\text{Facility, E}_1, \text{Fixt}_1, \text{Feat}_2) + \text{specifies_tolY}(\text{Facility, Feat}_3, \text{Feat}_6)$
   $\Leftrightarrow \infty \geq 0.01 + 0.001 + 0.001 + 0.01 + 0.001 + \infty \Leftrightarrow \infty \geq \infty$
For the sixth expression, the tolerance loop for \( \text{datum}_g = \text{Feat}_6 \) and \( \text{datum}_f = \text{Feat}_3 \) along the Facility element’s \( z \) axis is checked. Since neither the gripper nor the fixture specify the location of \( \text{Feat}_3 \) and \( \text{Feat}_6 \) along the \( z \) axis, the expression compares equally infinite tolerances to each other and is true.

\[
6. \quad \text{datum}_g = \text{Feat}_6, \quad \text{datum}_f = \text{Feat}_3 : \quad \text{open_tolZ} (\text{Facility}, \text{E}_2, \text{Fixt}_2, \text{Feat}_6) \geq \text{tol_posZ} (\text{Facility}, \text{Port}_6) + \text{accuracyZ} (\text{Facility}, \text{E}_2) + \text{fixture_pos_tolZ} (\text{Facility}, \text{Port}_6, \text{Fixt}_2) + \text{tol_posZ} (\text{Facility}, \text{Port}_1) + \text{fixture_pos_tolZ} (\text{Facility}, \text{Port}_1, \text{Fixt}_1) + \text{close_tolZ} (\text{Facility}, \text{E}_1, \text{Fixt}_1, \text{Feat}_3) + \text{specifies_tolZ} (\text{Facility}, \text{Feat}_3, \text{Feat}_6) \\
\iff \quad \infty \geq 0.01 + 0.001 + 0.01 + 0.001 + \infty + 0.001 \iff \infty \geq \infty
\]

For the seventh expression, the tolerance loop for \( \text{datum}_g = \text{Feat}_5 \) and \( \text{datum}_f = \text{Feat}_5 \) along the Facility element’s \( x \) axis is checked. Since neither the gripper nor the fixture specify the location of \( \text{Feat}_5 \) along the \( x \) axis, the expression compares equally infinite tolerances to each other and is true.

\[
7. \quad \text{datum}_g = \text{Feat}_5, \quad \text{datum}_f = \text{Feat}_5 : \quad \text{open_tolX} (\text{Facility}, \text{E}_2, \text{Fixt}_2, \text{Feat}_5) \geq \text{tol_posX} (\text{Facility}, \text{Port}_6) + \text{accuracyX} (\text{Facility}, \text{E}_2) + \text{fixture_pos_tolX} (\text{Facility}, \text{Port}_6, \text{Fixt}_2) + \text{tol_posX} (\text{Facility}, \text{Port}_1) + \text{fixture_pos_tolX} (\text{Facility}, \text{Port}_1, \text{Fixt}_1) + \text{close_tolX} (\text{Facility}, \text{E}_1, \text{Fixt}_1, \text{Feat}_5) \\
\iff \quad \infty \geq 0.01 + 0.001 + 0.001 + 0.01 + 0.001 + \infty \iff \infty \geq \infty
\]

For the eighth expression, the tolerance loop for \( \text{datum}_g = \text{Feat}_5 \) and \( \text{datum}_f = \text{Feat}_5 \) along the Facility element’s \( y \) axis is checked. Since neither the gripper nor the fixture specify the location of \( \text{Feat}_5 \) along the \( y \) axis, the expression compares equally infinite tolerances to each other and is true.

\[
8. \quad \text{datum}_g = \text{Feat}_5, \quad \text{datum}_f = \text{Feat}_5 : \quad \text{open_tolY} (\text{Facility}, \text{E}_2, \text{Fixt}_2, \text{Feat}_5) \geq \text{tol_posY} (\text{Facility}, \text{Port}_6) + \text{accuracyY} (\text{Facility}, \text{E}_2) + \text{fixture_pos_tolY} (\text{Facility}, \text{Port}_6, \text{Fixt}_2) + \text{tol_posY} (\text{Facility}, \text{Port}_1) + \text{fixture_pos_tolY} (\text{Facility}, \text{Port}_1, \text{Fixt}_1) + \text{close_tolY} (\text{Facility}, \text{E}_1, \text{Fixt}_1, \text{Feat}_5) \\
\iff \quad \infty \geq 0.01 + 0.001 + 0.001 + 0.01 + 0.001 + \infty \iff \infty \geq \infty
\]

For the ninth expression, the tolerance loop for \( \text{datum}_g = \text{Feat}_5 \) and \( \text{datum}_f = \text{Feat}_5 \) along the Facility element’s \( z \) axis is checked. In this case, the gripper and fixture do specify the location of \( \text{Feat}_5 \) along the \( z \) axis with a finite value. Furthermore, the fixture specifies the location of \( \text{Feat}_5 \) sufficiently for the \( \text{E}_2 \) robot to locate \( \text{Feat}_5 \).

\[
9. \quad \text{datum}_g = \text{Feat}_5, \quad \text{datum}_f = \text{Feat}_5 : \quad \text{open_tolZ} (\text{Facility}, \text{E}_2, \text{Fixt}_2, \text{Feat}_5) \geq \text{tol_posZ} (\text{Facility}, \text{Port}_6) + \text{accuracyZ} (\text{Facility}, \text{E}_2) + \text{fixture_pos_tolZ} (\text{Facility}, \text{Port}_6, \text{Fixt}_2) + \text{tol_posZ} (\text{Facility}, \text{Port}_1) + \text{fixture_pos_tolZ} (\text{Facility}, \text{Port}_1, \text{Fixt}_1) + \text{close_tolZ} (\text{Facility}, \text{E}_1, \text{Fixt}_1, \text{Feat}_5) \\
\iff \quad 0.08 \geq 0.01 + 0.001 + 0.001 + 0.01 + 0.001 + 0.03 \iff 0.08 \geq 0.053
\]

A part of type \( \text{Part}_1 \) may be loaded into fixture \( \text{Fixt}_1 \) and port \( \text{Port}_1 \) by the robot \( \text{E}_2 \) if the predicate \( \text{loadable} (\text{Part}_1, \text{Port}_1, \text{E}_2, \text{Fixt}_1) \) is true. The predicate \( \text{loadable} (\text{Part}_1, \text{Port}_1, \text{E}_2, \text{Fixt}_1) \) is true if the antecedents of statement \#3 are true. From the specification of the resource model, it is known that \( \text{Port}_1 \in \text{LoadPorts}_2, \text{Feat}_2 \in \text{Feat}, \text{Feat}_3 \in \text{Feat}, \text{Feat}_5 \in \text{Feat}, \text{Feat}_6 \in \text{Feat}, \text{owns} (\text{Part}_1, \text{Feat}_2), \text{owns} (\text{Part}_1, \text{Feat}_3), \text{owns} (\text{Part}_1, \text{Feat}_5), \text{owns} (\text{Part}_1, \text{Feat}_6), \text{owns} (\text{E}_2, \text{Fixt}_2), \text{locates} (\text{Fixt}_1, \text{Feat}_1, 1, 1), \text{locates} (\text{Fixt}_1, \text{Feat}_3, 1, 2), \text{locates} (\text{Fixt}_1, \text{Feat}_5, 1, 3), \text{locates}
(Fixt₂, Feat₂, 1, 1), *locates* (Fixt₂, Feat₆, 1, 2), *locates* (Fixt₂, Feat₅, 1, 3), *specifies* (Feat₅, Feat₆), and
*loadTrajExists* (E₂, Fixt₂, Fixt₁, Port₁, Part₁). This leaves nine mathematical expressions to be proven true because
the robot must be capable of locating three datums of the part within the equipment fixture’s open tolerances
along the Facility element’s x, y, and z coordinate axes. These expressions and their equivalent values are true as
follows.

For the first expression, the tolerance loop for datum₉ = Feat₂ and datum₁₇ = Feat₂ along the Facility element’s
x axis is checked. Since neither the gripper nor the fixture specify the location of Feat₂ along the x axis, the
expression compares equally infinite tolerances to each other and is true.

1. \[ \text{datum}_9 = \text{Feat}_2, \text{datum}_{17} = \text{Feat}_2 : \text{open}_X (\text{Facility}, E_1, \text{Fixt}_1, \text{Feat}_2) \geq \text{tol}_X (\text{Facility}, \text{Port}_6) + \text{accuracy}_X (\text{Facility}, E_2) + \text{fixture}_X (\text{Facility}, \text{Port}_1, \text{Fixt}_1) + \text{close}_X (\text{Facility}, E_2, \text{Fixt}_2, \text{Feat}_2) \]
\[ \Leftrightarrow \infty \geq 0.01 + 0.001 + 0.001 + 0.01 + 0.001 + \infty \Leftrightarrow \infty \geq \infty \]

For the second expression, the tolerance loop for datum₉ = Feat₂ and datum₁₇ = Feat₂ along the Facility
element’s y axis is checked. In this case, the gripper and fixture do specify the location of Feat₂ along the y axis
with a finite value. Furthermore, the fixture specifies the location of Feat₂ sufficiently for the E₂ robot to locate
Feat₂.

2. \[ \text{datum}_9 = \text{Feat}_2, \text{datum}_{17} = \text{Feat}_2 : \text{open}_Y (\text{Facility}, E_1, \text{Fixt}_1, \text{Feat}_2) \geq \text{tol}_Y (\text{Facility}, \text{Port}_6) + \text{accuracy}_Y (\text{Facility}, E_2) + \text{fixture}_Y (\text{Facility}, \text{Port}_1, \text{Fixt}_1) + \text{close}_Y (\text{Facility}, E_2, \text{Fixt}_2, \text{Feat}_2) \]
\[ \Leftrightarrow 0.03 \geq 0.01 + 0.001 + 0.001 + 0.01 + 0.001 + 0.001 \Leftrightarrow 0.03 \geq 0.023 \]

For the third expression, the tolerance loop for datum₉ = Feat₂ and datum₁₇ = Feat₂ along the Facility
element’s z axis is checked. Since neither the gripper nor the fixture specify the location of Feat₂ along the z axis,
the expression compares equally infinite tolerances to each other and is true.

3. \[ \text{datum}_9 = \text{Feat}_2, \text{datum}_{17} = \text{Feat}_2 : \text{open}_Z (\text{Facility}, E_1, \text{Fixt}_1, \text{Feat}_2) \geq \text{tol}_Z (\text{Facility}, \text{Port}_6) + \text{accuracy}_Z (\text{Facility}, E_2) + \text{fixture}_Z (\text{Facility}, \text{Port}_1, \text{Fixt}_1) + \text{close}_Z (\text{Facility}, E_2, \text{Fixt}_2, \text{Feat}_2) \]
\[ \Leftrightarrow \infty \geq 0.01 + 0.001 + 0.001 + 0.01 + 0.001 + \infty \Leftrightarrow \infty \geq \infty \]

For the fourth expression, the tolerance loop for datum₉ = Feat₆ and datum₁₇ = Feat₃ along the Facility
element’s x axis is checked. In this case, the gripper specifies the location of Feat₆ along the x axis with a finite
value. Furthermore, the fixture specifies the location of Feat₃ sufficiently for the E₂ robot to locate Feat₆.
4. \( \text{datum}_g = \text{Feat}_6, \text{datum}_f = \text{Feat}_3 : \text{open}_X \geq \text{tol}_X + \text{accuracy}_X + \text{close}_X + \text{specifies}_X \equiv 0.03 \geq 0.024 \)

For the fifth expression, the tolerance loop for \( \text{datum}_g = \text{Feat}_6 \) and \( \text{datum}_f = \text{Feat}_3 \) along the Facility element’s y axis is checked. Since neither the gripper nor the fixture specify the location of \( \text{Feat}_6 \) or \( \text{Feat}_3 \) along the y axis, the expression compares equally infinite tolerances to each other and is true.

5. \( \text{datum}_g = \text{Feat}_6, \text{datum}_f = \text{Feat}_3 : \text{open}_Y \geq \text{tol}_Y + \text{accuracy}_Y + \text{close}_Y + \text{specifies}_Y \equiv \infty \geq \infty \)

For the sixth expression, the tolerance loop for \( \text{datum}_g = \text{Feat}_6 \) and \( \text{datum}_f = \text{Feat}_3 \) along the Facility element’s z axis is checked. Since neither the gripper nor the fixture specify the location of \( \text{Feat}_6 \) or \( \text{Feat}_3 \) along the z axis, the expression compares equally infinite tolerances to each other and is true.

6. \( \text{datum}_g = \text{Feat}_6, \text{datum}_f = \text{Feat}_3 : \text{open}_Z \geq \text{tol}_Z + \text{accuracy}_Z + \text{close}_Z + \text{specifies}_Z \equiv \infty \geq \infty \)

For the seventh expression, the tolerance loop for \( \text{datum}_g = \text{Feat}_6 \) and \( \text{datum}_f = \text{Feat}_3 \) along the Facility element’s x axis is checked. Since neither the gripper nor the fixture specify the location of \( \text{Feat}_6 \) or \( \text{Feat}_3 \) along the x axis, the expression compares equally infinite tolerances to each other and is true.

7. \( \text{datum}_g = \text{Feat}_6, \text{datum}_f = \text{Feat}_3 : \text{open}_X \geq \text{tol}_X + \text{accuracy}_X + \text{close}_X + \text{specifies}_X \equiv \infty \geq \infty \)

For the eighth expression, the tolerance loop for \( \text{datum}_g = \text{Feat}_6 \) and \( \text{datum}_f = \text{Feat}_3 \) along the Facility element’s y axis is checked. Since neither the gripper nor the fixture specify the location of \( \text{Feat}_6 \) along the y axis, the expression compares equally infinite tolerances to each other and is true.

8. \( \text{datum}_g = \text{Feat}_6, \text{datum}_f = \text{Feat}_3 : \text{open}_Y \geq \text{tol}_Y + \text{accuracy}_Y + \text{close}_Y + \text{specifies}_Y \equiv \infty \geq \infty \)

For the ninth expression, the tolerance loop for \( \text{datum}_g = \text{Feat}_6 \) and \( \text{datum}_f = \text{Feat}_3 \) along the Facility element’s z axis is checked. In this case, the gripper specifies the location of \( \text{Feat}_6 \) along the z axis with a finite value. Furthermore, the fixture specifies the location of \( \text{Feat}_3 \) sufficiently for the \( \text{E}_2 \) robot to locate \( \text{Feat}_3 \).
9. \( \text{datum}_g = \text{Feat}_5, \text{datum}_f = \text{Feat}_4: \ \ open\_tolZ(\text{Facility}, \text{E}_1, \text{Fixt}_1, \text{Feat}_5) \geq \text{tol}_pZ(\text{Facility}, \text{Port}_6) + \text{accuracy}_{Z}(\text{Facility}, \text{Port}_1, \text{Fixt}_1) + \text{close\_tolZ}(\text{Facility}, \text{E}_2, \text{Fixt}_5, \text{Feat}_5) \)

\( \Leftrightarrow 0.03 \geq 0.01 + 0.001 + 0.01 + 0.001 + 0.001 \Leftrightarrow 0.03 \geq 0.023 \)

### 4.5.3 Material Processing Operation

Based on the information describing the vertical mill \( \text{E}_3 \) setup for making \( \text{Feat}_7 \) in Figure 4.12, the following functions are defined.

\[
\begin{align*}
\text{accuracy}_{X}(\text{E}_3, \text{T}_1, \text{Feat}_7) &= 0.001, \\
\text{accuracy}_{Y}(\text{E}_3, \text{T}_1, \text{Feat}_7) &= 0.001, \\
\text{accuracy}_{Z}(\text{E}_3, \text{T}_1, \text{Feat}_7) &= 0.001, \\
\text{accuracy}_{X}(\text{E}_3) &= 0.001, \\
\text{accuracy}_{Y}(\text{E}_3) &= 0.001, \\
\text{accuracy}_{Z}(\text{E}_3) &= 0.001, \\
\text{fixture\_pos\_tol}_{X}(\text{Port}_2, \text{Fixt}_3) &= 0.001, \\
\text{fixture\_pos\_tol}_{Y}(\text{Port}_2, \text{Fixt}_3) &= 0.001, \\
\text{fixture\_pos\_tol}_{Z}(\text{Port}_2, \text{Fixt}_3) &= 0.001, \\
\text{open\_tol}_{X}(\text{E}_3, \text{Fixt}_3, \text{Feat}_3) &= 0.03, \\
\text{open\_tol}_{Y}(\text{E}_3, \text{Fixt}_3, \text{Feat}_3) &= \infty, \\
\text{open\_tol}_{Z}(\text{E}_3, \text{Fixt}_3, \text{Feat}_3) &= \infty, \\
\text{close\_tol}_{X}(\text{E}_3, \text{Fixt}_3, \text{Feat}_3) &= 0.001, \\
\text{close\_tol}_{Y}(\text{E}_3, \text{Fixt}_3, \text{Feat}_3) &= \infty, \\
\text{close\_tol}_{Z}(\text{E}_3, \text{Fixt}_3, \text{Feat}_3) &= \infty, \\
\text{open\_tol}_{X}(\text{E}_3, \text{Fixt}_3, \text{Feat}_2) &= \infty, \\
\text{open\_tol}_{Y}(\text{E}_3, \text{Fixt}_3, \text{Feat}_2) &= 0.03, \\
\text{open\_tol}_{Z}(\text{E}_3, \text{Fixt}_3, \text{Feat}_2) &= \infty, \\
\text{close\_tol}_{X}(\text{E}_3, \text{Fixt}_3, \text{Feat}_2) &= \infty, \\
\text{close\_tol}_{Y}(\text{E}_3, \text{Fixt}_3, \text{Feat}_2) &= \infty, \\
\text{close\_tol}_{Z}(\text{E}_3, \text{Fixt}_3, \text{Feat}_2) &= \infty, \\
\text{specifies\_tol}_{X}(\text{Feat}_2, \text{Feat}_7) &= 0.001, \\
\text{specifies\_tol}_{Y}(\text{Feat}_2, \text{Feat}_7) &= \infty, \\
\text{specifies\_tol}_{Z}(\text{Feat}_2, \text{Feat}_7) &= \infty, \\
\text{specifies\_tol}_{X}(\text{Feat}_3, \text{Feat}_7) &= \infty, \\
\text{specifies\_tol}_{Y}(\text{Feat}_3, \text{Feat}_7) &= \infty, \\
\text{specifies\_tol}_{Z}(\text{Feat}_3, \text{Feat}_7) &= \infty.
\end{align*}
\]

![Figure 4.12. Tolerance Functions Associated with Making Feat\_7 with E\_3 along X\_E\_3](image)

The feature \( \text{Feat}_7 \) on \( \text{Part}_1 \) may be created by the vertical mill \( \text{E}_3 \) if the predicate \( \text{makes}(\text{E}_3, \text{Feat}_7, \text{Fixt}_5) \) is true. First, the predicate \( \text{makes}(\text{E}_3, \text{Feat}_7, \text{Fixt}_5) \) is true if the antecedents of statement #1 from Section 4.3.4 are true. From the specification of the resource model, it is known that the following predicates are true: \( \text{owns}(\text{E}_3, \text{T}_1), \text{makes}(\text{T}_1, \text{Feat}_5), \text{owns}(\text{E}_3, \text{Fixt}_5), \text{specifies}(\text{Feat}_2, \text{Feat}_7), \text{specifies}(\text{Feat}_3, \text{Feat}_7), \text{locates}(\text{Fixt}_5, \text{Feat}_5, 1, 1) \),

\[63\]
locates $\text{Fixt}_3$, $\text{Feat}_5$, $1$, $2$), and locates $\text{Fixt}_3$, $\text{Feat}_5$, $1$, $3$). This leaves nine mathematical expressions to be proven true because the vertical mill must be capable of making $\text{Feat}_7$ within tolerances relative to datums $\text{Feat}_2$, $\text{Feat}_3$, $\text{Feat}_5$ for the x, y, and z equipment coordinate axes. These expressions and their equivalent values are true based on the following descriptions.

In the first expression, the tolerance loop between the capability of $\text{E}_3$ to create a new feature $\text{Feat}_7$ with a given tolerance and the feature’s specified tolerance is checked based on $\text{Fixt}_3$’s specification of $\text{Feat}_2$’s location and $\text{Feat}_3$’s specification of $\text{Feat}_7$ along $\text{E}_3$’s x axis. Since $\text{Feat}_2$ does not specify $\text{Feat}_7$ along the x axis, both tolerances are infinite and equal.

1. $\text{specifies_{tolX}}(\text{E}_3, \text{Feat}_2, \text{Feat}_7) \geq \text{accuracyX}(\text{E}_3) + \text{accuracyX}(\text{E}_3, \text{T}_1, \text{Feat}_7) + \text{fixture_pos_{tolX}}(\text{E}_3, \text{Port}_2, \text{Fixt}_3) + \text{close_{tolX}}(\text{E}_3, \text{Fixt}_3, \text{Feat}_2)$
   $\Leftrightarrow \infty \geq 0.001 + 0.011 + 0.001 + \infty \Leftrightarrow \infty \geq \infty$

In the second expression, the tolerance loop between the capability of $\text{E}_3$ to create a new feature $\text{Feat}_7$ with a given tolerance and the feature’s specified tolerance is checked based on $\text{Fixt}_3$’s specification of $\text{Feat}_2$’s location and $\text{Feat}_3$’s specification of $\text{Feat}_7$ along $\text{E}_3$’s y axis. Since $\text{Feat}_2$ specifies $\text{Feat}_7$ along the y axis, and $\text{E}_3$ can create $\text{Feat}_7$ with sufficient accuracy relative to $\text{Feat}_2$ along the y axis, this expression is true.

2. $\text{specifies_{tolY}}(\text{E}_3, \text{Feat}_2, \text{Feat}_7) \geq \text{accuracyY}(\text{E}_3) + \text{accuracyY}(\text{E}_3, \text{T}_1, \text{Feat}_7) + \text{fixture_pos_{tolY}}(\text{E}_3, \text{Port}_2, \text{Fixt}_3) + \text{close_{tolY}}(\text{E}_3, \text{Fixt}_3, \text{Feat}_2)$
   $\Leftrightarrow 0.025 \geq 0.001 + 0.01 + 0.001 + 0.001 \Leftrightarrow 0.025 \geq 0.013$

In the third expression, the tolerance loop between the capability of $\text{E}_3$ to create a new feature $\text{Feat}_7$ with a given tolerance and the feature’s specified tolerance is checked based on $\text{Fixt}_3$’s specification of $\text{Feat}_2$’s location and $\text{Feat}_3$’s specification of $\text{Feat}_7$ along $\text{E}_3$’s z axis. Since $\text{Feat}_2$ does not specify $\text{Feat}_7$ along the z axis, both tolerances are infinite and equal.

3. $\text{specifies_{tolZ}}(\text{E}_3, \text{Feat}_2, \text{Feat}_7) \geq \text{accuracyZ}(\text{E}_3) + \text{accuracyZ}(\text{E}_3, \text{T}_1, \text{Feat}_7) + \text{fixture_pos_{tolZ}}(\text{E}_3, \text{Port}_2, \text{Fixt}_3) + \text{close_{tolZ}}(\text{E}_3, \text{Fixt}_3, \text{Feat}_2)$
   $\Leftrightarrow \infty \geq 0.001 + 0.011 + 0.001 + \infty \Leftrightarrow \infty \geq \infty$

In the fourth expression, the tolerance loop between the capability of $\text{E}_3$ to create a new feature $\text{Feat}_7$ with a given tolerance and the feature’s specified tolerance is checked based on $\text{Fixt}_3$’s specification of $\text{Feat}_2$’s location and $\text{Feat}_3$’s specification of $\text{Feat}_7$ along $\text{E}_3$’s x axis. Since $\text{Feat}_3$ specifies $\text{Feat}_7$ along the x axis, and $\text{E}_3$ can create $\text{Feat}_7$ with sufficient accuracy relative to $\text{Feat}_3$ along the x axis, this expression is true.
4. $\text{specifies_tol}\_X(E_3, \text{Feat}_5, \text{Feat}_7) \geq \text{accuracy}\_X(E_3) + \text{accuracy}\_X(E_3, T_1, \text{Feat}_7) + \text{fixture_pos_tol}\_X(E_3, \text{Port}_2, \text{Fixt}_3) + \text{close_tol}\_X(E_3, \text{Fixt}_3, \text{Feat}_7)$
\[\Leftrightarrow 0.025 \geq 0.001 + 0.01 + 0.001 + 0.001 \Leftrightarrow 0.025 \geq 0.013\]

In the fifth expression, the tolerance loop between the capability of E_3 to create a new feature Feat_7 with a given tolerance and the feature’s specified tolerance is checked based on Fixt_3’s specification of Feat_7’s location and Feat_3’s specification of Feat_7 along E_3’s y axis. Since Feat_5 does not specify Feat_7 and Fixt_3 does not specify Feat_3 along the y axis, both tolerances are infinite and equal.

5. $\text{specifies_tol}\_Y(E_3, \text{Feat}_5, \text{Feat}_7) \geq \text{accuracy}\_Y(E_3) + \text{accuracy}\_Y(E_3, T_1, \text{Feat}_7) + \text{fixture_pos_tol}\_Y(E_3, \text{Port}_2, \text{Fixt}_3) + \text{close_tol}\_Y(E_3, \text{Fixt}_3, \text{Feat}_7)$
\[\Leftrightarrow \infty \geq 0.001 + 0.01 + 0.001 + \infty \Leftrightarrow \infty \geq \infty\]

In the sixth expression, the tolerance loop between the capability of E_3 to create a new feature Feat_7 with a given tolerance and the feature’s specified tolerance is checked based on Fixt_3’s specification of Feat_7’s location and Feat_3’s specification of Feat_7 along E_3’s z axis. Since Feat_5 does not specify Feat_7 and Fixt_3 does not specify Feat_3 along the z axis, both tolerances are infinite and equal.

6. $\text{specifies_tol}\_Z(E_3, \text{Feat}_5, \text{Feat}_7) \geq \text{accuracy}\_Z(E_3) + \text{accuracy}\_Z(E_3, T_1, \text{Feat}_7) + \text{fixture_pos_tol}\_Z(E_3, \text{Port}_2, \text{Fixt}_3) + \text{close_tol}\_Z(E_3, \text{Fixt}_3, \text{Feat}_7)$
\[\Leftrightarrow \infty \geq 0.001 + 0.01 + 0.001 + \infty \Leftrightarrow \infty \geq \infty\]

In the seventh expression, the tolerance loop between the capability of E_3 to create a new feature Feat_7 with a given tolerance and the feature’s specified tolerance is checked based on Fixt_3’s specification of Feat_7’s location and Feat_3’s specification of Feat_7 along E_3’s x axis. Since Feat_5 does not specify Feat_7 and Fixt_3 does not specify Feat_3 along the x axis, both tolerances are infinite and equal.

7. $\text{specifies_tol}\_X(E_3, \text{Feat}_5, \text{Feat}_7) \geq \text{accuracy}\_X(E_3) + \text{accuracy}\_X(E_3, T_1, \text{Feat}_7) + \text{fixture_pos_tol}\_X(E_3, \text{Port}_2, \text{Fixt}_3) + \text{close_tol}\_X(E_3, \text{Fixt}_3, \text{Feat}_7)$
\[\Leftrightarrow \infty \geq 0.001 + 0.01 + 0.001 + \infty \Leftrightarrow \infty \geq \infty\]

In the eighth expression, the tolerance loop between the capability of E_3 to create a new feature Feat_7 with a given tolerance and the feature’s specified tolerance is checked based on Fixt_3’s specification of Feat_7’s location and Feat_3’s specification of Feat_7 along E_3’s y axis. Since Feat_5 does not specify Feat_7 and Fixt_3 does not specify Feat_3 along the y axis, both tolerances are infinite and equal.

8. $\text{specifies_tol}\_Y(E_3, \text{Feat}_5, \text{Feat}_7) \geq \text{accuracy}\_Y(E_3) + \text{accuracy}\_Y(E_3, T_1, \text{Feat}_7) + \text{fixture_pos_tol}\_Y(E_3, \text{Port}_2, \text{Fixt}_3) + \text{close_tol}\_Y(E_3, \text{Fixt}_3, \text{Feat}_7)$
\[\Leftrightarrow \infty \geq 0.001 + 0.01 + 0.001 + \infty \Leftrightarrow \infty \geq \infty\]

In the ninth expression, the tolerance loop between the capability of E_3 to create a new feature Feat_7 with a given tolerance and the feature’s specified tolerance is checked based on Fixt_3’s specification of Feat_7’s location
and Feat₅’s specification of Feat₇ along E₃’s z axis. Since Feat₅ specifies Feat₇ along the z axis, and E₃ can create Feat₇ with sufficient accuracy relative to Feat₅ along the z axis, this expression is true.

9. \[ \text{specifies}_{\text{tolZ}}(E₃, \text{Feat}_₅, \text{Feat}_7) \geq \text{accuracy}_Z(E₃) + \text{accuracy}_Z(E₃, \text{T}_1, \text{Feat}_7) + \text{fixture_pos_tolZ}(E₃, \text{Port}_2, \text{Fixt}_3) + \text{close_tolZ}(E₃, \text{Fixt}_3, \text{Feat}_₅) \] \[ \Leftrightarrow 0.025 \geq 0.001 + 0.01 + 0.001 + 0.001 \Leftrightarrow 0.025 \geq 0.013 \]

### 4.6 Chapter Summary

This chapter has presented a formal specification of the automatic generation of operations plans for flexible manufacturing facilities. In order to specify the desired mapping from the input domain for manufacturing operations planning to the output domain, both domains were formally defined. The input domain was defined as a combination of a model of the facility’s resources and models of parts that are manufacturable by those resources. These models were defined using first-order set theory. Furthermore, predicates and functions were defined that reflect the capabilities of resources to do different operations on the parts. These predicates and functions are listed in Appendix C. This input data defines the information that must be defined to plan manufacturing operations. The output domain was defined as a nonlinear sequence of operations that transform parts from an initial state to a final state describing the finished product. The operations data defines what the operations plan specifies and how it can provide data for diverse applications such as simulation, planning, scheduling, and execution. The desired mapping from the input to the output domain was given by a set of conditions for the sequence of operations. If this set of conditions is logically consistent with the operations and the resource and part models, then the mapping is correct. This formal specification creates an unambiguous description of the operations planner’s inputs and outputs. The unambiguous description of the resource model enables other information sources such as vendors and researchers to contribute to the resource data and behaviors. The unambiguous description of the operations plan allows diverse applications such as simulation models and production controllers to utilize the data describing individual operations and their sequences.
Chapter 5

OPERATIONS PLAN GENERATION METHODOLOGY

The previous chapter formally specified what results are expected from manufacturing operations planning. This chapter presents how manufacturing operations planning automatically generates these results using a new approach that integrates material processing with material handling. This approach borrows from artificial intelligence techniques to solve the problem in an original and useful way. Section 5.1 provides an overview of the different components of this solution to the planning task. Section 5.2 presents the high level planning methodology developed through this research. Section 5.3 describes the lower level geometric planning techniques that are called by the high level planner to check detailed geometric and tolerance constraints for material handling and processing. Lastly, Section 5.4 describes how the results of the planner are converted into a compact and usable description of the sequences of operations.

5.1 Introduction

Chapter 4 showed that an operations plan is only valid for a given facility and a part if all of the individual operations satisfy a set of criteria, and if every linear sequence of operations in the plan results in the final state of the part. The operations planning task is divided into high level planning, geometric tolerance planning, and kinematic trajectory planning. In this definition, high level planning generates a set of valid sequences of operations by reasoning about the discrete variables that are defined in the part’s state space. These variables describe information such as the port and fixture where the part is located in addition to the features added to the part. High level planning also reasons about individual operations using the set of logical validity checks for individual operations given in Chapter 4. For example, such reasoning includes verifying that an MP entity owns the tool that performs an MP operation on a part located in the equipment. A geometric tolerance planner checks to see if the geometric constraints to high level descriptions of operations can be satisfied. These geometric constraints were described in Chapter 4 as different sets of mathematical expressions that must be true in order for different types of operations to be valid. For MP operations, these geometric constraints include matching the
required tolerances of manufacturing features against the material processor equipment’s capability. For MH operations, these geometric constraints include matching the required tolerances of the part placement in fixtures against the capabilities of MH equipment to position the part in fixtures. MH geometric planning also includes matching the required position of the gripper to pick or place parts with the kinematic reach of the MH robot using a closed-form inverse kinematics algorithm for the robot, if such an algorithm exists. The last constraint on MH operations is that there must exist a collision-free trajectory from the home position of MH robots to unload or load positions of the part in fixtures and back to the home position again. This final constraint is described by the predicates \textit{unloadTrajExists} and \textit{loadTrajExists}. Due to the complexity of this trajectory constraint, trajectory planning will be discussed in Chapter 6.

