GENERATING MOTION-ECONOMICAL PLANS FOR MANUAL OPERATIONS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

ÖZGEN CANAN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

SEPTEMBER 2005

Approval of the Graduate School of Natural and Applied Sciences

_____

Prof. Canan Özgen
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science of Philosophy.

_____

Prof. Ayşe Kiper
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science of Philosophy.

_____

Dr. Ayşenur Birtürk
Supervisor

Examining Committee Members

Prof. Faruk Polat	(METU,CENG)	_____
Dr. Ayşenur Birtürk	(METU,CENG)	_____
Assoc. Prof. Ferda Nur Alparslan	(METU,CENG)	_____
Assoc. Prof. Nihan Kesim Çiçekli	(METU,CENG)	_____
Asst. Prof. Bilge Say	(METU, II)	_____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name :

Signature :

ABSTRACT

GENERATING MOTION-ECONOMICAL PLANS FOR MANUAL OPERATIONS

CANAN, ÖZGEN

Ms. Sc., Department of Computer Engineering

Supervisor : Ayşenur Birtürk

September 2005, 149 pages

This thesis discusses applying AI planning tools for generating plans for manual operations. Expertise of motion economy domain is used to select good plans among feasible ones. Motion economy is a field of industrial engineering, which deals with observing, reporting and improving manual operations. Motion economy knowledge is organized in principles regarding the sequences and characteristics of motions, arrangement of workspace, design of tools etc. A representation scheme is developed for products, workspace and hand motions of manual operations. Operation plans are generated using a forward chaining planner (TLPLAN). Planner and representation of domain have extensions compared to a standard forward chaining planner, for supporting concurrency, actions with resources and actions with durations. We formulated principles of motion economy as search control temporal formulas. In addition to motion economy rules, we developed rules for simulating common sense of humans and goal-related rules for preventing absurd sequences of actions in the plans. Search control rules constrain the problem and reduce search complexity. Plans are evaluated during search. Paths, which are not in conformity with the principles of motion economy, are pruned with motion economy rules. Sample problems are represented and solved. Diversity of types of these problems shows the generality of representation scheme. In experimental runs, effects of motion economy principles on the generation of plans are observed and analyzed.

Keywords: Motion Economy, Domain-specific Knowledge, Planning, Temporal Search Control

ÖZ

EL İLE YAPILAN İŞLEMLER İÇİN HAREKET AÇISINDAN EKONOMİK PLANLAR OLUŞTURULMASI

CANAN, ÖZGEN

Yüksek Lisans, Bilgisayar Mühendisliği Bölümü

Tez Yöneticisi : Ayşenur Birtürk

Eylül 2005, 149 sayfa

Bu çalışma yapay zeka planlama araçlarının el ile yapılan işlemler için planlar oluşturulmasında kullanılmasını inceler. Hareket ekonomisi uzmanlığı gerçekleştirilebilir planların arasından iyi olanların seçilmesinde kullanılmıştır. Hareket ekonomisi endüstri mühendisliğinin, el ile yapılan işlemleri gözlemleyen, sunan ve iyileştiren bir dalıdır. Hareket ekonomisi bilgi dağarcığı hareket tipleri ve sıralamaları, iş yeri düzenlemesi, alet tasarımı vb. ile ilgili prensipler halinde düzenlenmiştir. El ile yapılan işlemlerdeki ürünler, iş yeri ve el hareketleri için bir gösterim şeması geliştirilmiştir. El ile yapılan işlemler bir forward chaining planlayıcı (TLPLAN) kullanılarak oluşturulmuştur. Planlayıcının ve problem alanının gösteriminin standart bir ileri zincirleme planlayıcıya göre, eşzamanlılık, kaynaklı eylemler ve süreli eylemlerin desteklenmesi için uzantıları bulunmaktadır. Hareket ekonomisi prensiplerini zaman belirten arama kontrol formülleri olarak formüle ettik. Hareket ekonomisi kurallarına ek olarak, planlarda saçma hareket dizilerine engel olacak, sağduyuyu taklit eden kurallar ve amaçla ilgili kurallar geliştirdik. Arama kontrol kuralları problemi kısıtlamakta ve arama karmaşıklığını azaltmaktadır. Planlar arama zarfında değerlendirilmektedir. Hareket ekonomisi kurallarına uygun olmayan yollar hareket ekonomisi kuralları tarafından budanırlar. Örnek problemler sunuldu ve çözüldü. Bu problemlerin tiplerindeki çeşitlilik, gösterim mekanizmasının ifade edebilirliğini göstermektedirler. Deneysel çalıştırmalarda, hareket ekonomisi prensiplerinin oluşturulan planlar üzerindeki etkisi gözlemlendi ve incelendi.

Anahtar Kelimeler: Hareket ekonomisi, alana özel bilgi, planlama, zaman belirten arama kontrolü

To John Frusciante

ACKNOWLEDGEMENTS

I wish to express my deepest gratitude to my supervisor Dr. Ayşenur Birtürk for her guidance, advice, criticism, encouragements and insight throughout the research.

I would also like to thank Dr. Nilüfer Önder from Michigan Technological University for her suggestions and comments.

I would also like to thank my family for the motivation and self-confidence they provided me.

I would also like to thank Mrs. Naz Dino, Mrs. Fatma Ataman, Mr. Bülent Kandiller and Mr. Ali Alpay for their tolerances and allowances for the time I spent for this study.

The technical assistance of Ms.Seçil Arıduru, Mr. Erdem Yılgör, Mr. Cüneyt Yılmaz and Ms. İrem Aydın for developing and presenting sample problems is gratefully acknowledged.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

CHAPTER 1

INTRODUCTION

This study focuses on modeling problem solving process of motion economy. A representation scheme for the motion economy knowledge and a planner mechanism, which incorporates domain-specific knowledge, is constructed. Motion economy deals with generating and improving plans for a certain group of simple tasks in industries. Given a desired configuration of the product and/or arrangement of the workplace, a motion economy specialist tries to generate a plan, which is composed of fundamental hand motions and is economic from the point of view of motion economy expertise. The tasks, which are in the scope of motion economy, may be classified as manual operations. Operations, which require the use of arms on a limited space for completion, are defined as manual operations. They do not require traveling of the worker, use of the body parts other than arms and hands, like foot or trunk. Manual operations may take place in production systems, as well as daily life. Cooking tasks, typing, integration of small assemblies and painting are just a few examples of these. All sorts of manual operations are in the scope of this study. Motion economy formalizes a set of fundamental hand motions, which suffices to describe every manual operation, and deals with the modeling and analysis of these operations.

In many industries, there are production sequences, which require simple manual operations. Although there is a trend to eliminate these kinds of jobs in a production system by having industrial manipulators do the job instead, manual operations continue to occupy a major part of working hours. Manual operations take place not only in production systems. Work methods design deals with both process planning and design of the operation method. Process planning is one of the major fields that AI planning is applied over the years. Process planning provides a fruitful domain for

planning. However, AI planning tools have not been used in work-study so far. Saving from time or considering ergonomics has not been the concern of these studies. Although motion study has been used for workshop operations for decades, it requires intense examination of the job by specialists. The structure of the implementation is mostly composed of checklists. The job gets harder and harder as the number of primitive motions increases. Handling or aiding this work by computers will save valuable time and effort in methods design phase. Conventional methods for sequencing operations, such as project scheduling, search only possible orderings of the tasks disregarding cognitive and ergonomic constraints. Planning for the ease of the worker is essential for reducing operational disorders and productivity in repetitive tasks. As well as production, other sectors such as service can benefit from this study.

This study is also related to two other research areas, planning in AI and assembly planning. Knowledge representation of the products is adapted from assembly sequence planning studies. Operation plans are constructed from detailed specifications of products. Assembly planning is also used in representing the product configuration during phases of planning, where the product is partly established.

State-based planning methods of AI are employed for generating the plans. AI planning and motion economy cooperate to control the search of the planner. Temporal logic is used to represent motion economy principles, which guide the search of the planner. Quality of final plans is evaluated with the plans' conformity with motion economy rules. The model of the study is given in Figure 1.1.

Next chapter of the thesis gives a brief on motion economy. The general concepts of assembly sequence planning and planning in AI are also described in chapter 2. Chapter 3 gives the detailed explanation of our domain and presents the world model that we have developed and the knowledge representation. Chapter 4 details the

generation of the plans. Changes in world parameters during search are explained. Chapter 5 discusses search control mechanism. How search control is used for evaluation and selection of "good" plans is described. Chapter 6 presents sample problems and experimental results. Chapter 7 consists of possible extensions and future work. Chapter 8 is the conclusion section of the thesis.

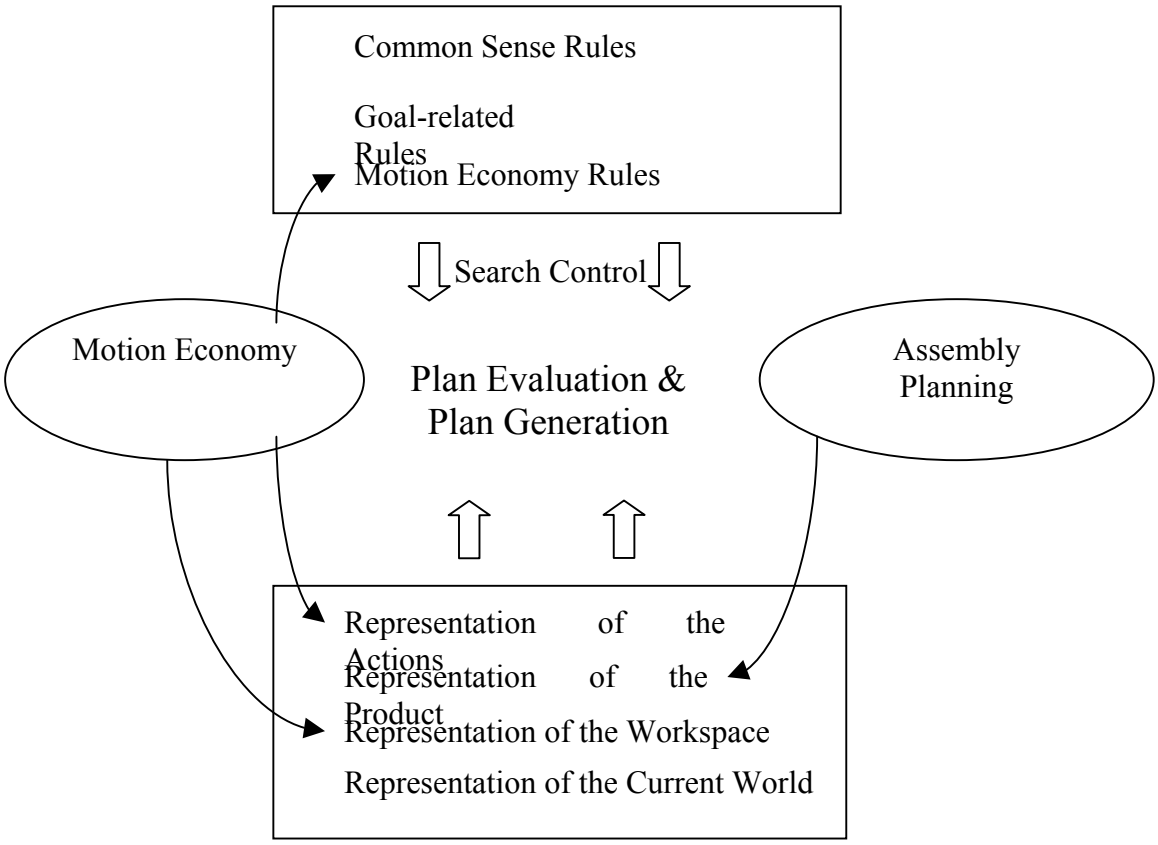

Figure 1.1 The Model

CHAPTER 2

BACKGROUND

2.1 Motion Economy

In the broadest view, manufacturing a new product is composed of three phases, which are planning, pre-production and production. Planning phase comprises the design of the product, the design of the processes, the design of the work method, the design of the tools and equipment, the design of the plant layout and determining the time standards.