5.2 High Level Operations planning

5.2.1 Introduction

The problem reduction algorithm, which was inspired by Newell and Simon (1963) and formalized by Nilsson (1998), automatically solves problems by reducing them into sub-problems that are easier to solve. This approach is well suited for discrete part manufacturing operations planning. Using the state space description, the solution to problems consists of finding the appropriate operators to change the state from the initial to goal state. In the discrete part manufacturing domain, part goal features and part locations are sub-goals and the planner searches for operators from the resource model that can change the part state into another state that satisfies these sub-goals. In general, a problem description consists of \{S, F, G\}, where S is the current state, F is a set of valid operators, and G is the goal state. The typical state space search solution consists of searching through a search tree by blindly applying the operators wherever possible until a set of correct sequences of operations is found that transform the state S into state G. However, this search process becomes simpler when the problem is decomposed into sub-problems. The sub-problems in turn become decomposed into sub-sub-problems until problems can be solved by a single operation. This problem decomposition is accomplished by providing procedures in the operators that not only determine the operator’s effect on the part state but also determine the preconditions for using the operator.
To illustrate this problem-reducing, backward-chaining approach, Figure 5.1 shows the initial decomposition of a problem given by \{S, F, G\}. In this terminology, \( f1(S) \) represents the resultant state after applying operator \( f1 \) to state \( S \) while \( S_{f1} \) represents the precondition state for operator \( f1 \). From the list of operators given by \( F, f1 \) and \( f2 \) are both known to eliminate some portion of the difference between the current state, \( S \), and the goal state, \( G \).

Thus, the search graph expands into two independent branches separated by an OR node. One possible solution presented by \( f1 \) is the left branch. This branch is further decomposed into two sub-problems. The left sub-problem is represented by \{S, F, S_{f1}\}, which is the problem of eliminating the difference between the current state, \( S \), and the state that is a precondition to using operator \( f1 \), \( S_{f1} \). The right sub-problem is represented by \{\( f1(S) \), F, G\}, which is the problem of eliminating the difference between the state that results as a consequence of applying \( f1 \) to \( S \) and the final goal state \( G \). These sub-problems are solved when the current state of the sub-problem is equal to the goal state. For example, in the right sub-problem, if \( f1(S) = G \). The two sub-problems on the left branch are connected by a sequence node that represents the knowledge that if these sub-problems are eventually solved, then the operators resulting from the solutions of the left sub-problem and its descendents that solve the precondition to the right sub-problem’s solution must be applied first, and then the operators resulting from the solutions of the right sub-problem and its descendents (including \( f1 \)) can be applied. In general, the search graph may be expanded in a breadth-first or depth-first order using standard algorithms outlined in Nilsson (1998). Nodes that are solved are not expanded any further while unsolved nodes without any solvable ancestors are marked as unsolvable. Since the goal of this research is to discover all non-cyclic solutions to the problem, this search graph is exhaustively explored.

![Figure 5.1 Illustration of OR search graph generation](image-url)
5.2.2 Generation of Operations Search Graph Without Alternatives

Applying this problem reduction technique to the discrete part manufacturing domain, the part state is defined to be a non-ordered set of features describing the geometry of the part, the current port the part is located at, the fixture that locates the part, and the fixture intention. Thus, the starting state of a part is the set of features defining the raw material, the port corresponding to the raw materials buffer storage entity, the fixture locating the part at the buffer, and a fixture intention that locates at least one part datum. The goal state of the part is obtained from the part model. This goal state is defined by the part’s raw material features plus a partially ordered set of features derived from the AND/OR graph of the part’s goal features, the port corresponding to the finished materials buffer storage entity, the fixture at the buffer that locates the part, and a fixture intent that locates at least one part datum. For a given part, the fixture intent determines the orientation of the part in the fixture. In general, the part state, \( S \), consists of the following set:

\[
S = \{ \text{FEAT}, P, \text{Fixt}, \text{FixtIntent} \},
\]

where \( \text{FEAT} = \{ \text{Feat1}, \text{Feat2}, \text{Feat3}, \ldots \} \), \( P = \) port, \( \text{Fixt} = \) fixture holding the part at \( P \), and \( \text{FixtIntent} = \) fixture intention representing the set of part datums located by fixture for this state.

The set \( \text{FEAT} \) in the part state is composed of a set of features that represent the current geometry of the part. For convenience, the features that belong to the raw material of the part will be omitted from the set \( \text{FEAT} \) in the subsequent discussion. This \( \text{FEAT} \) set is not ordered because the features in this set have no precedence once they are added to the part. A part state satisfies the part’s goal features if all of the goal features in some linear sequence from the AND/OR graph of the part’s goal features are elements of this \( \text{FEAT} \) set. If the difference between the start state and goal state is two features without precedence, then the operations planner will generate a search sub-graph with at least two solutions separated by an OR branch representing operations that produce each distinct feature. Each alternative branch defines the task of creating both features but the order of execution is different. For instance, if features \( \text{A} \) and \( \text{B} \) are in the goal state without any precedence and not in the current state, one generated OR branch begins by solving feature \( \text{A} \) first and then solves \( \text{B} \) while the other OR branch begins by solving feature \( \text{B} \) first and then solves \( \text{A} \). In general, as goal features are solved and added to the part state, the operations planner uses the goal feature precedence to add more unsolved features to the \( \text{FEAT} \) set from the AND/OR graph of the part’s goal features.
Other information in the part state includes the port the part is located at, the fixture that locates the part, and the specific intention of the fixture to locate datums of the part. Thus, the difference between the goal state and current part state may consist of the port, fixture, or fixture intention representing the orientation of the part in the fixture. Differences between the ports may be solved using MH or MT operations to move the part to a different port. Differences between the fixtures or fixture intention of the part are typically solved using MH operations to remove the part and place it back in a different fixture or with a different orientation.

A significant omission with the problem reduction technique is that there is no automatic precedence between different sub-goals in a goal. Thus, this planning strategy defines the precedence between sub-goals in the manufacturing problem domain. In the manufacturing problem domain, precedence between goal features is explicitly defined with an AND/OR feature graph. Precedence between the sub-goals of features, port, fixture, and fixture intent is not explicitly defined in the part model. However, physical reality dictates that the features must be solved first, then the difference between current and goal ports is solved, and then the difference in fixtures or part orientation in the same fixture is solved. Thus, for a given problem, the difference between current and goal ports can only be solved when all features are solved. Similarly, for another problem, the difference between current and goal fixture intentions can only be solved when the current port is equal to the goal port. This strategy allows operations plans that result from this planning strategy to be valid with respect to physical reality. For example, the initial problem statement for a part has differences between the current and goal state that consist of the goal features, the difference between the current and goal ports, the difference between part fixtures, and the difference between fixture intentions. It is nonsensical to begin solving the problem of getting the part from the raw material buffer to the finished parts buffer before the part is finished. Furthermore, the solution for changing the port of the part is dependent on the material state of the part. A robotic material handler may have to grip a part differently after features have been added to a part so the solution is completely different. Lastly, problems arising from differences between current and goal fixtures or fixture intents are only solved after differences between current and goal features and ports have been solved because the goal state fixture and fixture intent are intended to be at the goal port. It is not reasonable to plan how to move the part to a fixture in a port where the fixture will not be located.
The operations planner uses these sub-goal precedence rules in addition to key operators drawn only from resources available to the facility. Sometimes it may be desired to use the results of an operations planner to determine the benefits of procuring additional resources for a given facility. In this case, abstract resource classes would be first considered, and then specific equipment from vendors are considered for procurement. Although there is a drawback to the operations planning approach developed for this dissertation because the planner does not generate plans for abstract classes of resources, there is no difficulty in adding fictional resources belonging to these classes to the resource model database and using this operations planner to determine the benefits of these resources. Currently, the operations planner uses operators defined in behaviors in MP resource elements and their tooling elements to add missing goal part features to the part state. Operators are also defined in behaviors in MH and MT resource elements that move the part closer to the goal port or change the orientation of the part in a buffer. These operators correspond to operations that are formally defined in Chapter 4. These operators may only be applied to a problem solution if each operation that matches an operator is valid for the given facility.

First, the planning algorithm expands each unexplored problem node in the graph \( \{S, F, G\} \) by checking for goal features in \( G \) that are not found in \( S \). For each goal feature not in \( S \), the algorithm determines how many different ways each MP resource in the facility may create that feature. In turn, each MP resource queries methods in the tool entities that are defined as usable by the MP resource through the ownership relationship. The capability of a given tool entity to create the feature is determined by the procedural reasoning in the tool entity’s class method that accesses data about the tool entity and the MP entity using the tool. At this high level planning stage, all MP operators using equipment, fixtures, and tools that satisfy this following logical statement from Chapter 4 are valid.

\[
\forall E, \forall Fixt, \exists tool, \forall I, \forall L \ (\text{isaprocessor} (E) \land \text{isafixture} (Fixt) \land \text{isatool} (tool) \land \text{isafixtureintent} (I) \land \text{isafixturelocator} (L) \land \text{owns} (E, Fixt) \land \text{owns} (E, tool) \land \text{makes} (tool, feat) \land \forall \text{datum} \mid \text{specifies} (datum, feat) \land \text{locates} (Fixt, datum, I, L)
\]

To illustrate this high level planning of MP activities, let there be a part with a goal feature that is a hole specified by a position on a surface of the part, a depth, a positional tolerance, and a diametric tolerance. This part is to be manufactured by the facility described with the resource model in Figure 4.4. The raw and finished
material buffer for the part owns port #1. The twist drill entity T1 has an elimDifference method that returns a solution for feature Feat7 of type “hole” with 0.25 DIA, depth, position and diametric tolerances. The material processor E3 that represents a vertical milling MP entity with port #2 owns the tool element T1. Assuming that the predicate makes (E3, Feat7, Fixt3) is true, then a solution is found for E3 and the fixture Fixt3 with some fixture intention. In this case, the fixture intention #1 that specifies that Fixt3 locates Feat3, Feat5, and Feat7 is used. This solution creates two sub-branches from the original problem statement and is illustrated in Figure 5.2. In general, if a solution to a portion of the goal is found, precondition and resultant states are computed. Thus, the sub-problems from this solution include the left sub-problem that specifies the location of the part in the fixture Fixt3 at port #2 with fixture intention #1. The right sub-problem specifies that the part must be moved from the milling machine’s port #2 to the finished part buffer port #1. By tracing through this sub-graph depth-first going from left child node to right child node, an operations sub-plan is generated with the following sequence: move part to port #2 at machine with drill from port #1, drill part to 0.25 DIA and move the finished part from port #2 to port #1.

![Figure 5.2 Illustration of search graph generation for drill operation](image)

S = {{∅}, 1, Fixt3, 1}
F = set of operators, drill ∈ F.
G = {{0.25 DIA hole}, 1, Fixt3, 1}
drill = operator applicable to G – S = 0.25 DIA hole
drill(S) = resultant state (after applying drill to S) = {{0.25 DIA hole}, 2, Fixt3, 1}
S_p = precondition state for drill = {{}, 2, Fixt3, 1}

Given a sub-problem in the search graph, if the difference between the current state and goal state does not include any features, the planner attempts to solve the difference between current and goal ports using MH or MT operators from the resource model. This is accomplished using backward planning from the goal location to the current location. MH operators fall into two categories: load and unload. MH load and unload operators were previously defined in Chapter 4 and they change the port, fixture, and fixture intention of a part state. The MH load operator changes the state of the part from the MH’s home port and gripper-fixture to a buffer’s port, fixture, and intent. The MH unload operator changes the state of the part from a buffer’s port and fixture to the MH’s home port, gripper-fixture, and intent.
As illustrated in Figure 5.2, the right sub-problem of \( \{S = \{\{0.25 \text{ DIA hole}\}, 2, \text{Fixt}_3, 1\}, F, G = \{\{0.25 \text{ DIA hole}\}, 1, \text{Fixt}_1, 1\}\} \) is created by the \textit{drill} operation by \( E_3 \) on the part in \( \text{Fixt}_3 \) at port \#2. Since no features need to be solved for this sub-problem, the finished part movement from port \#2 to port \#1 needs to be solved. This sub-problem reflects the requirement to move the finished part to the finished parts buffer after machining. From the resource model, the MH \textit{load}_{6,1} \) operator is found that satisfies the sub-goal using \( E_2 \), an MH resource entity. \( E_2 \) has a home port = \#6 and port \#1 is in \( E_2 \)’s loadable port list. Figure 5.3 illustrates this first step towards solving this problem by decomposing the problem into the left sub-problem of moving the part from port \#2 to port \#6 \( \{S = \{\{0.25 \text{ DIA hole}\}, 2, \text{Fixt}_3, 1\}, F, G = \{\{0.25 \text{ DIA hole}\}, 6, \text{Fixt}_2, 1\}\} \) and the solved right sub-problem \( \{S = \{\{0.25 \text{ DIA hole}\}, 1, \text{Fixt}_1, 1\}, F, G = \{\{0.25 \text{ DIA hole}\}, 1, \text{Fixt}_1, 1\}\} \) using operator \textit{load}_{6,1} \). The remaining steps consist of decomposing the left unsolved sub-problem into sub-sub-problems using other MH \textit{unload} and \textit{load} operations.

\[
\begin{align*}
S &= \{\{0.25 \text{ DIA hole}\}, 2, \text{Fixt}_3, 1\} \\
F &= \text{set of operators, } \text{load} \in F. \\
G &= \{\{0.25 \text{ DIA hole}\}, 6, \text{Fixt}_2, 1\} \\
\text{unload}_{3,6} &= \text{operator applicable to } G \rightarrow S = \text{unload from port } \#6 \text{ to port } \#1 \\
\text{unload}_{3,6}(S) &= \text{resultant state (after applying } \text{unload}_{3,6} \text{ to } S) = \{\{0.25 \text{ DIA hole}\}, 1, \text{Fixt}_1, 1\} \\
S_{\text{unload}_{3,6}} &= \text{precondition state for } \text{unload}_{3,6} = \{\{0.25 \text{ DIA hole}\}, 6, \text{Fixt}_2, 1\} \\
\{S, F, G\} &= \{S, F, S_{\text{unload}_{3,6}}\} \\
\{\text{unload}_{3,6}(S), F, G\} &= \{\text{unload}_{3,6}(S), F, G\} \\
\{S, F, G\} &= \{S, F, S_{\text{unload}_{3,6}}\} \\
\{\text{unload}_{3,6}(S), F, G\} &= \{\text{unload}_{3,6}(S), F, G\}
\end{align*}
\]

Figure 5.3 Illustration of search graph generation for MH \textit{unload} from home port operator

The Figure 5.3 left unsolved sub-problem \( \{S = \{\{0.25 \text{ DIA hole}\}, 2, \text{Fixt}_3, 1\}, F, G = \{\{0.25 \text{ DIA hole}\}, 6, \text{Fixt}_2, 1\}\} \) can be decomposed into the left unsolved sub-problem \( \{S = \{\{0.25 \text{ DIA hole}\}, 2, \text{Fixt}_3, 1\}, F, G = \{\{0.25 \text{ DIA hole}\}, 1, \text{Fixt}_1, 1\}\} \) and the right solved sub-problem \( \{S = \{\{0.25 \text{ DIA hole}\}, 6, \text{Fixt}_2, 1\}, F, G = \{\{0.25 \text{ DIA hole}\}, 6, \text{Fixt}_2, 1\}\} \). This is a consequence of applying the MH \textit{unload}_{3,6} \) operator to the Figure 5.3 left sub-problem. The resulting search graph is illustrated in Figure 5.4.

\[
\begin{align*}
S &= \{\{0.25 \text{ DIA hole}\}, 2, \text{Fixt}_3, 1\} \\
F &= \text{set of operators, } \text{unload} \in F. \\
G &= \{\{0.25 \text{ DIA hole}\}, 6, \text{Fixt}_2, 1\} \\
\text{unload}_{3,6} &= \text{operator applicable to } G \rightarrow S = \text{unload from port } \#3 \text{ to port } \#6 \\
\text{unload}_{3,6}(S) &= \text{resultant state (after applying } \text{unload}_{3,6} \text{ to } S) = \{\{0.25 \text{ DIA hole}\}, 6, \text{Fixt}_2, 1\} \\
S_{\text{unload}_{3,6}} &= \text{precondition state for } \text{unload}_{3,6} = \{\{0.25 \text{ DIA hole}\}, 3, \text{Fixt}_6, 1\} \\
\{S, F, G\} &= \{S, F, S_{\text{unload}_{3,6}}\} \\
\{\text{unload}_{3,6}(S), F, G\} &= \{\text{unload}_{3,6}(S), F, G\} \\
\{S, F, G\} &= \{S, F, S_{\text{unload}_{3,6}}\} \\
\{\text{unload}_{3,6}(S), F, G\} &= \{\text{unload}_{3,6}(S), F, G\}
\end{align*}
\]
The MT move operator changes the port number of the part’s state if this satisfies a subgoal for the part. The fixture of a part is not changed after a move operator is applied. Necessary conditions for solving a problem using an MT move operator to solve a sub-problem are that the goal features are solved and that the sub-goal port is in the MT resource’s reachable port list. Figure 5.5 illustrates how the move4,3 operator that uses E6, which is an MT resource with a reachability list of ports #3 and #4, decomposes the unsolved left sub-problem from Figure 5.4.

The complete solution to the example manufacturing scenario in Chapter 4 results in the search graph that is illustrated in Figure 5.6. For simplicity, the table buffer E5 is omitted from the resource model so that no alternatives are present. This solution uses all of the operators illustrated in Figures 5.3 through 5.5 plus additional material handling and transportation operators.
In addition to unloading or loading parts from or to machines, MH resources can also be used to re-fixture a part on a machine. For instance, a part may have two features that can be machined on the same MP resource but require different fixturing. After machining one feature, the part must be unloaded to an MH resource’s home position and re-loaded into a different fixture on the same machine tool. The search graph generation algorithm handles this situation by using an MH load operator to solve the sub-problem where the part’s current features and ports are equal to the goal state’s features and ports, but the fixture or fixture intention differs between the current state and goal state. The precondition of this operator is that the part must be located at the MH resource entity’s home port. The solution of the next problem that results from this precondition is an MH unload operator that unloads the part from the machine to the MH entity’s home port. Thus, the generated sequence is an MH unload from the machine to the MH entity’s home position and then an MH load to the machine with the correct orientation. Figure 5.7 illustrates a search graph based on this solution for a part with two goal features that require different fixtures. The two features are features $A$ and $B$ and the correct fixtures for locating these features.
on the vertical milling machine with port #2 are FixtA and FixtB, respectively. If we trace through this graph depth first from left child nodes (providing the solution to the precondition of the right node) to right child nodes, the resulting operations plan is as follows: move part from port #1 to fixture FixtA at port #2, drill feature A (drillA), unload part from port #2 (unload2), load part to FixtB at port #2 (load7), drill feature B (drillB), and move part from port #2 to port #1. This search graph illustrates how physical causality in the operators’ preconditions is obeyed in the selection of operators. For instance, the part is loaded to the correct fixture the milling machine at port #2 before each drilling operation is executed at that machine. This example illustrates how solving the features first, then solving the difference between current and goal ports, and then solving the difference in part fixtures is correct. If a fixture were available that could locate features A and B using different fixture intents, then the same search graph structure would result. The difference is that the part would be reloaded into the same fixture with a different orientation matching the different fixture intent.

Figure 5.7 Illustration of search graph generation for re-fixturing of part in a machine.

5.2.3 Generation of Operations Search Graph With Duplication and Alternatives

In the previous sub-section, search graphs were generated manually without alternatives. In practice, an automated planner may find multiple operators to solve a given problem. Each one of these operators creates an alternative solution to the problem. A given solution may take the part state space back to a state that was already reached by an ancestor node. Such solutions generate cycles in the search graph that need to be detected and eliminated from the search graph. Other solutions may duplicate a solution that already exists in the graph
because the current problem being solved has already been solved elsewhere in the graph. If the existing solution is not an ancestor of the solution under consideration, the parent node of the new solution points instead to the existing solution. Thus, there may exist multiple parents for a given node, and the results of the search process is a graph rather than a tree.

In the manufacturing operations domain where the state space is defined to represent the state of the manufactured part, alternative solutions to a problem result from different operations that solve some part of the goal. These alternative operations may add the same goal feature in different ways to the part state using different equipment, fixtures, or tools. For example, Figure 5.8 illustrates two alternative ways of creating feature A by either drilling it or reaming it. This figure also illustrates how the reamₜ operator results in a precondition state that includes a precondition feature A'. Feature A' represents a starting hole that must be present before the reaming operation. Thus, the operator drillₜ must be applied before the reamₜ operator. Alternative operations may also add different goal features without any mutual precedence to the part state. Routing the part along different paths may create other alternative solutions. For instance, from the resource and part models given in Chapter 4, the solution to the problem of moving the part to E₃ from E₁ may include an alternative routing to the table buffer represented by E₅. The partial solution to this problem is illustrated by the search graph in Figure 5.9. This figure shows that there exist two operators that satisfy the problem of getting the raw material part to Port₇ or \{\{\}, 1, Fixt₁, 1\}, \{\}, 7, Fixt₄, 1\}. These two operators represent operations by the robot E₄ that unload the part from Port₅ to Port₇ and from Port₄ to Port₇. An additional operator also appears to solve the problem statement but only creates a cycle in the search graph. Applying this operator unload₄ to the problem does satisfy the immediate goal of the part being located at Port₇, but this operation undoes the previous operator’s task of loading the part to Port₅ from Port₇. Thus, checks have to be made that prevent such cycles from occurring.
The avoidance of cycles in the search graph is accomplished by detecting the cycles and then labeling the cyclic nodes “unsolvable”. Branches of the search graph whose nodes are labeled “unsolvable” are not expanded further and are ignored when developing an operations plan from the search graph. Graph cycles are detected by checking if left children nodes that result from an operator have the identical problem statement as an ancestor node. For instance, in Figure 5.9, the operator $unload_{2,7}$ generates a left child with the problem statement $\{\emptyset, 2, \text{Fixt}_3, 1\}$. This problem statement is identical to the initial problem statement in Figure 5.9. Thus, the whole alternative branch that is based on this operator is labeled “unsolvable”. An exception to this rule must occur when auxiliary buffers for parts can be used for deadlock avoidance or throughput optimization.
Researchers have proposed using auxiliary part buffers for scheduling [Hussain & Joshi, 1999] and deadlock avoidance [Kumaran et al., 1994]. If it is physically possible, parts may be temporarily placed in these buffers to allow other parts to bypass them based on algorithms that increase throughput and avoid deadlocks. In order to incorporate such algorithms for automated flexible manufacturing facilities, the material handling plans must be generated to move parts to these auxiliary buffers and back out again. In the resource and part models from Chapter 4, the table buffer E5 with port #5 may be used as an auxiliary buffer for parts arriving to be machined on the vertical mill E3. When the robot E4 loads the part to the buffer and unloads it again, the part reenters the same part state as when the robot initially unloaded the part from the AGV port. Figure 5.9 illustrates the search graph in this situation and displays the repeated part state \( \{\{\}, 1, \text{Fixt}_1, 1\}, \{\{\}, 7, \text{Fixt}_4, 1\} \). In this case, the repeated part state is desirable and a cycle is avoided because the initial repeated part state occurrence has an alternative branch resulting from the operator \( \text{unload}_{3,7} \). Thus, the general rule is that if a left child node is generated that repeats the same problem statement as an ancestor of this child node, its branch is labeled as unsolvable unless the ancestor has an alternative branch that does not lead to this child node and this alternative branch is associated with an auxiliary buffer.

Figure 5.9 Illustration of search graph generation with material handling alternative.
It should be apparent to the reader that alternatives in the search graph will generate duplicated problem statements that are not necessarily cyclic nodes. Figures 5.8 and 5.9 both illustrate situations where problem statements are duplicated in nodes that are not directly related (i.e., one is not a direct ancestor of the other). In order to avoid the computational effort of solving the same problem twice, the descendents of the new and duplicated node become the descendents of the original duplicated node. Thus, these descendents of the original duplicated node have two or more parent nodes. For example, in Figure 5.8, both nodes that have the same problem statement, \{\}, 1, Fixt1, 1\}, \{\}, 2, Fixt1, 1\}, point to the same block that represents the solution of moving the part from Port1 to Port2. This solution represents the fact that regardless of which machining operations will occur on E3 (drill or drill + ream), they both require that the part is located identically in the same fixture. Thus, both alternatives use the same set of material handling operations to bring the part to the machining fixture.

5.3 Geometric Planning

5.3.1 Introduction

In Chapter 4, functions were defined and used to specify the validation criteria for operations generated by manufacturing operations planning. These functions model the tolerances associated with different operations measured along specific coordinate axes. The high level planner described in Section 5.2 calls the lower level reasoning to verify that operators are valid. This section defines the lower level geometric tolerance information that defines tolerances and transformations in the resource model of the facility. This information must be stored in the resource model to perform operations planning. Next, this section defines the algorithms for computing the functions from Chapter 4 using the information from the resource model.

5.3.2 Geometric Definitions in Resource and Part Models

In order to compute these functions for tolerance stacking along different coordinate axes, it is necessary to know the transformation matrices between different coordinate frames. The coordinate frames used include the
frame of the facility, a frame for each equipment entity attached to the hardware, a frame for each port, and a frame for each fixture. These frames for equipment E₁, E₂ and their fixtures and ports are illustrated in Figure 5.10. The matrices that define geometric transformations between these frames are stored as attributes of elements in the resource model. Using the notation developed by Craig (1989), a transformation matrix that defines the geometrical transformation from frame A to frame B is named $^A T^B$. A point defined in A’s frame is written as $^A P$ while a vector defined in A’s frame is named $^A V$. The unit vectors of a coordinate system B are written as $X_B$, $Y_B$, $Z_B$. These unit vectors expressed in A’s coordinate system are written as $^A X_B$, $^A Y_B$, $^A Z_B$. A transformation matrix defines the displacement between frames as well as rotation in three dimensions such that for a given point defined in B’s frame $^B P$, the same point defined in A’s frame is named $^A P$ and computed with the following equation: $^A P = ^A T^B \cdot ^B P$. The product of transform matrices results in a new transform matrix such that $^A T^C = ^A T^B \cdot ^B T^C$. The inverse of a transform matrix is the inverse of the transform such that $^A T^{-1} = (^A T)^{-1}$. Each of these transform matrices consists of a rotational sub-matrix and a translational vector. For a given matrix $^A T$, its rotational sub-matrix $^A R$ consists of three column vectors $[^A X_B, ^A Y_B, ^A Z_B]$. These column vectors are unit vectors giving the principal directions of coordinate frame B described in coordinate frame A. For example, the column vector in the x direction, $^A X_B$, is the following column vector $[X_B \cdot X_A, X_B \cdot Y_A, X_B \cdot Z_A]^T$, where each vector pair may be expressed in either the A or B coordinate system. The translational vector, $^A P_{A \text{origin}}$, is a column vector specifying the translation from the A’s origin to B’s origin in A’s coordinate system.
In the resource model, the location of ports is defined with respect to the facility using the transform $\text{Facility}_{\text{port}T}$. There is such a transform associated with every port in the facility. Each of these ports is attached to an equipment entity, and the matrix $\text{equip}_{\text{port}T}$ defines the transform from the equipment frame to the port frame. This $\text{equip}_{\text{port}T}$ transform exists in the resource model for every port owned by an equipment entity. Fixtures are assumed to be mounted relative to port frames so a transform from port frames to fixture frames is named $\text{port}_{\text{fixt}T}$.

This transform exists in the resource model for every fixture that can be located at a port. The locator points and unit vectors for fixtures are defined in the fixture coordinate frame. Assuming that the fixture has a primary locator, secondary locator, and a tertiary locator, points for the three locators in the closed fixture position are named $\text{fixtP}_{\text{close},1}$, $\text{fixtP}_{\text{close},2}$, and $\text{fixtP}_{\text{close},3}$. The points for the three locators in the opened fixture position are named $\text{fixtP}_{\text{open},1}$, $\text{fixtP}_{\text{open},2}$, and $\text{fixtP}_{\text{open},3}$. The unit vectors pointing from the locators to the located datums are named $\text{fixtV}_{\text{close},1}$, $\text{fixtV}_{\text{close},2}$, $\text{fixtV}_{\text{close},3}$, and $\text{fixtV}_{\text{open},1}$, $\text{fixtV}_{\text{open},2}$, $\text{fixtV}_{\text{open},3}$. All fixtures have these points and vectors defined in the resource model. Formally, the existence of these transforms, points, and vectors that are stored in the resource model is defined as follows.

$\forall \text{port} \ (\text{isaport} \ (\text{port}) \land \exists \ \text{Facility}_{\text{port}T})$

$\forall \text{E}, \forall \text{port} \ (\text{isanequipment} \ (\text{E}) \land \text{isaport} \ (\text{port}) \land \text{owns} \ (\text{E}, \text{port}) \Rightarrow \exists \ \text{equip}_{\text{port}T})$

$\forall \text{E}, \forall \text{port}, \forall \text{fixt} \ (\text{isanequipment} \ (\text{E}) \land \text{isaport} \ (\text{port}) \land \text{isafixture} \ (\text{fixt}) \land \text{owns} \ (\text{equip}, \text{fixt}) \land \text{owns} \ (\text{equip}, \text{port}) \Rightarrow \exists \ \text{port}_{\text{fixt}T})$

$\forall \text{fixt} \ (\text{isafixture} \ (\text{fixt}) \Rightarrow \exists \ \text{fixtP}_{\text{closed},1} \land \exists \ \text{fixtP}_{\text{open},1}, \exists \ \text{fixtP}_{\text{closed},2} \land \exists \ \text{fixtP}_{\text{open},2} \land \exists \ \text{fixtP}_{\text{closed},3} \land \exists \ \text{fixtP}_{\text{open},3})$

$\forall \text{fixt} \ (\text{isafixture} \ (\text{fixt}) \Rightarrow \exists \ \text{fixtV}_{\text{closed},1} \land \exists \ \text{fixtV}_{\text{open},1}, \exists \ \text{fixtV}_{\text{closed},2} \land \exists \ \text{fixtV}_{\text{open},2} \land \exists \ \text{fixtV}_{\text{closed},3} \land \exists \ \text{fixtV}_{\text{open},3})$
5.3.3 Geometric Computations

Chapter 4 defined that tolerance stacking can only be accomplished using components of tolerances along the same coordinate axis. Thus, some of the criteria for valid operations consist of mathematical expressions that compare sums of components of tolerances along the same coordinate axis. These expressions use functions that return the results of the transformation of tolerance data in the resource and part models. One important transformation that must be calculated in order to transform tolerances for material handling is the transform from a robot’s port frame to its gripper-fixture frame when the robot’s gripper is at a buffer fixture in order to load or unload parts. This transform port \text{fixt}T, which represents the transform from a resource’s port to its fixture, is fixed for non-material handler resources, but port \text{fixt}T changes for material handler resources as the material handler moves its gripper-fixture around. An algorithm has been developed that computes this transform port \text{fixt}T for a given robot resource \text{R} with port \text{r} and fixt \text{r} that unloads or loads part \text{i} from or to buffer resource \text{E} with port \text{b} and fixt \text{b}. From the rules for matrix transformation [Craig, 1989], the transform port \text{fixt}T may be calculated with the following expression: port \text{fixt}T = [\text{Facility port}T]^{-1} * \text{Facility portbT} * \text{portb fixt}T * \text{fixt fixtrT}.

This expression is the product of four matrices whose values are returned by functions in the resource model. The transform [\text{Facility port}T]^{-1} is the matrix inverse of \text{Facility port}T, which is the fixed transform of port \text{r}’s frame with respect to the facility coordinate frame, and is an attribute of the port \text{r} element. The transform \text{Facility portbT}, which is the fixed transform of port \text{b}’s frame with respect to the facility coordinate frame, is stored as an attribute of the port \text{b} element. The matrix \text{portb fixt}T, which is the fixed transform of fixt \text{b}’s frame with respect to port \text{b}’s coordinate frame, represents the location of the buffer’s fixture in the buffer and is stored as an attribute of the buffer element. The matrix \text{fixt fixtrT}, which is the transform of fixt \text{b}’s frame with respect to fixt \text{r}’s frame, represents the transformation from the buffer fixture to the gripper-fixture when the gripper is in position to grasp or release the part in the buffer fixture for the unload or load operations.

To compute this transform \text{fixt fixtrT} for some part state, it is assumed that the current fixture intents of both fixtures are known. Next, it is necessary to compare the part’s feature-datums located by these fixtures, and
determine how both fixtures may simultaneously locate the part. Thus, the algorithm to compute the transform \( \text{fixt}_{b} \text{fixt}_{r} T \) compares each feature-datum datum \( \text{datum}_{b} \) located by \( \text{fixt}_{b} \) and determines which feature-datum datum \( \text{datum}_{r} \) located by \( \text{fixt}_{r} \) is either identical or parallel to \( \text{datum}_{b} \). These matching feature-datum pairs are then used with the locator vectors from the fixtures to feature-datum surfaces to compute the transform \( \text{fixt}_{b} \text{fixt}_{r} T \). Three direction cosine column vectors define the rotation submatrix \( \text{fixt}_{b} \text{fixt}_{r} R \) for \( \text{fixt}_{b} \text{fixt}_{r} T \). Thus, \( \text{fixt}_{b} \text{fixt}_{r} R = [\text{fixt}_{b}X_{\text{fixt}_{r}}, \text{fixt}_{b}Y_{\text{fixt}_{r}}, \text{fixt}_{b}Z_{\text{fixt}_{r}}] \).

These direction cosine vectors are computed by the projection of the \( \text{fixt}_{r} \) vector onto the unit directions of the \( \text{fixt}_{b} \) coordinate frame. Thus, along the x axis of the \( \text{fixt}_{b} \) frame, the column vector \( \text{fixt}_{b}X_{\text{fixt}_{b}} \) is equivalent to \( [X_{\text{fixt}_{b}}X_{\text{fixt}_{b}}, X_{\text{fixt}_{b}}Y_{\text{fixt}_{b}}, X_{\text{fixt}_{b}}Z_{\text{fixt}_{b}}]^T \). If these unit vectors are defined in the \( \text{fixt}_{b} \) frame, then the column vector \( \text{fixt}_{b}X_{\text{fixt}_{r}} \) is equivalent to \( [\text{fixt}_{b}X_{\text{fixt}_{r}}X_{\text{fixt}_{r}}, \text{fixt}_{b}X_{\text{fixt}_{r}}Y_{\text{fixt}_{r}}, \text{fixt}_{b}X_{\text{fixt}_{r}}Z_{\text{fixt}_{r}}]^T \). Since the unit locator vectors for the two fixtures are known and are parallel to the coordinate axes of each fixture, the columns of the rotation submatrix \( \text{fixt}_{b} \text{fixt}_{r} R \) can be computed. Since each column vector in \( \text{fixt}_{b} \text{fixt}_{r} R \) is associated with a coordinate axis in the \( \text{fixt}_{r} \) frame, each unit locator vector for the \( \text{fixt}_{b} \) is simply placed in the column that matches the coordinate axis of its parallel unit locator vector for the \( \text{fixt}_{r} \). If the directions of the two unit locator vectors are opposite, then the sign of this column vector is changed.

Specifically, the algorithm for computing \( \text{fixt}_{b} \text{fixt}_{r} T \) begins by determining the coordinate axis of the fixture frame that is parallel to each part datum \( \text{datum}_{i} \) located by \( \text{fixt}_{b} \), where \( i \) is 1 for the primary datum, 2 for the secondary datum, and 3 for the tertiary datum. This coordinate axis is known as \( \text{fixtAxis} \). Next, the algorithm determines the coordinate axis of the fixture frame that is parallel to each part datum \( \text{datum}_{j} \) located by \( \text{fixt}_{r} \), where \( j \) is 1 for the primary datum, 2 for the secondary datum, and 3 for the tertiary datum. This coordinate axis is known as \( \text{gripperAxis} \). If these part datums corresponding to locators \( i \) and \( j \) are the same, then both fixtures locate the same part datum and a variable \( \text{dir} \) is set to +1. Otherwise, if part datums corresponding to locators \( i \) and \( j \) are not the same, and if datum \( i \) specifies the location of datum \( j \) or vice-versa, then \( \text{dir} \) is set to −1. If none of these conditions are met, then \( \text{dir} \) is set to 0. If \( \text{dir} \) is −1 or +1, then one directional cosine of the transform \( \text{fixt}_{b} \text{fixt}_{r} T \) can be calculated from this datum pair. If the operation is an unload operation, then a column directional cosine vector corresponding to the \( \text{gripperAxis} \) is computed by multiplying the \( \text{fixt}_{b}V_{\text{close}} \) vector by \( \text{dir} \) and the component of \( \text{fixt}_{b}V_{\text{open}} \) along the \( \text{gripperAxis} \) coordinate axis. For example, if the operation is unload and \( \text{gripperAxis} \) represents the x axis, then the first column of the rotation submatrix is the vector \( \text{fixt}_{b}X_{\text{fixt}_{r}} \) that is the product of the
\(\text{fixtb} \text{V}_{\text{close}, i}\) vector, \(\text{dir}\), and the component of \(\text{fixtv} \text{V}_{\text{open}, j}\) along the x coordinate axis. For the gripperAxis coordinate axis, this column vector is expressed by the following: \(\text{fixtb} \text{fixtr} \text{R}(\text{gripperAxis}) = \text{dir}* \text{fixtv} \text{V}_{\text{close}, j}(\text{gripperAxis})*\text{fixtb} \text{V}_{\text{open}, i}\). Otherwise, if the operation is a load operation, the column vector corresponding to the gripperAxis is computed with the following expression: \(\text{fixtb} \text{fixtr} \text{R}(\text{gripperAxis}) = \text{dir}* \text{fixtv} \text{V}_{\text{open}, j}(\text{gripperAxis})*\text{fixtb} \text{V}_{\text{close}, i}\). These calculations produce the rotational submatrix \(\text{fixtb} \text{fixtr} \text{R} = \begin{bmatrix} \text{fixtb} \text{X} \text{fixtr} & \text{fixtb} \text{Y} \text{fixtr} & \text{fixtb} \text{Z} \text{fixtr} \end{bmatrix}\). The other part of the transform \(\text{fixtb} \text{fixtr} \text{T}\) is the translational vector \(\text{fixtb} \text{P}_{\text{fixtr} \text{origin}}\) that specifies the vector from \(\text{fixtb}\)’s origin to \(\text{fixtr}\)’s origin in \(\text{fixtb}\)’s coordinate system.

To calculate the translational vector \(\text{fixtb} \text{P}_{\text{fixtr} \text{origin}}\) for the transformation \(\text{fixtb} \text{fixtr} \text{T}\), it is necessary to compute the distance from \(\text{fixtb}\)’s origin to \(\text{fixtr}\)’s origin along the coordinate axes of \(\text{fixtb}\)’s coordinate frame. For each pair of locators of \(\text{fixtb}\) and \(\text{fixtr}\), that have parallel vectors, the component of this distance along one of \(\text{fixtb}\)’s coordinate axis is calculated by determining the distance between fixture origins along their locator vectors. These vector components are calculated by finding the pairs of fixture locators who have parallel vectors. For fixture pairs that do not locate the same feature-datum, the \(\text{dist}\) variable is set to the distance between the two located feature-datums. Next, the distances from both fixtures to the point of contact at the surface of the feature-datum that is located by the gripper-fixture are calculated. These distances are \(\text{fixtDisp}\) for the buffer fixture and \(\text{gripperDisp}\) for the gripper fixture. The distance \(\text{fixtDisp}\) is equal to the \(\text{fixtAxis}\) component of its locator point plus the \(\text{fixtAxis}\) component of its locator vector multiplied times its vector length and the distance \(\text{dist}\) to the feature-datum located by the gripper-fixture. Mathematically, this is expressed as follows: \(\text{fixtDisp} = \text{fixtP}_{\text{open}, i}(\text{fixtAxis}) + \text{fixtV}_{\text{open}, i}(\text{fixtAxis})*\text{vect_length}(\text{fixtV}_{\text{open}, i} + \text{dist})\). The distance \(\text{gripperDisp}\) is equal to the gripperAxis component of its locator point plus the gripperAxis component of its locator vector multiplied times its vector length. Mathematically, this is expressed as follows: \(\text{gripperDisp} = \text{fixtrP}_{\text{close}, j}(\text{gripperAxis}) + \text{fixtrV}_{\text{close}, j}(\text{gripperAxis})*\text{vect_length}(\text{fixtrV}_{\text{close}, j})\). Finally, the distance between the two fixtures is calculated by subtracting the product of transformed distance \(\text{gripperDisp}\) from \(\text{fixtDisp}\). The distance \(\text{gripperDisp}\) is transformed by multiplying it by \(\text{fixtb} \text{fixtr} \text{T}(\text{fixtAxis}, \text{gripperAxis})\), which is +1 if gripperAxis and \(\text{fixtAxis}\) point in the same direction and –1 if they point in opposite directions. Mathematically, this is expressed as follows: \(\text{fixtb} \text{fixtr} \text{T}(\text{fixtAxis}, 3) = \text{fixtDisp} - \text{fixtb} \text{fixtr} \text{T}(\text{fixtAxis}, \text{gripperAxis})*\text{gripperDisp}\). The complete algorithm for computing \(\text{fixtb} \text{fixtr} \text{T}\) is given in pseudo-code as follows.
Algorithm for computing $\mathbf{fixt}_{\mathbf{fixt}} \mathbf{r}_T$:
Initialize $\mathbf{fixt}_{\mathbf{fixt}} \mathbf{T}$ to be all zeros except $\mathbf{fixt}_{\mathbf{fixt}} \mathbf{T}(4, 4) \leftarrow 1$
for all part datums $i$ located by $\mathbf{fixt}_b$ using this fixture intent
if (operation is unload)
    $\mathbf{fixt}\text{Axis} \leftarrow$ axis that is parallel to $\mathbf{fixt}_{\mathbf{fixt}}V_{\text{close},i}$
else if (operations is load)
    $\mathbf{fixt}\text{Axis} \leftarrow$ axis that is parallel to $\mathbf{fixt}_{\mathbf{fixt}}V_{\text{open},i}$
for all part datums $j$ located by $\mathbf{fixt}_r$ using this gripper-fixture intent
if (operation is unload)
    $\mathbf{gripper}\text{Axis} \leftarrow$ axis that is parallel to $\mathbf{fixt}_{\mathbf{fixt}}V_{\text{open},j}$
else if (operation is load)
    $\mathbf{gripper}\text{Axis} \leftarrow$ axis that is parallel to $\mathbf{fixt}_{\mathbf{fixt}}V_{\text{close},j}$
if (datum $i = \text{datum } j$) // gripper-fixture and buffer fixture locate the same part feature
    $\text{dir} \leftarrow 1$
else // check if gripper-fixture and buffer fixture locate features that locate each other
    $\text{feat}_r \leftarrow \text{feature} (\text{part}, j)$
    $\text{feat}_b \leftarrow \text{feature} (\text{part}, i)$
    if ($\text{specifies} (\text{feat}_b, \text{feat}_r)$ is true)
        $\text{dir} \leftarrow -1$
    if ($\text{dir} \neq 0$) // compute one column vector of the rotation sub-matrix
        for the values of $k \leftarrow 0, 1, 2$
        if (operation is unload)
            $\mathbf{fixt}_{\mathbf{fixt}}T(k, \text{gripper}\text{Axis}) \leftarrow \text{dir} * \mathbf{fixt}_{\mathbf{fixt}}V_{\text{open}, j}(\text{gripper}\text{Axis}) * \mathbf{fixt}_{\mathbf{fixt}}V_{\text{close}, i}(k)$
        else if (operation is load)
            $\mathbf{fixt}_{\mathbf{fixt}}T(k, \text{gripper}\text{Axis}) \leftarrow \text{dir} * \mathbf{fixt}_{\mathbf{fixt}}V_{\text{close}, j}(\text{gripper}\text{Axis}) * \mathbf{fixt}_{\mathbf{fixt}}V_{\text{open}, i}(k)$
for all part datums $i$ located by $\mathbf{fixt}_b$ using this fixture intent
if (operation is unload)
    $\mathbf{fixt}\text{Axis} \leftarrow$ axis that is parallel to $\mathbf{fixt}_{\mathbf{fixt}}V_{\text{close},i}$
else if (operations is load)
    $\mathbf{fixt}\text{Axis} \leftarrow$ axis that is parallel to $\mathbf{fixt}_{\mathbf{fixt}}V_{\text{open},i}$
for all part datums $j$ located by $\mathbf{fixt}_r$ using this gripper-fixture intent
if (operation is unload)
    $\mathbf{gripper}\text{Axis} \leftarrow$ axis that is parallel to $\mathbf{fixt}_{\mathbf{fixt}}V_{\text{open},j}$
else if (operation is load)
    $\mathbf{gripper}\text{Axis} \leftarrow$ axis that is parallel to $\mathbf{fixt}_{\mathbf{fixt}}V_{\text{close},j}$
if (buffer fixture datum $i = \text{gripper datum } j$)
    // gripper-fixture and buffer fixture locate the same part feature
    $\text{dir} \leftarrow 1$
    $\text{dist} \leftarrow 0.0$
else // check if gripper-fixture and buffer fixture locate features that locate each other
    $\text{feat}_r \leftarrow \text{feature} (\text{part}, j)$
    $\text{feat}_b \leftarrow \text{feature} (\text{part}, i)$
    if ($\text{specifies} (\text{feat}_b, \text{feat}_r)$ is true)
        $\text{dir} \leftarrow -1$
        $\text{dist} \leftarrow \text{specifies\_dist} (\text{feat}_b, \text{feat}_r)$
    if ($\text{dir} \neq 0$) // compute vector components from fixture frame origin to gripper origin
        if (operation is unload)
            $\mathbf{fixt}\text{Disp} \leftarrow \mathbf{fixt}_{\mathbf{fixt}}P_{\text{close}, i}(\text{fixt}\text{Axis}) + \mathbf{fixt}_{\mathbf{fixt}}V_{\text{close}, i}(\text{fixt}\text{Axis}) * (\text{vect\_length} (\mathbf{fixt}_{\mathbf{fixt}}V_{\text{close}, i}) + \text{dist})$
            $\mathbf{gripper}\text{Disp} \leftarrow \mathbf{fixt}_{\mathbf{fixt}}P_{\text{open}, j}(\text{gripper}\text{Axis}) + \mathbf{fixt}_{\mathbf{fixt}}V_{\text{open}, j}(\text{gripper}\text{Axis}) * (\text{vect\_length} (\mathbf{fixt}_{\mathbf{fixt}}V_{\text{open}, j}))$
        else if (operation is load)
            $\mathbf{fixt}\text{Disp} \leftarrow \mathbf{fixt}_{\mathbf{fixt}}P_{\text{open}, i}(\text{fixt}\text{Axis}) + \mathbf{fixt}_{\mathbf{fixt}}V_{\text{open}, i}(\text{fixt}\text{Axis}) * (\text{vect\_length} (\mathbf{fixt}_{\mathbf{fixt}}V_{\text{open}, i}) + \text{dist})$
            $\mathbf{gripper}\text{Disp} \leftarrow \mathbf{fixt}_{\mathbf{fixt}}P_{\text{close}, j}(\text{gripper}\text{Axis}) + \mathbf{fixt}_{\mathbf{fixt}}V_{\text{close}, j}(\text{gripper}\text{Axis}) * (\text{vect\_length} (\mathbf{fixt}_{\mathbf{fixt}}V_{\text{close}, j}))$
        $\mathbf{fixt}_{\mathbf{fixt}}T(\text{fixt}\text{Axis}, 3) \leftarrow \mathbf{fixt}\text{Disp} - \mathbf{fixt}_{\mathbf{fixt}}T(\text{fixt}\text{Axis}, \text{gripper}\text{Axis}) * \text{gripDisp}$
Functions that return the components of tolerances between features and their datums in the equipment coordinate frame were used in Chapter 4 to define the criteria for valid material processing operations. These functions transform the built-in tolerance specification between features on a part into the x, y, and z components of a tolerance vector defined in the equipment coordinate system. For a given equipment entity \( E \) that holds a part with features \( \text{Feat}_a \) and \( \text{Feat}_b \) such that \( \text{Feat}_a \) locates \( \text{Feat}_b \), these functions are \( \text{specifies_tol}_X(E, \text{Feat}_a, \text{Feat}_b) \), \( \text{specifies_tol}_Y(E, \text{Feat}_a, \text{Feat}_b) \), and \( \text{specifies_tol}_Z(E, \text{Feat}_a, \text{Feat}_b) \). The tolerance vector is computed by transforming the locator vector of a fixture that locates one of the features into the equipment coordinate frame.

The transformation assumes that the locator vector is parallel to the vector between the features. This unit locator vector is scaled by the tolerance between \( \text{Feat}_a \) and \( \text{Feat}_b \) given by the part model and returned by the function \( \text{specifies_tol} \left( \text{Feat}_a, \text{Feat}_b \right) \). Once the tolerance vector is computed, the dot products of x, y, and z vectors with the tolerance vector are the components of the tolerance vector. The equations to compute these tolerance functions that are as follows for each x, y, and z coordinate axis of the equipment frame.

\[
\forall E, \forall \text{Part}, \forall \text{port}, \forall \text{fixt}, \forall \text{Feat}_a, \forall \text{Feat}_b, \forall I, \forall L \ (\text{isanequipment} \ (E) \land \text{isapart} \ (\text{Part}) \land \text{isafixture} \ (\text{fixt}) \land \text{isafeature} \ (\text{Feat}_a) \land \text{isafeature} \ (\text{Feat}_b) \land \text{isafixtureintent} \ (I) \land \text{isafixturelocator} \ (L) \land \text{owns} \ (E, \text{port}) \land \text{owns} \ (E, \text{fixt}) \land \text{locates} \ (\text{fixt}, \text{Feat}_a, I, L) \Rightarrow \text{specifies_tol}_X \left( E, \text{Feat}_a, \text{Feat}_b \right) = X \left( I \right) \left( \text{T}_{\text{port}} \left( \text{fixt} \right) \ast \text{T}_{\text{closed}} \left( I \right) \ast \text{specifies_tol} \left( \text{Feat}_a, \text{Feat}_b \right) \right)
\]

\[
\forall E, \forall \text{Part}, \forall \text{port}, \forall \text{fixt}, \forall \text{Feat}_a, \forall \text{Feat}_b, \forall I, \forall L \ (\text{isanequipment} \ (E) \land \text{isapart} \ (\text{Part}) \land \text{isafixture} \ (\text{fixt}) \land \text{isafeature} \ (\text{Feat}_a) \land \text{isafeature} \ (\text{Feat}_b) \land \text{isafixtureintent} \ (I) \land \text{isafixturelocator} \ (L) \land \text{owns} \ (E, \text{port}) \land \text{owns} \ (E, \text{fixt}) \land \text{locates} \ (\text{fixt}, \text{Feat}_a, I, L) \Rightarrow \text{specifies_tol}_Y \left( E, \text{Feat}_a, \text{Feat}_b \right) = Y \left( I \right) \left( \text{T}_{\text{port}} \left( \text{fixt} \right) \ast \text{T}_{\text{closed}} \left( I \right) \ast \text{specifies_tol} \left( \text{Feat}_a, \text{Feat}_b \right) \right)
\]

\[
\forall E, \forall \text{Part}, \forall \text{port}, \forall \text{fixt}, \forall \text{Feat}_a, \forall \text{Feat}_b, \forall I, \forall L \ (\text{isanequipment} \ (E) \land \text{isapart} \ (\text{Part}) \land \text{isafixture} \ (\text{fixt}) \land \text{isafeature} \ (\text{Feat}_a) \land \text{isafeature} \ (\text{Feat}_b) \land \text{isafixtureintent} \ (I) \land \text{isafixturelocator} \ (L) \land \text{owns} \ (E, \text{port}) \land \text{owns} \ (E, \text{fixt}) \land \text{locates} \ (\text{fixt}, \text{Feat}_a, I, L) \Rightarrow \text{specifies_tol}_Z \left( E, \text{Feat}_a, \text{Feat}_b \right) = Z \left( I \right) \left( \text{T}_{\text{port}} \left( \text{fixt} \right) \ast \text{T}_{\text{closed}} \left( I \right) \ast \text{specifies_tol} \left( \text{Feat}_a, \text{Feat}_b \right) \right)
\]

Functions that return the components of specified tolerances between features and their datums in the facility coordinate frame were used in Chapter 4 to define the criteria for valid material handling operations. These functions transform the built-in tolerance specification between features on a part into the x, y, and z components of a tolerance vector defined in the facility coordinate system. For a given \( \text{Facility} \) element with an equipment entity \( E \) that holds a part with features \( \text{Feat}_a \) and \( \text{Feat}_b \) such that \( \text{Feat}_a \) locates \( \text{Feat}_b \), these functions are
specifies_tolX (Facility, E, Feat_a, Feat_b), specifies_tolY (Facility, E, Feat_a, Feat_b), and specifies_tolZ (Facility, E, Feat_a, Feat_b). The tolerance vector is computed by transforming the locator vector of a fixture that locates one of the features into the facility coordinate frame. The transformation assumes that the locator vector is parallel to the vector between the features. This unit locator vector is scaled by the tolerance between Feat_a and Feat_b from the part model and the result returned by the function specifies_tol (Feat_a, Feat_b). Once the tolerance vector is computed, the dot products of x, y, and z vectors with the tolerance vector are the components of the tolerance vector. The equations to compute these tolerance functions are as follows for each x, y, and z coordinate axis for the Facility element’s frame.

∃ Facility, ∀ E, ∀ Part, ∀ Feat_a, ∀ Feat_b, ∀ I, ∀ L (isafacility (Facility) ∧ isanequipment (E) ∧ isapart (Part) ∧ isafixture (fixt) ∧ isafeature (Feat_a) ∧ isafeature (Feat_b) ∧ isafixtureintent (I) ∧ isafixturelocator (L) ∧ owns (E, port) ∧ owns (E, fixt) ∧ locates (fixt, Feat_a, I, L) ⇒ specifies_tolX (Facility, E, Feat_a, Feat_b) = X • Facility portT* port *specifies_tol (Feat_a, Feat_b))

∃ Facility, ∀ E, ∀ Part, ∀ Feat_a, ∀ Feat_b, ∀ I, ∀ L (isafacility (Facility) ∧ isanequipment (E) ∧ isapart (Part) ∧ isafixture (fixt) ∧ isafeature (Feat_a) ∧ isafeature (Feat_b) ∧ isafixtureintent (I) ∧ isafixturelocator (L) ∧ owns (E, port) ∧ owns (E, fixt) ∧ locates (fixt, Feat_a, I, L) ⇒ specifies_tolY (Facility, E, Feat_a, Feat_b) = Y • Facility portT* port *specifies_tol (Feat_a, Feat_b))

∃ Facility, ∀ E, ∀ Part, ∀ Feat_a, ∀ Feat_b, ∀ I, ∀ L (isafacility (Facility) ∧ isanequipment (E) ∧ isapart (Part) ∧ isafixture (fixt) ∧ isafeature (Feat_a) ∧ isafeature (Feat_b) ∧ isafixtureintent (I) ∧ isafixturelocator (L) ∧ owns (E, port) ∧ owns (E, fixt) ∧ locates (fixt, Feat_a, I, L) ⇒ specifies_tolZ (Facility, E, Feat_a, Feat_b) = Z • Facility portT* port *specifies_tol (Feat_a, Feat_b))

Functions that return the components of tolerances between closed fixtures and located features in the equipment coordinate frame were used in Chapter 4 to define the criteria for valid material processing operations. These functions transform the built-in tolerance specification between a fixture and a feature that it locates into the x, y, and z components of a tolerance vector defined in the equipment coordinate system. For a given equipment entity E that holds a fixture fixt that locates a feature feat, these functions are close_tolX (E, fixt, feat), close_tolY (E, fixt, feat), and close_tolZ (E, fixt, feat). The tolerance vector is computed by transforming the locator vector of a fixture that locates one of the features into the equipment coordinate frame. This transformation assumes that the locator vector is parallel to the vector between the features. The locator vector is scaled by the tolerance between fixture fixt and feature feat specified in the fixture model and returned by the function close_tol (fixt, feat). Once the tolerance vector is computed, the dot products of x, y, and z vectors with
the tolerance vector are the components of the tolerance vector. The equations to compute these tolerance functions are as follows for each x, y, and z coordinate axis of the equipment frame.

$$\forall E, \forall fixt, \forall feat, \forall port, \forall I, \forall L (\text{isanequipment} (E) \land \text{isapart} (\text{Part}) \land \text{isafixture} (\text{fixt}) \land \text{isafeature} (\text{feat}) \land \text{isafixtureintent} (I) \land \text{isafixturelocator} (L) \land \text{owns} (E, \text{port}) \land \text{locates} (\text{fixt}, \text{feat}, I, L) \Rightarrow \text{close_tol}X (E, \text{fixt}, \text{feat}) = X \bullet \text{port}T \bullet \text{port} \text{fixt}T \bullet \text{fixt}V \text{close, L} \bullet \text{close_tol} (\text{fixt}, \text{feat}))$$

$$\forall E, \forall fixt, \forall feat, \forall port, \forall I, \forall L (\text{isanequipment} (E) \land \text{isapart} (\text{Part}) \land \text{isafixture} (\text{fixt}) \land \text{isafeature} (\text{feat}) \land \text{isafixtureintent} (I) \land \text{isafixturelocator} (L) \land \text{owns} (E, \text{port}) \land \text{locates} (\text{fixt}, \text{feat}, I, L) \Rightarrow \text{close_tol}Y (E, \text{fixt}, \text{feat}) = Y \bullet \text{port}T \bullet \text{port} \text{fixt}T \bullet \text{fixt}V \text{close, L} \bullet \text{close_tol} (\text{fixt}, \text{feat}))$$

$$\forall E, \forall fixt, \forall feat, \forall port, \forall I, \forall L (\text{isanequipment} (E) \land \text{isapart} (\text{Part}) \land \text{isafixture} (\text{fixt}) \land \text{isafeature} (\text{feat}) \land \text{isafixtureintent} (I) \land \text{isafixturelocator} (L) \land \text{owns} (E, \text{port}) \land \text{locates} (\text{fixt}, \text{feat}, I, L) \Rightarrow \text{close_tol}Z (E, \text{fixt}, \text{feat}) = Z \bullet \text{port}T \bullet \text{port} \text{fixt}T \bullet \text{fixt}V \text{close, L} \bullet \text{close_tol} (\text{fixt}, \text{feat}))$$

Functions that return the components of tolerances between open or closed fixtures and located features in the facility coordinate frame were used in Chapter 4 to define the criteria for valid material handling operations. These functions transform the built-in tolerance specification between a fixture and a feature that it locates into the x, y, and z components of a tolerance vector defined in the facility coordinate system. For a given Facility element with an equipment entity E that holds a fixture fixt that locates a feature feat in the closed state, these functions are close_tolX (Facility, E, fixt, feat), close_tolY (Facility, E, fixt, feat), and close_tolZ (Facility, E, fixt, feat). When fixtures are open, these functions are open_tolX (Facility, E, fixt, feat), open_tolY (Facility, E, fixt, feat), and open_tolZ (Facility, E, fixt, feat). The tolerance vector is computed by transforming the locator vector of a fixture that locates one of the features into the Facility element’s coordinate frame. This transformation assumes that the locator vector is parallel to the vector between the features. The locator vector is scaled by the tolerance between fixture fixt and feature feat specified in the fixture model and returned by the function close_tol (fixt, feat) or open_tol (fixt, feat). Once the tolerance vector is computed, the dot products of x, y, and z vectors with the tolerance vector are the components of the tolerance vector. The equations to compute these tolerance functions are as follows for each x, y, z coordinate axis of the Facility element’s frame.

$$\forall E, \forall fixt, \forall feat, \forall port, \forall I, \forall L (\text{isanequipment} (E) \land \text{isapart} (\text{Part}) \land \text{isafixture} (\text{fixt}) \land \text{isafeature} (\text{feat}) \land \text{isafixtureintent} (I) \land \text{isafixturelocator} (L) \land \text{owns} (E, \text{port}) \land \text{locates} (\text{fixt}, \text{feat}, I, L) \Rightarrow \text{close_tol}X (\text{Facility}, E, \text{fixt}, \text{feat}) = X \bullet \text{Facility}_\text{port}T \bullet \text{port} \text{fixt}T \bullet \text{fixt}V \text{close, L} \bullet \text{close_tol} (\text{fixt}, \text{feat}))$$

$$\forall E, \forall fixt, \forall feat, \forall port, \forall I, \forall L (\text{isanequipment} (E) \land \text{isapart} (\text{Part}) \land \text{isafixture} (\text{fixt}) \land \text{isafeature} (\text{feat}) \land \text{isafixtureintent} (I) \land \text{isafixturelocator} (L) \land \text{owns} (E, \text{port}) \land \text{locates} (\text{fixt}, \text{feat}, I, L) \Rightarrow \text{open_tol}X (\text{Facility}, E, \text{fixt}, \text{feat}) = X \bullet \text{Facility}_\text{port}T \bullet \text{port} \text{fixt}T \bullet \text{fixt}V \text{span, L} \bullet \text{open_tol} (\text{fixt}, \text{feat}))$$
∀E, ∀fixt, ∀feat, ∀port, ∀I, ∀L (isanequipment (E) ∧ isaport (port) ∧ isafixture (fixt) ∧ isafixtureintent (I) ∧ isafixturelocator (L) ∧ owns (E, port) ∧ locates (fixt, feat, I, L) ⇒ close_tolY (Facility, E, fixt, feat) = Y • Facility • portT * portT * fixtT • V_{close, Z} * close_tol (fixt, feat))

∀E, ∀fixt, ∀feat, ∀port, ∀I, ∀L (isanequipment (E) ∧ isaport (Part) ∧ isafixture (fixt) ∧ isafeature (feat) ∧ isafixtureintent (I) ∧ isafixturelocator (L) ∧ owns (E, port) ∧ locates (fixt, feat, I, L) ⇒ open_tolY (Facility, E, fixt, feat) = Y • Facility • portT * portT * fixtT * V_{open, Y} * open_tol (fixt, feat))

∀E, ∀fixt, ∀feat, ∀port, ∀I, ∀L (isanequipment (E) ∧ isaport (Part) ∧ isafixture (fixt) ∧ isafeature (feat) ∧ isafixtureintent (I) ∧ isafixturelocator (L) ∧ owns (E, port) ∧ locates (fixt, feat, I, L) ⇒ close_tolZ (Facility, E, fixt, feat) = Z • Facility • portT * portT * fixtT • V_{close, Z} * close_tol (fixt, feat))

∀E, ∀fixt, ∀feat, ∀port, ∀I, ∀L (isanequipment (E) ∧ isaport (Part) ∧ isafixture (fixt) ∧ isafeature (feat) ∧ isafixtureintent (I) ∧ isafixturelocator (L) ∧ owns (E, port) ∧ locates (fixt, feat, I, L) ⇒ open_tolZ (Facility, E, fixt, feat) = Z • Facility • portT * portT * fixtT * V_{open, Z} * open_tol (fixt, feat))

Functions that return the components of tolerances between the fixtures’ positions relative to equipments’ ports in the equipment coordinate frame were used in Chapter 4 to define the criteria for valid material processing operations. These functions transform the built-in tolerance specification between a port and a fixture located in the port into the x, y, and z components of a tolerance vector defined in the equipment coordinate system. For a given equipment entity E that owns a port port with a fixture fixt, these functions are fixture_pos_tolX (E, port, fixt), fixture_pos_tolY (E, port, fixt), and fixture_pos_tolZ (E, port, fixt). A tolerance vector defined in the port frame is equivalent to the sum of the vector components returned by the functions fixture_pos_tolX (port, fixt), fixture_pos_tolY (port, fixt), and fixture_pos_tolZ (port, fixt). Once this tolerance vector is computed, it is transformed into the equipment frame using $E_{portT}$. Lastly, the dot products of x, y, and z vectors with the transformed tolerance vector are the components of the tolerance vector defined in the equipment frame. The equations to compute these tolerance functions are as follows for each x, y, z coordinate axis of the equipment frame.

∀E, ∀port, ∀fixt (isanequipment (E) ∧ isaport (port) ∧ isafixture (fixt) ∧ owns (E, fixt) ∧ owns (E, port) ⇒ fixture_pos_tolX (E, port, fixt) = X • E_{portT} * (X • fixture_pos_tolX (port, fixt) + Y • fixture_pos_tolY (port, fixt) + Z • fixture_pos_tolZ (port, fixt)))

∀E, ∀port, ∀fixt (isanequipment (E) ∧ isaport (port) ∧ isafixture (fixt) ∧ owns (E, fixt) ∧ owns (E, port) ⇒ fixture_pos_tolY (E, port, fixt) = Y • E_{portT} * (X • fixture_pos_tolX (port, fixt) + Y • fixture_pos_tolY (port, fixt) + Z • fixture_pos_tolZ (port, fixt)))

∀E, ∀port, ∀fixt (isanequipment (E) ∧ isaport (port) ∧ isafixture (fixt) ∧ owns (E, fixt) ∧ owns (E, port) ⇒ fixture_pos_tolZ (E, port, fixt) = Z • E_{portT} * (X • fixture_pos_tolX (port, fixt) + Y • fixture_pos_tolY (port, fixt) + Z • fixture_pos_tolZ (port, fixt)))
These functions that return the components of tolerance between fixtures’ positions in the facility coordinate system were used in Chapter 4 to define the criteria for valid material handling operations. These functions transform the built-in tolerance specification between a port and a fixture located in the port into the x, y, and z components of a tolerance vector defined in the facility coordinate system. For a given facility with an equipment entity \( E \) that owns a port \( \text{port} \) with a fixture \( \text{fixt} \), these functions are \( \text{fixture\_pos\_tolX}(E, \text{port}, \text{fixt}) \), \( \text{fixture\_pos\_tolY}(E, \text{port}, \text{fixt}) \), and \( \text{fixture\_pos\_tolZ}(E, \text{port}, \text{fixt}) \). A tolerance vector defined in the port frame is equivalent to the sum of the vector components returned by the functions \( \text{fixture\_pos\_tolX}(\text{port}, \text{fixt}) \), \( \text{fixture\_pos\_tolY}(\text{port}, \text{fixt}) \), and \( \text{fixture\_pos\_tolZ}(\text{port}, \text{fixt}) \). Once this tolerance vector is computed, it is transformed into the facility frame using \( \text{Facility}_{\text{portT}} \). Lastly, the dot products of x, y, and z vectors with the transformed tolerance vector are the components of the tolerance vector defined in the facility frame. The equations to compute these tolerance functions are as follows for each x, y, and z coordinate axis of the facility frame.

\[
\exists \text{Facility}, \forall E, \forall \text{port}, \forall \text{fixt} \ (\text{isafacility}(\text{Facility}) \land \text{isanequipment}(E) \land \text{isaport}(\text{port}) \land \text{isafixture}(\text{fixt}) \land \text{owns}(E, \text{fixt}) \land \text{owns}(E, \text{port}) \Rightarrow \text{fixture\_pos\_tolX}(\text{Facility}, \text{port}, \text{fixt}) = X \cdot \text{Facility}_{\text{portT}} \cdot (X \cdot \text{fixture\_pos\_tolX}(\text{port}, \text{fixt}) + Y \cdot \text{fixture\_pos\_tolY}(\text{port}, \text{fixt}) + Z \cdot \text{fixture\_pos\_tolZ}(\text{port}, \text{fixt}))
\]

\[
\exists \text{Facility}, \forall E, \forall \text{port}, \forall \text{fixt} \ (\text{isafacility}(\text{Facility}) \land \text{isanequipment}(E) \land \text{isaport}(\text{port}) \land \text{isafixture}(\text{fixt}) \land \text{owns}(E, \text{fixt}) \land \text{owns}(E, \text{port}) \Rightarrow \text{fixture\_pos\_tolY}(\text{Facility}, \text{port}, \text{fixt}) = Y \cdot \text{Facility}_{\text{portT}} \cdot (X \cdot \text{fixture\_pos\_tolX}(\text{port}, \text{fixt}) + Y \cdot \text{fixture\_pos\_tolY}(\text{port}, \text{fixt}) + Z \cdot \text{fixture\_pos\_tolZ}(\text{port}, \text{fixt}))
\]

\[
\exists \text{Facility}, \forall E, \forall \text{port}, \forall \text{fixt} \ (\text{isafacility}(\text{Facility}) \land \text{isanequipment}(E) \land \text{isaport}(\text{port}) \land \text{isafixture}(\text{fixt}) \land \text{owns}(E, \text{fixt}) \land \text{owns}(E, \text{port}) \Rightarrow \text{fixture\_pos\_tolZ}(\text{Facility}, \text{port}, \text{fixt}) = Z \cdot \text{Facility}_{\text{portT}} \cdot (X \cdot \text{fixture\_pos\_tolX}(\text{port}, \text{fixt}) + Y \cdot \text{fixture\_pos\_tolY}(\text{port}, \text{fixt}) + Z \cdot \text{fixture\_pos\_tolZ}(\text{port}, \text{fixt}))
\]

5.4 Operations Plan Graph Generation

5.4.1 Introduction

Section 5.2 described a high level view of techniques for generating an OR search graph for a part while Section 5.3 defined the lower level algorithms used by the high level planner. The OR search graph represents different sequences of operations that result in the finished part located at the finished parts buffer. One OR search graph without alternatives is illustrated in Figure 5.6. However, these search graphs are not efficient to
visualize or to store because there may be duplication in alternative branches and there may be OR branches that are sequence independent. Mettala (1989) shows that an AND/OR graph representation of process plans is efficient and suitable for controlling production. Thus, techniques have been developed for this dissertation to convert the OR search graph into an AND/OR operations plan. This section describes how the OR search graph is converted into an OR operations plan, then how redundant nodes are eliminated from this OR operations graph, and lastly how the graph is simplified by converting OR nodes whose branches fit all possible sequences into simpler AND nodes. The resulting output is an AND/OR operations plan graph that is defined according to the description from Mettala (1989). This operations plan graph consists of AND nodes, OR nodes, and operation nodes.