Motion and time study is a field of industrial engineering, which deals with the design and improvement of the operations in a production system. Motion study is related to designing a preferred method to achieve a task, which is also referred as methods study, work methods design, work-study and work design. It deals both with the design of the processes and with the design of the work method. The design of the processes is deciding the production operations and their sequences, whereas the design of the work method focuses on each of these operations one by one and aims to fit the operator to the job, rearranging the order of motions in the operation. Main approaches to be considered in finding the preferred method are eliminating all unnecessary work, combining operations or elements, changing the sequence of operations and simplifying the necessary operations.

Motion economy is a sub-field of motion and time study, where efficiency gain is achieved by reorganizing a proposed method. Frank B. Gilbreth and his wife Lillian M. Gilbreth originated motion economy. In [2] Gilbreth states that

> *Motion economy consists of dividing work into the most fundamental elements possible; studying these elements separately and in relation to one another; and from these studied elements, when timed, building methods of least waste.*

Gilbreth introduced micromotion study to analyze the operations by breaking them into primitive steps. Micromotion study consists of filming the operation, generating operation charts from the film and analyzing operations for improvement. However, for most of the operations the analyst can derive a chart by merely observing the operator at work. No timing is needed to conduct motion study.

Most of the work is done with two hands and few fundamental motions are common in all of these tasks. Gilbreth came up with fundamental subdivisions of motion to describe all manual operations. He named these primitive actions as *therbligs*, which is Gilbreth spelled backwards. There are 17 *therbligs*, which are necessary to cover all the operations. The descriptions of the *therbligs* are given below, taken from [3].

Search (Sh): that part of the cycle during which the eyes or the hands are hunting or groping for the object. Search begins when the eyes or hands begin to hunt for the object, and ends when the object has been found.
Select (Se): the choice of one object from among several. In many cases, it is difficult if not impossible to determine where the boundaries lie between search and select. For this reason it is often the practice to combine them, referring to both as the one therblig select. Using this broader definition, select than refers to the hunting and locating of one object from among several. Select begins with when the eyes or hands begin to hunt for the object and ends with the desired object have been located.
Grasp (G): taking hold of an object, closing the fingers around it prepatory to picking it up, holding it or manipulating it. Grasp begins when the hand or fingers first make contact with the object, and ends when the hand has obtained control of it.
Transport empty (TE): moving the empty hand in reaching for an object. It is assumed that the hand moves without resistance toward or away from the object.

Transport empty begins when the hand begins to move without load or resistance, and ends when the hand stops moving.

Transport loaded (TL): moving an object from one place to another. The object may be carried in the hands or fingers, or it may be moved from one place to another by sliding, dragging, or pushing it along. Transport loaded also refers to moving the empty hand against resistance. Transport loaded begins when the hand begins to move an object or encounter resistance, and ends when the hand stops moving.

Hold (H): retention of an object after it has been grasped, no movement of the object taking place. Hold begins when the movement of the object stops, and ends with the start of next therblig.

Release load (RL): letting go of the object. Release load begins when the object starts to leave the hand, and ends when the object has been completely separated from the hand or fingers.

Position (P): turning or locating an object in such a way that it will be properly oriented to fit into the location for which it is intended. It is possible to position an object during the motion transport loaded. The carpenter, for example, may turn the nail into position for using while carrying it to the board into which it will be driven. Position begins when the hand begins to turn or locate the object, and ends when the object has been placed in the desired position or location.

Pre-position (PP): locating an object in a predetermined place, or locating it in the correct position for some subsequent motion. Pre-position is the same as position except that the object is located in the approximate position that will be needed later. Usually a holder, bracket, or special container of some kind holds the object in a way that permits it to be grasped easily in the position in which it will be used. Pre-position is the abbreviated term used for pre-position for the next operation.

Inspect (I): examining an object to determine whether or not it complies with standard size, shape, color, or other qualities previously determined. The inspection may employ sight, hearing, touch, odor, or taste. Inspect is predominantly a mental reaction and may occur simultaneously with other therbligs. Inspect begins when the

eyes or other parts of the body begin to examine the object, and ends when the examination has been completed.

Assemble (A): placing one object into or on another object with which it becomes an integral part. Assemble begins as the hand starts to move the part into its place in the assembly, and ends when the hand has completed the assembly.

Disassemble (DA): separating one object from another object of which it becomes an integral part. Disassemble begins when the hand starts to remove one part from the assembly, and ends when the hand has separated the part completely from the remainder of the assembly.

Use (U): manipulating a tool, device, or piece of apparatus for the purpose for which it was intended. Use may refer to an almost infinite number of particular cases. It represents the motion for which the preceding motions have been more or less preparatory and for which the ones that follow are supplementary. Use begins when the hand starts to manipulate the tool or device, and ends when the hand ceases the application.

Unavoidable delay (UD): a delay beyond the control of the operator. Unavoidable delay may result from either of the following causes: a) a failure or interruption in the process: b) an arrangement of the operation that prevents one part of the body from working while other body members are busy. Unavoidable delay begins when the hand stops its activity, and ends when activity is resumed.

Avoidable delay (AD): any delay of the operator for which he or she is responsible and over which he or she has control. It refers to delays, which the operator may avoid if desired. Avoidable delay begins when the prescribed sequence of motions is interrupted, and ends when the standard work method is resumed.

Plan (P): a mental reaction, which precedes the physical movement that is, deciding how to proceed with the job. Plan begins at the point where the operator begins to work out the next step of the operation, and ends when the procedure to be followed has been determined.

Rest for overcoming fatigue (R): a fatigue or delay factor or allowance provided to permit the worker to recover from the fatigue incurred by the work. Rest begins when the operator stops working, and ends when work is resumed.

Besides cameras and stopwatches, which are used for filming and timing the operations, the only tool that guides the analysts in motion economy is the operation chart. Operation charts list the fundamental motions, therbligs, of the two hands, while they are performing a task, simultaneously. Operation chart is also called right hand-left hand chart. A right hand-left hand chart for a simple operation is given in Figure 2.1. The task is to assemble two washers with a bolt, which are stored originally in a container. Left hand picks up bolt from the container and holds it. Right hand picks up a washer from the container and assembles it to the washer. Then it picks up the other type of washer and assembles it to the subassembly. Finally, left hand carries the final assembly to a bin. Recall that same sequence of therbligs could define some other operation from any domain.

Planning and improvement process is usually the responsibility of industrial engineers and staff specialists. The aim in developing better methods is to reduce costs in terms of time, materials and energy. The number of primitive motions or amount of time required to complete a task can be a measure of improvement, as well as ergonomic concerns such as human fatigue and comfort. Human fatigue can occur in both physical and mental terms. Asymmetrical motions tire the muscles of the arms, while selecting continuously from several alternatives challenges the brain.

LEFT HAND		RIGHT HAND	
Therblig	**Description**	**Therblig**	**Description**
Transport empty	Reaches for bolt		
Select	Selects and grasps bolt		
Grasp			
Transport loaded	Carries bolt to working position	Transport empty	Reaches the container for first washer
Position	Positions bolt	Select	Selects and grasps the first washer
		Grasp	
Hold	Holds bolt	Transport loaded	Carries washer to bolt
		Assemble	Assembles washer and releases
		Release	
		Transport empty	Reaches the container for second washer
		Select	Selects and grasps the washer
		Grasp	
		Transport loaded	Carries washer to bolt
		Assemble	Assembles washer and releases
		Release	
Transport loaded	Carries assembly to the bin		
Release	Releases assembly to the bin		

Figure 2.1 Right hand-left hand chart

The designs, represented by standard charts, guide the improvement process. Although there are standard methods for timing, observing and evaluating work, there are no standard procedures in the methods design and improvement cycle. The expertise is organized as checklists or some rules of thumb. There are principles of motion economy related the use of human body, the arrangement of the work place and the design of tools and equipment. First three principles of motion economy related to the use of human body are as follows [3],

- *The two hands should begin as well as complete their motions at the same time.*
- *The two hands should not be idle at the same time except during rest periods.*
- *Motions of the arms should be made in opposite and symmetrical directions and should be made simultaneously.*

For each therblig, there are general considerations, which can be organized as checklists. Below are some of the questions arising with the use of the therblig *select* [3]:

- *Is the layout such as to eliminate searching for articles?*
- *Are parts and materials properly labeled?*
- *Can parts be pre-positioned during preceding operation?*

LEFT HAND		RIGHT HAND	
Therblig	**Description**	**Therblig**	**Description**
Transport empty	Reaches for bolt		
Select	Selects and grasps bolt		
Grasp			
Transport loaded	Carries bolt to working position		
Position	Positions bolt to fixture		
Transport empty	Reaches the container for second washer	Transport empty	Reaches the container for first washer
Select	Selects and grasps the washer	Select	Selects and grasps the washer
Grasp		Grasp	
Transport loaded	Carries washer to bolt	Transport loaded	Carries washer to bolt
Assemble	Assembles washer and releases	Assemble	Assembles washer and releases
Release		Release	
Select	Selects and grasps the assembly		
Grasp			
Transport loaded	Carries assembly to the bin		
Release	Releases assembly to the bin		

Figure 2.2 Right hand-left hand chart for the improved operation

The application of these principles is a though task for it requires too much attention and expertise from the analyst. The assembly operation shown in Figure 2.1 is rather an inefficient way to do the task. In light of the general principles, we can conclude that the holding of the bolt must be eliminated by the use of a fixture, and selecting parts from the same container must be eliminated with the use of additional containers. This way the operation will be less tiring and quicker. When the operation is longer and there are many more therbligs in the design, the job of the

analyst becomes harder. It will be much more difficult to capture the opportunities of improvement in the orderings of the therbligs. The right hand left hand chart for the improved operation is given in Figure 2.2.

2.2 Assembly Sequence Planning

Assembly sequence planning involves the generation, evaluation and selection of feasible assembly sequences of a product [1]. Assembly sequence planning has an important role in designing efficient production systems. Sequence of assembly operations affects production costs, when production time and setup costs are considered.

Representation of assemblies must contain not only information about parts and contacts between the parts, but also information about the feasibility constraints of the connections. Bourjault [4] used graph of connections for representing assemblies. There is one node for each part of the assembly and one edge between parts, which have at least one contact between them. For generating feasible assembly sequences, he used a questioning technique. The designer is asked questions in pairs [4].

- *Is it true that L_i cannot be established if L_j, L_k ... have been established?*
- *Is it true that L_i cannot be established if L_j, L_k ... are still not established?*

L_i, L_j, L_k are connections (liasion in French) in the connection graph. From the answers of these questions, the precedence relations between connections are extracted and a complete representation of the assembly can be constructed for planning phase. De Fazio and Whitney developed a different questioning technique, which decreases the total number of questions needed [5, 6].

- *Which liasions must be done prior to doing liasion i?*
- *Which liasions must be left to be done prior to doing liasion i?*

A pair of questions for each connection is enough for generating feasible assembly sequences; however, it is more difficult to answer this pair of questions.

One of the other approaches to model assemblies is to build a relational graph of contacts and parts and store mechanical and geometrical information in contact entities. A graph of contacts is given in Figure 2.1 (b), for the assembly shown in Figure 2.1 (a) [1].

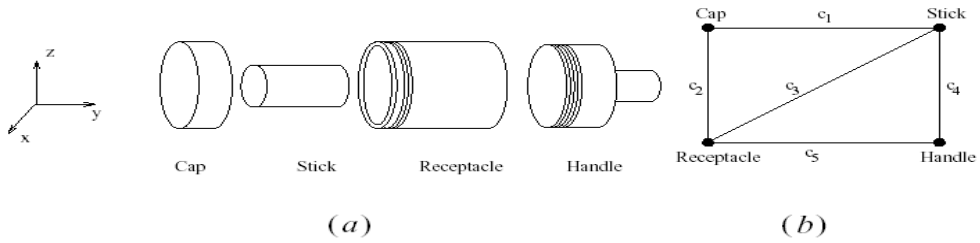

Figure 2.3 Exploded view of assembly (a), and graph of connections (b)

Mechanical CAD design systems are used to extract geometric and mechanical relationships of the assembly. Mello and Sanderson use geometric algorithms for the generation of feasible assembly sequences [7]. Expert systems have also been developed to capture feasible assembly sequences from the relational graph representation [1].

Different representation schemes were proposed for assembly sequences. Directed graphs are directed state-space graphs where nodes are the set of possible parts and subassemblies so far formed, and edges are assembly operations. States in the graph are represented by subassemblies already formed, such as {A, B}, or by the set of connections already established, such as vector {FFFTF} for 5 contacts. All paths from the start to end represent a feasible assembly sequence. Directed graph

representation of feasible assembly sequences for the assembly shown in Figure 2.1 is given in Figure 2.2 [1].

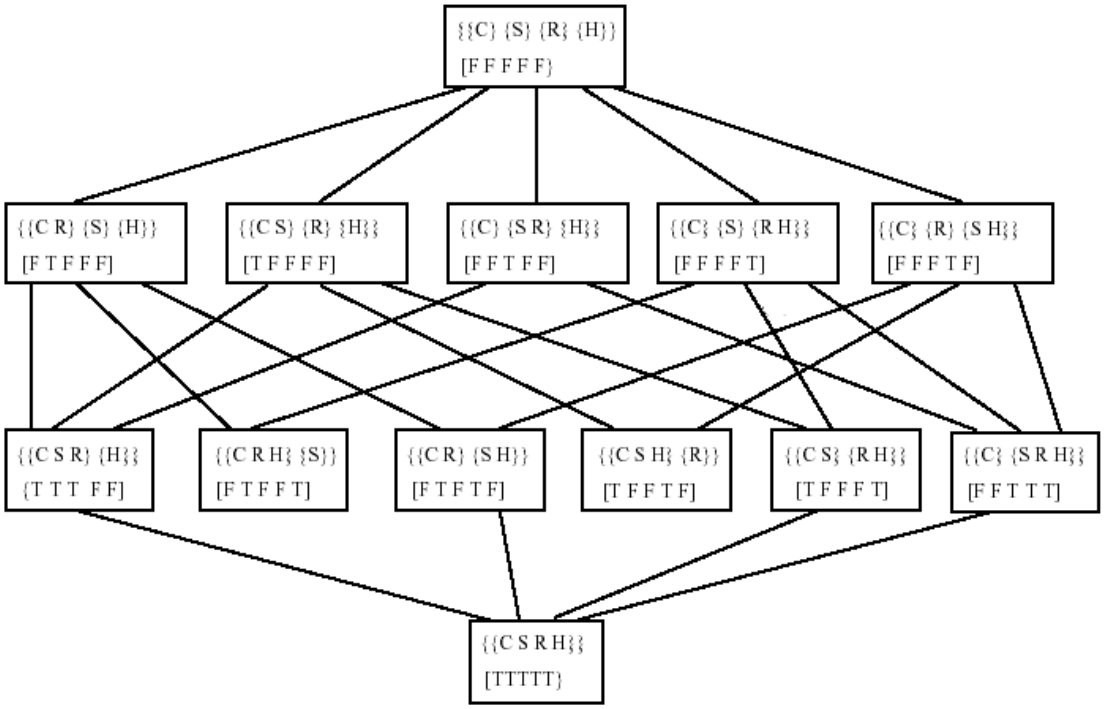

Figure 2.4 Directed graph of feasible assembly sequences for the assembly given in Figure 2.3

Another representation method is an assembly tree, which is a rooted tree where the final assembly is the root, nodes represent subassemblies and leaves are the raw parts. An assembly tree represents just one feasible assembly sequence. Assembly tree representation is shown in Figure 2.3 [1].

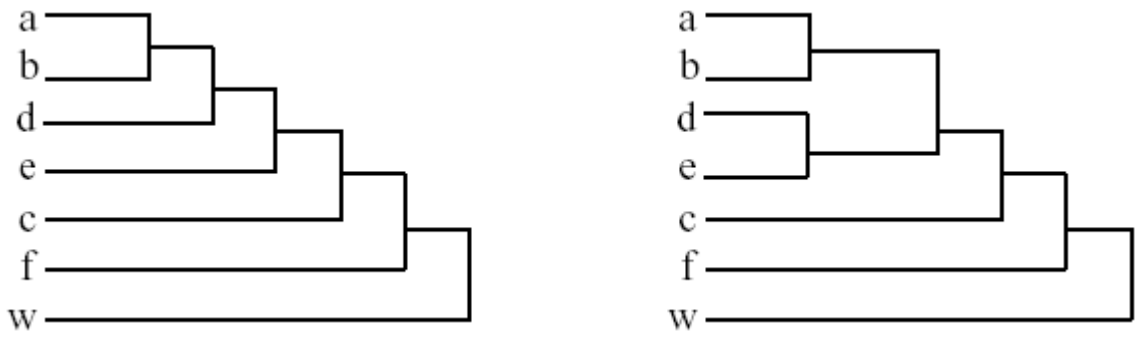

Figure 2.5 Two assembly trees for an assembly with seven parts

In AND/OR graphs, nodes represent subassemblies already formed and there are 2-connector hyper arcs, corresponding to assembly operations. In contrast to directed graphs, AND/OR graphs contain each assembly operation only once, however both of them represent each assembly state just once. AND/OR graph for the sample assembly given in Figure 2.1 is depicted in Figure 2.4 [1].

These methods deal with the generation of all feasible assembly sequences. Evaluation and selection of a best sequence is another issue. Several evaluation criteria are proposed. These are tooling, and fixture cost, subassemblies reorientation time, tool change time and opportunity to group operations on a single workstation [1]. These techniques rely on additional assembly-specific knowledge. Feasible sequences grow exponentially with every part in the assembly, and deciding upon optimal one is an NP problem. Evaluation and selection can be applied after all feasible sequences are generated or while they are being generated, avoiding the combinatorial explosion problem.

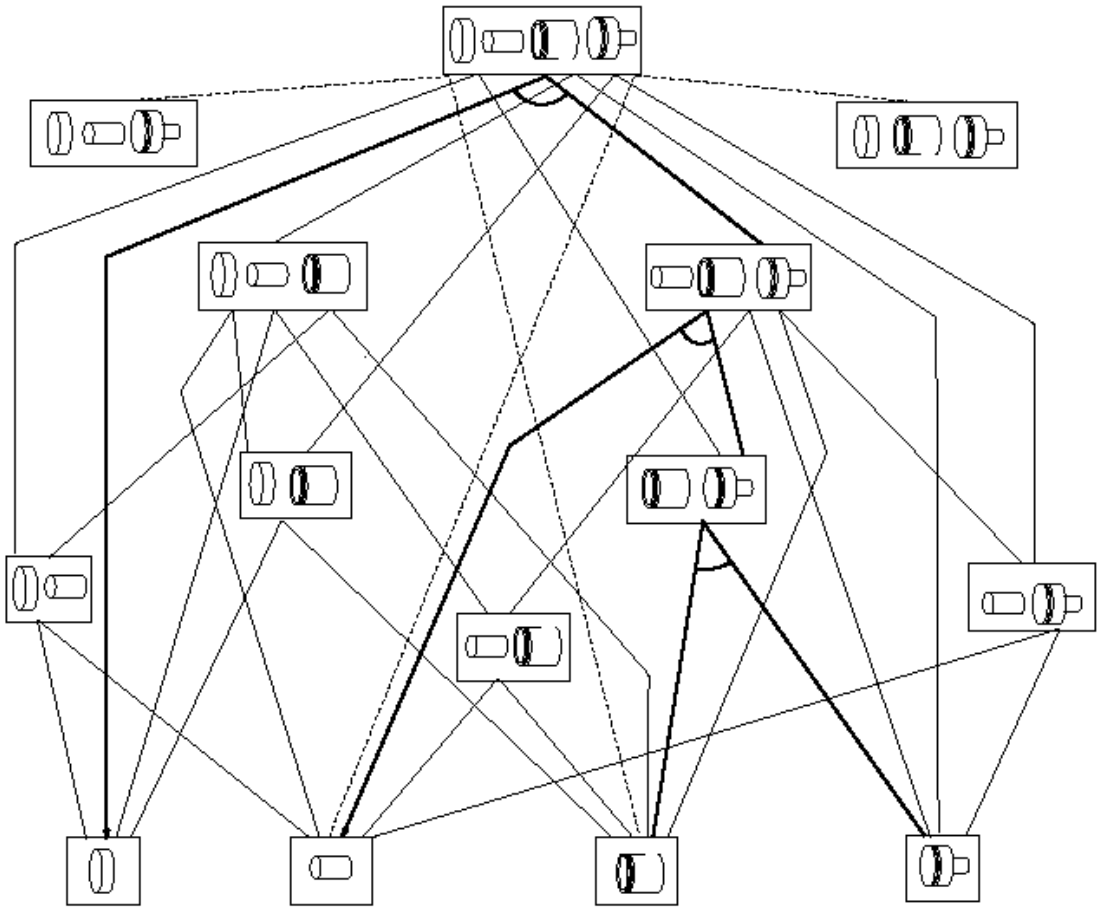

Figure 2.6 AND/OR graph of feasible assembly sequences for the assembly given in Figure 2.3