5.4.2 OR Graph Trace Algorithm

The initial OR operations plan graph is generated from the OR search graph using techniques that are illustrated in the flow chart in Figure 5.11. This procedure consists of tracing through the search graph depth-first and replacing sequential children nodes with a simple sequence of nodes starting with the left child node and ending with the right child node. Only nodes representing operations are copied into the operations plan graph. OR nodes in the search graph translate directly into OR nodes in the operations plan graph. Since there are nodes with multiple parents in the search graph due to repeated sub-problem descriptions, operation nodes may be duplicated in this initial OR operations plan graph. Figure 5.12 illustrates the linear operations plan that corresponds to the OR search graph in Figure 5.6 for the example manufacturing scenario in Chapter 4. Since there are no alternative branches in the graph in Figure 5.6, the resulting operations plan is linear without alternatives. Operations plans generated in this manner from OR search graphs with alternatives may result in duplication of the same nodes in alternative branches. Applying the algorithm to the OR search graph in Figure 5.9 results in an operations plan with alternatives that is illustrated in Figure 5.13. The nodes representing operators move$_{3,4}$ and unload$_{4,7}$ are duplicated at the beginning of both OR branches in the operations plan. For efficient storage of the operations plan graph, an algorithm has been developed that removes such redundancy.
5.4.3 Elimination of Redundant Nodes from OR Operations Plan Graph

Due to the requirement to minimize the number of nodes in the operations plan graph, an algorithm has been developed that checks for duplicated nodes at the beginning and end of OR branches. An example of the output of this algorithm is a more compact operations plan graph derived from the operations plan graph in Figure 5.13. This more compact operations plan graph is displayed in Figure 5.14. Note that the duplicated operation nodes move3,4 and unload4,7 are no longer present on both OR branches but form a single sequence in Figure 5.14. This...
algorithm checks through all branches of each OR node in the graph, starting with the OR nodes deepest in the graph structure. If there exists the same sequence of nodes at the beginning of all OR branches stemming from an OR node, then these nodes are removed from the OR branches and placed in sequence in front of the OR node. Similarly, if there exists the same sequence of nodes at the end of all OR branches stemming from an OR node, then these nodes are removed from the OR branches and placed in sequence in back of the OR node.

Figure 5.14. Operations Plan without duplication.

5.4.4 Addition of AND Nodes to OR Operations Plan Graph

As a consequence of the previous procedures, operations plan graphs are generated that are not minimized through the use of AND branches. Based on the definition of an AND/OR graph in Mettala (1989), such AND branches specify that the order of execution of the nodes between branches is not deterministic. In contrast, the order of execution of the nodes in the same AND branch is deterministic. For instance, if nodes A, B, and C are nodes in three different AND branches, then six execution sequences are possible: A, B, C; A, C, B; B, A, C; B, C, A; C, A, B; C, B, A. Starting with the deepest OR node, the algorithm to convert OR branches into AND branches checks each OR node to see if the nodes in its branches have some non-deterministic sequence. For instance, if there is an OR node with six branches with the six execution sequences listed above, then this OR node can be converted into an AND node with three children branches consisting of A, B, and C. Mettala (1989) presents an algorithm (Algorithm 4-3 – “AND” Graph Constructor) that constructs an AND graph from a list of execution sequences if possible. This algorithm checks if the nodes in OR branches can be merged into an AND node and if so, it generates the children of the new AND node and replaces the original OR node with this new AND node. This algorithm performs this check by creating a table of possible operation precedences from one
OR branch and marking precedence pairs whose precedence is violated by other operation branches of the OR node. If any precedence pairs remain after this elimination, two AND branches may be defined.

To illustrate how AND nodes can be extracted from an OR operations plan graph, assume that there is a requirement to plan for the manufacture of a part with goal featuresFeat_7andFeat_8using the resource model from Chapter 4. The part is a modification of the part illustrated in Figure 4.9. Feat_7 and Feat_8 do not have any precedence and they are located on the same surfaceFeat_1of the part. Furthermore, they are both specified by datums Feat_2, Feat_3, and Feat_5. Figure 5.15 illustrates the OR search graph generated for this part while Figure 5.16 illustrates the OR operations plan graph generated from the search graph. The results of applying the algorithm to remove duplicate nodes from the OR operations plan graph is the graph illustrated in Figure 5.17. The AND node algorithm generates the compact operation plan graph illustrated in Figure 5.18. For this example, an AND node replaces the OR node because the OR node has two branches with the following node sequences: 2, 3; 3, 2. Since these two sequences represent every possible sequence of nodes 2 and 3, the OR node is replaced by an AND node with children branches of nodes 2 and 3.

Figure 5.15. Search Graph with AND Sequence of Features.
5.5 Chapter Summary

This chapter has described the planning techniques for generating an AND/OR operations plan graph that merge material handling and material processing operations. First, the high level planning algorithm generates an OR search graph from the resource and part models. This planning algorithm checks lower level constraints that
deal with geometry, tolerances, and transformation. Lastly, the techniques for converting this OR search graph into a compact and useful AND/OR operations plan graph have been presented. This planner integrates product design with production constraints because new plans can quickly be generated from a part model. Once in production, this data-driven plan is useful because simulation models may use this plan to quickly test out production capacity for new resources and products, and production control systems may use this plan to control flexible manufacturing systems with less startup problems. Furthermore, bottleneck control systems and schedulers can quickly utilize the physical capability of shop floor resources with this plan to increase production throughput using auxiliary buffers. Lastly, the tight integration of this planner with the resource model means that the production capacity of new resources can quickly be estimated and incorporated into the shop floor through the generation of new operation plans that use the new resources.
Chapter 6

MATERIAL HANDLING TASK PLANNING METHODOLOGY

This chapter explains how tasks for robotic material handlers can be planned at the kinematic trajectory level using the resource model. This description is significant because it links progress in robot path planning research with manufacturing resource modeling and operations planning. Section 1 introduces the chapter by giving an overview of path planning techniques and the specific solution implemented by this research. Section 2 defines the problems solved by material handling trajectory planning. These problems are represented by the beginning and end of trajectories to load and unload parts. Section 3 describes how potential fields may be computed which model the desired attraction of the robot towards the goal while robot is being repulsed by obstacles. Section 4 describes the search strategy used with these potential fields to generate collision free trajectories for material handling robot resources.

6.1 Introduction

In this model, both robot and human material handling resources can be defined to have the capability to load and unload parts to and from buffers. For a given resource, $E_i$, this capability is defined in Chapter 4 by the predicates $loadable$ (Part, $E_i$, port, fixt) and $unloadable$ (Part, $E_i$, port, fixt). From Chapter 4, these functions can only be true if certain conditions are true. These conditions are completely specified in Chapter 4 except for the truth of $unloadTrajExists$ ($E_i$, gripper, fixt, port_b, Part) and $loadTrajExists$ ($E_i$, gripper, fixt, port_l, Part). For human material handlers, it is assumed that trajectories always exist for loading or unloading parts if the part is below a maximum threshold of weight. For robotic material handlers, these conditions are only true if it is possible to compute collision-free trajectories that satisfy the requirements of unload or load tasks. In the literature, this type of planning is referred to as task-level robot planning (Lozano-Perez, 1983). Son (1998) decomposes task-level robot planning into unload and load tasks. For example, task-level robot planning for an
unload operation is divided into the following planning tasks: determining an appropriate grasp of the part by the robot gripper; planning a gross motion trajectory from the robot’s initial position to the general area of the part; planning a fine motion trajectory to a grasp position; grasping the part; planning a fine motion trajectory away from the part’s fixture with the part; and then planning a gross motion trajectory to a safe robot position.

The motion planning tasks are difficult because the typical robot arm has many degrees of freedom that must be exploited in order to move around obstacles in the robot’s workspace. The most successful robot motion planning algorithms can be divided into configuration space algorithms and workspace algorithms. The configuration space of a robot arm is a space where each point represents a different set of joint position values. Configuration space algorithms first convert the obstacles in the workspace into their equivalent obstacles in the robot’s configuration space. This is accomplished by computing which joint configurations of the robot result in collisions with the robot’s obstacles. Lozano-Perez (1983) and Donald (1984) have contributed methods for computing these obstacles in configuration space. Given this decomposition of obstacles in configuration space, the planning task becomes a simpler task of moving a point from the initial configuration of the robot to the goal configuration without intersecting with polyhedral obstacles. The goal configuration can be computed from the goal position of the end effector through the use of an inverse kinematics algorithm for the robot. Problems with this approach are that it becomes computationally intensive for high degree of freedom configuration spaces and that there may exist multiple inverse kinematic solutions if the robot has redundant degrees of freedom. On the other hand, workspace planning techniques consist of planning the motion of the robot in the normal 3D workspace. These techniques use the proximity of the robot’s end effector to the goal position to determine how to move the robot to the goal and rely on collision checking between the robot’s body and obstacles in the 3D workspace to determine where not to move. Barraquand & Latombe (1990) have developed solutions using this approach. The main problem with the workspace planning approach is that since an exact mapping between configuration space and workspace has not been made, the planner takes an unknown amount of time to find a solution.
In general, the grasp planning task is also difficult because there may be many different ways for a robot’s gripper to grasp the part and because each different grasp requires a different robot end effector position. Grasp planning consists of matching the part geometry with the gripper geometry and ensuring that the grasp selected results in a robot position that is attainable without collision in the unload and load positions. Furthermore, there must exist obstacle-free trajectories for the robot material handler to unload and load the part using this grasp. Troccaz (1988) has developed algorithms for determining grasp locations for grippers on parts while Lozano-Perez and colleagues (1987) have developed algorithms for grasping the part and moving it.

In this work, one main contribution is how robotic planning techniques are applied to flexible manufacturing. This is important because the models for using the aforementioned robot planning algorithms in industry do not exist. Thus, this work incorporates algorithm for gross motion trajectory generation using workspace planning developed by Barraquand & Latombe (1990) in this operations planning architecture. This algorithm first computes potential fields computed in the robot’s workspace based on the initial positions of control points on the robot gripper, the obstacles in the workspace, and the goal positions of control points on the gripper. The 3D workspace is discretized into a rectangloid grid where array values representing cubes that intersect with obstacles are assigned the value of 1. Array values representing free space cubes are assigned the value of 0. The difference between gross and fine motion is the resolution of this discretization of the workspace. A potential field may be defined over the workspace using an algorithm to assign values to an array whose elements represent cubes of the workspace. If the goal position for a control point lies within a specific cube, then that cube becomes a goal cube for the control point. Each potential field that corresponds to a control point has only one minimum at the control point’s goal cube. The robot’s path is determined by guiding a control point to follow the decreasing gradient of these potential fields away from obstacles towards its goal position. For each control point, potential fields are computed for the gross and fine grids that discretize the workspace.

Given an initial joint configuration, the gross motion algorithm searches by finding the best neighboring configuration for each step that decreases the potentials of control points on the end effector. Competition between satisfying different goals due to the different control points is resolved using an arbitration function that is the sum of all of the control point potentials. The joint configuration change during each planning step is
minimized so that no part on the robot will move more than one workspace cube. Potential configurations that result in robot collisions with obstacles are not used. This search continues until the robot’s configuration is in a local minimum such that no neighboring configurations decrease the potential field value of the control points. If any of the control points are one cube away from its goal cube, then the gross motion search is finished. Otherwise, random motions are used to move the robot’s configuration out of the local minimum and continue searching. Once a configuration is reached so that at least one control point on the robot is close to its goal position, other searches are done using increasingly finer resolution of space until all control points are at their goal positions. These fine motion searches results in sequences of joint value configurations that represent smaller joint value changes per configuration than the gross motion trajectory. The final trajectory is the gross motion trajectory followed by the fine motion trajectories.

6.2 Material Handling Problem Statement

6.2.1 Introduction

In this model, the robot task planning algorithm is incorporated by using the fixture intention of the robot gripper to define the grasp position for a given part. This grasp position defines the goal positions for the end effector for trajectories that unload and load the part. The geometric models of equipment that intersect with the workspace of the robot represent obstacles that must be avoided by the robot while it executes these trajectories. Lastly, as proposed by Lozano-Perez (1983), for each desired trajectory, a gross motion trajectory that moves the robot’s end effector close to the goal position is first computed and then a series of fine motion trajectories that brings the end effector exactly to the correct position within robot tolerances are computed.

6.2.2 Trajectory Goals

In this research, material handling for robot entities is divided into load and unload tasks. For the unload task, a robot moves its open gripper from its home position to a position determined by the predefined intent of its gripper-fixture and the part’s location in an equipment fixture at a port. This intent defines how the gripper locates the datums of the part. Next, the robot moves its closed gripper with the part from this location back to its
home position. Thus, for the unload task there are two trajectories to be planned. The function, \( unloadTrajExists \) (\( E_i, \) gripper, \( fixt, \) port\( _b \), Part) is true if these two trajectories exist. The three locator points on the gripper-fixture that grasps the part are used to defined this trajectory. In the robot port’s coordinate frame, the beginning and ending values of these points are defined as follows for robot entity \( R \), robot port \( port_1 \), equipment port \( port_2 \), gripper fixture \( fixt_1 \), and equipment fixture \( fixt_2 \) located in a facility \( Facility \). Note that for a robot entity \( R \), when the gripper-fixture is in its home position, the gripper-fixture frame’s location with respect to the robot’s port frame is defined by the transformation \( port_1fixt_1T = home_trans (R) \). Furthermore, when the fixture is in an unload position, the gripper-fixture frame’s location is defined with respect to the robot’s port frame by the transformation \( port_1fixt_1T \) as computed by the algorithm in Section 5.3.3.

Trajectory #1  

begin:  
\[
\begin{align*}
\text{port}_1\text{Popen, 1} &= home_trans (R)* fixt_1\text{Popen, 1} \\
\text{port}_1\text{Popen, 2} &= home_trans (R)* fixt_1\text{Popen, 2} \\
\text{port}_1\text{Popen, 3} &= home_trans (R)* fixt_1\text{Popen, 3}
\end{align*}
\]

end:  
\[
\begin{align*}
\text{port}_1\text{Popen, 1} &= port_1fixt_1T* fixt_1\text{Popen, 1} \\
\text{port}_1\text{Popen, 2} &= port_1fixt_1T* fixt_1\text{Popen, 2} \\
\text{port}_1\text{Popen, 3} &= port_1fixt_1T* fixt_1\text{Popen, 3}
\end{align*}
\]

Trajectory #2  

begin:  
\[
\begin{align*}
\text{port}_1\text{Pclose, 1} &= port_1fixt_1T* fixt_1\text{Pclose, 1} \\
\text{port}_1\text{Pclose, 2} &= port_1fixt_1T* fixt_1\text{Pclose, 2} \\
\text{port}_1\text{Pclose, 3} &= port_1fixt_1T* fixt_1\text{Pclose, 3}
\end{align*}
\]

end:  
\[
\begin{align*}
\text{port}_1\text{Pclose, 1} &= home_trans (R)* fixt_1\text{Pclose, 1} \\
\text{port}_1\text{Pclose, 2} &= home_trans (R)* fixt_1\text{Pclose, 2} \\
\text{port}_1\text{Pclose, 3} &= home_trans (R)* fixt_1\text{Pclose, 3}
\end{align*}
\]

For the load task, a robot moves its closed gripper with the part from its home position to a position determined by the predefined part location in the equipment fixture at a port. Next, the robot moves its open gripper without the part from this position to its home position. Thus, for the unload task there are two trajectories to be planned. The function, \( loadTrajExists \) (\( E_i, \) gripper, \( fixt, \) port\( _b \), Part) is true if these two trajectories exist. The three locator points on the gripper-fixture that grasps the part are used to defined this trajectory. In the robot port’s coordinate frame, the initial and final values of these points are defined as follows. Note that for a robot entity \( R \), when the gripper-fixture is in its home position, the gripper-fixture frame’s location is defined with respect to the robot’s port frame by the transformation \( port_1fixt_1T = home_trans (R) \). Furthermore, when the fixture is in a load position, the gripper-fixture frame’s location is defined with respect to the robot’s port frame by the transformation \( port_1fixt_1T \) as computed by the algorithm in Section 5.3.3.
Trajectory #1

begin: $\text{port1Pclose, 1} = \text{home_trans (R)} \cdot \text{fixt1Pclose, 1}$
$\text{port1Pclose, 2} = \text{home_trans (R)} \cdot \text{fixt1Pclose, 2}$
$\text{port1Pclose, 3} = \text{home_trans (R)} \cdot \text{fixt1Popen, 3}$

end: $\text{port1Pclose, 1} = \text{home_trans (R)} \cdot \text{fixt1Pclose, 1}$
$\text{port1Pclose, 2} = \text{home_trans (R)} \cdot \text{fixt1Pclose, 2}$
$\text{port1Pclose, 3} = \text{home_trans (R)} \cdot \text{fixt1Pclose, 3}$

Trajectory #2

begin: $\text{port1Popen, 1} = \text{home_trans (R)} \cdot \text{fixt1Popen, 1}$
$\text{port1Popen, 2} = \text{home_trans (R)} \cdot \text{fixt1Popen, 2}$
$\text{port1Popen, 3} = \text{home_trans (R)} \cdot \text{fixt1Popen, 3}$

end: $\text{port1Popen, 1} = \text{home_trans (R)} \cdot \text{fixt1Popen, 1}$
$\text{port1Popen, 2} = \text{home_trans (R)} \cdot \text{fixt1Popen, 2}$
$\text{port1Popen, 3} = \text{home_trans (R)} \cdot \text{fixt1Popen, 3}$

6.2.3 Workspace

The trajectories that are based on the preceding goals are ordered sequences of joint angle value configurations for a given robot arm. Each one of these joint angle configurations must result in a robot position such that none of the robot’s geometry intersects with the geometry of equipment in the robot’s workspace. A circle around the robot’s base with a given radius defines the robot’s workspace. The position of equipment on the factory floor determines if equipment is in the robot’s workspace. In general, if the geometric model of an equipment entity intersects with a cylinder defined by the 2D robot workspace circle, then the equipment is in the robot’s workspace. All obstacles in the workspace are checked for collisions during gross motion trajectory planning. The workspace grid for gross motion planning is generated using a predefined spatial resolution. During fine motion planning, only a subspace of the workspace is checked for collisions. This subspace consists of the area around the initial positions and the target positions of the control points on the robot end effector. Thus, the workspace for fine motion planning only includes obstacles due to the geometrical model of the buffer being unloaded or loaded and the fixture. The workspace grid for fine motion planning is generated using different spatial resolutions that are closer to the accuracy of the robot’s end effector placement.
6.3 Potential Field Generation for Trajectory Planning

6.3.1 Introduction

Khatib (1986) first proposed to use potential fields for robotics as part of a real time trajectory control system that guides robots around unexpected obstacles. Lengyel and colleagues (1990) used potential fields to plan the paths of robots around known obstacles. The algorithm developed by Lengyel and colleagues (1990) uses a wave front expansion algorithm on a rectangloid grid to generate a potential field with one minimum at the goal cube. This potential field is represented by a table lookup function that has a specific value for workspace locations. A disadvantage of this potential field is that the gradient of the potential decreases towards the goal but it does not decrease away from obstacles. Thus, a point following the negative gradient of this potential will graze along the boundaries of obstacles towards the goal. While this is sufficient for planning the motion of a single point occupying one workspace cube at a time, if the point is attached to a robot arm it is desirable to keep the point away from obstacles. Thus, a different potential that was developed by Barranquand & Latombe (1990) and titled the Improved Numerical Navigation Function by Latombe (1991) is selected for this research. This potential’s gradient decreases away from obstacles as well as towards the goal. A cube in the workspace grid that intersects with the goal position represents this goal. The potential function is computed by first generating a list of workspace cubes that lie between obstacles in the workspace. In a 2D workspace, the skeleton that this list represents is known as a Voronoi Diagram [O’Dunlaing et al, 1984]. Next, the workspace cube representing the goal position is connected to this skeleton with an extension to the skeleton. Next, the potential field is computed for the cubes in the skeleton based on the distance from the goal. Lastly, the potential field in the rest of the free space is computed by moving wave fronts from the obstacle boundaries and decreasing their potentials until the waves reach the skeleton.

6.3.2 Workspace Grid Computation

Before potential fields can be computed, the workspace grid must be created. The boundaries of this grid envelop the robot’s workspace. This grid has a coordinate frame that is specified relative to the robot’s port coordinate frame. The geometric models of equipment, floors, and walls are obstacles in the robot workspace. The size of the grid cubes is determined by the size of obstacles in the environment. Since any intersection between an
obstacle and a grid cube removes the cube from free space, it is necessary to make the cubes small enough such that there exists some free space in which to plan robot motion. However, if the cubes are too small, excessive computation is required to plan the motion of the robot. The size of workspace cubes are defined in the model that describes the robot resource.

Given established boundaries for the workspace grid and a set resolution for the size of the grid cubes, a 3-dimensional array can be computed that represents this grid. For each array element representing a grid cube, the element is set to 1 if the grid cube intersects with any geometrical model of obstacles in the workspace. Boundary representations (B-Rep) are used to define solid geometries of elements. B-Rep uses the topological notion that a physical element is bounded by a set of faces [Zeid, 1991]. The solid geometries of these obstacles are represented by B-Reps in their resource models. For instance, the Facility resource entity has representations of the floor and walls in its model. Each equipment resource also has a geometric model described by boundary representation. Intersection tests can be performed between the cubes in the workspace grid and the obstacles by transforming the B-Rep models of the obstacles into the workspace coordinate frame and comparing coordinate values in all 3 dimensions. Thus, in order to build the grid array, each cube in the workspace is checked to see if it intersects with any obstacle in the workspace. If there is an intersection, then the corresponding array element is set to 1.

### 6.3.3 Skeleton Computation

Given the grid array that models free space and obstacles in the workspace, the list of workspace cubes that define the skeleton may be generated. The following is an algorithm adapted from Latombe (1991) that generates this skeleton list $S$ from the grid array $G$. Note that for a given $p$ that represents a grid cube, $G(p)$ returns 0 or 1 depending on whether the cube is in free space or not. In this algorithm, a distance called the $L^1$ or Manhattan distance, is used to define the distance between cubes in the grid. This distance is the number of cubes that must be traveled through in order for a point on the robot to move from one cube to another. This algorithm first initializes the potential function, $U(p)$, to infinity for every position $p$ in free space. Next, the algorithm inserts all boundary obstacle cubes into the list $L_0$. This algorithm works by moving wave fronts of grid cubes away from obstacles and the skeleton forms where these waves meet. Each step of these waves is represented by a $L_t$ list.
where $L_0$ is the first step of the waves. Each subsequent step of these waves is represented by a list of cubes that are direct neighbors of the previous cubes. Note that cubes that are direct neighbors of each other share a face in common. Using these wave lists, this algorithm computes a function, $d_1(p)$, which returns the $L^1$ distance from the grid cube $p$ to its nearest obstacle.

**Skeleton Algorithm**

For every $p$ such that $G(p) = 0$

- $U(p) \leftarrow \infty$
- $d_1(p) \leftarrow 0$

// $L_i$, $i \leftarrow 0, 1, 2, \ldots$, is a list of grid cubes that is initially empty

for every $p$ such that $G(p) = 1$

- if there exists a direct neighbor $p'$ of $p$ such that $G(p') = 0$
  - $d_1(p) \leftarrow 0$
  - $O(p) \leftarrow p$
  - insert $p$ at the end of $L_0$

for $i=0, 1, \ldots$, until $L_i$ is empty

for every $p$ in $L_i$

- for every direct neighbor $p'$ of $p$ such that $G(p') = 0$
  - if $(d_1(p') = \infty)$
    - $d_1(p') \leftarrow i + 1$
    - $O(p') \leftarrow O(p)$
    - insert $p'$ at the end of $L_{i+1}$
  - else if $L^1$ distance between $O(p')$ and $O(p) > 2$
    - if $p \notin S$ then insert $p'$ into $S$

### 6.3.4 Computation of Potential Field for Skeleton

Given the list of grid cubes that represent the skeleton of the workspace, $S$, and the distance function, $d_1$, the potential field lookup values can be computed for the skeleton grid cubes and the rest of the free space. First, this skeleton must be connected to the goal cube so that a potential can be created that has a negative gradient towards the goal. Then the potential can be computed in this augmented skeleton. The following algorithm from Latombe (1991) specifies how this potential is computed for the elements of the skeleton. In this algorithm, the goal cube is specified by $p_{goal}$. The algorithm begins by building a list $\sigma$ that consists of $p_{goal}$ and a succession of its neighbors that lead to the skeleton by always selecting the neighbor with the largest value of $d_1$. This works because $d_1$ is a measure of the distance to obstacles and the elements of the skeleton are the farthest away from obstacles. Next, the potential field, represented by the function $U$, is computed for this augmented skeleton. First, the potential at the goal cube, $U(p_{goal})$, is set to 0 because it is the minimum of the potential well. Also, this goal cube, $p_{goal}$, is added to the list $Q$ that is always sorted by ascending values of $d_1$. The potential field is computed for the
augmented skeleton by expanding out from \( p_{\text{goal}} \) to its neighbors in the skeleton and steadily increasing the potential field values based on their \( d_1 \) values. At the end of this loop, the list \( L_0 \) contains all of the cubes in the augmented skeleton that are accessible from \( p_{\text{goal}} \) and the potential field is defined for these cubes.

**Algorithm for Computing Potential Field**

// create list \( \sigma \) that contains \( p_{\text{goal}} \) and a link to the skeleton

1. Insert \( p_{\text{goal}} \) into list \( \sigma \)
2. \( p \leftarrow p_{\text{goal}} \)
3. While (\( p \notin S \))
   1. Select the direct neighbor \( p' \) of \( p \) that has the largest value of \( d_1 \)
   2. Insert \( p' \) into list \( \sigma \)
   3. \( p = p' \)
4. // add list \( \sigma \) to the skeleton \( S \)
5. Insert all elements of \( \sigma \) into \( S \)

// compute the potential field \( U \) in \( S \)

- \( U(p_{\text{goal}}) \leftarrow 0 \)
- // list \( Q \) is a list of grid cubes always sorted by ascending values of \( d_1 \). It is initially empty.
- Insert \( p_{\text{goal}} \) into list \( Q \)

// \( L_i, i \leftarrow 0, 1, 2, \ldots \), is a list of grid cubes that is initially empty

- While (\( Q \) is not empty)
  1. \( p = \text{element removed from beginning of } Q \) list
  2. Insert \( p \) at the end of \( L_0 \)
  3. For every possible neighbor \( p' \) of \( p \) in \( S \)
     1. If \( U(p') = \infty \)
        1. \( U(p') \leftarrow U(p) + 1 \)
        2. Insert \( p' \) into \( Q \)

### 6.3.5 Computation of Potential Field for Rest of Free Space

Given the results of the previous algorithm in Section 6.3.4, the potential field must still be computed for the rest of the free space grid in the workspace. The following algorithm, which is a modification of an algorithm presented by Latombe (1991), assumes that \( L_0 \) contains all of the augmented skeleton cubes and that the potential field was initialized to a large value represented by \( \infty \). This algorithm begins by assigning incremented potential field values to the neighbors of the cubes in the skeleton list. In turn, these neighbors in the \( L_1 \) list assign incremented potential field values to their neighbors. This process continues until all free space cubes in the grid that are accessible from the skeleton cubes are assigned a potential field value.

**Algorithm for Computing Potential Field in Remaining Free Space**

// compute the potential field \( U \) in the rest of the free space accessible from \( p_{\text{goal}} \)

- For \( i \leftarrow 0, 1, \ldots \), until \( L_i \) is empty
  1. For every \( p \) in \( L_i \)
for every direct neighbor \( p' \) of \( p \) such that \( G(p') = 0 \)
if \( U(p') = \infty \)
\[ U(p') \leftarrow U(p) + 1 \]
insert \( p' \) in the end of \( L_{4+1} \)

### 6.4 Search Strategy

#### 6.4.1 Introduction

According to Barraquand & Latombe (1990), it is possible to plan trajectories for robots with many degrees of freedom after workspace potential fields for the robot’s control points have been calculated. Accordingly, this research has developed models to implement the search algorithms proposed by Barraquand & Latombe (1990). Thus, the search strategy that relies on workspace potential fields to determine which joint motions result in a robot position that is closer to the goal is used. Although these workspace potentials only have one minimum value, this search process does result in a local minimum because the robot arm attached to a control point may be wrapped around an obstacle. The control point cannot continue following the workspace potential field to its minimum if the robot arm is unable to move it further. Other local minimas may result because each control point independently drives the search towards its own goal in such a manner that may prevent all control points from simultaneously reaching their goals. In such situations, a random walk is used to randomly move the robot away from the local minimum and begin searching again.

#### 6.4.2 Configuration Space Search

The configuration space search begins with an initial joint configuration, \( \Theta_{\text{init}} \), and selects the legal neighbor of \( \Theta_{\text{init}} \) that results in the lowest value for the sum of potential fields of the control points. This continues for \( \Theta_{\text{init}} \)’s successor configurations until there does not exist a legal neighbor configuration that results in a lower value for the sum of potential fields of the control points. A legal neighbor configuration is a neighboring joint configuration that does not exceed joint limits and does not cause the robot to collide with external obstacles.

For a given joint configuration, its neighbor configurations are joint values that do not move the control points by more than one workspace grid cube. Thus, the entire configuration neighborhood for a given joint configuration consists of some configurations with certain joint values increased or decreased by a delta and
others kept constant. Furthermore, the delta for rotary joints changes as a function of the joint configuration. In order to prevent a rotary joint change from moving a control point beyond one workspace cube or one cube length, the maximum rotation a joint may move is $\Delta \theta = \text{cube length}/R$, where $R$ is the distance from the center of rotation to the control point.

The sum of potential field values of the control points for a given joint configuration is computed using a kinematic description of the robot. This kinematic description, modeled using the Denevit-Hartenberg parameters (Denavit & Hartenburg, 1955), defines the transformation from the robot’s port frame to its gripper-fixture frame using forward kinematics (Craig, 1989). Given a configuration description $\Theta$, the function $\text{forward\_kin}(\Theta)$ returns the transform $\text{port\_fixT}$. This forward kinematic calculation determines the position of a control point in the workspace grid as a function of the joint configuration. For example, given a control point on the gripper-fixture that coincides with the locator point for the primary datum when the gripper is closed, $\text{fixT}_{\text{close, 1}}$, the control point’s position relative to the robot’s workspace grid frame can be computed as follows: $\text{gridT}_{\text{close, 1}} = \text{gridT}_{\text{port}} * \text{forward\_kin}(\Theta) * \text{fixT}_{\text{close, 1}}$. The potential field value for this control point is the potential field value for the workspace grid cube that contains the position $\text{gridT}_{\text{close, 1}}$.