2.3 AI Planning

Planning is a state-space search, where states are snap-shots of the world and movement operators between the states are the actions. Planners represent possible configurations of the world as states. A state is a conjunction of positive literals. States include positive conditions that are relevant to the planner. Any other conditions that are not mentioned are believed to be false. Most AI planners work on environments that are observable, discrete, finite and static. Functions are not allowed in representation of the states. Goal is a specific state of the world. It contains the conditions that are to be achieved. Actions are formed of two parts, which are preconditions and effects. Actions can activate when preconditions match with a state of the world. Activation of an action changes the state of the world by the introduction of new positive or negative literals. In other words, an action affects the world by adding new literals to a state and deleting literals from a state. The sequence of actions, which transform the initial state to the goal state, is a plan. In our case, the goal is the realization of the final product and a sequence of fundamental motions to achieve this goal will be the plan [8].

State-space search is a straightforward approach for deriving plans. Both forward and backward state-space search can be used due to the two-sided structure of actions. Forward state-space search, which is also called progression planning, starts with the initial state and generates successors of the states by applying applicable actions on them. The search continues until the goal state is generated. In the absence of functions, any graph search algorithm that is complete may be used in the search. Forward search algorithms are inefficient since the state space of most planning problems is huge and there are many misleading actions. Finding a good heuristic to guide the search is very important. In backward state-space search, actions' effects are matched to at least one of the conjuncts of the goal or a sub-goal. Predecessors are generated by adding the action's preconditions to the state and by applying effects in reverse [8].

Both forward and backward chaining planners have certain limitations when real-world applications are to be modeled. Major requirements that cannot be satisfied with these conventional methods are concurrent actions, actions, which stimulate or inhibit effects of other actions, durative actions and planning with resources. Various extensions to ordinary planning representations have been proposed to accomplish these capabilities.

Forward chaining planners suffer from the exponential explosion of the search space. Search space branches with every possible match of actions and states. If actions contain variables, number of the successor states increase with the number of variable substitutions. To decrease CPU usage and total amount of time spent to reach a solution, good heuristics must be used. Other planning techniques have been devised to outperform forward chaining, such as partial-order planning and planning graphs. One way of narrowing the search space is to use domain dependent knowledge to guide the search and avoid visiting misleading states. This way some possible successor states are doomed before they are explored. Planning is fastened by reducing the number of branches at every depth.

2.4 TLPLAN

TLPLAN is a forward chaining planner with a search control mechanism, which uses domain specific knowledge [9]. It is developed by Fahiem Bacchus and Froduald Kabanza.

Standard forward chaining formulation does not support concurrent actions, durative actions and resources. In [10], Bacchus and Ady present an approach to facilitate these features in TLPLAN. Addition list of the actions are separated into two, instant effects and delayed effects. First group of effects are realized instantly when the action is applied. Delayed effects are realized in a world that will occur when the

duration of the action is over. Durations of the actions are specified by the domain designer and may be varied with the arguments of the action or global variables. Instant and delayed effects mark the start and finish of an action. Every state a time stamp, starting with a fixed start time in the initial state. Time is discrete, but intermediate points in time never occur. Delayed effects are put in an event queue with associated time stamps. When there are no applicable actions to the current state, the planner calls wait-for-next-event action, which pops the action from the event queue with the earliest time of occurrence. Effects change the state and time advances as much as the duration of the action. Resources can also be modeled with the use of delayed effect concept. Instant effects insert a predicate defining the occupancy of the resource, and delayed effects free the resource. This restricts the usage of the same resource unless the time advances forward the duration of the action. Resources with two or more, or nonlinear capacities can also be modeled by using functions in descriptions of the durative actions.

Approach of TLPLAN to search control is to use modal checking of temporal formulas, relating to the states in the timeline starting from the current state [11]. Temporal logic enables the planner to reason not only on the current state and candidate actions but also the following states and sequences of actions. The idea of temporal logic is that a formula is not statically true or false in a model, as it is in propositional and predicate logic. Instead, the models of temporal logic contain several states and a formula can be true in some states and false in others. Contrary to static notion of truth, truth values of the formulas may change as the system changes states. The view of time varies in different applications of temporal logic. Time can be linear or branching, as well as being continuous or discrete [12]. In first-order linear temporal logic, temporal formulas are obtained by adding temporal modalities to first-order formulas.

CHAPTER 3

REPRESENTATION OF WORLD MODEL

This chapter explains our formalization of the motion economy domain in detail. Knowledge representation for motion economy domain can be grouped into four subdivisions. These are representation of the product, representation of the workspace, representation of the current world and representation of actions. Actions, workspace and current world are represented for the first time for planning in motion economy domain. Representation of the products is adapted from assembly planning studies to define the tasks for manual operations. All the predicates used in the world model are summarized in Table A.1, given in Appendix A.

Hands are agents that perform the actions. Two hands are defined for left and right hands. Each hand has a fatigue attribute to keep track of the number of actions performed by them in a plan.

3.1 Representation of the Product

An extension of a relational graph, which is described in Chapter 2, is used for representing products. Relational graphs contain connection features in addition to graph of connections. Atomic parts of the assembly make up the nodes in the connection graph and there is an edge for every contacting part. These edges are referred as contacts in this thesis.

Atomic **parts** are the materials that form the final product. No further decomposition of these parts will take place anywhere in the plan. Parts have an attribute, which defines their location in the workspace. Same part may take place more than once in a product. When this is the case, parts are defined as separate entities but they are

assigned to same *type*, such as bolts. *Type* information is used within motion economy knowledge for the arrangement of the workplace.

Product specification may also contain **tools**. Some contacts can only be established with the use of tools. Tools have a location attribute, like parts, related to carrying and storing. Tools must be used in the same location where the contact is to be established.

In the contact graph, nodes represent parts and there is an arc for every contact between parts. Attributes of the **contacts** are given in the Table 3.1.

Table 3.1 Attributes of contacts

Name	**Description**
Part	Part, which has the contact. A contact has one or two parts.
Established	A Boolean value specifying whether the contact is established in the current state of the plan.
Cbei	Set of contacts, which makes it infeasible to establish the contact if established. Stands for "can not be established if".
Cbein	Set of contacts, which makes it infeasible to establish the contact if **not** established. Stands for "can not be established if not".
Tool	Tool, wherever a tool is required to establish the contact.

Contacts implicitly contain the feasible assembly sequences. A kind of precedence relationship is represented by using two kinds of contact set for each contact. In fact, these sets represent two types of constraints. One set is formed of contacts which make it infeasible to establish the contact if they are established. The other set contains the contacts which make it infeasible to establish the contact if they are not

established. Contact can only be established if no contacts in the first set are established and all contacts in the second set are established. These constraints were first introduced by Bourjault; he also developed a method to obtain the constraints in which the designer is interrogated with two types of questions for two constraint types. Here we assume the constraints are known prior to planning, and given with the product description. Formalization of precedence rules is borrowed from immediate dominance/linear precedence (ID/LP) formalism [13].

Contact graph representation is devised for generating and evaluating assembly sequences. However, in motion economy domain, we aim to cover all sorts of manual operations. Thus, we extended contact graph representation as follows.

- If a tool is listed in the attributes of a contact, the operator must use it while establishing the contact between the parts. If more than one tool is listed, then the tools must be used simultaneously while realizing the connection. If no tool is listed then the parts are to be assembled only with the use of the hands.
- We use arcs from and to the same part as another extension to the relational graph. These arcs represent a transformation concerning a single part. We call these arcs self-contacts. A self-contact represents irreversible transformations of the part like chemical transformations such as painting and drilling. This extension stems from the fact that we plan not only for assembly sequences, but for the manual operations to perform them. Having transformations of atomic parts or subassemblies in our model expands the type of operations we are capable to model and plan. The conventional problem of generating feasible assembly sequences does not address planning of setup operations like these. These operations may be included in a relational model of a product as setup costs. However, if evaluation of assembly sequences does not account for grouping of tooling, fixtures or workstations, these costs will be incurred in same amount in every plan independent of the sequence of connection establishments. Since our major concern for better plans is the

ease of the hands, planning for the sequence of such non-connecting operations has the same degree of importance with assembly operations. These self-contacts have a tool listed in the attributes; otherwise, the contact must represent an assembly operation of two parts. As these are not in the scope of assembly planning, self-contacts were not used before. These operations are planned in the same manner with assembly operations, which require the use of tools, such as screwing a bolt.

Predicates, which are used to represent the product in the domain, are given in the Table 3.2. Prefix notation is used when predicates and functions in the model are described.

3.2 Representation of the Workspace

All activities of the hands take place, and all the materials used in the plan are stored on a pre-defined workspace. Representation of the workspace is partly taken from motion economy studies. There are normal and maximum work areas for the left hand and for the right hand [3].

> *Normal working area is determined by an arc drawn with a sweep of the hand across the table. The forearm is only extended, and the upper arm hangs at the side of the body in a natural position until it tends to swing away as the hand moves toward the outer part of the work place. Maximum working area for a hand is determined by an arc drawn with a sweep of the right hand across the table, with the arm pivoted at the shoulder* [3].

Table 3.2 Predicates used in the representation of the current world.

Predicate	**Description**
(contact C)	C is a contact.
(part P)	P is an atomic part of the product.
(ctool C T)	Contact C requires tool T to be established.
(cbei C1 C2)	Contact C1 cannot be established if contact C2 is established.
(cbein C1 C2)	Contact C1 cannot be established if contact C2 is NOT established.
(cpart C P)	Part P has contact C.
(tool T)	T is a tool.

The overlapping area of the normal working areas for the right and the left hand constitutes a zone which two-handed work may be done most conveniently. Dimensions of these half circles for male and female have been measured for numerous process development studies in special industries.

Workspace representation, which we are using for this research, is an abstraction of these imaginary areas, defined above. We partition the workspace into nine discrete locations. One of them is called *working area*, where two-handed work can be done. Other locations are defined with three properties for each.

- First property is called *side*. It determines whether the location is at the left-hand or right-hand side of the worker.
- Second property determines the distance of the location to the working area. This property is related to which ring the location belongs. The outermost ring seen in the figure is called *maximum reach area*, and the ring between maximum reach area and the working area is called the *normal reach area*. This property is called *reach distance.* It may take two values, which are normal and maximum.

- Third property determines the adjacent axis to the location on the horizontal plane where the workspace lies. The *axis* property of a location is either vertical or horizontal.

Representation of the workspace is depicted in the Figure 3.1. Although this figure shows a partition of a perfect half circle, locations should be considered lying on fuzzy regions for the two hands. Borders of these regions are not exact. These fuzzy regions can be defuzzified in the applications. Figure 3.2 shows dimensions of normal and maximum work areas in the horizontal plane as developed and used by the Process Development Section of the General Motors Manufacturing Staff [3]. Locations are named with alphabetic codes of their properties. Descriptions of the eight locations are summarized in the Table 3.3.

Objects are stored in **containers** in the workplace. **Fixtures** are used to position objects. Setting the location of a container or a fixture in the workplace is a part of the problem, and it is decided as the plan is constructed, when a hand is to grasp a part from a container, or a hand is to position a part to use a tool on it. Containers and fixtures are always available. There is no limit on the number of containers and fixtures.

Locations of all entities, which are hands, tools, parts, containers, fixtures, are defined with the *at* predicate anywhere in the plan.

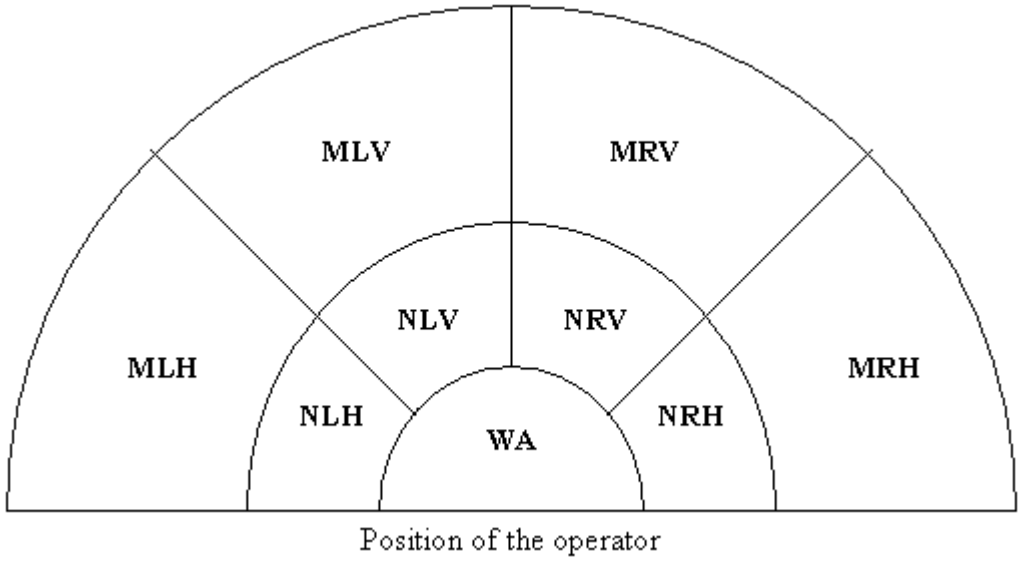

Figure 3.1 Representation of the workspace

Table 3.3 Properties of the locations

Location	**Property**		
	Reach distance	**Side**	**Axis**
NLH	Normal	Left	Horizontal
NRH	Normal	Right	Horizontal
NLV	Normal	Left	Vertical
NRV	Normal	Right	Vertical
MLH	Maximum	Left	Horizontal
MRH	Maximum	Right	Horizontal
MLV	Maximum	Left	Vertical
MRV	Maximum	Right	Vertical

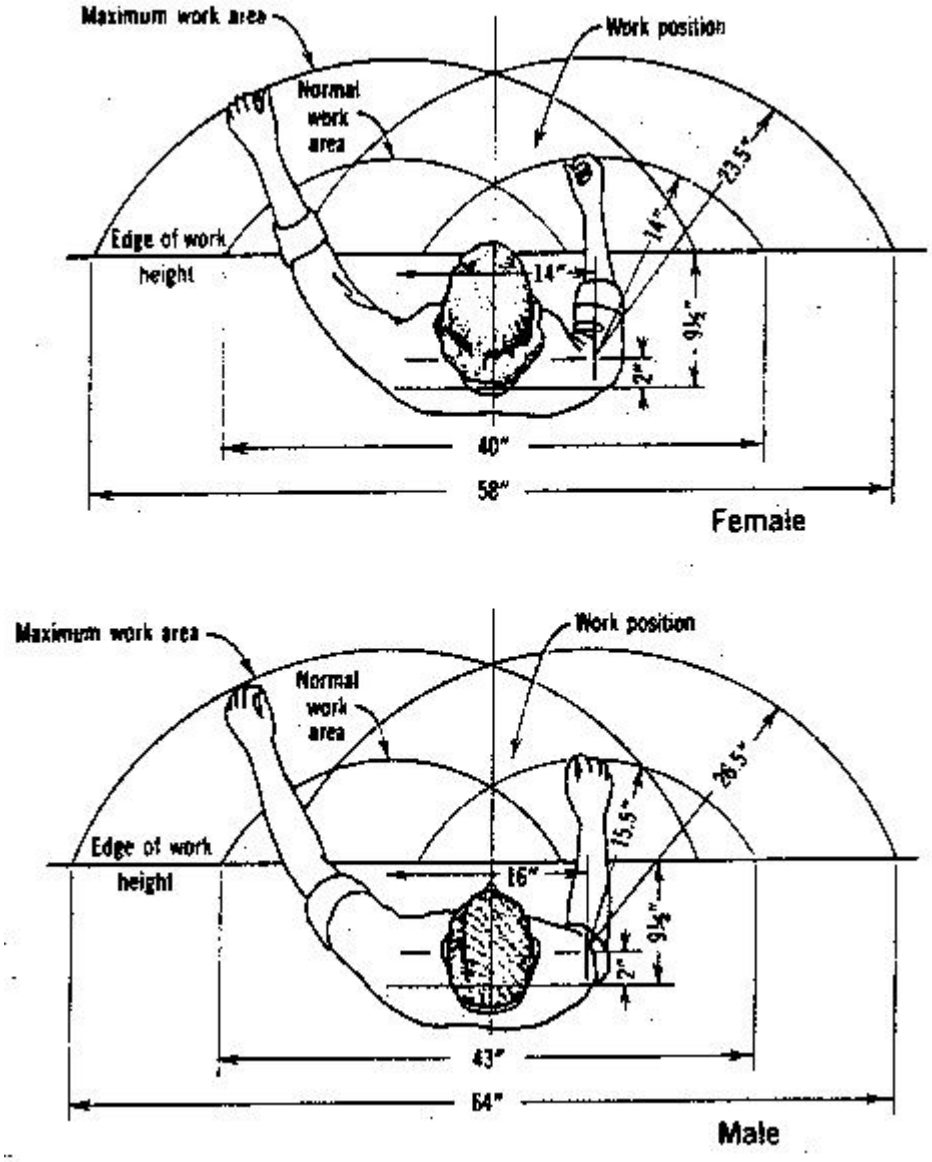

Figure 3.2 Dimensions of normal and maximum work areas in the horizontal plane as developed and used by the Process Development Section of the General Motors Manufacturing Staff

3.3 Representation of Current World

The representation of product and the workspace are given as inputs to the planner. They remain unchanged throughout the planning phase. Predicates describing the current world are manipulated by the actions. They change through states. World description of the initial and goal states defines the problem. Initial set of these predicates and the ones in the goal state depend on the choice of the problem owner. Manipulation of these predicates by the actions to reach the goal state generates the plan. List of the predicates, which are used in the representation of the current world, are given in the Table 3.4.

Table 3.4 Predicates, representing the current world

Predicate	**Description**
(pos X)	Part X is positioned.
(at X L)	Location of X (part, hand or tool) is L.
(holds H X)	Hand H holds part X or tool X.
(established C)	Contact C is established
(container P L)	There is a container of P at L
(fixture P L)	There is a fixture of P at L

3.4 Representation of the Actions

As introduced by Gilbreth, therbligs are fundamental subdivisions of motion. Any operation performed with the hands can be decomposed into sequences of *therbligs*. We construct our operation plans with a subset of the therbligs. *Therbligs* correspond to actions in our planning problem. The actions of our motion economy domain can be separated into two groups.

i. One group of actions is used to plan for establishing or disassembling contacts, these are *assemble*, *disassemble*, *use* and *disuse*. These actions will be referred as contact establishment actions.
ii. Other group of actions is used to plan for auxiliary operations, which satisfy the conditions for an *assemble* or *use* action. These actions are *transport, grasp, position* and *release*. Necessary conditions for an assembly or use operations range are simply the location of the parts, tools and hands. Usage of auxiliary actions varies. Hands may grasp a tool or a part. Transport actions change the location attributes of the hands and the objects from their old locations to new locations. Hand can release the object to the other hand or onto the workspace.

Planning in motion economy domain requires modeling of concurrent actions, durative actions and resources.

- Concurrency of actions is needed to generate plans where right and left hand simultaneously achieve different tasks.
- We have the need to model durative actions, for representing distribution of actions over time and to decide for the reorganization of the action sequences. In motion economy, fundamental motions are timed and placed on charts for proper analysis. Although in the method improvement task, specialists assume that these atomic motions have absolute durations for a trained worker and motions should not be shorter or longer, amount of time occupied by fundamental motions serves for rearranging them on the timeline with others. There is a need to decide if a hand can make two moves when the other is doing one job, and such. In our current model, all actions have unit durations since no study is conducted to measure durations based on the attributes of actions.
- Hands must be modeled as resources with unit capacity so as not to allow concurrent actions, which are done with the same hand.