The criteria for a legal joint configuration are that the joints do not exceed their assigned limits and that the robot does not collide with obstacles. Checking the joint limits is straightforward while checking for collisions is difficult and computationally expensive. Since each moving part of a robot must be checked for collisions with external obstacles for a given configuration, collision checking requires a geometric model of the moving parts of the robot. Thus, each robot resource has a B-Rep geometric model of each of its links. Each link model is also coupled with the Denevit-Hartenberg parameters so that the transformation of each model’s coordinate frame is known for a given joint configuration. A “divide and conquer” approach is used for collision checking in this research. Typically, this approach first uses an inexpensive test to eliminate a subset of problems while a more expensive test is required if the test fails. In this case, an inexpensive test for collisions is set up by bounding each robot link with a solid ellipsoid. This test uses the distance function $d_1$ defined in Section 6.3 and the principle that the sum of distances to a point on the ellipsoid surface from the foci is equal to the ellipsoid length. For a given joint configuration, these ellipsoids are transformed into the correct position and orientation relative to the
workspace grid. Furthermore, the grid cubes containing the foci of an ellipsoid are calculated. For such grid cubes \( p_1 \) and \( p_2 \) that contain foci of an ellipsoid, an obstacle collides with the ellipsoid if \( d_1(p_1) + d_1(p_2) \leq \text{the ellipsoid length in grid cubes} \). Since this bounding ellipsoid may occupy a larger volume than the robot link, a detected collision must be checked more thoroughly using an expensive test. This test consists of calculating which workspace grid cubes intersect with the robot link and checking if any of these cubes are obstacle cubes.

### 6.4.3 Random Walk Away from Local Minima

This path planning strategy uses configuration planning to move the end effector control points along the negative gradient of their potential fields towards the goal. If this strategy fails at some point because there are no legal neighbor configurations that decrease the potential field values for the control points, then this strategy is no longer effective. At this point, the robot is moved randomly in configuration space for a random number of steps. After each step, the resulting potential field values are computed and compared with the previous local minimum value. If the resulting potential field values are less than the previous local minimum values, then a configuration search is continued from that point. Otherwise, if these random moves do not result in a configuration with lower potential field values, then a configuration search is performed at the end of the random motion.

The robot is moved to a randomly selected legal neighbor configuration for each random walk step. A significant issue is how to determine the number of steps. The problem is that the distance from the current local minimum to the hill of the next local minimum is unknown. Latombe (1991) proposes using a random number of such steps with an expected value that is related to the length of the workspace in one direction in grid cubes. Thus, the number steps is estimated to be a random distribution about the expected value of \( L/\delta^2 \), where \( L \) is the length of the workspace along one axis and \( \delta \) is the length of a grid cube.

### 6.4.4 Random Motion Planning

For the complete motion planner that integrates the configuration space planning with random walks, an algorithm proposed by Barraquand & Latombe (1990) is used. This algorithm begins with a configuration space search that ends in a potential field minimum at configuration \( q_{loc} \). If the resulting configuration is not at the goal, then the planner generates a set of 20 random walks and configuration space searches. If one of these 20 searches
results in a potential field minimum at $q_{loc}'$ that is less than the original minimum at $q_{loc}$, then the planner inserts
the path from $q_{loc}$ to $q_{loc}'$ to the original path $\tau$ that led to $q_{loc}$ from the initial configuration at $q_{init}$. Next, the
planner continues the search starting at $q_{loc}'$. If none of the 20 motions executed from a local minimum result in a
better local minimum, then the planner backtracks to a randomly selected configuration in the path that was
generated using a random walk. If a new configuration space search from this configuration results in a better
local minimum than achieved thus far, then the search commences from this new local minimum. This algorithm
that is adapted from the description in Latombe (1991) is defined as follows for an initial configuration $q_{init}$ and a
goal configuration $q_{goal}$. In this algorithm, the function $potential\_field\_path(q)$ returns a path that is found using
the configuration space planning based on workspace potential fields as described above. The function,
random_path($q_{loc}$, $t$) returns the path that results from a random walk of $t$ steps. The function, backtrack($\tau$, $\tau_1$, $\tau_2$, . . . $\tau_{20}$) returns a path starting from $q_{init}$ and ending at some configuration randomly selected from the
configurations in the paths that were generated by random walks.

**Motion Planning Algorithm**

\[
\begin{align*}
\tau & \leftarrow potential\_field\_path(q_{init}) \\
q_{loc} & \leftarrow \text{last configuration in } \tau \\
\text{while } q_{loc} & \neq q_{goal} \\
\text{escape} & \leftarrow \text{FALSE} \\
\text{for } i & \leftarrow 1 \text{ to } 20 \text{ until escape } = \text{TRUE} \\
\tau_i & \leftarrow \text{random number of steps} \\
q_{rand} & \leftarrow \text{last configuration in } \tau_i \\
\tau_i & \leftarrow \tau_i + potential\_field\_path(q_{rand}) \\
q_{loc}' & \leftarrow \text{last configuration in } \tau_i \\
\text{if } U(q_{loc}') & < U(q_{loc}) \\
\text{escape} & \leftarrow \text{TRUE} \\
\tau & \leftarrow \tau + \tau_i \\
\text{if escape } = \text{FALSE} \\
\tau & \leftarrow \text{backtrack}(\tau, \tau_1, \tau_2, \ldots \tau_{20}) \\
q_{back} & \leftarrow \text{last configuration in } \tau \\
\tau & \leftarrow \tau + potential\_field\_path(q_{back}) \\
q_{loc} & \leftarrow \text{last configuration in } \tau
\end{align*}
\]

**6.4.5 Application in Flexible Manufacturing**

In the path planning work reported by Barraquand & Latombe (1990), basic trajectory planning algorithms
are developed. These strategies are described in previous sections. However, Barraquand & Latombe (1990) do
not apply these algorithms to real flexible manufacturing environments. Application of these algorithms requires
determination of suitable grasping positions, selection of control points on the robot, a function to arbitrate between different goals of the control points, and a separation between gross and fine motion planning for part retrieval.

For trajectories that enable a robot to retrieve a part, the goal locations of the robot’s control points determine the grasp position on the part. Since these control points coincide with the locator points on the robot’s gripper-fixture, the goal of these trajectories is to position these control points so that the gripper precisely locates the part. From the description of the fixture model given in Chapter 4, if the gripper-fixture is designed with the intent to locate the part in this manner, then it is capable of holding the part securely during robot motion.

During motion planning, it is necessary to arbitrate between the different goals of the robot’s control points. Different arbitration schemes that are proposed by Latombe (1991) include the following: moving in a direction to reduce the potential field value of the control point that is furthest away from its goal, moving to reduce the potential field value of the control point that is the closest plus a small epsilon times the potential field value of the furthest control point, and moving to reduce the sum of the potential field values of all of the control points. The goal of this motion planning is to move the robot’s end effector to a position and orientation such that a very simple move to the final goal can be accomplished. Furthermore, it is undesirable to move the robot’s end effector to the target approach in the wrong orientation by letting one control point dominate the motion planning. Typically, at this point, the robot will be unable to easily reorient the end effector to the correct orientation because the robot is extended at the approach position and the end effector is close to obstacles such as the buffer and the fixture. Thus, the arbitration between different control points uses the sum of their potential field values during motion planning. While this strategy creates more local minimas, it appears to drive the end effector to the goal in the correct orientation.

Since this motion planning is done using a discretized grid of the workspace, planning the complete path of the end effector to precise positions required for part unloading or loading is computationally intensive. For instance, in the illustration given in Chapter 4, the required robot accuracy of placement of its end effector is given as +/- 0.001. In order to illustrate how to fully utilize this accuracy, assume that this tolerance is +/- 0.001
inch and the robot’s workspace is 6 feet wide by 6 feet long by 6 feet high. Then, in order to place the control points of the robot’s end effector to a position within +/- 0.001 inch, each workspace grid cube must have a length of 0.001 inch. Then the planner must work with a grid of 72000 cubes wide by 72000 cubes long by 72000 cubes high or a total of 3.73E14 cubes. In order to reduce this complexity, the motion planning task is separated into a gross level and a succession of more precise levels of motion planning. Gross motion planning uses a grid of larger resolution and stops planning when one of the control points reaches a potential field value of 1. The first fine motion planner uses a smaller grid with a finer resolution and stops planning when one of the control points reaches a potential field value of 1. This smaller workspace consists of a bounding box around the position of the control points and their goal positions. The resolution of this smaller grid is based on a maximum number of 50 grid cubes per side. When this first fine motion planner is finished, motion planning is done with a smaller grid and smaller cubes. This continues until it is no longer possible to construct a bounding box around the control points and their goal positions with a finer resolution and keep the maximum number of 50 grid cubes per side. At this point, an inverse kinematics solution (Craig, 1989) is used to determine the precise joint configuration to put the control points at their goal positions. Formally, the function, \( \text{inverse}_\text{kin} \left( \theta_{\text{final}} \right) \), returns this final configuration \( \theta_{\text{final}} \). The addition of the configuration \( \theta_{\text{final}} \) to the computed trajectories completes the motion planning to the goal position.

To satisfy the requirements of flexible manufacturing, the trajectories computed for unload or load operations must be converted into instruction sets, and the times required to execute the motions must be calculated. For a given operation \( \text{OP} \), the estimated time for the operation is represented by the function, \( \text{time} \left( \text{OP} \right) \). The file name of the file containing the instruction set for the operation is represented by the function, \( \text{instruction}_\text{set} \left( \text{OP} \right) \). The cost of the operation is simply the estimated time for the operation multiplied by the cost per time of the robot resource \( E \) or \( \text{cost} \left( \text{OP} \right) = \text{cost}_\text{per}_\text{time} \left( E \right) \times \text{time} \left( \text{OP} \right) \).
6.5 Chapter Summary

This chapter has described some algorithms for planning material handling tasks from the literature and how these algorithms are incorporated into new models for flexible manufacturing. The generation of trajectories as described in this chapter allows the constraints on part material handling to be completely characterized. Previous chapters have defined higher level material handling constraints based on the ability of robot grippers to hold parts, robot accuracy, and location of the robot on the factory floor. This chapter completes the characterization of material handling constraints by providing a means to determine if a given robot is able to use its kinematic flexibility to move a given gripper to load and unload parts. This means that the planning of the automated material handling of parts may be completely integrated with material processing at the shop floor. Given this new capability, production constraints arising from material handling constraints can be incorporated into the part design process. Furthermore, the complex kinematic flexibility of robots on the shop floor can be smoothly utilized for new tasks with this planner. By quickly generating data-driven operations plans with precise robot trajectories for new part designs, simulating the production of new parts for every design iteration is easier and more accurate. Lastly, production delays due to robot programming are minimized because the robot programs are automatically generated with this operations planner.
Chapter 7

IMPLEMENTATION ARCHITECTURE

The principles of good information systems architecture are as follows:
1. consistency
2. orthogonality (elements relatively independent of each other)
3. propriety (proper to functions, no unnecessary function)
4. parsimony (no functional redundancy in different forms)
5. transparency (functions introduced in implementation no imposed on the user)
6. generality (multipurpose)
7. open-endedness (alternate uses of needed function)
8. completeness (in solving needs and desires of user) [Blaauw, 1972]

Chapter 4 defined the formal specifications of the operations planner presented in this research. Chapters 5 and 6 described the algorithms that satisfy these software specifications. This chapter describes how the planner is implemented in software and how input and output data is separated from the software using external data storage models. Results from the software implementation complete this description. Section 7.1 introduces the chapter by describing the benefits of separating data from logic and using object-oriented design to design software. Section 7.2 describes the object-oriented information model used to translate the formal resource and model sets that were defined in Chapters 4 and 5 into object classes more applicable to software design. Section 7.3 describes how the input data consisting of resource and part data is stored and how the output data describing the generated process plan is stored. Lastly, Section 7.4 shows the results of the C++ software implementation by illustrating the resource and part models that were used to verify the software and by presenting different operation plans that were automatically generated and reflect specific changes in these resource and product models.

7.1 Introduction

Given the formal specification of the operations planner from Chapter 4 and the algorithms from Chapters 5 and 6, there still remain the problems of determining the format of the input and output data and implementing the algorithms intelligently. Historically, software development and usage became much easier when it became commonplace to separate the logic of software from the data supplied by the user. Then multiple users could use
the same software by simply changing the input data based on their individual requirements. This data was originally in the form of flat files that were typically customized to the specific software application. However, gradually tools in the form of databases and software to read the databases were developed to make this process easier. Expert systems took this process one step further by also separating the software logic from the implementation. This is accomplished by using externally defined rules that are executed by meta-rules in the implementation. However, these expert systems present problems in real life applications because there are not sufficient boundaries between different rules. A change in one rule may create unintended consequences in the overall performance of the expert system that can only be detected after the problem’s occurrence. Object-oriented programming languages were developed as a simpler way to automatically reason about the behavior of real or imagined objects. These languages automatically provide boundaries between portions of the software by limiting reasoning about a given object to code attached to a representation of the object and its data. This programming technique also enables a different approach to programming that consists of modeling the actual objects that are simulated rather than modeling the algorithms that simulate the objects. The advantages of this approach are clear for the task of simulating manufacturing resources’ actions in order to determine which sequences of these actions will result in the product being manufactured. Thus, software that reasons about the behavior of specific resource types can be embedded in software classes that represent these resources. Furthermore, reasoning about actions that involve different resources, such as how a milling machine deploys a drill tool, can be implemented using natural relationships between objects such as the ownership of a tool by a machine.

7.2 Object-Oriented Information Model

7.2.1 Introduction

The object-oriented information model used in this research models manufacturing resources, parts, and their relationships following the methodology proposed by Rumbaugh and colleagues (1991). According to this object-oriented modeling approach, classes can be designed from the desired data flow between objects. Since these classes are meant to integrate manufacturing enterprises beyond just operations planning, the data flow diagram illustrated in Figure 3.1 is utilized to design the resource and part classes. These classes also map directly to the
elements and sets in the formal specification developed in Chapter 4. Booch (1986) defined object classes to have
data, called class attributes, and functions, called class behaviors. Instances of these classes are objects. These
classes may be defined as sub-classes of other classes called super classes. The inheritance relationship between a
sub-class and its super class means that objects of the sub-class inherit attributes and behaviors from its super
class. Another relationship between classes is that one object of a class may own an object of another class. If
object a owns another object b, then a has access to b’s information and can control b’s behavior. The set relation
owns from Chapter 4 directly maps to this relationship. For instance, if element x ∈ X, and element y ∈ Y, then x
corresponds to an object x′ of class X′ and y corresponds to an object y′ of class Y′. Furthermore, if owns (x, y) is
true for elements x and y, then object x′ owns object y′.

7.2.2 Resource Model Classes

Using an object-oriented classification based on the specifications in Chapter 4, manufacturing resource
classes include a facility class that may own instances of the work center class. Each work center class may own
instances of the manufacturing equipment class E. Equipment objects may own objects of the fixture class.
Subclasses of class E include the Material Processor, Material Handler, Material Transporter, and Buffer Storage
classes. Material Processor objects may own objects of the tool class. In this model, the tool class has a HoleTool
subclass which is further subdivided into three subclasses that represent twist drills, reamers, and bores. These
classes and some of their attributes are illustrated in Figure 7.1. These attributes include information used
throughout this research that models the capability of resources to change the state of parts. Additional attributes,
such as cost and status, enable the integration between resource model and engineering functions such as design
for manufacturability assessment and production scheduling. Using the Unified Modeling Language (UML)
symbolology defined by Eriksson and Penker (1998), class ownership is denoted by the “owns” association while a
superclass and its subclass have an inheritance relationship. This class structure representing resources is used to
minimize the duplication of data attribute fields among classes. For instance, all equipment resources have an
associated activity-based cost rate that represents the cost of usage per some time unit. Thus, the super class,
Equipment, has an attribute representing this cost rate. For a given equipment entity E, the function cost_per_time
(E) returns this cost rate. All of the four equipment sub-classes inherit this cost rate attribute from the super class
Equipment. Other attributes, such as the lists of loadable and unloadable ports in the Material Handler class,
directly correspond to functions defined in Chapter 4. For instance, if object $E \in MH$ and object $\text{port} \in \text{Ports}$, the function $\text{loadable}(E, \text{port})$ is true if the port is in the list of loadable ports that is an attribute of the object that represents the equipment $E$. The design of the tool classes also follows this strategy.

**Figure 7.1 The Resource Model Classes used for Problem Illustration (UML format)**

In order to illustrate how resource modeling is implemented, Figure 7.2 shows the limited number of tool classes and their attributes that can be owned by Material Processor objects modeled in this research. United States Cutting Tool Institute (1989) and Walsh (1994) provided the data for the tooling attributes in this resource model. In Chapter 4, the function $\text{makes\_hole}(\text{tool}, \text{hole diameter, diametric tolerance, hole surface finish, material type, material hardness})$ was defined to be true if the tool is capable of making the hole feature with the required diameter, diametric tolerance, and surface finish on the material with the specified type and hardness. The tool class attributes in Figure 7.2 provide the data to compute whether this function is true for a given hole-making tool, feature, and base material. Other functions are necessary to compute the cost and time of machining operations by tools. Since the data representing the tool’s life, cost, cost/time, setup time, and change time can be represented in the same way for all tool sub-classes, attributes representing this data are part of the Tool class. Some of these attributes are represented for a given tool $T$ by the functions $\text{change\_time}(T)$, $\text{life}(T)$, and $\text{cost}(T)$.

Next, a function is needed to compute the precondition feature for a given tool if it is able to make the goal feature. For instance, in Section 5.2.2, an example was presented where a reaming operation required a
precondition feature of an initial hole. In order to facilitate this computation, the attribute of the minimum starting
diameter is added. A tool that does not require a starting hole simply has a zero minimum starting diameter. Since
all hole-making tools have some minimum starting diameter as well as the maximum depth of hole that they can
make, the minimum starting diameter and the maximum depth attributes are part of the Hole Tool class.

In this model, the sub-classes of the Hole Tool class include the Twist Drill, Reamer, and Bore classes. The
Twist Drill class has the Final Dia, Pos Accuracy, and Dia Accuracy attributes. For a given element \( \text{tool} \in \) TwistDrill,
there is a corresponding software object \( \text{tool}' \) with these attributes. For a given feature \( \text{feat}, \)
\( \text{accuracyX} (E, \text{tool}, \text{feat}), \text{accuracyY} (E, \text{tool}, \text{feat}), \) and \( \text{accuracyZ} (E, \text{tool}, \text{feat}) \) return values based on the Pos
Accuracy attribute of the object \( \text{tool}' \). The best surface finish achievable by a twist drill tool is dependent on the
base material’s type, material hardness and the machining parameters. Each Twist Drill software object may own
one or more Twist Drill Material Spec objects that specify these parameters. The Twist Drill Material Spec class
has Material Type, Min Hardness, Max Hardness, Surface Finish, Speed, and Feed attributes. If a Twist Drill
object owns a given object of the Twist Drill Material Spec class, then that tool is capable of achieving the
Surface Finish on material with the given Material Type and hardness between Min Hardness and Max Hardness
if machined using the given Speed and Feed.
In this model, the Reamer class has the Final Dia, Roughing Pos Accuracy, Roughing Dia Accuracy, Finishing Pos Accuracy, and Finishing Dia Accuracy attributes. Reamers may either be used in a roughing or finishing mode. The nominal final diameter of the hole feature created by a reamer is fixed, but the position and diametric accuracy is dependent on whether the operation is roughing or finishing. The operation is roughing or finishing based on the hole feature’s position tolerance, diametric tolerance, and surface finish. For a given \( \text{tool} \in \text{Reamers} \), there is a corresponding software object \( \text{tool}' \) with these attributes. For a given feature \( \text{feat} \), \( \text{accuracyX} (E, \text{tool}, \text{feat}) \), \( \text{accuracyY} (E, \text{tool}, \text{feat}) \), and \( \text{accuracyZ} (E, \text{tool}, \text{feat}) \) return values based on the Roughing Pos Accuracy attribute or the Finishing Pos Accuracy attribute of the object \( \text{tool}' \). The best surface finish achievable by a reamer tool is dependent on the base material’s type, material hardness, and the machining parameters. Each Reamer object may own one or more Reamer Material Spec objects that specify these parameters. The Reamer Material Spec class has Material Type, Min Hardness, Max Hardness, Roughing Surface Finish, Roughing Speed, Roughing Feed, Finishing Surface Finish, Finishing Speed, Finishing Feed attributes. If a Reamer object owns a given object of the Reamer Material Spec class, then in the roughing mode that tool is capable of achieving the Roughing Surface Finish on material with the given Material Type and hardness between Min Hardness and Max Hardness if machined using the given Roughing Speed and Roughing Feed. Similarly, in the finishing mode the tool is capable of achieving the Finishing Surface Finish on material with the given Material Type and hardness between Min Hardness and Max Hardness if machined using the given Finishing Speed and Finishing Feed.
The Bore class has the Pos Accuracy and Dia Accuracy attributes. For a given tool \( \in \text{Bores} \), there is a corresponding software object tool’ with these attributes. For a given feature \( \text{feat}, \) \( \text{accuracyX} \) \( (E, \text{tool}, \text{feat}) \), \( \text{accuracyY} \) \( (E, \text{tool}, \text{feat}) \), and \( \text{accuracyZ} \) \( (E, \text{tool}, \text{feat}) \) return values based on the Pos Accuracy attribute of the object tool’. In this model, given a material with a specific hardness, boring tools may create holes with a surface finish based on the depth of cut, speed, and feed. Each Bore object may own one or more Bore Material Spec objects that specify the parameters for the material: Material Type, Min Hardness, and Max Hardness. Each of these Bore Material Spec objects may own one or more Bore Feed/Speed Spec objects that specify different depth of cuts, speeds, and feeds.

In this model, it is assumed that material handler robots are closed kinematic chains. Thus, robot objects can be defined using a set of Denevit-Hartenberg parameters [Craig, 1989] for each joint link. Therefore, the Robot class has the following attributes: the number of joint links, sets of Denevit-Hartenberg parameters, its workspace radius used in path planning, and geometric models of each joint link. These attributes are used in robot behaviors such as forward kinematics that computes the transformation from the robot’s port to its gripper-fixture based on the current joint configuration. Formally, \( \text{port}_{\text{fixt}}T = \text{forward kin} (\Theta) \). Another behavior of the Robot class is path planning, described in Chapter 6. Subclasses of the Robot class are specific vendor supplied models of robots. For example, the Puma 560 robot is a subclass of the Robot class. Since a closed form inverse kinematics solution is available for the Puma 560 robot from Craig (1989), inverse kinematics is a behavior of the Puma 560 class. The Robot class and its subclasses are displayed in Figure 7.3 using the UML format.

![Figure 7.3 Resource Model Robot Classes (UML Format)](image-url)
7.2.3 Part Model Classes

The Part class and related classes also reflect the formal definitions concerning parts given in Chapter 4. These classes are illustrated in Figure 7.4. A part object is defined by a list of features, the AND/OR graph of these features that defines precedence between the features, and a start and end buffer. From Chapter 4, the predicate \( \text{owns} \) \((\text{part}, \text{feat})\) may be true for an element \( \text{part} \in \text{Parts} \) and element \( \text{feat} \in \text{Feat} \). Similarly, objects of the Part class may own one or more objects of the Feature class that are in the list of features for the part. A part object also owns an object of the Material class that defines the material type and hardness. For a given element \( \text{part} \in \text{Parts} \), there is a corresponding object \( \text{part}' \). Furthermore, the functions \( \text{material}(\text{part}) \) and \( \text{hardness}(\text{part}) \) defined in Chapter 4 return values represented by the attributes of the Material object owned by an object \( \text{part}' \).

Objects of the Feature class define the discrete features of the part. A flag indicating if the feature belongs to the original blank and a list of datums define an object of the Feature class. A Feature object may own one or more Datum objects. Datum objects link the datum features that specify the location of a feature with the object representing the feature. Thus, Datum objects have Distance and Tolerance attributes. The Distance attribute defines the nominal distance between the datum and the feature while the Tolerance attribute defines the tolerance associated with this distance. In this model, features can be classified as flat surfaces or holes. Thus, the Feature class has subclasses FlatSurface and Hole. Flat surfaces can be characterized using the boundary representation by a list of points. The cross product of these points is a vector that points to the outside of the part. A surface finish specification also characterizes surfaces. Thus, the FlatSurface object has Surface Finish and List of Points attributes. Similarly, cylindrical holes can be characterized by a depth, diameter, internal surface finish, and a diametric tolerance. Thus, objects of the Hole class have Depth, Diameter, Surface Finish, and Diametric Tolerance attributes.
7.2.4 Class Behaviors for Planning

The attributes of the classes representing the resource and part models and the class relationships are used to create class behaviors to facilitate planning. The high level logic of these planning activities is represented by formal functions from Chapter 4 such as \textit{makes\_hole}, \textit{loadable}, \textit{unloadable}, and \textit{movable}. Behaviors of resource classes implement these functions. These behaviors are capable of answering if a specific manufacturing facility is capable of changing the part state to a desired goal state and how it accomplishes this. In object-oriented modeling, behaviors may be modeled as virtual or polymorphic functions. A polymorphic function is a function defined for a set of sub-classes of a super class that performs differently for each sub-class. This enables the interface to the function to remain the same regardless of changes to the sub-classes. A polymorphic function, \textit{elimDifference}, is defined throughout the resource model class hierarchy that checks if different resource entities can eliminate part of the difference between a part current state and goal state. Thus, executing the \textit{elimDifference} behavior of the facility object causes the facility’s work station objects to check if they can eliminate the difference. Similarly, this causes each work station object to execute the \textit{elimDifference} behavior of its equipment resource objects. Each of these equipment resource objects interacts with behaviors of their tools and fixtures in order to determine if and how they can eliminate the difference. For instance, material processing resources interact with the behaviors and attributes of their tool objects to determine if and how their tools may make hole
features. If the predicate makes_hole is true for a given machining center, then time and cost of the operation is calculated using the following equation from Chang and colleagues (1998).

\[
\forall E, \forall \text{tool}, \forall \text{feat}, \\
(\text{isanprocessor} (E) \land \text{isatool} (\text{tool}) \land \text{isafeature} (\text{feat}) \land \text{makes_hole} (\text{tool}, \text{diameter} (\text{feat}), \text{finish} (\text{feat}), \\
\text{material} (\text{feat}), \text{hardness} (\text{feat}))) \Rightarrow ( \\
\text{machining time} = 60 \pi \times \text{diameter} (\text{feat}) \times \text{depth} (\text{feat}) / (12 \times \text{feed} (\text{tool}) \times \text{speed} (\text{tool})) \land \\
\text{time} (\text{OP}) = \text{machining time} + \text{change_time} (\text{tool}) \times \text{machining time/life} (\text{tool}) \land \\
\text{cost} (\text{OP}) = \text{time} (\text{OP}) \times \text{cost_per_time} (E) + \text{machining time} \times \text{cost} (\text{tool}) / \text{life} (\text{tool})))
\]

7.3 External Data Organization

7.3.1 Introduction

Efforts to separate input and output data from logic in software have resulted in several generations of data storage techniques. According to Stonebraker and colleagues (1991), these range from storing data in customized, flat ASCII files to hierarchical databases [Dadam, 1986] to relational databases to object-oriented databases. While flat ASCII files are still useful for certain data storage applications such as neutral part and image representations that do not require much direct human interaction, it is easier to store more complicated data in databases. Business spreadsheet applications have led to relational databases that store data in tabular forms that are easy for humans to read and modify. These relational databases allow relations between tables and provide standard software tools for accessing the databases such as Structured Query Language (SQL). The new generation of object-oriented databases go beyond tabular data storage with predefined data types by allowing data to be stored in an object-oriented fashion. In this architecture, the input resource model is stored using a relational database, the input part model is stored using an ASCII file, and the output data is stored using a relational database.

7.3.2 Resource Model Database

A relational database is used to store the resource model. This takes advantage of the human interfaces provided by commercial relational databases. A good human interface is important because engineers and technicians need to change the resource model data as new resources are purchased and old resources are removed from the facility. While commercial object-oriented databases are currently available and provide simple
methods for directly storing object data [Stonebraker et al., 1991], they do not provide good human readable interfaces due to the complexity of object relationships such as ownership and inheritance. The class methods that implement the different layers of the operations planner are embedded in object-oriented software. This software creates instances of resource classes by reading the relational database tables and converting the tabular data into object data. This enables humans to directly monitor and adjust the data records for the resources in a manufacturing environment. Naturally, the structure of the database is determined by the standard resource classification so that the object-oriented software can easily be designed to read the object-data from the database records. To keep this simple, one table is created for each resource class and each table is named using the same name as the resource class. Then each record in the table corresponds to the data for one instance of the class. Furthermore, the fields containing class attribute data have the same names as the class attribute names. The ownership relation between two classes is expressed using a one-to-many relation between primary keys of the owning table and foreign keys of the owned table. The inheritance relation is expressed using a one-to-one relation between the primary keys of the super-class table and the primary keys of the sub-class table. Figure 7.4 illustrates some of the tabular relationships of the relational database that reflects the resource classification in Figure 7.1.
7.3.3 Part Model File Specification

The input data to the operations planner includes the part data that is stored in a flat ASCII file. An ASCII file is used to specify the part because commercial CAD software typically provides neutral ASCII file representations of parts. Thus, a translator simply must translate between a file describing the part geometry and the file specification developed through this research. This part data consists of a list of feature definitions, an AND/OR graph defining the feature precedences, the names of the raw material and finished material buffers, and the type of base material and its hardness. A file specification was developed to allow modifications to the part definition in this file without requiring changes to the planner software. This file specification is illustrated with an example in Appendix B. In this file specification, individual features are first defined using the specific information for each feature type that has been discussed in previous chapters. Next, the feature AND/OR
precedence graph is defined using a string of symbols and numbers representing feature numbers. The character “*” represents a sequence, “&” represents an AND node, and “|” represents an OR node. Thus, the string “(*7 9*)” represents a sequence of feature 7 and then feature 9. Thus, feature 7 must be made and then feature 9 must be made. The string “(| 7 8 |)” represents feature 7 OR feature 8. Thus, either feature 7 or feature 8 must be made on the part. The string “(&7 9&)” represents feature 7 AND feature 9. Thus, either feature 7 and then feature 9 must be made or feature 9 and then feature 7 must be made. Lastly, part specific information is defined in the part specification file. This information includes the material type and hardness of the discrete part.

### 7.3.4 Operations Plan Database

The output data that consists of an operations plan is stored in a relational database. The relational database interface enables other software such as schedulers, simulators, and manufacturing execution systems to read the operations plan and make decisions accordingly. Furthermore, the human interface provided with the relational database allows engineers and technicians to check the validity of the plan and modify it when necessary.

The operations plan database consists of four tables that are illustrated in Figure 7.5. Three tables define the unique material processing, material handling, and material transportation operations that are required to manufacture the part. The other table, GraphNodes, represents the generated AND/OR graph of the operations. Each node in this AND/OR graph is represented by a row in the GraphNodes table. A field in the GraphNodes table, titled OpNumb, links the operation number of each operation node to its description in one of the three tables describing operation data. Another field, titled NodeType, indicates whether the node is an operation, a beginning OR node, a beginning AND node, an ending OR node, or an ending AND node. This is indicated using a text string that can take the following values: OP, BEGIN_OR, BEGIN_AND, END_OR, and END_AND. Each set of OR branches begins with a row with the value BEGIN_OR that is followed by a list of operations or other sub branches and then ends with END_OR. Similarly, each set of AND branches begins with a row with the value BEGIN_AND that is followed by a list of operations or other sub branches and then ends with END_AND. Other fields in the GraphNodes table specify data that is common to all operations such as the name of the equipment that executes the operation, the fixture holding the part during the operation, the workstation where the operation occurs, and the time and cost associated with the operation. The last field, EquipProgram, stores the file
name of a file containing an instruction set associated with the operation. For instance, Chapter 6 describes how the planner generates an instruction set that defines the trajectory an automated material handler takes when performing an unload or load operation. The names of files containing numerical control (NC) programs may also be set in the EquipProgram field for a material processing operation. These instruction sets may be downloaded to machines and executed at production time.

Figure 7.5 RDBMS Relationship Diagram of Operations Plan Database

7.4 Results of Software Implementation

7.4.1 Introduction

The architecture described in Sections 7.2 and 7.3 has been implemented in software using the C++ language. This implementation represents a validation of the formal models described in this dissertation as well as a practical tool for manufacturing operations planning. For this implementation, a resource model in a Microsoft
ACCESS relational database and a part specification in an ASCII file represents the initial implementation for input to the software. An AND/OR graph of operations stored in another Microsoft ACCESS relational database defines an interpretation of the output domain. The C++ code for the implementation reads the information defining the input domain and generates information defining the output domain if at least one valid sequence of operations is found. If no valid sequences of operations are found for a specific interpretation of the input domain, then the software reports back to the user that no solution was found. The formal models and algorithms that define the validity of a sequence of operations and individual operations were used along with a planning algorithm to create this software. These models and algorithms were defined in Chapters 4, 5, and 6.

The inspiration for this dissertation is that the responsiveness of manufacturing operations plans to changes in resources or their reconfiguration as well as to changes in the part being manufactured or redesigned is very useful for flexible manufacturing. Thus, this section shows how the software implementation generates different operation plans based on changes in the resource or part models. These changes are made by modifying records in the resource model database or by modifying the data in the part specification file and then running the software to generate a new operations plan database. In this section, a resource model layout is presented that reflects the resources of the Penn State Computer Integrated Manufacturing laboratory described in http://www.engr.psu.edu/cim/cim.html. Four different configurations of these resources are specified along with a part specification. The software planner was executed for each of these configurations and a part specification and the resulting four operation plans are presented. Next, the software planner was executed for five different part specifications and a resource configuration and the resulting five operation plans are presented. For simplicity, tolerances, kinematics, and trajectories will be ignored during the discussion of these results.