In [10], Bacchus and Ady present an approach to facilitate these features in a forward chaining planner as described in Chapter 2. Their model is used to represent the actions of motion economy domain.

Busy predicates: The model for concurrency for the forward chaining planner first tries to match as many actions as possible with the current state before enhancing time. Hands must be defined as limited resources, to prevent making them perform more than one action at a time level. To define times when the hand is performing an action at a time level, *busy* predicate is used. *(busy H)* is added in the instant effects and deleted in the delayed effects of every action. Existence of this predicate in the world states that the hand is busy performing one of the actions. Every action check if the busy predicate does not exist in the world, otherwise the action can not instantiate for the hand.

transport_empty: *Transport_empty* changes the location attributes of the hands from their old locations to new locations. This action is used for transport when the hand is empty.

Action (transport_empty (h : hand, from : location, to : location)),
PRECOND: ¬(busy h) ∧ (empty h) ∧ ¬(= from to) ∧ (at h from)
INSTANT EFFECTS
DELETE LIST: (at h from)
ADD LIST: (will transport_empty h to) ∧ (busy h) ∧
(+= (tr-made h) 1) ∧
(+= (fatigue h) 1) ∧ (+= (rel-actions h) 1)
DELAYED EFFECTS
DELETE LIST: (will transport_empty h to) ∧ (busy h)
ADD LIST: (at h to) ∧ (init-action-counter h).

(tr-made h) is used to store number of transports that are made by hand h. *(rel-actions h)* counts the number of actions, which are done by hand h, while the other hand is idle. *(init-action-counter h)* is used to initialize *(rel-actions h)* when hands

operate simultaneously. These functions are used in search control and will be discussed in Chapter 5.

transport_loaded: When the hand is holding something, *transport_loaded* is used for transport. *Transport_loaded* changes the location attributes of the hands and the objects it holds from their old locations to new locations.

Action (transport_loaded (h : hand, from : location, to : location)),
PRECOND: ¬(busy h) ∧ ¬(empty h) ∧ ¬(= from to) ∧ (at h from)
INSTANT EFFECTS
DELETE LIST: (at h from) ∧ ∀[x:(holds h x)] (at x from)
ADD LIST: (will transport_loaded h to) ∧ (busy h) ∧
(+= (tr-made h) 1) ∧
(+= (fatigue h) 1) ∧ (+= (rel-actions h) 1)
DELAYED EFFECTS
DELETE LIST: (will transport_loaded h to) ∧ (busy h)
ADD LIST: (at h to) ∧ (init-action-counter h) ∧
∀[x:(holds h x)] (at x to)

select-grasp: We combine search, select and grasp therbligs into one action**,** since deciding the boundaries between these three therbligs is very hard, and most of the time, they follow each other in this order. A tool or a part can be grasped. *Select-grasp* is defined in three different ways and three different actions are written. These actions correspond to grasping a part, which is located in a container or fixture, grasping a tool, which is located in a container or fixture, and grasping a part or tool, which is located in the other hand. Locations of the grasping hand and the object must be the same. *Select-grasp* does not check for the presence of a tool or atomic parts at the location. Planner can choose any location to pick them up. Addition of containers to the world is an effect of this action. In other words by each grasping of a tool or an atomic part the initial location of them on the workspace is arranged. Nonatomic parts, assemblies, must be ready at the location of grasp for maintaining consistency with the preceding actions in the plan.

Action (select-grasp-part (h : hand, p : part)),

PRECOND: ¬(busy h) ∧ (empty h) ∧ ¬∃[x:(hand x)] (holds x p) ∧
(check-sim-grasp p) ∧ ¬∃[l:(at p l)] ¬(at h l) ∧
¬(isatomic p) ⇒ ∃[l:(isloc l)]((at h l) ∧ (at p l))
INSTANT EFFECTS
DELETE LIST:
ADD LIST: (will grasp h p) ∧ (busy h) ∧ (= (tr-made h) 0) ∧
(+= (fatigue h) 1) ∧ (+= (rel-actions h) 1)
DELAYED EFFECTS
DELETE LIST: (will grasp h p) ∧ (busy h)
ADD LIST: (init-action-counter h) ∧ (grasp-all-contact h p) ∧
∃[l:(at h l)]((isatomic p) ∧ ¬(at p l)) ⇒
((at p l) ∧ (¬(container (type p) l) ⇒ (container (type p) ?l))))

Action (select-grasp-tool (h : hand, t : tool)),
PRECOND: ¬(busy h) ∧ (empty h) ∧ ¬∃[x:(hand x)] (holds x t) ∧
¬∃[x:(hand x)] (will grasp x t) ∧ ¬∃[l:(at t l)]
INSTANT EFFECTS
DELETE LIST:
ADD LIST: (will grasp h t) ∧ (busy h) ∧ (= (tr-made h) 0) ∧
(+= (fatigue h) 1) ∧ (+= (rel-actions h) 1) ∧
∃[l:(at h l)] ¬(at t l) ⇒
((at t l) ∧ (¬(container (type t) l) ⇒ (container (type t) ?l)))
DELAYED EFFECTS
DELETE LIST: (will grasp h t) ∧ (busy h)
ADD LIST: (init-action-counter h) ∧ (holds h t)

Action (select-grasp_fh (h1 : hand, h2 : hand, o : object)),
PRECOND: ¬(busy h1) ∧ ¬(busy h1) ∧ (empty h1) ∧ (holds h2 o) ∧
∃[l:(isloc l)]((at h1 l) ∧ (at h2 l))
INSTANT EFFECTS
DELETE LIST:
ADD LIST: (will grasp h1 o) ∧ (busy h1) ∧
(= (tr-made h1) 0) ∧ (+= (fatigue h1) 1) ∧
(+= (rel-actions h1) 1) ∧ (will release h2 o) ∧
(busy h2) ∧ (= (tr-made h2) 0) ∧
(+= (fatigue h2) 1) ∧ (+= (rel-actions h2) 1)
DELAYED EFFECTS
DELETE LIST: (will grasp h1 o) ∧ (will release h2 o) ∧
(busy h1) ∧ (busy h2) ∧ (release-all-contact h2 o)
ADD LIST: (init-action-counter2) ∧ (grasp-all-contact h1 o) ∧

(check-sim grasp p) checks if part *p* is being grasped by another hand at the same time. If there is another concurrent action to grasp *p*, only instant effects of that action have changed the world. The part is not held by the hand and it still seems to be available for grasp. *Check-sim-grasp* must account for the case in which *p* belongs to an assembly. A recursive check is made for all parts which connect to each other by established contacts. *(grasp-all-contact h p)* adds *(holds h x)* predicate to the current world for all *x* that are connected to *p*. Similarly *(release-all-contact h p)* deletes *(holds h x)* predicate for all *x* that are connected to *p*.

release: A tool or a part can be released. There are two types of *release*. Hand can release the object to a container or onto the workspace, and hand can release the object to the other hand. The object is deleted from the holding list of the hand. If the object is transferred to another holder, it is added to its holding list.

Action (release (h : hand, o : object)),
 PRECOND: ¬(busy h) ∧ (holds x o)
 INSTANT EFFECTS
  DELETE LIST:
  ADD LIST: (will release h o) ∧ (busy h) ∧ (= (tr-made h) 0) ∧
    (+= (fatigue h) 1) ∧ (+= (rel-actions h) 1)
 DELAYED EFFECTS
  DELETE LIST: (will release h o) ∧ (busy h) ∧
                (release-all-contact h o)
  ADD LIST: (init-action-counter h)

Action (release_th (h1 : hand, h2 : hand, o : object)),
 PRECOND: ¬(busy h1) ∧ ¬(busy h1) ∧ (empty h2) ∧ (holds h1 o) ∧
          ∃[l:(isloc l)]((at h1 l) ∧ (at h2 l))
 INSTANT EFFECTS
  DELETE LIST:
  ADD LIST: (will release h1 o) ∧ (busy h1) ∧
                (= (tr-made h1) 0) ∧ (+= (fatigue h1) 1) ∧
                (+= (rel-actions h1) 1) ∧ (will grasp h2 o) ∧
                (busy h2) ∧ (= (tr-made h2) 0) ∧
    (+= (fatigue h2) 1) ∧ (+= (rel-actions h2) 1)
 DELAYED EFFECTS
  DELETE LIST: (will release h1 o) ∧ (will grasp h2 o) ∧

(busy h1) ∧ (busy h2)
ADD LIST: (init-action-counter2) ∧ (grasp-all-contact h2 o) ∧
(release-all-contact h1 o) ∧

assemble: *Assemble* realizes the establishment of a contact. If a tool is required to establish the contact, *assemble* cannot be used, other actions regarding the use of a tool must be used. Parts of the contact must be brought to the same location. The contact must be feasible for this operation to happen. Feasibility check is made by using the constraint sets of the contact, which were explained in the previous section. No contacts in the cbei set must be established and all the contacts in the cbein set must be established.

One of the hands is the operator hand. The other hand fixes one of the parts for the assembly operation. This hand holds one of the parts of the assembly. After *assemble*, the contact is established, parts form an assembly. Operator hand holds nothing. The assembly is held by the second hand. Assembled parts will behave as a unit after the contact is established. For instance, *transport_loaded* will change the location attributes of all the parts of the assembly they form. Assemblies are not represented as new entities in the world; atomic parts form a list and they are treated as an assembly.

Action (assemble (h1 : hand, h2 : hand, c : contact)),
PRECOND: ¬(busy h1) ∧ ¬(busy h1) ∧ ¬(= h1 h2) ∧
∃[l:(isloc l)]((at h1 l) ∧ (at h2 l)) ∧
¬∃[x:(ctool c x)] ∧ ¬(established c) ∧ (feasible c) ∧
(check-holding-contact c h1 h2)
INSTANT EFFECTS
DELETE LIST:
ADD LIST: (will assemble1 h1 c) ∧ (busy h1) ∧
(= (tr-made h1) 0) ∧ (will assemble2 h2 c) ∧
(busy h2) ∧ (= (tr-made h2) 0)
DELAYED EFFECTS
DELETE LIST: (will assemble1 h1 c) ∧
(will assemble2 h2 c) ∧
(busy h1) ∧ (busy h2) ∧

∀[x:(holds h1 x)] (holds h1 x)
ADD LIST: (init-action-counter2) ∧ (established c) ∧
∀[x:(holds h1 x)] (holds h2 x)

(check-holding-contact c h1 h2) checks if hands hold each of the parts that make up the contact.

use: *Use* realizes contacts for which a tool is required. If a tool is listed in the attributes of a contact, it can only be established with *use*. Tools can be used to transform a part or to assemble two parts together. As in *assemble*; the contact must be feasible before *use*. Contacts, which are listed in the *cbei* and *cbein* attributes of the contact, must be established and must not be established respectively. Two different actions are defined for *use* regarding the type of the contact being established. In all versions, there is one operator hand. Hands and the parts of the contact must be in the same location. The establishment of a contact with *use* has the same effects with *assemble* in the way parts behave as an assembly.

In the first case, *use-self-contact*, contact represents an operation on a single part and contact is from and onto the same part. There will be no change in the assembly configurations after the establishment of this contact, but the contact will become established. Operator hand holds the tool. Second hand positions the part for the operation.

In the second case, *use-with-fixture*, two parts form a new assembly and the tool must be used in this operation. Two parts are to be assembled with *use*. Operator hand holds the tool, second hand holds one of the parts and the other part has to be positioned on a fixture. This is controlled by the predicate *(pos p)*. Fixture is not included in the preconditions of the action. It is added to the world description, when this action is included in the plan.

Action (use_sc (h1 : hand, h2 : hand, c : contact, t : tool)),

PRECOND: ¬(busy h1) ∧ ¬(busy h1) ∧ ¬(= h1 h2) ∧ (ctool c t) ∧
!∃[p:(cpart c p)] ∧ (holds h1 t) ∧
∃[p:(cpart c p)] (holds h2 p) ∧
∃[l:(isloc l)]((at h1 l) ∧ (at h2 l)) ∧
¬(established c) ∧ (feasible c) ∧
INSTANT EFFECTS
DELETE LIST:
ADD LIST: (will use_sc1 h1 c) ∧ (busy h1) ∧
(= (tr-made h1) 0) ∧ (= (tr-made h2) 0) ∧
(will use_sc2 h2 c) ∧ (busy h2)
DELAYED EFFECTS
DELETE LIST: (will use_sc1 h1 c) ∧ (will use_sc2 h2 c) ∧
(busy h1) ∧ (busy h2)
ADD LIST: (init-action-counter2) ∧ (established c) ∧

Action (use_wf (h1 : hand, h2 : hand, c : contact, t : tool)),
PRECOND: ¬(busy h1) ∧ ¬(busy h1) ∧ ¬(= h1 h2) ∧ (ctool c t) ∧
¬!∃[p:(cpart c p)] ∧ (holds h1 t) ∧
∃[l:(isloc l)]((at h1 l) ∧ (at h2 l)) ∧
¬(established c) ∧ (feasible c) ∧
∃[x,y:(cpart c x) ∧ (cpart c y)] (¬(= x y) ∧
(holds h2 y) ∧ (pos x) ∧ ∃[l :(isloc l)] ((at h1 l) ∧ (at x l))))
INSTANT EFFECTS
DELETE LIST:
ADD LIST: (will use_wf1 h1 c) ∧ (busy h1) ∧
(= (tr-made h1) 0) ∧ (will use_wf2 h2 c) ∧
(busy h2) ∧ (= (tr-made h2) 0)
DELAYED EFFECTS
DELETE LIST: (will use_wf1 h1 c) ∧ (will use_wf2 h2 c) ∧
(busy h1) ∧ (busy h2) ∧
∀[x:(holds h2 x)] (holds h2 x) ∧
∃[x:(cpart c x),y:(cpart c y)] (¬(= x y) ⇒ (pos x))
ADD LIST: (init-action-counter2) ∧ (established c) ∧
∀[x:(holds h1 x)] (holds h2 x) ∧
∃[x:(cpart c x),y:(cpart c y)] (¬(= x y) ⇒ (holds h2 y))

position: *Position* must proceed every *use-with-fixture*. In *use-with-fixture*, one hand holds the tool and the other hand holds one of the parts. Other part of the assembly must be ready at the location and positioned on a fixture. *Position* is a kind of *release* in which a fixture and *(pos x)* predicate are added to the current world. Preconditions

and other effects of this action are similar to *release*. A tool can not be positioned. One additional precondition prevents positioning of parts, which do not require *use-with-fixture* action in the problem.

Action (position (h : hand, p : part)),
PRECOND: ¬(busy h) ∧ (holds x p) ∧ (part p) ∧
∃[c,t:(cpart c p) ∧ (ctool c t)] (¬∃![p:(cpart c p)] ∧
(feasible c) ∧ ¬(established c))
INSTANT EFFECTS
DELETE LIST:
ADD LIST: (will position h p) ∧ (busy h) ∧ (= (tr-made h) 0) ∧
(+= (fatigue h) 1) ∧ (+= (rel-actions h) 1)
DELAYED EFFECTS
DELETE LIST: (will position h p) ∧ (busy h) ∧
(release-all-contact h p)
ADD LIST: (init-action-counter h)

In fact, parts and tools must be positioned before every contact establishment operation. P*osition* must proceed every *assemble*, *disassemble*, *use* and *disuse*. However, *position* is only used for *use-with-fixture*. For other operations, it is implied that objects are positioned before these actions to save time and effort in planning and analyzing plans. Representation of *position* before *assemble* will have no effect on the evaluation and selection of plans for its place in a plan will be determined according to the sequence of contact establishments, which is the real decision to make.

disassemble: Disassembling a part is a flaw in every operation plan, for parts, which are in the graph of connections. Assembly sequence planning systems do not account for the possibility of disassembling, and reassembling parts. However, we go beyond the domain of assembly sequence planning, and intend to plan for a superior set of operations than assembly sequence planning. *Disassemble* may be used when the operation has to start from a point, where some or all of the parts are joined, and different formation of their assembly is wanted. Think of a kid disassembling his Lego toy to make atomic parts for another toy he wants to build. Another possible

use of *disassemble* is when a part has to be assembled during the course of the operation but it has to be removed at some final step. This kind of apparatus can usually be modeled as a fixture or tool, but when one or more assemble operations has to be made before this part is removed, it is more convenient to add it to the graph of connections, and distract its establishment from the goal. Precedence relations may be used to inform the planner between which establishments of contacts this apparatus must be effective.

Same feasibility check is made for the disassemble operation as with the assemble operation. *Disassemble* has the reverse effects of *assemble*. *Disassemble* makes the established property of a contact false. As the parts in an assembly act together checking location conditions for one is enough. Prior to the operation, one hand holds the assembly. The operator must hold nothing prior to the operation. A contact between two parts is broken with *disassemble* and hands hold each part and other parts that are connected to them. Which part will left or right hand hold after the action is not distinguished by the planner.

Action (disassemble (h1 : hand, h2 : hand, c : contact)),
PRECOND: ¬(busy h1) ∧ ¬(busy h1) ∧ ¬(= h1 h2) ∧ (empty h2)
∃[p:(cpart c p)] (holds h1 p)
∃[l:(isloc l)]((at h1 l) ∧ (at h2 l)) ∧
¬∃[x:(ctool c x)] ∧ (established c) ∧ (feasible c)
INSTANT EFFECTS
DELETE LIST: (established c)
ADD LIST: (will disassemble1 h1 c) ∧ (busy h1) ∧
(= (tr-made h1) 0) ∧ (will disassemble2 h2 c) ∧
(busy h2) ∧ (= (tr-made h2) 0)
DELAYED EFFECTS
DELETE LIST: (will disassemble1 h c) ∧
(will disassemble2 h c) ∧
(busy h1) ∧ (busy h2) ∧
∃[x:(cpart c x), y:(cpart c y)]
(¬(= x y) ⇒ (release-all-contact h1 y))
ADD LIST: (init-action-counter2) ∧
∃[x:(cpart c x), y:(cpart c y)]
(¬(= x y) ⇒ (grasp-all-contact h2 y))

disuse: *Disuse* is not one of the therbligs listed by Gilbreth. A new action is added to the domain to model *disassemble* operations that require a tool. Motion economy has used *use* therblig when a new subassembly was formed or one is deformed. *Disuse* is the reverse operation of *use* as *disassemble* is the reverse operation for *assemble*. It has reverse effects of *use*. On contrary to *use*, only one version of *disuse* is defined. In first version of *use*, a self-contact is realized for a single part. These operations are regarded as irreversible transformations such as drilling and painting. Their effects cannot be taken back with the use of same tool. *Disuse* is defined for *use-with-fixture* version of *use*. Dismantling a screw from an assembly with a screwdriver is an example of *disuse*. Name of this action may be confusing for the tool is used in a normal way just as in the case of *use*. This action is called disuse for its similarity with *disassemble*, not for awkward utilization of the tool. Same feasibility check is made with *disassemble*. There is one operator hand, which uses the tool. Other hand holds the part, which is being taken apart. After *disuse* is completed, secondary hand holds the disassembled part, or the assembly that the part belongs to, operator hand holds the tool and other dismantled subassembly is left on the workspace.

Action (disuse_wf (h1 : hand, h2 : hand, c : contact, t : tool)),
 PRECOND: ¬(busy h1) ∧ ¬(busy h1) ∧ ¬(= h1 h2) ∧ (ctool c t) ∧
  ¬!∃[p:(cpart c p)] ∧ (holds h1 t) ∧
  ∃[l:(isloc l)]((at h1 l) ∧ (at h2 l)) ∧
  (established c) ∧ (feasible c) ∧
  ∃[x:(cpart c x)] (holds h2 x)
 INSTANT EFFECTS
  DELETE LIST: (established c)
  ADD LIST: (will disuse_wf1 h1 c) ∧ (busy h1) ∧
  (= (tr-made h1) 0) ∧ (will disuse_wf2 h2 c) ∧
  (busy h2) ∧ (= (tr-made h2) 0)
 DELAYED EFFECTS
  DELETE LIST: (will disuse_wf1 h1 c) ∧
  (will disuse_wf2 h2 c) ∧
  (busy h1) ∧ (busy h2) ∧
  ∃[x:(cpart c x),y:(cpart c y)]
  (¬(= x y) ⇒ (release-all-contact h2 y)

ADD LIST: (init-action-counter2)

These actions, which are described so far, constitute all the action to the planners' domain. Other therbligs were discarded, since they are considered improper for representation or analysis of motion economy principles.

- *Hold* therblig is not exclusively included in the domain, but *hold* take place in the plans when the hands are not empty and perform no action.
- *Pre-position therblig* is also excluded for similar reasons. Moreover savings in time when pre-positioning is difficult to measure in the scope of this study. Pre-positioning overlaps other actions and changes the way grasp and transport actions are made.
- *Avoidable delay* is a *therblig* that may be observed but will not be used in the design of a process. This *therblig* is meant to be used in developing right hand left hand charts by observing an operator performing a task. When the specialist notices a pause of the operator, that elapsed time is shown with an avoidable delay in the chart. As we plan motions for an imaginary, perfect operator, there will be no flaws or pauses for which the operator is responsible. Therefore, *avoidable delay* is not included in the representation of the motion economy domain.
- *Unavoidable delay therblig* is meant to be used to fill vacancies of a hand, which stems from the characteristics of the process. *Unavoidable delay* is implied in final plans, but it is not represented as an action, since it is never intentionally used to complete a task.
- *Inspect* and *plan therbligs* rather represent unobservable mental processes. These are not performed by the domains' operators, which are hands, but only may be shown as pauses in the plans. In a trained process, they should not be observed. It is assumed that the operator will not have the need to inspect or plan anywhere in the plan. These *therbligs* are not used in the plans.
- *Rest for overcoming fatigue therblig* is not also included in the current model.