The resource models used to validate the software implementation are reconfigurations of the system resources in the layout illustrated in Figure 7.6. This facility consists of two processing workcenters with machining centers, one storage workcenter, and a transportation workcenter. For these examples, the resource model reconfigurations consist of changing the tooling that are accessible to different machining centers and specifying a table buffer to be an auxiliary buffer or not. This resource model is stored in a database and its formal specification is in Table 7.1.
Table 7.1. Formal Specification of Resource Model for PSU CIM Laboratory

<table>
<thead>
<tr>
<th>Resource</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>$W$</td>
<td>$W = W_p \cup W_t \cup W_s$</td>
</tr>
<tr>
<td>$W_p$</td>
<td>$W = {W_1, W_2}$</td>
</tr>
<tr>
<td>$W_t$</td>
<td>$W = {W_4}$</td>
</tr>
<tr>
<td>$W_s$</td>
<td>$W = {W_3}$</td>
</tr>
<tr>
<td>$F$</td>
<td>$F = {\text{Fixt}_1, \text{Fixt}_2, \text{Fixt}_3, \text{Fixt}_4, \text{Fixt}_5, \text{Fixt}_6, \text{Fixt}<em>7, \text{Fixt}<em>8, \text{Fixt}<em>9, \text{Fixt}</em>{10}, \text{Fixt}</em>{11}, \text{Fixt}</em>{12}}$</td>
</tr>
<tr>
<td>$T_{\text{drill}}$</td>
<td>$T_{\text{drill}} = {T_4, T_9, T_{39}}$</td>
</tr>
<tr>
<td>$T_{\text{ream}}$</td>
<td>$T_{\text{ream}} = {T_{71}}$</td>
</tr>
<tr>
<td>$T_{\text{bore}}$</td>
<td>$T_{\text{bore}} = {\emptyset}$</td>
</tr>
<tr>
<td>$diameter(T_4)$</td>
<td>$diameter(T_4) = 0.5$</td>
</tr>
<tr>
<td>$diameter(T_9)$</td>
<td>$diameter(T_9) = 0.25$</td>
</tr>
<tr>
<td>$diameter(T_{39})$</td>
<td>$diameter(T_{39}) = 0.1875$</td>
</tr>
<tr>
<td>$diameter(T_{71})$</td>
<td>$diameter(T_{71}) = 0.25$</td>
</tr>
<tr>
<td>$Ports$</td>
<td>$Ports = {\text{Port}_1, \text{Port}_2, \text{Port}_3, \text{Port}_4, \text{Port}_5, \text{Port}_6, \text{Port}<em>7, \text{Port}<em>8, \text{Port}<em>9, \text{Port}</em>{10}, \text{Port}</em>{11}, \text{Port}</em>{12}}$</td>
</tr>
<tr>
<td>$E$</td>
<td>$E = {E_1, E_2, E_3, E_4, E_5, E_6, E_7, E_8, E_9, E_{10}}$</td>
</tr>
<tr>
<td>$MH$</td>
<td>$MH = {E_1, E_2, E_3}$</td>
</tr>
<tr>
<td>$BS$</td>
<td>$BS = {E_4, E_6, E_{12}}$</td>
</tr>
<tr>
<td>$MT$</td>
<td>$MT = {E_1, E_7, E_8, E_9, E_{10}}$</td>
</tr>
<tr>
<td>$Lathes$</td>
<td>$Lathes = {E_5}$</td>
</tr>
<tr>
<td>$VertMills$</td>
<td>$VertMills = {E_6}$</td>
</tr>
<tr>
<td>$HorizMills$</td>
<td>$HorizMills = {E_3}$</td>
</tr>
<tr>
<td>$MP$</td>
<td>$MP = {E_1, E_7, E_9, E_{10}}$</td>
</tr>
<tr>
<td>$Lathes$</td>
<td>$Lathes = {E_5}$</td>
</tr>
<tr>
<td>$VertMills$</td>
<td>$VertMills = {E_6}$</td>
</tr>
<tr>
<td>$HorizMills$</td>
<td>$HorizMills = {E_3}$</td>
</tr>
<tr>
<td>$MP$</td>
<td>$MP = {E_1, E_7, E_9, E_{10}}$</td>
</tr>
<tr>
<td>$Rmh$</td>
<td>$Rmh = {E_1, E_2, E_3}$</td>
</tr>
<tr>
<td>$Wmh$</td>
<td>$Wmh = {\emptyset}$</td>
</tr>
<tr>
<td>$UnloadPorts_1$</td>
<td>$UnloadPorts_1 = {\text{Port}_2, \text{Port}_4, \text{Port}_5, \text{Port}_6}$</td>
</tr>
<tr>
<td>$UnloadPorts_2$</td>
<td>$UnloadPorts_2 = {\text{Port}_7, \text{Port}_8, \text{Port}<em>9, \text{Port}</em>{10}}$</td>
</tr>
<tr>
<td>$ReachPorts_1$</td>
<td>$ReachPorts_1 = {\text{Port}_5, \text{Port}_7}$</td>
</tr>
<tr>
<td>$locates$</td>
<td>$locates(\text{Fixt}_1, \text{Feat}_5, 1, 1)$</td>
</tr>
<tr>
<td>$lodates$</td>
<td>$lodates(\text{Fixt}_1, \text{Feat}_4, 1, 2)$</td>
</tr>
<tr>
<td>$locates$</td>
<td>$locates(\text{Fixt}_1, \text{Feat}_3, 1, 3)$</td>
</tr>
<tr>
<td>$locates$</td>
<td>$locates(\text{Fixt}_2, \text{Feat}_2, 2, 1)$</td>
</tr>
<tr>
<td>$lodates$</td>
<td>$lodates(\text{Fixt}_2, \text{Feat}_1, 2, 2)$</td>
</tr>
<tr>
<td>$locates$</td>
<td>$locates(\text{Fixt}_2, \text{Feat}_3, 2, 3)$</td>
</tr>
<tr>
<td>$locates$</td>
<td>$locates(\text{Fixt}_3, \text{Feat}_5, 1, 1)$</td>
</tr>
<tr>
<td>$lodates$</td>
<td>$lodates(\text{Fixt}_3, \text{Feat}_4, 1, 2)$</td>
</tr>
<tr>
<td>$locates$</td>
<td>$locates(\text{Fixt}_3, \text{Feat}_3, 1, 3)$</td>
</tr>
<tr>
<td>$locates$</td>
<td>$locates(\text{Fixt}_4, \text{Feat}_5, 1, 1)$</td>
</tr>
<tr>
<td>$lodates$</td>
<td>$lodates(\text{Fixt}_4, \text{Feat}_4, 1, 2)$</td>
</tr>
<tr>
<td>$locates$</td>
<td>$locates(\text{Fixt}_4, \text{Feat}_3, 1, 3)$</td>
</tr>
</tbody>
</table>
The part models used to validate the software implementation are reconfigurations of the part file specification in Appendix B. The part reconfigurations consist of changing which features are goal features and changing the precedence of these goal features. The part model is formally specified as follows:

\[
\begin{align*}
\text{Parts} &= \{\text{Part1}\}, \\
\text{Feat} &= \{\text{Feat1, Feat2, Feat3, Feat4, Feat5, Feat6, Feat7, Feat8, Feat9}\} \\
&\text{beginbuffer (Part1) = E_6, } \text{endbuffer (Part1) = E_6} \\
&\text{material (Part1) = Aluminum 1108, hardness (Part1) = 48} \\
&\text{diameter (Feat7) = 0.25, diameter (Feat8) = 0.50, diameter (Feat9) = 0.50} \\
&\text{specifies (Feat5, Feat7), specifies (Feat4, Feat7), specifies (Feat3, Feat7)} \\
&\text{specifies (Feat4, Feat8), specifies (Feat1, Feat8), specifies (Feat3, Feat8)} \\
&\text{specifies (Feat5, Feat9), specifies (Feat4, Feat9), specifies (Feat3, Feat9)} \\
&\text{specifies (Feat1, Feat5), specifies (Feat2, Feat4), specifies (Feat3, Feat6)}
\end{align*}
\]

### 7.4.2 Examples of Operations Plans for Different Resource Models

To illustrate the different operations plans that are generated from different resource models, one part model will be used. This part model is described by the part file specification in Appendix B except that there is only one goal feature Feat7. From the formal description of the part model, it should be noted that Feat7 is a hole feature, it is specified by datums Feat5, Feat4, Feat3, and Feat7 has a diameter of 0.25. Part1 is also defined to have its raw material located at the Kardex AS/RS E1 buffer and the finished material must be brought to E1.

For the first example, the MP entity E7, which represents a Haas VF-5 Vertical Milling Center, is assigned a twist drill tool T9, named “DR1-4M1”, that is designed to make 0.25” diameter holes. This tool had been included
in the general set of tools available to the facility but not specifically to E₇. Adding a new record to the MaterialProcessorToolUsage table in the resource model database makes this assignment. The new record specifies the unique tool ID number (#9) and the unique MP entity ID number (#7). Formally, *owns* (E₇, T₉) is now a true statement. After this modification to the resource model database, the operations planning software is executed and generates the operations plan that is illustrated in Figure 7.7. The planner first determines that E₇ can make the feature because E₇ owns a tool T₉ that can make the feature and a fixture Fixt₁ that locates the required datums for Feat₇ (Feat₅, Feat₄, Feat₃). This creates the sub-problems of using resources to move the raw material part from E₁ (Port₁) to E₇ (Port₆) and the finished part back to the storage buffer represented by E₁ (Port₁). These sub-problems are solved by using *unload* and *load* operators from the resource model. The generated solution to this problem is as follows: Puma Robot #3 unloads part from Port₁ to its home Port₁₂, Puma Robot #3 loads part from its home Port₁₂ to Port₅, Puma Robot #1 unloads part from Port₂ to its home Port₁₁, Puma Robot #1 loads part from its home Port₁₁ to E₇ at Fixt₁ in Port₆, VF-5 machine uses Tₙ to make Feat₇ in Fixt₁, Puma Robot #1 unloads part from Port₆ to its home Port₁₁, Puma Robot #1 loads part from its home Port₁₁ to Port₅, Puma Robot #3 unloads part from Port₂ to its home Port₁₂, and finally, Puma Robot #3 loads part from its home Port₁₂ to Port₁.

![Figure 7.7 Results from Resource Model with ¼ Drill at Machine VF-5 and Port #6](image)

For the second example, two more tools are made available to the MP entity E₇. These tools are named DR3-16M1 and R1-4M2 and are represented by entities T₃₉ and T₇₁ in the resource model. Formally, *owns* (E₇, T₉), *owns* (E₇, T₃₉), and *owns* (E₇, T₇₁) are now true statements. Because there are now two solutions to making the feature using either T₉ or T₇₁, the software implementation creates two alternative branches in the search graph. The branch corresponding to T₉’s operator results in the operations sequence illustrated in Figure 7.7. The branch corresponding to T₇₁’s operator uses T₃₉’s *drill* operator to create the starting hole that satisfies the starting hole precondition for T₇₁’s *ream* operator. This type of solution is illustrated in Figure 5.8 in Section 5.2.3. After
generating the search graph, tracing through the graph to generate an OR graph, and searching the OR graph to eliminate duplicate nodes, the software implementation generates the solution illustrated in Figure 7.8. Thus, in a similar fashion to the operations plan in Figure 7.7, the Puma Robot #3 and Puma Robot #1 bring the part to Fixt₁ in the machine VF-5 at Port₆. Since both alternative machining sequences work on the part in the same orientation and feature intention in Fixt₁, an OR node splits the solution after the part is located at Port₆. One OR branch makes Feat₇ using T₉ while the other OR branch makes Feat₇ using T₃₉ and then T₇₁. These OR branches represent two separate alternatives to produce Feat₇. At the end of both machining sequences, the part state is the same so that the same material handling operations are used to bring the finished part back to the Kardex AS/RS at Port₁. These material handling operations were defined for the previous example.

![Figure 7.8 Results from Resource Model with ¼ Drill, 3/16 Drill, ¼ Reamer at Machine VF-5 and Port #6](image)

For the third example, E₇, representing the VF-5 machine, continues to own the three tools from the previous example but the buffer represented by E₄ at Port₄ is set to be an auxiliary buffer. For the implementation, checking the Auxiliary box for E₄’s record in the BufferStorage table and then running the software again generates a new operations plan reflecting the new status of E₄. Formally, now the predicate \texttt{auxiliary}(E₄) is true. An auxiliary buffer may create material handling alternatives in the operations plan if there exists material handling resources to load and unload the buffer. One search graph that results from such an auxiliary buffer is illustrated in Figure 5.9 in Section 5.2.3. The operations plan generated from this third example is illustrated in Figure 7.9. As illustrated in this figure, the generated operations plan specifies that when the part is unloaded from Port₂ by Puma Robot #1, the part may be loaded to the auxiliary buffer at Port₄ and then unloaded from Port₄, or the part may immediately be loaded to E₇ at Port₆. After creating Feat₇ on E₇ using either material processing alternative, the part is unloaded by Puma Robot #1. At this point, the part may be loaded to the
auxiliary buffer again at Port 4 for temporary storage or loaded to Port 5. Lastly, the part is brought back to the finished parts buffer at the Kardex AS/RS as in previous examples.

For the fourth example of how the automatic generation of operation plans is responsive to changes in the resource model, the tools from the previous examples are made unavailable to E 7, and T 9 is made available to another MP milling machine resource, VF-0E. This resource, represented by entity E 9 with Port 8, is at the end of another material movement chain in another workcenter. Removing the records for E 7 and adding a new record to the MaterialProcessorToolUsage table in the resource model database reconfigures the resource model database. The new record specifies the unique tool ID number (#9) and the unique MP entity ID number (#9). Formally, owns (E 9, T 9) is now a true statement. After this modification to the resource model database, the operations planning software is executed and generates the operations plan that is illustrated in Figure 7.10. The planner first determines that E 9 can make the feature because E 9 owns a tool T 9 that can make the feature and a fixture Fixt 4 that locates the required datums forFeat 7 (Feat 5, Feat 4, Feat 3). This creates the sub-problems of how to use the resources to move the raw material part from E 1 (Port 1) to E 9 (Port 8) and the finished part back to the storage buffer represented by E 1 (Port 1). These sub-problems are solved by using unload, load, and move operators from the resource model. The generated solution to this problem is as follows: Puma Robot #3 unloads part from Port 1 to its home Port 12, Puma Robot #3 loads part from its home Port 12 to the AGV station at Port 3, AGV moves part from Port 3 to Port 7, Puma Robot #2 unloads part from Port 7 to its home Port 13, Puma Robot #2 loads part from its home Port 13 to E 9 at Fixt 4 in Port 8, VF-0E machine uses T 9 to make Feat 7 in Fixt 4, Puma Robot #2 unloads part
from Port8 to its home Port13, Puma Robot #2 loads part from its home Port13 to Port7, AGV moves part from Port7 to Port3, Puma Robot #3 unloads part from Port3 to its home Port12, and finally, Puma Robot #3 loads part from its home Port12 to Port1.

![Diagram of operations plan](image.png)

**Figure 7.10 Results from Resource Model with ¼ Drill at Machine VF-0E and Port #8**

### 7.4.3 Examples of Operations Plans for Different Part Models

To illustrate the responsiveness of the operations planner to changes in the part model, five examples of operations plans that were generated from different permutations of the part model are presented. These different permutations involve defining different combinations of Feat7, Feat8, and Feat9 as goals for the part. From the part model, it should be noted that Feat7 and Feat9 are specified using the same datums (Feat5, Feat4, Feat3) while Feat8 is specified using a different set of datums (Feat4, Feat3, Feat1). These goal features are hole features and Feat7 has a diameter of 0.25", Feat8 has a diameter of 0.50", and Feat9 has a diameter of 0.50". The resource model for these examples is the same as in the previous section except that E7 has two tools T4 and T9 named DR1-2M1 and DR1-4M1. Formally, the predicates $owns(E7, T4)$ and $owns(E7, T9)$ are true statements.

For the first example of the responsiveness of the generated operations plans to changes in the part model, the goal feature precedence string is specified to be (*7 8*). Thus, Feat7 must be made and then Feat8 must be made. Based on this part model, the operations planner generated the operations plan illustrated in Figure 7.11. The resource model is capable of making this part because tool T9 is capable of making Feat7 and tool T4 is capable of making Feat8. Furthermore, the fixture Fixt1 is capable of locating the datums for Feat7 with its fixture intention #1 and it is capable of locating the datums for Feat8 with its fixture intention #2. This plan resembles plans in the previous section except that after Feat7 is made using tool T9 on E7, the part must be refixtured because Feat8 is
specified using a different set of datums. Figure 5.7 in Section 5.2.2 described the planning algorithm for choosing material handling actions to refixture a part in the same machine. This algorithm was used by the operations planner implementation to determine that the Puma Robot #1 must unload the part from Port6 after Feat7 is made and load the part back into Fixt1 with fixture intention #2 in Port6. After Feat8 is finished, the part is brought back to the finished parts buffer as before.

For the second example of the responsiveness of the generated operations plans to changes in the part model, the goal feature string is specified to be (*7 8 9*). Thus, Feat9 must be made after features Feat7 and Feat8 are made according to the tasks specified in the previous example. The complete operations plan for this example is illustrated in Figure 7.12. In this operations plan, after Feat8 is made on the milling machine represented by E7, the part must be refixtured to make Feat9 because Feat9 requires the fixture intent #1 of Fixt1. Unloading the part and then loading it again into E7 refixtures the part. After Feat9 is made, the part is unloaded and brought back to the finished parts buffer as in previous examples.
For the third example of the responsiveness of the generated operations plans to changes in the part model, the goal feature string is specified to be (\(|7\ 8\ |\)). According to this specification, either Feat\(_7\) or Feat\(_8\) need to be made on the part. The operations plan that was generated from this specification is illustrated in Figure 7.13. Since the two goal features are specified by different datums and require different fixture intentions on Fixt\(_1\), loading the raw material part and unloading the finished part are different for each feature. Thus, the operations plan includes an OR node with branches for each goal feature and corresponding load and unload operations.

The fourth example of operations planning with different part models uses a part model with the feature string specified as (\(|\&\ 7\ 9\ \&\)). Thus, features Feat\(_7\) and Feat\(_9\) may be made in any order. Because these goal features use the same fixture intention of Fixt\(_1\) on E\(_7\), this part specification is satisfied by an operations plan that brings the part to Fixt\(_1\) with fixture intention #1 at Port\(_6\), and then has two AND branches with the operations to make the features. Such an operations plan was generated by the software implementation and it is illustrated in Figure 7.14.
The fifth example of operations planning with different part models uses a part model with the feature string specified as (& 7 8 &). Thus, features Feat7 and Feat8 may be made in any order. Because these goal features use the different fixture intentions of Fixt1 on E7, this part specification is satisfied by an operations plan that brings the part to the Puma Robot #1 at Port11, and then has two AND branches with the operations to load the part in the correct fixture intent, make the feature, and then unload the part from the correct fixture intent. Such an operations plan was generated by the software implementation and it is illustrated in Figure 7.15.

**Figure 7.15 Results from Part Model with Goal Features Graph (& 7 8 &)**

7.4.4 Limitations

While the results of the implementation validate the formal resource and part models and the operations planner, there are limitations that need to be addressed by future research. The limitations of the software implementation of the operations planning architecture consist of not being able to pinpoint to the user why a resource configuration is unable to make a part, the complexity of the search process has not been analyzed, and the software has not been tested for complex AND/OR feature precedence graphs and resource models with many manufacturing alternatives. The difficulty with pinpointing to the user why a resource configuration cannot make
a part is that in a search graph many different alternative paths are explored. No solution to the planning problem is found if all alternative paths are unsolvable. It is difficult to predict which unsolvable alternative path is important to the user and how the reasons for its unsolvability should be reported to the user. The result of the research for this dissertation is the first framework for operations planning. Given this framework, future research can approach issues of complexity and develop more robust implementations capable of handling more complex part and resource models.

7.5 Chapter Summary

This chapter has presented an overview of the architecture used to create the operations planner and satisfy the formal specifications given in Chapter 4. This architecture includes the object-oriented information model used to represent the real world objects found in manufacturing facilities as well as discrete parts that can be machined in these facilities. This architecture also includes the methods used to store the input data representing instances of resources and parts and how the operations plan generated by the planner is stored. This object-oriented architecture is useful because it integrates a human-friendly relational database and part file specification with automatic decision-making. The database allows users to update operation plans based on new resource information and capabilities by simply changing records and then running the operations planner again. The part file specification could be translated from commercial CAD software specifications, which enables the integration of standard product design techniques with this operations planner. Simulation models, schedulers, and production controllers can easily utilize the operation instructions, times, cost, and sequences in the operations plan database generated by the operations planner. This software implementation of this architecture generated plans with alternatives for a shop with thirteen equipment entities, ninety tools, fifteen fixtures, and prismatic parts with varying numbers of goal features. This implementation validates the formal model of the architecture that was presented in Chapters 4 and 5. This was illustrated by presenting the results from the software implementation that generated different operations plans for different resource models and part models. Four operation plans were generated for four different resource model configurations and the same part model, while five operation plans were generated for five different part models and the same resource model. This
demonstrates how the operations planner, developed for this dissertation, generates manufacturing plans in a manner that is driven by the information in the resource and part models.
Chapter 8

RESEARCH CONTRIBUTIONS, FUTURE RESEARCH, AND CONCLUSIONS

8.1 Research Contributions

The contributions of the research include a new resource-oriented framework for manufacturing operations planning that integrates critical engineering functions, a part state space formulation that enables planning of material processing and material handling operations for the first time with precise physical constraints, and a new, formally defined operations planning algorithm that was tested by a software implementation. The object-oriented resource model provides sufficient information for detailed geometric and trajectory planning of intra-equipment operations as well as operations that involve multiple resource interactions such as part loading and unloading. The structure of the resource model also facilitates integration with other engineering functions such as design, accounting, scheduling, and production control by incorporating cost and other data with these same resource classes. The part state space formulation is another contribution because it enables simultaneous planning of material processing and material movement tasks using this resource model. The formal definition of the input and output domains for the operations planner is another contribution because it firmly establishes the necessary mapping from resource and part models to the operations plan. Furthermore, the search techniques developed to satisfy this formal definition illustrate how to incorporate artificial intelligence into manufacturing operations planning in a more complete and useful way than has been previously described in the literature. Lastly, the algorithms for converting the generated search graph into a compact operations plan with alternatives are useful for real world implementation. This operations plan provides a model of the physical capability of the facility’s resources to manufacture a part. The alternative operations in this plan may be subsequently selected for execution based on considerations such as activity-based cost, throughput optimization, equipment and tool availability, and bottleneck avoidance.
8.2 Future Research

Future extensions of this research include distributing the objects in the resource and part models, creating translators between commercial part specifications and this dissertation’s facility specific part model, planning automated material handling of tools and fixtures, adding an interface to the planner that informs the user why a given resource set is unable to make the part, and integrating other engineering functions using the resource model framework. Distributing the objects in the resource and part models allows different knowledge sources in a large manufacturing enterprise to smoothly interact and share common information. Thus, vendors could easily provide equipment specific attributes, and researchers could provide new class behaviors within a standardized resource model framework. Future translators between commercial CAD geometric specifications and this dissertation’s facility-specific part model enable “real world” implementations of this research. Automatic planning of the material handling of fixtures and tools adds another real world constraint to this manufacturing operations planner. For instance, it makes no sense to plan an operation by a tool on a machine in a “hands-off” facility if there does not exist a resource capable of loading the machine with the tool. For actual implementation of this planning tool to test the capability of a facility layout or resource set to manufacture a given part, additional development work must be performed to specify exactly why a facility is unable to manufacture a part. Given this output, an engineer may be able to quickly reconfigure the resource model to meet the product’s requirements. Future integration of other engineering functions using this resource model may consist of developing manufacturability ratings for design feedback based on the operations plan generated by this planner. Other integration efforts may consist of developing schedulers, bottleneck avoidance strategies, and execution systems that use the operations plan that is generated through this research to control automated flexible manufacturing facilities.

8.3 Thesis Conclusions

In conclusion, it is possible to develop formal and practical models for operations planning that combine the planning of material movement tasks with material processing activities. These models directly contribute to software implementation. Furthermore, artificial intelligence search techniques can be used to solve real world
problems such as “is a part manufacturable by a facility’s resources and how can it be manufactured”? These artificial intelligence techniques can be integrated with object-oriented information modeling to facilitate automated decision-making using the modeled capabilities of physical resources to perform manufacturing operations. These results imply that automatic and flexible decision-making can be successfully integrated with manufacturing planning and execution on the shop floor.
REFERENCES


Appendix A

EXAMPLE VERIFICATION OF OPERATIONS PLAN

In order to illustrate how operations plans are specified and verified, Section 4.5 presented an example interpretation of the domain that represents a simple manufacturing facility and a simple part. The facility’s resources and their layout are given in Figure 4.4 and the part is illustrated in Figure 4.5. Three operations that could be part of an operations plan were validated according to the formal criteria established in Chapter 4. In this appendix, the complete operations plan for manufacturing the part using the facility is presented and validated by proving that the operations plan is valid for the facility and part. This validation is based on the necessary conditions for the validity of each individual operation and that each linear sequence in the operation plan results in the goal state for the part.

The start node for the example part in part state-space is represented by the node \((p, \text{flist}, \text{fixture}, \text{fixtureintent})\), where \(p = \text{Port}_1\), \(\text{flist} = \{\text{Feat}_1, \text{Feat}_2, \text{Feat}_3, \text{Feat}_4, \text{Feat}_5, \text{Feat}_6\}\), \(\text{fixture} = \text{Fixt}_1\), and \(\text{fixtureintent} = \#1\). The goal condition for this part is given by \(p = \text{Port}_1 \Rightarrow \text{goal}p(\text{Part}_1, p), \text{goal}flist(\text{Part}_1) = \{\text{Feat}_7\}, \text{goal}fixture(\text{Part}_1) = \text{Fixt}_1\), and \(\text{goal}fixtureintent(\text{Part}_1) = \#1\). Based on these start and goal conditions, an operations plan can be generated. This operations plan specifies that robot (E₂) first unloads the part from the AS/RS (E₁) at port #1. Next, the robot (E₂) loads the part to the AGV (E₆) at its station labeled port #3. The AGV moves with the part to its other station labeled port #4. Next, the robot (E₄) unloads part from the AGV. The robot (E₄) then loads the part into the vertical milling machine (E₃) at port #2. The vertical milling machine uses its tool, T₁, to make feature #7 in the part. Next, the robot (E₄) unloads the part from the vertical milling machine and loads it into the AGV (E₆) at port #4. Note that the table with port #5 can be used as an auxiliary buffer for the vertical milling machine. The AGV subsequently moves with the part to port #3. The robot (E₂) unloads the finished part from the AGV and loads it into the AS/RS at port #1. This operations plan can be expressed as an AND/OR graph that incorporates alternative routes that use the auxiliary buffer at port #5. Figure A.1 illustrates this graph and the individual operations in the graph are listed in Table A.1.

151
Figure A.1. AND/OR operations plan graph for example part.

Table A.1. Operations for Part 1

<table>
<thead>
<tr>
<th>Operation Num</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>unload (Port₁, E₂, Fixt₁, 1)</td>
</tr>
<tr>
<td>2</td>
<td>load (Port₁, E₂, Fixt₁, 1)</td>
</tr>
<tr>
<td>3</td>
<td>move (Port₆, Port₁, E₆, Fixt₆, 1)</td>
</tr>
<tr>
<td>4</td>
<td>unload (Port₄, E₄, Fixt₆, 1)</td>
</tr>
<tr>
<td>5</td>
<td>load (Port₅, E₄, Fixt₅, 1)</td>
</tr>
<tr>
<td>6</td>
<td>unload (Port₅, E₄, Fixt₅, 1)</td>
</tr>
<tr>
<td>7</td>
<td>load (Port₂, E₄, Fixt₃, 1)</td>
</tr>
<tr>
<td>8</td>
<td>process (Feat₇, E₃, Fixt₃, 1)</td>
</tr>
<tr>
<td>9</td>
<td>unload (Port₂, E₄, Fixt₃, 1)</td>
</tr>
<tr>
<td>10</td>
<td>load (Port₅, E₄, Fixt₅, 1)</td>
</tr>
<tr>
<td>11</td>
<td>unload (Port₅, E₄, Fixt₅, 1)</td>
</tr>
<tr>
<td>12</td>
<td>load (Port₄, E₄, Fixt₆, 1)</td>
</tr>
<tr>
<td>13</td>
<td>move (Port₄, Port₃, E₆, Fixt₆, 1)</td>
</tr>
<tr>
<td>14</td>
<td>unload (Port₁, E₂, Fixt₁, 1)</td>
</tr>
<tr>
<td>15</td>
<td>load (Port₁, E₂, Fixt₁, 1)</td>
</tr>
</tbody>
</table>

According to the given specifications for an operations plan to be valid, the individual operations in the operations plan satisfy a schema of necessary conditions, and the ending state for each linear sequence in the plan must satisfy the goal conditions for the part. The following formal language proves that the fifteen individual operations in the preceding table are valid for this interpretation of the operations planning domain. For simplicity, the equations for the tolerances and datums in the vertical direction (Z) are left out of the following proofs.
Operation 1. $OP_1 = \text{unload (Port}_1, E_2, \text{Fixt}_1, 1)$. Because the operation is of the unload type, $\text{unload (OP}_1)$ is true. Thus, the only applicable condition to be satisfied is $\text{unload (OP}_1) \Rightarrow \text{unloadable (part (OP}_1), \text{port (OP}_1), \text{equip (OP}_1), \text{from fixture (OP}_1))$. From the definition of $OP_1$, $\text{part (OP}_1) = \text{Part}_1$, $\text{port (OP}_1) = \text{Port}_1$, and $\text{equip (OP}_1) = E_2$. The truth of $\text{unloadable (Part}_1, \text{Port}_1, E_2, \text{Fixt}_1)$ has already been established in Section 4.5. Thus, $\text{unloadable (Part}_1, \text{Port}_1, E_2, \text{Fixt}_1)$ is true, and $OP_1$ is valid for this interpretation.

Operation 2. $OP_2 = \text{load (Port}_3, E_2, \text{Fixt}_6, 1)$. Because the operation is of the load type, $\text{load (OP}_2)$ is true. Thus, the only applicable condition to be satisfied is $\text{load (OP}_2) \Rightarrow \text{loadable (part (OP}_2), \text{port (OP}_2), \text{equip (OP}_2), \text{to fixture (OP}_2))$. From the definition of $OP_2$, $\text{part (OP}_2) = \text{Part}_1$, $\text{port (OP}_2) = \text{Port}_3$, $\text{equip (OP}_2) = E_2$, and $\text{to fixture (OP}_2) = \text{Fixt}_6$. The truth of $\text{loadable (Part}_1, \text{Port}_3, E_2, \text{Fixt}_6)$ can be proved by proving that the antecedents of statement #3 are true. From the interpretation of the domain given previously, it is known that $\text{is a port (Port}_3) \land \text{loadable (E}_2, \text{Port}_3), \text{is a feature (Feat}_2), \text{is a feature (Feat}_3), \text{is a feature (Feat}_6), \text{owns (Part}_1, \text{Feat}_2), \text{owns (Part}_1, \text{Feat}_3), \text{owns (Part}_1, \text{Feat}_6), \text{owns (E}_6, \text{Fixt}_6), \text{locates (Fixt}_6, \text{Feat}_2, 1, 1), \text{locates (Fixt}_6, \text{Feat}_3, 1, 2), \text{specifies (Feat}_5, \text{Feat}_6)$, and $\text{loadTrajExists (E}_2, \text{Fixt}_2, \text{Fixt}_6, \text{Port}_3, \text{Part}_1)$. This leaves four mathematical expressions to be proven true because the robot must be capable of locating both datums of the part within the equipment fixture’s open tolerance along the Facility’s x and y coordinate axes. These expressions and their equivalent values are true as follows:

1. $\text{open Tol}_X (\text{Facility, E}_6, \text{Fixt}_6, \text{Feat}_2) \geq \text{tol pos}_X (\text{Facility, E}_2) + \text{accuracy}_X (\text{Facility, E}_2) + \text{fixture pos tol}_X (\text{Facility, Port}_6, \text{Fixt}_2) + \text{tol pos}_X (\text{Facility, E}_6) + \text{close tol}_X (\text{Facility, Port}_6, \text{Fixt}_2, \text{Feat}_2)$
   \[ \Leftrightarrow \infty \geq 0.01 + 0.001 + 0.001 + 0.01 + \infty \Leftrightarrow \infty \geq \infty \]

2. $\text{open Tol}_Y (\text{Facility, E}_6, \text{Fixt}_6, \text{Feat}_2) \geq \text{tol pos}_Y (\text{Facility, E}_6) + \text{accuracy}_Y (\text{Facility, E}_6) + \text{fixture pos tol}_Y (\text{Facility, Port}_6, \text{Fixt}_6) + \text{tol pos}_X (\text{Facility, E}_6) + \text{close tol}_Y (\text{Facility, E}_6, \text{Fixt}_6, \text{Feat}_2)$
   \[ \Leftrightarrow 0.03 \geq 0.01 + 0.001 + 0.001 + 0.01 + 0.001 + 0.001 \Leftrightarrow 0.03 \geq 0.023 \]

3. $\text{open Tol}_X (\text{Facility, E}_6, \text{Fixt}_6, \text{Feat}_2) \geq \text{tol pos}_X (\text{Facility, E}_6) + \text{accuracy}_X (\text{Facility, E}_6) + \text{fixture pos tol}_X (\text{Facility, Port}_6, \text{Fixt}_6) + \text{tol pos}_X (\text{Facility, E}_6) + \text{close tol}_X (\text{Facility, E}_6, \text{Fixt}_6, \text{Feat}_2) + \text{specifies tol}_X (\text{Facility, Feat}_5, \text{Feat}_1)$
   \[ \Leftrightarrow \infty \geq 0.01 + 0.001 + 0.001 + 0.01 + 0.001 + 0.001 \Leftrightarrow \infty \geq \infty \]

4. $\text{open Tol}_Y (\text{Facility, E}_6, \text{Fixt}_6, \text{Feat}_2) \geq \text{tol pos}_Y (\text{Facility, E}_6) + \text{accuracy}_Y (\text{Facility, E}_6) + \text{fixture pos tol}_Y (\text{Facility, Port}_6, \text{Fixt}_6) + \text{tol pos}_X (\text{Facility, E}_6) + \text{close tol}_Y (\text{Facility, E}_6, \text{Fixt}_6, \text{Feat}_2) + \text{specifies tol}_Y (\text{Facility, Feat}_5, \text{Feat}_1)$
   \[ \Leftrightarrow 0.03 \geq 0.01 + 0.001 + 0.001 + 0.01 + 0.001 + 0.001 \Leftrightarrow 0.03 \geq 0.024 \]
Operation 3. OP₃ = move (Port₃, Port₄, E₆, Fixt₆, 1). Because the operation is of the move type, \textit{move} (OP₃) is true. Thus, the only applicable condition to be satisfied is \textit{move} (OP₃) ⇒ \textit{moveable} (part (OP₃), equip (OP₃), startport (OP₃), endport (OP₃), fixture (OP₃)). From the definition of OP₃, part (OP₃) = Part₁, startport (OP₃) = Port₅, endport (OP₃) = Port₄, equip (OP₃) = E₆, fixture (OP₃) = Fixt₆. The truth of \textit{reachable} (Part₁, Port₅, E₆, Fixt₆) and \textit{reachable} (Part₁, Port₄, E₆, Fixt₆) can be proved by proving that the antecedents of statement #4 are true. From the interpretation of the domain given previously, it is known that \textit{isaport} (Port₅) ∧ \textit{reachable} (E₆, Port₅), \textit{isafeature} (Feat₃), \textit{owns} (Part₁, Feat₃), \textit{owns} (E₆, Fixt₆), \textit{locates} (Fixt₆, Feat₁, 1, 1). Thus, \textit{reachable} (Part₁, Port₅, E₆, Fixt₆) is true. Furthermore, since \textit{isaport} (Port₄) ∧ \textit{reachable} (E₆, Port₄), \textit{reachable} (Part₁, Port₄, E₆, Fixt₆) is also true. From statement #5, it is known that \∀E, \∀Part, \∀p₅, \∀p₆ (isatransporter (E) ∧ \textit{isaport} (Part) ∧ \textit{isaport} (p₅) ∧ \textit{isaport} (p₆) ∧ j ≠ k ∧ \textit{reachable} (Part, E, p₅, Fixt₆) ∧ \textit{reachable} (Part, E, p₆, Fixt₆) ⇒ \textit{movable} (Part, E, p₅, p₆, Fixt₆)). Since 3 ≠ 4, \textit{reachable} (Part₁, Port₅, E₆, Fixt₆), and \textit{reachable} (Part₁, Port₄, E₆, Fixt₆) are true, \textit{movable} (Part, E₆, Port₅, Port₄, Fixt₆) is also true and OP₃ is valid for this interpretation.

Operation 4. OP₄ = unload (Port₄, E₄, Fixt₆, 1). Because the operation is of the unload type, \textit{unload} (OP₄) is true. Thus, the only applicable condition to be satisfied is \textit{unload} (OP₄) ⇒ \textit{unloadable} (part (OP₄), port (OP₄), equip (OP₄), fromfixture (OP₄)). From the definition of OP₄, part (OP₄) = Part₁, port (OP₄) = Port₄, equip (OP₄) = E₄, and fromfixture (OP₄) = Fixt₆. The truth of \textit{unloadable} (Part, port, E₄, Fixt₆) can be proved by proving that the antecedents of statement #2 are true. First, it is known that \textit{isaport} (Port₄) ∧ \textit{loadable} (E₄, Port₄), \textit{isafeature} (Feat₂), \textit{isafeature} (Feat₃), \textit{isafeature} (Feat₆), \textit{owns} (Part₁, Feat₂), \textit{owns} (Part₁, Feat₃), \textit{owns} (Part₁, Feat₆), \textit{owns} (E₄, Fixt₆), \textit{locates} (Fixt₆, Feat₂, 1, 1), \textit{locates} (Fixt₆, Feat₃, 1, 2), \textit{owns} (E₄, Fixt₆), \textit{locates} (Fixt₆, Feat₆, 1, 1), \textit{locates} (Fixt₆, Feat₆, 1, 2), \textit{specifies} (Feat₃, Feat₆), and \textit{unloadTrajExists} (E₄, Fixt₆, Fixt₆, Port₄, Part₁). This leaves four mathematical expressions to be proven true because the robot must be capable of locating both datums of the part within the gripper’s open tolerance along the Facility’s x and y coordinate axes. These expressions and their equivalent values are true as follows:

1. \textit{open_tolX} (Facility, E₄, Fixt₆, Feat₂) ≥ \textit{tol_posX} (Facility, E₄) + \textit{accuracyX} (Facility, E₄) + \textit{fixture_pos_tolX} (Facility, Port₄, Fixt₆) + \textit{tol_posX} (Facility, E₄) + \textit{fixture_pos_tolX} (Facility, Port₄, Fixt₆) + \textit{close_tolX} (Facility, E₆, Fixt₄, Feat₂)
   ⇔ 0.08 ≥ 0.01 + 0.001 + 0.01 + 0.001 + 0.03 ⇔ 0.08 ≥ 0.053
2. $open_{tolY}(Facility, E_a, Fixt_5, Feat_5) \subseteq tol_{posY}(Facility, E_a) + accuracyY(Facility, E_a) + fixture_{pos_tolY}(Facility, Port_5, Fixt_5) + tol_{posY}(Facility, E_a) + fixture_{pos_tolY}(Facility, Port_5, Fixt_5) + close_{tolY}(Facility, E_a, Fixt_5, Feat_5)$
\[\Leftrightarrow \infty \geq 0.01 + 0.001 + 0.001 + 0.01 + 0.001 + \infty \Leftrightarrow \infty \geq \infty \]

3. $tol_{posX}(Facility, E_a) + accuracyX(Facility, E_a) + fixture_{pos_tolX}(Facility, Port_5, Fixt_5) + tol_{posY}(Facility, E_a) + fixture_{pos_tolY}(Facility, Port_5, Fixt_5) + close_{tolX}(Facility, E_a, Fixt_5, Feat_5) \subseteq open_{tolX}(Facility, E_a, Fixt_5, Feat_5)$
\[\Leftrightarrow 0.08 \geq 0.01 + 0.001 + 0.001 + 0.01 + 0.001 + \infty + 0.001 \Leftrightarrow \infty \geq \infty \]

4. $open_{tolY}(Facility, E_a, Fixt_5, Feat_5) \subseteq tol_{posY}(Facility, E_a) + accuracyY(Facility, E_a) + fixture_{pos_tolY}(Facility, Port_5, Fixt_5) + tol_{posX}(Facility, E_a) + fixture_{pos_tolX}(Facility, Port_5, Fixt_5) + close_{tolY}(Facility, E_a, Fixt_5, Feat_5) + specifies_{tolY}(Facility, Feat_5, Feat_5)$
\[\Leftrightarrow 0.08 \geq 0.01 + 0.001 + 0.001 + 0.01 + 0.001 + 0.03 + 0.001 \Leftrightarrow 0.08 \geq 0.054 \]

Operation 5. $OP_5 = load(Port_5, E_4, Fixt_5, 1)$. Because the operation is of the load type, $load(OP_5)$ is true. Thus, the only applicable condition to be satisfied is $load(OP_5) \Rightarrow loadable(part(OP_5), port(OP_5), equip(OP_5), tofixture(OP_5))$. From the definition of $OP_5$, $part(OP_5) = Part_1$, $port(OP_5) = Port_5$, $equip(OP_5) = E_4$, $tofixture(OP_5) = Fixt_5$. The truth of $loadable(Part_1, Port_5, E_4, Fixt_5)$ can be proved by proving that the antecedents of statement #3 are true. From the interpretation of the domain given previously, it is known that $isaport(Port_5) \land loadable(E_4, Port_5), isafeature(Feat_5), isafeature(Feat_5), isafeature(Feat_5), owns(Part_1, Feat_5), owns(Part_1, Feat_5)$, $owns(Part_1, Feat_5), owns(E_4, Fixt_5), locates(Fixt_5, Feat_5, 1, 1), locates(Fixt_5, Feat_5, 1, 2), owns(E_4, Fixt_5), locates(Fixt_5, Feat_5, 1, 1), locates(Fixt_5, Feat_5, 1, 2), specifies(Feat_5, Feat_5), and $loadTrajExists(E_4, Fixt_5, Fixt_5, Port_5, Part_1)$. This leaves four mathematical expressions to be proven true because the robot must be capable of locating both datums of the part within the equipment fixture’s open tolerance along the Facility’s $x$ and $y$ coordinate axes. These expressions and their equivalent values are true as follows:

1. $open_{tolX}(Facility, E_a, Fixt_5, Feat_5) \subseteq tol_{posX}(Facility, E_a) + accuracyX(Facility, E_a) + fixture_{pos_tolX}(Facility, Port_5, Fixt_5) + tol_{posX}(Facility, E_a) + fixture_{pos_tolX}(Facility, Port_5, Fixt_5) + close_{tolX}(Facility, E_a, Fixt_5, Feat_5)$
\[\Leftrightarrow 0.03 \geq 0.01 + 0.001 + 0.001 + 0.01 + 0.001 + 0.001 \Leftrightarrow 0.03 \geq 0.023 \]

2. $open_{tolY}(Facility, E_a, Fixt_5, Feat_5) \subseteq tol_{posY}(Facility, E_a) + accuracyY(Facility, E_a) + fixture_{pos_tolY}(Facility, Port_5, Fixt_5) + tol_{posY}(Facility, E_a) + fixture_{pos_tolY}(Facility, Port_5, Fixt_5) + close_{tolY}(Facility, E_a, Fixt_5, Feat_5)$
\[\Leftrightarrow 0.03 \geq 0.01 + 0.001 + 0.001 + 0.01 + 0.001 + 0.001 \Leftrightarrow 0.03 \geq 0.023 \]

3. $open_{tolX}(Facility, E_a, Fixt_5, Feat_5) \subseteq tol_{posX}(Facility, E_a) + accuracyX(Facility, E_a) + fixture_{pos_tolX}(Facility, Port_5, Fixt_5) + tol_{posX}(Facility, E_a) + fixture_{pos_tolX}(Facility, Port_5, Fixt_5) + close_{tolX}(Facility, E_a, Fixt_5, Feat_5) + specifies_{tolX}(Facility, Feat_5, Feat_5)$
\[\Leftrightarrow 0.03 \geq 0.01 + 0.001 + 0.001 + 0.01 + 0.001 + 0.001 \Leftrightarrow 0.03 \geq 0.023 \]

4. $open_{tolY}(Facility, E_a, Fixt_5, Feat_5) \subseteq tol_{posY}(Facility, E_a) + accuracyY(Facility, E_a) + fixture_{pos_tolY}(Facility, Port_5, Fixt_5) + tol_{posY}(Facility, E_a) + fixture_{pos_tolY}(Facility, Port_5, Fixt_5) + close_{tolY}(Facility, E_a, Fixt_5, Feat_5) + specifies_{tolY}(Facility, Feat_5, Feat_5)$
\[\Leftrightarrow 0.03 \geq 0.01 + 0.001 + 0.001 + 0.01 + 0.001 + 0.001 \Leftrightarrow 0.03 \geq 0.024 \]
Operation 6. OP₆ = unload (Port₅, E₄, Fixt₅, 1). Because the operation is of the unload type, unload (OP₆) is true. Thus, the only applicable condition to be satisfied is unload (OP₆) ⇔ unloadable (part (OP₆), port (OP₆), equip (OP₆), fromfixture (OP₆)). From the definition of OP₆, part (OP₆) = Part₁, port (OP₆) = Port₅, equip (OP₆) = E₄, and fromfixture (OP₆) = Fixt₅. The truth of unloadable (Part₁, Port₅, E₄, Fixt₅) can be proved by proving that the antecedents of statement #2 are true. First, it is known that isaport (Port₅) ∧ unloadable (E₄, Port₅), isafeature (Feat₂), isafeature (Feat₃), isafeature (Feat₆), owns (Part₁, Feat₂), owns (Part₁, Feat₃), owns (Part₁, Feat₆), owns (E₄, Fixt₅), locates (Fixt₅, Feat₂, 1, 1), locates (Fixt₅, Feat₃, 1, 2), owns (E₄, Fixt₅), locates (Fixt₄, Feat₂, 1, 1), locates (Fixt₄, Feat₆), specifies (Feat₃, Feat₆, 1, 2), and unloadTrajExists (E₄, Fixt₄, Fixt₅, Port₅, Part₁). This leaves four mathematical expressions to be proven true because the robot must be capable of locating both datums of the part within the gripper’s open tolerance along the Facility’s x and y coordinate axes. These expressions and their equivalent values are true as follows:

1. \[ \text{open_tolX (Facility, E₄, Fixt₄, Feat₂)} \geq \text{tol_posX (Facility, E₄)} + \text{accuracyX (Facility, E₄)} + \text{fixture_pos_tolX (Facility, Port₅, Fixt₅, Feat₂)} \]
\[ \Leftrightarrow \infty \geq 0.01 + 0.001 + 0.001 + 0.01 + 0.001 + \infty \Leftrightarrow \infty \geq \infty \]

2. \[ \geq \text{tol_posY (Facility, E₄)} + \text{accuracyY (Facility, E₄)} + \text{fixture_pos_tolY (Facility, Port₅, Fixt₅, Feat₂)} \leq \text{open_tolY (Facility, E₄, Fixt₄, Feat₂)} \]
\[ \Leftrightarrow 0.01 + 0.001 + 0.001 + 0.01 + 0.001 + 0.03 \leq 0.08 \Leftrightarrow 0.08 \geq 0.053 \]

3. \[ \text{open_tolX (Facility, E₄, Fixt₄, Feat₆)} \geq \text{tol_posX (Facility, E₄)} + \text{accuracyX (Facility, E₄)} + \text{fixture_pos_tolX (Facility, Port₅, Fixt₅, Feat₆)} + \text{specifies_tolX (Facility, Feat₃, Feat₆)} \]
\[ \Leftrightarrow \infty \geq 0.01 + 0.001 + 0.001 + 0.01 + 0.001 + 0.03 + 0.001 \leq 0.08 \Leftrightarrow 0.08 \geq 0.054 \]

4. \[ \text{open_tolY (Facility, E₄, Fixt₄, Feat₆)} \geq \text{tol_posY (Facility, E₄)} + \text{accuracyY (Facility, E₄)} + \text{fixture_pos_tolY (Facility, Port₅, Fixt₅, Feat₆)} + \text{specifies_tolY (Facility, Feat₃, Feat₆)} \]
\[ \Leftrightarrow \infty \geq 0.01 + 0.001 + 0.001 + 0.01 + 0.001 + \infty + 0.001 \Leftrightarrow \infty \geq \infty \]

Operation 7. OP₇ = load (Port₂, E₄, Fixt₅, 1). Because the operation is of the load type, load (OP₇) is true. Thus, the only applicable condition to be satisfied is load (OP₇) ⇔ loadable (part (OP₇), port (OP₇), equip (OP₇)). From the definition of OP₇, part (OP₇) = Part₁, port (OP₇) = Port₂, and equip (OP₇) = E₄. The truth of loadable (Part₁, port₂, E₄, Fixt₅) can be proved by proving that the antecedents of statement #3 are true. From the interpretation of the domain given previously, it is known that isaport (Port₅) ∧ loadable (E₄, Port₅), isafeature (Feat₂), isafeature (Feat₃), isafeature (Feat₆), owns (Part₁, Feat₂), owns (Part₁, Feat₃), owns (Part₁, Feat₆), owns (E₄, Fixt₅), locates (Fixt₅, Feat₂, 1, 1), locates (Fixt₅, Feat₃, 1, 2), owns (E₄, Fixt₅), locates (Fixt₄, Feat₂, 1, 1),
locates (Fixt4,Feat6,1,2), specifies (Feat3,Feat6), and loadTrajExists (E4,Fixt4,Fixt3,Port3,Part1). This leaves four mathematical expressions to be proven true because the robot must be capable of locating both datums of the part within the equipment fixture’s open tolerance along the Facility’s x and y coordinate axes. These expressions and their equivalent values are true as follows:

1. \(\text{open_tolX} (\text{Facility}, E_3, \text{Fixt}_3, \text{Feat}_2) \geq \text{tol_posX} (\text{Facility}, E_4) + \text{accuracyX} (\text{Facility}, E_4) + \text{fixture_pos_tolX} (\text{Facility}, \text{Port}_7, \text{Fixt}_4) + \text{tol_posX} (\text{Facility}, E_3) + \text{fixture_pos_tolX} (\text{Facility}, \text{Port}_2, \text{Fixt}_3) + \text{close_tolX} (\text{Facility}, E_4, \text{Fixt}_4, \text{Feat}_2) \Leftrightarrow \infty \geq 0.01 + 0.001 + 0.001 + 0.01 + 0.001 + \infty \Leftrightarrow \infty \geq \infty\)

2. \(\text{open_tolY} (\text{Facility}, E_3, \text{Fixt}_3, \text{Feat}_2) \geq \text{tol_posY} (\text{Facility}, E_4) + \text{accuracyY} (\text{Facility}, E_4) + \text{fixture_pos_tolY} (\text{Facility}, \text{Port}_7, \text{Fixt}_4) + \text{tol_posY} (\text{Facility}, E_3) + \text{fixture_pos_tolY} (\text{Facility}, \text{Port}_2, \text{Fixt}_3) + \text{close_tolY} (\text{Facility}, E_4, \text{Fixt}_4, \text{Feat}_2) \Leftrightarrow 0.03 \geq 0.01 + 0.001 + 0.001 + 0.01 + 0.001 + 0.001 \Leftrightarrow 0.03 \geq 0.023\)

3. \(\text{open_tolX} (\text{Facility}, E_3, \text{Fixt}_3, \text{Feat}_3) \geq \text{tol_posX} (\text{Facility}, E_4) + \text{accuracyX} (\text{Facility}, E_4) + \text{fixture_pos_tolX} (\text{Facility}, \text{Port}_7, \text{Fixt}_4) + \text{tol_posX} (\text{Facility}, E_3) + \text{fixture_pos_tolX} (\text{Facility}, \text{Port}_2, \text{Fixt}_3) + \text{specifies_tolX} (\text{Facility}, \text{Feat}_3, \text{Feat}_6) + \text{close_tolX} (\text{Facility}, E_4, \text{Fixt}_4, \text{Feat}_6) \Leftrightarrow \infty \geq 0.01 + 0.001 + 0.001 + 0.01 + 0.001 + \infty + \infty \Leftrightarrow \infty \geq \infty\)

4. \(\text{open_tolY} (\text{Facility}, E_3, \text{Fixt}_3, \text{Feat}_3) \geq \text{tol_posY} (\text{Facility}, E_4) + \text{accuracyY} (\text{Facility}, E_4) + \text{fixture_pos_tolY} (\text{Facility}, \text{Port}_7, \text{Fixt}_4) + \text{tol_posY} (\text{Facility}, E_3) + \text{fixture_pos_tolY} (\text{Facility}, \text{Port}_2, \text{Fixt}_3) + \text{specifies_tolY} (\text{Facility}, \text{Feat}_3, \text{Feat}_6) \Leftrightarrow 0.03 \geq 0.01 + 0.001 + 0.001 + 0.01 + 0.001 + 0.001 + 0.001 \Leftrightarrow 0.03 \geq 0.024\)

Operation 8. \(\text{OP}_8 = \text{process} (\text{Feat}_7, E_3, \text{Fixt}_3, 1)\). Because the operation is of the process type, \(\text{process} (\text{OP}_8)\) is true. Thus, the only applicable condition to be satisfied is \(\text{process} (\text{OP}_8) \Rightarrow \text{makes} (\text{equip} (\text{OP}_8), \text{feat} (\text{OP}_8), \text{fixture} (\text{OP}_8))\). From the definition of \(\text{OP}_8\), it is known that \(\text{equip} (\text{OP}_8) = E_3, \text{feat} (\text{OP}_8) = \text{Feat7},\) and \(\text{fixture} (\text{OP}_8) = \text{Fixts}\). The truth of \(\text{makes} (E_3, \text{Feat}_7, \text{Fixt}_3)\) has already been established in Section 4.5. Thus, \(\text{makes} (E_3, \text{Feat}_7, \text{Fixt}_3)\) is true, and \(\text{OP}_8\) is valid for this interpretation.

Operation 9. \(\text{OP}_9 = \text{unload} (\text{Port}_2, E_4, \text{Fixt}_3, 1)\). Because the operation is of the unload type, \(\text{unload} (\text{OP}_9)\) is true. Thus, the only applicable condition to be satisfied is \(\text{unload} (\text{OP}_9) \Rightarrow \text{unloadable} (\text{part} (\text{OP}_9), \text{port} (\text{OP}_9), \text{equip} (\text{OP}_9), \text{fromfixture} (\text{OP}_9))\). From the definition of \(\text{OP}_9\), \(\text{part} (\text{OP}_9) = \text{Part}_1, \text{port} (\text{OP}_9) = \text{Port}_2, \text{equip} (\text{OP}_9) = E_4,\) and \(\text{fromfixture} (\text{OP}_9) = \text{Fixt}_3\). The truth of \(\text{unloadable} (\text{Part}_1, \text{Port}_2, E_4, \text{Fixt}_3)\) can be proved by proving that the antecedents of statement #2 are true. First, it is known that \(\text{isaport} (\text{Port}_2) \land \text{unloadable} (E_4, \text{Port}_2), \text{isafeature} (\text{Feat}_2), \text{isafeature} (\text{Feat}_3), \text{isafeature} (\text{Feat}_6), \text{owns} (\text{Part}_1, \text{Feat}_2), \text{owns} (\text{Part}_1, \text{Feat}_3), \text{owns} (\text{Part}_1, \text{Feat}_6), \text{owns} (E_4, \text{Fixt}_3), \text{locates} (\text{Fixt}_3, \text{Feat}_5, 1, 1), \text{locates} (\text{Fixt}_3, \text{Feat}_6, 1, 2), \text{owns} (E_4, \text{Fixt}_3), \text{locates} (\text{Fixt}_4, \text{Feat}_5, 1, 1), \text{locates} (\text{Fixt}_4, \text{Feat}_6, 1, 2), \text{specifies} (\text{Feat}_3, \text{Feat}_6),\) and \(\text{loadTrajExists} (E_4, \text{Fixt}_3, \text{Fixt}_3, \text{Port}_2, \text{Part}_1)\). This leaves
4 mathematical expressions to be proven true because the robot must be capable of locating both datums of the part within the gripper’s open tolerance along the Facility’s x and y coordinate axes. These expressions and their equivalent values are true as follows:

1. \[\text{open_tolX} (\text{Facility, E}_4, \text{Fixt}_4, \text{Feat}_2) \geq \text{tol_posX} (\text{Facility, E}_4) + \text{accuracyX} (\text{Facility, E}_4) + \text{fixture_pos_tolX} (\text{Facility, Port}_7, \text{Fixt}_4) + \text{tol_posX} (\text{Facility, E}_3) + \text{fixture_pos_tolX} (\text{Facility, Port}_2, \text{Fixt}_3) + \text{close_tolX} (\text{Facility, E}_3, \text{Fixt}_3, \text{Feat}_2) \]
   \[\Leftrightarrow 0.08 \geq 0.01 + 0.001 + 0.01 + 0.001 + 0.01 + 0.001 \Leftrightarrow 0.08 \geq 0.024\]

2. \[\text{open_tolY} (\text{Facility, E}_4, \text{Fixt}_4, \text{Feat}_2) \geq \text{tol_posY} (\text{Facility, E}_4) + \text{accuracyY} (\text{Facility, E}_4) + \text{fixture_pos_tolY} (\text{Facility, Port}_7, \text{Fixt}_4) + \text{tol_posY} (\text{Facility, E}_3) + \text{fixture_pos_tolY} (\text{Facility, Port}_2, \text{Fixt}_3) + \text{close_tolY} (\text{Facility, E}_3, \text{Fixt}_3, \text{Feat}_2) \]
   \[\Leftrightarrow \infty \geq 0.01 + 0.001 + 0.01 + 0.001 + \infty \Leftrightarrow \infty \geq \infty\]

3. \[\text{open_tolX} (\text{Facility, E}_4, \text{Fixt}_4, \text{Feat}_6) \geq \text{tol_posX} (\text{Facility, E}_4) + \text{accuracyX} (\text{Facility, E}_4) + \text{fixture_pos_tolX} (\text{Facility, Port}_7, \text{Fixt}_4) + \text{tol_posX} (\text{Facility, E}_3) + \text{fixture_pos_tolX} (\text{Facility, Port}_2, \text{Fixt}_3) + \text{close_tolX} (\text{Facility, E}_3, \text{Fixt}_3, \text{Feat}_3) + \text{specifies_tolX} (\text{Facility, Feat}_3, \text{Feat}_6) \]
   \[\Leftrightarrow \infty \geq 0.01 + 0.001 + 0.01 + 0.001 + \infty + 0.001 \Leftrightarrow \infty \geq \infty\]

4. \[\text{open_tolY} (\text{Facility, E}_4, \text{Fixt}_4, \text{Feat}_6) \geq \text{tol_posY} (\text{Facility, E}_4) + \text{accuracyY} (\text{Facility, E}_4) + \text{fixture_pos_tolY} (\text{Facility, Port}_7, \text{Fixt}_4) + \text{tol_posY} (\text{Facility, E}_3) + \text{fixture_pos_tolY} (\text{Facility, Port}_2, \text{Fixt}_3) + \text{close_tolY} (\text{Facility, E}_3, \text{Fixt}_3, \text{Feat}_3) + \text{specifies_tolY} (\text{Facility, Feat}_3, \text{Feat}_6) \]
   \[\Leftrightarrow 0.08 \geq 0.01 + 0.001 + 0.01 + 0.001 + 0.001 + 0.001 \Leftrightarrow 0.08 \geq 0.025\]

Operation 10. \(\text{OP}_{10} = \text{load} (\text{Port}_5, \text{E}_4, \text{Fixt}_5, 1)\). Because the operation is of the load type, \(\text{load} (\text{OP}_{10})\) is true. Thus, the only applicable condition to be satisfied is \(\text{load} (\text{OP}_{10}) \Rightarrow \text{loadable} (\text{part} (\text{OP}_{10}), \text{port} (\text{OP}_{10}), \text{equip} (\text{OP}_{10}), \text{tofixture} (\text{OP}_{10}))\). From the definition of \(\text{OP}_{10}\), \(\text{part} (\text{OP}_{10}) = \text{Part}_1\), \(\text{port} (\text{OP}_{10}) = \text{Port}_5\), \(\text{equip} (\text{OP}_{10}) = \text{E}_4\), and \(\text{tofixture} (\text{OP}_{10}) = \text{Fixt}_5\). The truth of \(\text{loadable} (\text{Part}_1, \text{Port}_5, \text{E}_4, \text{Fixt}_5)\) has already been proven by the validation for Operation 5.

Operation 11. \(\text{OP}_{11} = \text{unload} (\text{Port}_5, \text{E}_4, \text{Fixt}_5, 1)\). Because the operation is of the unload type, \(\text{unload} (\text{OP}_{11})\) is true. Thus, the only applicable condition to be satisfied is \(\text{unload} (\text{OP}_{11}) \Rightarrow \text{unloadable} (\text{part} (\text{OP}_{11}), \text{port} (\text{OP}_{11}), \text{equip} (\text{OP}_{11}), \text{fromfixture} (\text{OP}_{11}))\). From the definition of \(\text{OP}_{11}\), \(\text{part} (\text{OP}_{11}) = \text{Part}_1\), \(\text{port} (\text{OP}_{11}) = \text{Port}_5\), \(\text{equip} (\text{OP}_{11}) = \text{E}_4\), and \(\text{fromfixture} (\text{OP}_{11}) = \text{Fixt}_5\). The truth of \(\text{unloadable} (\text{Part}_1, \text{Port}_5, \text{E}_4, \text{Fixt}_5)\) has already been proven by the validation for Operation 6.

Operation 12. \(\text{OP}_{12} = \text{load} (\text{Port}_4, \text{E}_4, \text{Fixt}_6, 1)\). Because the operation is of the load type, \(\text{load} (\text{OP}_{12})\) is true. Thus, the only applicable condition to be satisfied is \(\text{load} (\text{OP}_{12}) \Rightarrow \text{loadable} (\text{part} (\text{OP}_{12}), \text{port} (\text{OP}_{12}), \text{equip} (\text{OP}_{12}), \text{tofixture} (\text{OP}_{12}))\). From the definition of \(\text{OP}_{12}\), \(\text{part} (\text{OP}_{12}) = \text{Part}_1\), \(\text{port} (\text{OP}_{12}) = \text{Port}_4\), \(\text{equip} (\text{OP}_{12}) = \text{E}_4\),
and tofixture \((\text{OP}_{13}) = \text{Fixt}_6\). The truth of loadable \((\text{Part}_1, \text{Port}_4, E_6, \text{Fixt}_6)\) can be proved by proving that the antecedents of statement #3 are true. From the interpretation of the domain given previously, it is known that isaport \((\text{Port}_4) \land loadable \((E_6, \text{Port}_4), \text{isafeature}(\text{Feat}_2), \text{isafeature}(\text{Feat}_3), \text{isafeature}(\text{Feat}_6), \text{owns}(\text{Part}_1, \text{Feat}_2), \text{owns}(\text{Part}_1, \text{Feat}_3), \text{owns}(\text{Part}_1, \text{Feat}_6), \text{locates}(\text{Fixt}_6, \text{Feat}_2, 1, 1), \text{locates}(\text{Fixt}_6, \text{Feat}_3, 1, 2), \text{locates}(\text{Fixt}_4, \text{Feat}_2, 1, 1), \text{locates}(\text{Fixt}_4, \text{Feat}_6, 1, 2), \text{specifies}(\text{Feat}_3, \text{Feat}_6), \text{loadTrajExists}(E_4, \text{Fixt}_4, \text{Fixt}_6, \text{Port}_4, \text{Part}_1)\). This leaves four mathematical expressions to be proven true because the robot must be capable of locating both datums of the part within the equipment fixture’s open tolerance along the Facility’s x and y coordinate axes. These expressions and their equivalent values are true as follows:

1. \(\text{open}_\text{tolX}(\text{Facility}, E_6, \text{Fixt}_6, \text{Feat}_2) \geq \text{tol}_\text{posX}(\text{Facility}, E_4) + \text{accuracyX}(\text{Facility}, E_4) + \text{fixture}_\text{pos}_\text{tolX}(\text{Facility}, \text{Port}_7, \text{Fixt}_4) + \text{tol}_\text{posX}(\text{Facility}, E_6) + \text{fixture}_\text{pos}_\text{tolX}(\text{Facility}, \text{Port}_4, \text{Fixt}_6) + \text{close}_\text{tolX}(\text{Facility}, E_4, \text{Fixt}_4, \text{Feat}_2) \iff \infty \geq 0.01 + 0.001 + 0.01 + 0.001 + 0.001 + \infty \iff \infty \geq \infty\)

2. \(\text{open}_\text{tolY}(\text{Facility}, E_6, \text{Fixt}_6, \text{Feat}_2) \geq \text{tol}_\text{posY}(\text{Facility}, E_4) + \text{accuracyY}(\text{Facility}, E_4) + \text{fixture}_\text{pos}_\text{tolY}(\text{Facility}, \text{Port}_7, \text{Fixt}_2) + \text{tol}_\text{posY}(\text{Facility}, E_6) + \text{fixture}_\text{pos}_\text{tolY}(\text{Facility}, \text{Port}_4, \text{Fixt}_6) + \text{close}_\text{tolY}(\text{Facility}, E_4, \text{Fixt}_4, \text{Feat}_2) \iff 0.03 \geq 0.01 + 0.001 + 0.001 + 0.01 + 0.001 + 0.001 \iff 0.03 \geq 0.023\)

3. \(\text{open}_\text{tolX}(\text{Facility}, E_6, \text{Fixt}_6, \text{Feat}_3) \geq \text{tol}_\text{posX}(\text{Facility}, E_4) + \text{accuracyX}(\text{Facility}, E_4) + \text{fixture}_\text{pos}_\text{tolX}(\text{Facility}, \text{Port}_7, \text{Fixt}_4) + \text{tol}_\text{posX}(\text{Facility}, E_6) + \text{fixture}_\text{pos}_\text{tolX}(\text{Facility}, \text{Port}_4, \text{Fixt}_6) + \text{close}_\text{tolX}(\text{Facility}, E_4, \text{Fixt}_4, \text{Feat}_6) + \text{specifies}_\text{tolX}(\text{Facility}, \text{Feat}_2, \text{Feat}_3) \iff \infty \geq 0.01 + 0.001 + 0.01 + 0.001 + 0.001 + \infty \iff \infty \geq \infty\)

4. \(\text{open}_\text{tolY}(\text{Facility}, E_6, \text{Fixt}_6, \text{Feat}_3) \geq \text{tol}_\text{posY}(\text{Facility}, E_4) + \text{accuracyY}(\text{Facility}, E_4) + \text{fixture}_\text{pos}_\text{tolY}(\text{Facility}, \text{Port}_7, \text{Fixt}_4) + \text{tol}_\text{posY}(\text{Facility}, E_6) + \text{fixture}_\text{pos}_\text{tolY}(\text{Facility}, \text{Port}_4, \text{Fixt}_6) + \text{close}_\text{tolY}(\text{Facility}, E_4, \text{Fixt}_4, \text{Feat}_6) + \text{specifies}_\text{tolY}(\text{Facility}, \text{Feat}_2, \text{Feat}_3) \iff 0.03 \geq 0.01 + 0.001 + 0.01 + 0.001 + 0.001 + 0.001 \iff 0.03 \geq 0.024\)

Operation 13. \(\text{OP}_{13} = \text{move} (\text{Port}_4, \text{Port}_3, E_6, \text{Fixt}_6, 1)\). Because the operation is of the move type, \(\text{move} (\text{OP}_{13})\) is true. Thus, the only applicable condition to be satisfied is \(\text{move} (\text{OP}_{13}) \Rightarrow \text{moveable} (\text{Part} (\text{OP}_{13}), \text{equip} (\text{OP}_{13}), \text{startport} (\text{OP}_{13}), \text{endport} (\text{OP}_{13}), \text{fixture} (\text{OP}_{13}))\). From the definition of \(\text{OP}_{13}\), \(\text{part} (\text{OP}_{13}) = \text{Part}_4, \text{startport} (\text{OP}_{13}) = \text{Port}_4, \text{endport} (\text{OP}_{13}) = \text{Port}_3, \text{equip} (\text{OP}_{13}) = E_6, \text{fixture} (\text{OP}_{13}) = \text{Fixt}_6\). The truth of reachable \((\text{Part}_1, \text{Port}_4, E_6, \text{Fixt}_6)\) and reachable \((\text{Part}_1, \text{Port}_3, E_6, \text{Fixt}_6)\) were proved in the validation of Operation 3. From statement #5, since \(4 \neq 3\), reachable \((\text{Part}_1, \text{Port}_4, E_6, \text{Fixt}_6)\) and reachable \((\text{Part}_1, \text{Port}_3, E_6, \text{Fixt}_6)\) are true, moveable \((\text{Part}, E_6, \text{Port}_4, \text{Port}_3, \text{Fixt}_6)\) is also true, and \(\text{OP}_{13}\) is valid for this interpretation.

Operation 14. \(\text{OP}_{14} = \text{unload} (\text{Port}_3, E_2, \text{Fixt}_6, 1)\). Because the operation is of the unload type, \(\text{unload} (\text{OP}_{14})\) is true. Thus, the only applicable condition to be satisfied is \(\text{unload} (\text{OP}_{14}) \Rightarrow \text{unloadable} (\text{part} (\text{OP}_{14}), \text{port} \ldots)\).
(OP14), equip (OP14), fromfixture (OP14)). From the definition of OP14, part (OP14) = Part1, port (OP14) = Port3, equip (OP14) = E2, and fromfixture (OP14) = Fixt6. The truth of unloadable (Part1, Port3, E2, Fixt6) can be proved by proving that the antecedents of statement #2 are true. First, it is known that isaport (Port3) ∧ unloadable (E2, Port3), isafeature (Feat2), isafeature (Feat3), isafeature (Feat6), owns (Part1, Feat2), owns (Part1, Feat3), owns (Part1, Feat6), owns (E6, Fixt2), locates (Fixt6, Feat2, 1, 1), locates (Fixt6, Feat3, 1, 2), owns (E2, Fixt2), locates (Fixt2, Feat2, 1, 2), specifies (Feat3, Feat6), and unloadTrajExists (E2, Fixt2, Fixt6, Port3, Part1). This leaves four mathematical expressions to be proven true because the robot must be capable of locating both datums of the part within the gripper’s open tolerance along the Facility’s x and y coordinate axes. These expressions and their equivalent values are true as follows:

1. open_tolX (Facility, E2, Fixt2, Feat2) ≥ tol_posX (Facility, E2) + accuracyX (Facility, E2) + fixture_pos_tolX (Facility, Port6, Fixt2) + toll_posX (Facility, E6) + fixture_pos_tolX (Facility, Port3, Fixt6) + close_tolX (Facility, E6, Fixt6, Feat2)
   ⇔ ∞ ≥ 0.01 + 0.001 + 0.01 + 0.001 + ∞ ⇔ ∞ ≥ ∞

2. open_tolY (Facility, E2, Fixt2, Feat2) ≥ tol_posY (Facility, E2) + accuracyY (Facility, E2) + fixture_pos_tolY (Facility, Port6, Fixt2) + toll_posY (Facility, E6) + fixture_pos_tolY (Facility, Port3, Fixt6) + close_tolY (Facility, E6, Fixt6, Feat2)
   ⇔ 0.08 ≥ 0.01 + 0.001 + 0.01 + 0.001 + 0.03 ⇔ 0.08 ≥ 0.053

3. open_tolX (Facility, E2, Fixt2, Feat3) ≥ tol_posX (Facility, E2) + accuracyX (Facility, E2) + fixture_pos_tolX (Facility, Port6, Fixt2) + toll_posX (Facility, E6) + fixture_pos_tolX (Facility, Port3, Fixt6) + close_tolX (Facility, E6, Fixt6, Feat3) + specifies_tolX (Facility, Feat3, Feat6)
   ⇔ ∞ ≥ 0.01 + 0.001 + 0.01 + 0.001 + 0.03 + 0.001 ⇔ ∞ ≥ 0.054

4. open_tolY (Facility, E2, Fixt2, Feat3) ≥ tol_posY (Facility, E2) + accuracyY (Facility, E2) + fixture_pos_tolY (Facility, Port6, Fixt2) + toll_posY (Facility, E6) + fixture_pos_tolY (Facility, Port3, Fixt6) + close_tolY (Facility, E6, Fixt6, Feat3) + specifies_tolY (Facility, Feat3, Feat6)
   ⇔ ∞ ≥ 0.01 + 0.001 + 0.01 + 0.001 + ∞ + 0.001 ⇔ ∞ ≥ ∞

Operation 15. OP15 = load (Port1, E2, Fixt1, 1). Because the operation is of the load type, load (OP15) is true. Thus, the only applicable condition to be satisfied is load (OP15) → loadable (part (OP15), port (OP15), equip (OP15), tofixture (OP15)). From the definition of OP15, part (OP15) = Part1, port (OP15) = Port1, equip (OP15) = E2, tofixture (OP15) = Fixt1. The truth of loadable (Part1, Port1, E2, Fixt1) has already been established in Section 4.5. Thus, loadable (Part1, Port1, E2, Fixt1) is true, and OP15 is valid for this interpretation.

Since all 15 operations for this example interpretation have been proved to be valid, the next step is to prove that all linear sequences of operations in the operations plan result in a part state that satisfies the goal condition.

From Figure 5, four linear sequences of operations can be derived from the AND/OR graph. These sequences are as follows.
Sequence #1: 1, 2, 3, 4, 5, 6, 7, 8, 9, 12, 13, 14, 15
Sequence #2: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15
Sequence #3: 1, 2, 3, 4, 7, 8, 9, 12, 13, 14, 15
Sequence #4: 1, 2, 3, 4, 7, 8, 9, 10, 11, 12, 13, 14, 15

These sequences and their effects on the state of the part are listed in Table A.2, Table A.3, Table A.4, and Table A.4 as follows.

### Table A.2. Sequence #1

<table>
<thead>
<tr>
<th>Operation Numb</th>
<th>Operation</th>
<th>Part State after operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>unload</td>
<td>(Port_{t}, {Feat_{1}, Feat_{2}, Feat_{3}, Feat_{4}, Feat_{5}, Feat_{6}}, Fixt_{1}, 1)</td>
</tr>
<tr>
<td>1</td>
<td>unload</td>
<td>(Port_{t}, {Feat_{1}, Feat_{2}, Feat_{3}, Feat_{4}, Feat_{5}, Feat_{6}}, Fixt_{1}, 1)</td>
</tr>
<tr>
<td>2</td>
<td>load</td>
<td>(Port_{t}, {Feat_{1}, Feat_{2}, Feat_{3}, Feat_{4}, Feat_{5}, Feat_{6}}, Fixt_{1}, 1)</td>
</tr>
<tr>
<td>3</td>
<td>move</td>
<td>(Port_{t}, {Feat_{1}, Feat_{2}, Feat_{3}, Feat_{4}, Feat_{5}, Feat_{6}}, Fixt_{1}, 1)</td>
</tr>
<tr>
<td>4</td>
<td>unload</td>
<td>(Port_{t}, {Feat_{1}, Feat_{2}, Feat_{3}, Feat_{4}, Feat_{5}, Feat_{6}}, Fixt_{1}, 1)</td>
</tr>
<tr>
<td>5</td>
<td>load</td>
<td>(Port_{t}, {Feat_{1}, Feat_{2}, Feat_{3}, Feat_{4}, Feat_{5}, Feat_{6}}, Fixt_{1}, 1)</td>
</tr>
<tr>
<td>6</td>
<td>unload</td>
<td>(Port_{t}, {Feat_{1}, Feat_{2}, Feat_{3}, Feat_{4}, Feat_{5}, Feat_{6}}, Fixt_{1}, 1)</td>
</tr>
<tr>
<td>7</td>
<td>load</td>
<td>(Port_{t}, {Feat_{1}, Feat_{2}, Feat_{3}, Feat_{4}, Feat_{5}, Feat_{6}}, Fixt_{1}, 1)</td>
</tr>
<tr>
<td>8</td>
<td>process</td>
<td>(Port_{t}, {Feat_{1}, Feat_{2}, Feat_{3}, Feat_{4}, Feat_{5}, Feat_{6}}, Fixt_{1}, 1)</td>
</tr>
<tr>
<td>9</td>
<td>unload</td>
<td>(Port_{t}, {Feat_{1}, Feat_{2}, Feat_{3}, Feat_{4}, Feat_{5}, Feat_{6}}, Fixt_{1}, 1)</td>
</tr>
<tr>
<td>12</td>
<td>load</td>
<td>(Port_{t}, {Feat_{1}, Feat_{2}, Feat_{3}, Feat_{4}, Feat_{5}, Feat_{6}}, Fixt_{1}, 1)</td>
</tr>
<tr>
<td>13</td>
<td>move</td>
<td>(Port_{t}, {Feat_{1}, Feat_{2}, Feat_{3}, Feat_{4}, Feat_{5}, Feat_{6}}, Fixt_{1}, 1)</td>
</tr>
<tr>
<td>14</td>
<td>unload</td>
<td>(Port_{t}, {Feat_{1}, Feat_{2}, Feat_{3}, Feat_{4}, Feat_{5}, Feat_{6}}, Fixt_{1}, 1)</td>
</tr>
<tr>
<td>15</td>
<td>load</td>
<td>(Port_{t}, {Feat_{1}, Feat_{2}, Feat_{3}, Feat_{4}, Feat_{5}, Feat_{6}}, Fixt_{1}, 1)</td>
</tr>
</tbody>
</table>

### Table A.3. Sequence #2

<table>
<thead>
<tr>
<th>Operation Numb</th>
<th>Operation</th>
<th>Part State after operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>unload</td>
<td>(Port_{t}, {Feat_{1}, Feat_{2}, Feat_{3}, Feat_{4}, Feat_{5}, Feat_{6}}, Fixt_{1}, 1)</td>
</tr>
<tr>
<td>1</td>
<td>unload</td>
<td>(Port_{t}, {Feat_{1}, Feat_{2}, Feat_{3}, Feat_{4}, Feat_{5}, Feat_{6}}, Fixt_{1}, 1)</td>
</tr>
<tr>
<td>2</td>
<td>load</td>
<td>(Port_{t}, {Feat_{1}, Feat_{2}, Feat_{3}, Feat_{4}, Feat_{5}, Feat_{6}}, Fixt_{1}, 1)</td>
</tr>
<tr>
<td>3</td>
<td>move</td>
<td>(Port_{t}, {Feat_{1}, Feat_{2}, Feat_{3}, Feat_{4}, Feat_{5}, Feat_{6}}, Fixt_{1}, 1)</td>
</tr>
<tr>
<td>4</td>
<td>unload</td>
<td>(Port_{t}, {Feat_{1}, Feat_{2}, Feat_{3}, Feat_{4}, Feat_{5}, Feat_{6}}, Fixt_{1}, 1)</td>
</tr>
<tr>
<td>5</td>
<td>load</td>
<td>(Port_{t}, {Feat_{1}, Feat_{2}, Feat_{3}, Feat_{4}, Feat_{5}, Feat_{6}}, Fixt_{1}, 1)</td>
</tr>
<tr>
<td>6</td>
<td>unload</td>
<td>(Port_{t}, {Feat_{1}, Feat_{2}, Feat_{3}, Feat_{4}, Feat_{5}, Feat_{6}}, Fixt_{1}, 1)</td>
</tr>
<tr>
<td>7</td>
<td>load</td>
<td>(Port_{t}, {Feat_{1}, Feat_{2}, Feat_{3}, Feat_{4}, Feat_{5}, Feat_{6}}, Fixt_{1}, 1)</td>
</tr>
<tr>
<td>8</td>
<td>process</td>
<td>(Port_{t}, {Feat_{1}, Feat_{2}, Feat_{3}, Feat_{4}, Feat_{5}, Feat_{6}}, Fixt_{1}, 1)</td>
</tr>
<tr>
<td>9</td>
<td>unload</td>
<td>(Port_{t}, {Feat_{1}, Feat_{2}, Feat_{3}, Feat_{4}, Feat_{5}, Feat_{6}}, Fixt_{1}, 1)</td>
</tr>
</tbody>
</table>
Table A.4. Sequence #3

<table>
<thead>
<tr>
<th>Operation Numb</th>
<th>Operation</th>
<th>Part State after operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>start (Port1, {Feat1, Feat2, Feat3, Feat4, Feat5, Feat6}, Fixt1, 1)</td>
<td>(Port1, {Feat1, Feat2, Feat3, Feat4, Feat5, Feat6, Feat7}, Fixt1, 1)</td>
</tr>
<tr>
<td>2</td>
<td>unload (Port1, E2, Fixt1, 1)</td>
<td>(Port1, {Feat1, Feat2, Feat3, Feat4, Feat5, Feat6, Feat7}, Fixt1, 1)</td>
</tr>
<tr>
<td>3</td>
<td>load (Port3, E2, Fixt6, 1)</td>
<td>(Port3, {Feat1, Feat2, Feat3, Feat4, Feat5, Feat6, Feat7}, Fixt6, 1)</td>
</tr>
<tr>
<td>4</td>
<td>move (Port3, Port4, E6, Fixt6, 1)</td>
<td>(Port3, {Feat1, Feat2, Feat3, Feat4, Feat5, Feat6, Feat7}, Fixt6, 1)</td>
</tr>
<tr>
<td>5</td>
<td>unload (Port4, E4, Fixt6, 1)</td>
<td>(Port4, {Feat1, Feat2, Feat3, Feat4, Feat5, Feat6, Feat7}, Fixt6, 1)</td>
</tr>
<tr>
<td>6</td>
<td>load (Port2, E4, Fixt3, 1)</td>
<td>(Port2, {Feat1, Feat2, Feat3, Feat4, Feat5, Feat6, Feat7}, Fixt3, 1)</td>
</tr>
<tr>
<td>7</td>
<td>process (Feat7, E3, Fixt3, 1)</td>
<td>(Port2, {Feat1, Feat2, Feat3, Feat4, Feat5, Feat6, Feat7}, Fixt3, 1)</td>
</tr>
<tr>
<td>8</td>
<td>unload (Port2, E4, Fixt3, 1)</td>
<td>(Port2, {Feat1, Feat2, Feat3, Feat4, Feat5, Feat6, Feat7}, Fixt3, 1)</td>
</tr>
<tr>
<td>9</td>
<td>load (Port5, E4, Fixt5, 1)</td>
<td>(Port5, {Feat1, Feat2, Feat3, Feat4, Feat5, Feat6, Feat7}, Fixt5, 1)</td>
</tr>
<tr>
<td>10</td>
<td>unload (Port5, E4, Fixt5, 1)</td>
<td>(Port5, {Feat1, Feat2, Feat3, Feat4, Feat5, Feat6, Feat7}, Fixt5, 1)</td>
</tr>
</tbody>
</table>

Table A.5. Sequence #4

<table>
<thead>
<tr>
<th>Operation Numb</th>
<th>Operation</th>
<th>Part State after operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>start (Port1, {Feat1, Feat2, Feat3, Feat4, Feat5, Feat6}, Fixt1, 1)</td>
<td>(Port1, {Feat1, Feat2, Feat3, Feat4, Feat5, Feat6, Feat7}, Fixt1, 1)</td>
</tr>
<tr>
<td>2</td>
<td>unload (Port1, E2, Fixt1, 1)</td>
<td>(Port1, {Feat1, Feat2, Feat3, Feat4, Feat5, Feat6, Feat7}, Fixt1, 1)</td>
</tr>
<tr>
<td>3</td>
<td>load (Port3, E2, Fixt6, 1)</td>
<td>(Port3, {Feat1, Feat2, Feat3, Feat4, Feat5, Feat6, Feat7}, Fixt6, 1)</td>
</tr>
<tr>
<td>4</td>
<td>move (Port3, Port4, E6, Fixt6, 1)</td>
<td>(Port3, {Feat1, Feat2, Feat3, Feat4, Feat5, Feat6, Feat7}, Fixt6, 1)</td>
</tr>
<tr>
<td>5</td>
<td>unload (Port4, E4, Fixt6, 1)</td>
<td>(Port4, {Feat1, Feat2, Feat3, Feat4, Feat5, Feat6, Feat7}, Fixt6, 1)</td>
</tr>
<tr>
<td>6</td>
<td>load (Port2, E4, Fixt3, 1)</td>
<td>(Port2, {Feat1, Feat2, Feat3, Feat4, Feat5, Feat6, Feat7}, Fixt3, 1)</td>
</tr>
<tr>
<td>7</td>
<td>process (Feat7, E3, Fixt3, 1)</td>
<td>(Port2, {Feat1, Feat2, Feat3, Feat4, Feat5, Feat6, Feat7}, Fixt3, 1)</td>
</tr>
<tr>
<td>8</td>
<td>unload (Port2, E4, Fixt3, 1)</td>
<td>(Port2, {Feat1, Feat2, Feat3, Feat4, Feat5, Feat6, Feat7}, Fixt3, 1)</td>
</tr>
<tr>
<td>9</td>
<td>load (Port5, E4, Fixt5, 1)</td>
<td>(Port5, {Feat1, Feat2, Feat3, Feat4, Feat5, Feat6, Feat7}, Fixt5, 1)</td>
</tr>
<tr>
<td>10</td>
<td>unload (Port5, E4, Fixt5, 1)</td>
<td>(Port5, {Feat1, Feat2, Feat3, Feat4, Feat5, Feat6, Feat7}, Fixt5, 1)</td>
</tr>
<tr>
<td>11</td>
<td>unload (Port5, E4, Fixt5, 1)</td>
<td>(Port5, {Feat1, Feat2, Feat3, Feat4, Feat5, Feat6, Feat7}, Fixt5, 1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>12</td>
<td>load (Port₄, E₄, Fixt₆, 1)</td>
<td>(Port₄, {Feat₁, Feat₂, Feat₃, Feat₄, Feat₅, Feat₆, Feat₇}, Fixt₆, 1)</td>
</tr>
<tr>
<td>13</td>
<td>move (Port₄, Port₃, E₆, Fixt₆, 1)</td>
<td>(Port₃, {Feat₁, Feat₂, Feat₃, Feat₄, Feat₅, Feat₆, Feat₇}, Fixt₆, 1)</td>
</tr>
<tr>
<td>14</td>
<td>unload (Port₃, E₂, Fixt₁, 1)</td>
<td>(Port₆, {Feat₁, Feat₂, Feat₃, Feat₄, Feat₅, Feat₆, Feat₇}, Fixt₁, 1)</td>
</tr>
<tr>
<td>15</td>
<td>load (Port₁, E₂, Fixt₁, 1)</td>
<td>(Port₁, {Feat₁, Feat₂, Feat₃, Feat₄, Feat₅, Feat₆, Feat₇}, Fixt₁, 1)</td>
</tr>
</tbody>
</table>

In order to prove that each of the four linear sequences result in a state that satisfies the goal condition, the ending state of each sequence \((p, flist)\) must satisfy the following condition from Chapter 4:

\[
\forall Part, \forall p, \forall flist, \forall fixt, \forall intent \ (\text{isapart} (Part) \land \text{isaport} (p) \land \text{isalistoffeatures} (flist) \land \text{isafixture} (fixt) \land \text{isafixtureintent} (intent) \land p = \text{goalp} (Part) \land \text{goalfeaturespresent} (\text{node} (Part, p, flist, fixt, intent)) \land fixt = \text{goalfixture} (Part) \land intent = \text{goalfixtureintent} (Part) \Rightarrow \text{goal} (\text{node} (Part, p, flist, fixt, intent)))
\]

First, note that all four sequences end with the part state \((p, flist, fixture, intent) = (\text{Port}₁, \{\text{Feat}₁, \text{Feat}₂, \text{Feat}₃, \text{Feat}₄, \text{Feat}₅, \text{Feat}₆, \text{Feat}₇\}, \text{Fixt}₁, 1)\). Furthermore, \textbf{GoalFList₁} = \{\text{Feat}₁, \text{Feat}₂, \text{Feat}₃, \text{Feat}₄, \text{Feat}₅, \text{Feat}₆, \text{Feat}₇\}, \text{goalp} (\text{Part}₁) = \text{Port}₁, \text{goalfixture} (\text{Part}₁) = \text{Fixt}₁, and \text{goalfixtureintent} (\text{Part}₁) = 1. Since these statements satisfy the conditions for the goal, \text{goal} (\text{node} (\text{Port}₁, \{\text{Feat}₁, \text{Feat}₂, \text{Feat}₃, \text{Feat}₄, \text{Feat}₅, \text{Feat}₆, \text{Feat}₇\}, \text{Fixt}₁, 1)) is true, and all linear sequences satisfy the goal condition for \text{Part}₁.
ILLUSTRATION OF PART FILE SPECIFICATION

"twoholes"  # part name
1  # part number
9  # total number of features in this list

# the features that define the raw material
"flatsurface"  # feature is a flat surface
1  # feature 1
1  # feature is part of blank
1  # number of locating surfaces
5  # primary datum
1.0  # nominal distance to datum
0.025  # +/- position tolerance
4  # number of boundary points to surface
0.0, 0.0, 0.0 # point 1
1.0, 0.0, 0.0 # point 2
1.0, 0.0, 1.0 # point 3
0.0, 0.0, 1.0 # point 4
64  # surface finish

"flatsurface"  # feature is a flat surface
2  # feature 2
1  # feature is part of blank
1  # number of locating surfaces
4  # primary datum
1.0  # nominal distance to datum
0.025  # +/- position tolerance
4  # number of boundary points to surface
1.0, 0.0, 0.0 # point 1
1.0, 1.0, 0.0 # point 2
1.0, 1.0, 1.0 # point 3
1.0, 0.0, 1.0 # point 4
64  # surface finish

"flatsurface"  # feature is a flat surface
3  # feature 3
1  # feature is part of blank
1  # number of locating surfaces
6  # primary datum
1.0  # nominal distance to datum
0.025  # +/- position tolerance
4  # number of boundary points to surface
1.0, 0.0, 1.0 # point 1
1.0, 1.0, 1.0 # point 2
0.0, 1.0, 1.0 # point 3
0.0, 0.0, 1.0 # point 4
64  # surface finish

"flatsurface"  # feature is a flat surface
4  # feature 4
1  # feature is part of blank
1  # number of locating surfaces
2  # primary datum
1.0  # nominal distance to datum
0.025  # +/- position tolerance
4  # number of boundary points to surface
0.0, 0.0, 0.0 # point 1
0.0, 0.0, 1.0 # point 2
0.0, 1.0, 1.0 # point 3
0.0, 1.0, 0.0 # point 4
64   # surface finish
"flatsurface"  # feature is a flat surface
5   # feature 5
1   # feature is part of blank
1   # number of locating surfaces
1.0   # primary datum
0.025   # +/- position tolerance
4   # number of boundary points to surface
0.0, 1.0, 0.0  # point 1
1.0, 1.0, 0.0  # point 2
1.0, 1.0, 1.0  # point 3
0.0, 1.0, 1.0  # point 4
64   # surface finish
"flatsurface"  # feature is a flat surface
6   # feature 6
1   # feature is part of blank
1   # number of locating surfaces
1.0   # primary datum
0.025   # +/- position tolerance
4   # number of boundary points to surface
0.0, 0.0, 0.0  # point 1
1.0, 0.0, 0.0  # point 2
1.0, 1.0, 0.0  # point 3
0.0, 1.0, 0.0  # point 4
64   # surface finish
"hole"  # feature is a hole
7   # feature number 7
0   # feature is not part of blank
3   # number of locating surfaces
1.0   # primary datum
0.025   # +/- position tolerance
4   # secondary datum
0.5   # nominal distance to datum
0.025   # +/- position tolerance
3   # tertiary datum
0.5   # nominal distance to datum
0.025   # +/- position tolerance
0.50   # diameter
0.008   # +/- diametric tolerance
100   # surface finish of hole
"hole"  # feature is a hole
8   # feature number 8
0   # feature is not part of blank
3   # number of locating surfaces
4   # primary datum
1.0   # secondary datum
0.025   # +/- position tolerance
1   # tertiary datum
0.5   # nominal distance to datum
0.025   # +/- position tolerance
3   # tertiary datum
0.5   # nominal distance to datum
0.025   # +/- position tolerance
0.50   # diameter
0.008   # +/- diametric tolerance
100   # surface finish of hole
"hole"
9    # feature number 9
0    # feature is not part of blank
3    # number of locating surfaces
5    # primary datum
1.0  # nominal distance to datum
0.025 # +/- position tolerance
4    # secondary datum
0.5  # nominal distance to datum
0.025 # +/- position tolerance
3    # tertiary datum
0.5  # nominal distance to datum
0.025 # +/- position tolerance
0.50  # depth of hole
0.50  # diameter
0.008 # +/- diametric tolerance
100  # surface finish of hole
"(*7 8*)" # feature precedence sequence (only includes 2 features)

"Kardex ASRF" # name of start buffer (for raw material)
"Kardex ASRF" # name of end buffer (for finished part)

"aluminum"   # base material name
"1108"        # base material code
148           # hardness of base material
Appendix C

LIST OF PREDICATES AND FUNCTIONS

**Predicates**

- **fixt_intent (fixt, I)**  true if I is a fixture intention of fixt
- **goalfeaturespresent (node)** true if all goal features for a part are in the node representing the part’s state.
- **isabuffer (a)**  true if a is a material storage buffer entity
- **isafacility (f)**  true if f is a facility
- **isafixture (f)**  true if f is a fixture entity
- **isafixtureintent (a)** true if a is a fixture intent
- **isafixturelocator (a)** true if a is a fixture locator
- **isafeature (a)**  true if a is a feature entity
- **isagoal (n)**  true if node n is a goal part state node
- **isagoalfeature (a)** true if a is a goal feature entity
- **isahandler (a)**  true if a is a material handling entity
- **isahole (a)** true if a is a hole feature
- **isalistoffeatures (flist)** true if flist is a list of features
- **isanequipment (a)** true if a is an equipment entity
- **isanoperation (OP)** true if OP is an operation
- **isaprocessor (a)** true if a is a material processor entity
- **isaprocessworkstation (a)** true if a is a material processing workstation
- **isastorageworkstation (a)** true if a is a material storage workstation
- **isasurface (a)**  true if a is a surface feature
- **isastool (t)**  true if t is a tool entity
- **isatransporter (a)** true if a is a material transporter entity
- **isatransportworkstation (a)** true if a is a material transporting workstation
- **isworkstation (a)** true if a is a workstation
- **locates (fixt, f, I, L)**  true if fixt is designed to locate f’s position with intention I using locator L
- **load (OP)** true if operation OP is of MH type and a load operation
- **loadable (E, P)**  true if E is intended to load to port P
- **loadable (p, E, P, fixt)** true if E is intended to load part p to fixture fixt at port P
- **loadTrajExists (E, g, f, P, p)**  true if trajectory exists so that E can load part p to port P in fixture f using gripper fixture g.
- **make (OP)** true if operation OP is of MP type
- **makes (t, f)** true if tool t is designed to make feature f
- **makes (E, f, fixt)** true if material processor E can make feature f using tool t
- **makes_hole (t, d, fin, mat, h)** true if tools is capable of making hole feature with diameter d and finish fin on material mat with hardness h.
- **move (OP)** true if operation OP is of MT type
- **moveable (part, E, p1, p2, fixt)** true if material transporter E can move the part using fixt from ports p1 to p2
- **owns (a, b)** true if a owns b
- **precondition (a, b)** true if a must exist before b may exist
- **reachable (P, E)** true if equipment E can move to port P
- **reachable (p, P, E, fixt)** true if equipment E can move part p to port P using fixture fixt.
- **specifies (a, b)** true if a is designed to specify b’s position
- **storabe (p, E, fixt)** true if equipment E can store part p using fixture fixt.
- **unload (OP)** true if operation OP is of MH type and an unload operation
- **unloadable (E, P)** true if E is intended to unload from port P
- **unloadable (p, E, P, fixt)** true if E is intended to unload part p from fixture fixt at port P
- **unloadTrajExists (E, g, f, P, p)**  true if trajectory exists so that E can unload part p from port P in fixture f using gripper fixture g.
Functions

accuracyX(E)  returns accuracy of placement of tools or grippers by E along E’s X axis
accuracyY(E)  returns accuracy of placement of tools or gripper fixtures by E along E’s Y axis
accuracyZ(E)  returns accuracy of placement of tools or gripper fixtures by E along E’s Z axis
accuracyX(E, t, f)  returns accuracy of the position of f along E’s X axis if created by t without placement errors
accuracyY(E, t, f)  returns accuracy of the position of f along E’s Y axis if created by t without placement errors
accuracyZ(E, t, f)  returns accuracy of the position of f along E’s Z axis if created by t without placement errors

close_tolX(E, fixt, f)  returns tolerance of position of f if located by closed fixture fixt at equipment E along E’s X axis
close_tolY(E, fixt, f)  returns tolerance of position of f if located by closed fixture fixt at equipment E along E’s Y axis
close_tolZ(E, fixt, f)  returns tolerance of position of f if located by closed fixture fixt at equipment E along E’s Z axis
close_tolX(F, E, fixt, f)  returns tolerance of position of f if located by closed fixture fixt at equipment E along F’s X axis
close_tolY(F, E, fixt, f)  returns tolerance of position of f if located by closed fixture fixt at equipment E along F’s Y axis
close_tolZ(F, E, fixt, f)  returns tolerance of position of f if located by closed fixture fixt at equipment E along F’s Z axis

goalfixture(Part)  returns the goal fixture for part Part
goalfixtureintent(Part)  returns the goal fixture intent for part Part
goalp(Part)  returns the goal port for part Part
home_trans(E)  returns transformation of gripper fixture at home position relative to port of robot equipment E.

inverse_kin(T)  returns a robot joint configuration for the given transformation matrix T.
life(T)  returns tool life of tool T
material(part)  returns material of part
open_tol(fixt, f)  returns tolerance of position of f if located by opened fixture fixt
open_tolX(E, fixt, f)  returns tolerance of position of f if located by opened fixture fixt at equipment E along E’s X axis
open_tolY(E, fixt, f)  returns tolerance of position of f if located by opened fixture fixt at equipment E along E’s Y axis
open_tolZ (E, fixt, f)  returns tolerance of position of f if located by opened fixture fixt at equipment E along E’s Z axis
open_tolX (F, E, fixt, f)  returns tolerance of position of f if located by opened fixture fixt at equipment E along F’s X axis
open_tolY (F, E, fixt, f)  returns tolerance of position of f if located by opened fixture fixt at equipment E along F’s Y axis
open_tolZ (F, E, fixt, f)  returns tolerance of position of f if located by opened fixture fixt at equipment E along F’s Z axis
port (E)  returns the port owned by equipment E
specifies_dist (datum, feat)  returns nominal distance of datum’s specification of feat’s position
specifies_tol (datum, feat)  returns tolerance of datum’s specification of feat’s position
specifies_tolX (E, datum, feat)  returns tolerance of datum’s specification of feat’s position along E’s X axis
specifies_tolY (E, datum, feat)  returns tolerance of datum’s specification of feat’s position along E’s Y axis
specifies_tolZ (E, datum, feat)  returns tolerance of datum’s specification of feat’s position along E’s Z axis
specifies_tolX (F, E, datum, feat)  returns tolerance of datum’s specification of feat’s position along F’s X axis
specifies_tolY (F, E, datum, feat)  returns tolerance of datum’s specification of feat’s position along F’s Y axis
specifies_tolZ (F, E, datum, feat)  returns tolerance of datum’s specification of feat’s position along F’s Z axis
startport (OP)  returns starting port of MT operation OP
tol_posX (F, port)  returns tolerance of port’s position along F’s X axis
tol_posY (F, port)  returns tolerance of port’s position along F’s Y axis
tol_posZ (F, port)  returns tolerance of port’s position along F’s Z axis
tool (OP)  returns tool used in MP operation OP
tofixture (OP)  returns fixture loaded by MH operation OP
time (OP)  returns estimated time of operation OP
vect_length (^V)  returns length of vector ^V