Handling assemblies: Parts, which have no established contacts, are defined as atomic parts. As contacts are established, parts are connected together and form assemblies. These assemblies are subjected to complete set of actions in the domain. They are transported, grasped, released and form other assemblies, with other parts or assemblies, until the goal state is reached. When two parts form an assembly or a part joins an assembly, current world has the slightest change. Regarding formation of the new assembly, only the establishment of the contact is represented by adding a predicate to the world. No entity representing the assembly is added to the world. The behavior of the group of parts is controlled by continuously checking the established contacts in the whole planning phase. For preventing a fault in the plan, regarding the common behavior of assembled parts, following conditions must be satisfied.

i. All of the parts of the assembly must be at the same location when they are to be grasped. If a part is atomic, no location condition is checked, it can be grasped from anywhere. This means determining the location of the container of the part. Grasping an atomic part more than once is permitted. If a part is grasped before and released somewhere, *select-grasp* can not be applied in a different location. Since each usage of every part, even if their types are same, is represented as a unique entity in the contact graph, grasping a part once brings no drawbacks. Grasping a part more than once, although it will be used just once, is never desired and never will take place in final plans.

$$\neg\exists[l{:}(at\ p\ l)]$$

If the part is not atomic, it is satisfactory if the location of one of the parts is equal to the location of the hand.

$$\neg(isatomic\ p) \Rightarrow \exists[l{:}(isloc\ l)]((at\ h\ l) \wedge (at\ p\ l))$$

Other parts of the assembly are not tested in the preconditions of *select-grasp*. Other regulations, listed above, assure all of the parts to be in the same location any time. Hence, if one of the parts and the hand are at the same location, assembly can be grasped.

ii. All of the parts of the assembly must be held in case of a *select-grasp*. Different actions are not defined for grasping atomic parts and grasping assemblies. In both cases, a part and a hand are the only arguments of this action. *Select-grasp* can be instantiated with any part of an assembly, all having the same consequences. *(holds h p)* predicates are added to the world for other parts, which have connections to the part being grasped. Established contacts of the other parts are also checked until whole assembly is held by the hand. Although *select-grasp* is instantiated for one of the parts of the assembly, effects of the action applies for all of the parts of the assembly.

(add (holds h p)) $\wedge$ $\forall$[c,x:(cpart c p),(cpart c x)]
((established c) $\Rightarrow$ (grasp-all-contact h x)) $\rightarrow$ (grasp-all-contact h p)

iii. Locations of all of the parts of the assembly must change when the assembly is transported. Tools, parts or assemblies are not listed in the arguments of *transport_loaded*. *Transport_loaded* changes the location of the hand and whatever it is holding. There is no need for testing whether parts have established contacts to maintain assemblies. Location attributes are updated for every object that hand holds.

$\forall$[x:(holds h x)] (del (at x from)) $\wedge$ $\forall$[x:(holds h x)] (add (at x to))

iv. None of the parts of the assembly must be held after *release*. As in *select-grasp*, *release* is instantiated for a single part and affects the parts that are connected to it.

Items explained above guarantee that the hand holds all of the assembly any time, so *release* have no preconditions regarding assemblies.

(del (holds h p)) $\wedge$ $\forall$[c,x:(cpart c p),(cpart c x)]
((established c) $\Rightarrow$ (release-all-contact h x)) $\rightarrow$ (release-all-contact h p)

v. All of the parts of the assembly must be at the same location when a new assembly is to be formed. All of the parts of the assembly must be held when a new assembly is to be formed. Argument of the contact establishment operations is the contact. Contacts have defined with two parts. If one hand holds one part of the contact and the other holds the other part, contacts can be established. This is the test that is included in the preconditions of these actions. If parts are not atomic, it is guaranteed that hands hold other parts of the part's assembly with the use of regulations described above.

(check-holding-contact c h1 h2) $\rightarrow$ $\exists$[x,y:(cpart c x),(cpart c y)]
$\neg$(= x y) $\wedge$ (holds h1 x) $\wedge$ (holds h2 y)

vi. All of the parts of the assembly must be held after a new assembly is formed. Depending on the selection of the operator hand and the secondary hand, parts transform from one hand to the other in assemble action. *Use-self-contact* does not affect what hands are holding and *use-with-fixture* ends with all of the parts being on the fixture. Assemblies can be formed from atomic parts or other assemblies. With the regulations explained above, these actions only shift the holder of all of the parts involved. Contacts of the parts are not tested if they are established.

$\forall$[x:(holds h1 x)] (del (holds h1 x)) $\wedge$ $\forall$[x:(holds h1 x)] (add (holds h2 x))

Cycle-check predicates: Exploration of identical states, which were once already visited during the search, causes cycles in the search. Cycle-checking feature of the planner is used to avoid this problem. Built-in feature of the planner used, TLPLAN,

prevents cycles by checking effects of an action to see if it produces a known state, before applying it to the current state. However, the planner looks only for instant effects of the actions. It does not consider the delayed effects. Although the action will generate a new state with the addition and deletion of delayed effects, action will not be included in the plan if instant effects does not change current state or generate an already visited state. This is because every match of a concurrent action is interpreted as a state change, as the change caused by wait-for-next-event.

One good example is the *transport_empty* therblig. This action deletes the location of the hand in instant effects and adds the new location in the delayed effects. When the hand is in the working area *transport_empty* has eight possible instantiations each for other eight locations. These eight instantiations differ in addition of the new location of the hand to the world, which is written in the delayed effects. When they match current state, they produce same state when instant effects are realized and delayed effects are put into the event queue. Cycle checking permits usage of one of them, and prunes states produced by other seven, as a repetition of states is discovered. Even though these actions instantiate by different values for arguments, cycle checking decides only by looking to the immediate next state. This makes it impossible to explore all possible ways of transportation from a location.

will predicate is introduced to overcome this problem. This predicate includes name and arguments of the action. It may be interpreted as “hand will perform this action with these parameters”. It is added to the world as an instant effect and deleted as a delayed effect. It has a validity period between the start and end of an action. For different instantiations of an action, this predicate produces a difference in the immediate next state even if these instantiations have the same instant effects.

Transport_empty for eight different locations will immediately add the predicate *(will transport_empty h from to)*. Cycle checking will not exclude any of the actions since a different next state is produced for each of them. In contact establishment

actions, two hands have not identical roles. Consequences of actions change with the selection of the operator and secondary hand. Two *will* predicates, one for each hand, are used. Actions are numbered in these predicates to imply the role of the hands. Otherwise cycle-checking does not distinguish instantiations of actions where the choice of the operator hand is different.

We introduced these predicates to work with cycle checking. Later they were found useful for aiding search control. Their major advantage is to make pruning possible without waiting the delayed effects of an action. This also shortens the range where the temporal formula will be active and makes it easy to formulize search control rules.

Representation of the world model is given in this chapter. The model is given in four sections; representation of the product, representation of the workspace, representation of the current world and representation of actions. Next chapter discusses the way the planner generates plans for manual operations.

CHAPTER 4

GENERATING PLANS FOR MANUAL OPERATIONS

Forward chaining planning approach is used for generating plans. The initial state is composed of a specification of parts, tools, contacts and hands. Contrary to the general assembly sequence planning problems, initial state may contain established contacts. Hands are located in the working area at the start of every plan. Tools and parts are defined in the product representation but they do not have initial locations. Locations of tools and parts are decided in plan generation. This is a part of the solution of the problems. They can be grasped from any location on the workplace. Every time they are grasped, a container is added to that location, marking the initial location of the parts and representing the arrangement of the workspace. Fixtures are added to the workspace in the same manner as they are required. Hence, a plan additionally proposes a design of the workspace with the locations of containers and fixtures.

Goal state is composed of the final set of contacts, which must be established, and optionally the final arrangement of the workplace. The operator may be forced to place the final product in a specific location. One extreme case is to plan only for the arrangement of the workplace, such as to tidy up a table. The outcome of the plan is the sequence of actions that make the goal state true.

Unfinished plans are sequence of actions, which are unsatisfactory to reach the goal. With the inclusion of actions to the plan, goal is partially realized, meaning that not all but some of the contacts are formed, or some of the final locations are not satisfied. The algorithm halts when the sequence of actions gives a complete operation plan to form the product.

Representation of goal state and intermediate product configurations is adapted from the state representation of directed graph of assembly sequences from assembly planning. Homem de Mello and Sanderson suggest that an assembly state can be represented either by the set of connections already established, or by a partition of the set of parts in the product corresponding to the subassemblies so far formed [7]. Establishment of a contact is represented with a Boolean value. For the whole contact set, a true-false vector represents contacts already established. An *assemble* or a *use* action make one of this values true at a time, while a *disassemble* action makes it false. Completing a product means having a certain subset of these values true (complete set in a conventional assembly sequencing problem). The goal state is represented by the vector where contacts, which have to be made, have truth-values. Other method, proposed by Homem de Mello and Sanderson suggests representing the goal state as the unison of all atomic parts forming the assembly, and is semantically equal to the approach that we use [1]. During the course of the search of the planner, adding actions will aim to make certain contacts true and will implicitly merge sets of subassemblies into one. For an assembly with five contacts and parts, A, B, C, D and E, Figure 4.1 shows how the configuration of the product change as the plan is constructed.

The reader must keep in mind that configuration of the product does not necessarily change in consecutive states of the search ($i \leq j \leq k$). As *assemble* and *use* actions alter the contacts' truth-values, other auxiliary actions, such as *transport* or *select-grasp*, may occur in between. In general, this representation does not correspond to a branch of the search space of the planner. Addition of any action to the plan changes the state of the plan; this means a change of the unfinished sequence of actions. However not every action, changing the world model, changes the product configuration. A change in the product configuration is achieved by more than one change in the states of the search space. Changes in product configuration correspond to the addition of more than one action to the current plan. Nevertheless, sequence of certain instantiations of actions implies a certain assembly sequence. This new way

of generating assembly sequences is a major difference from other assembly planning systems. Indirectly, planner's choices on the order of actions make decisions on the sequence of establishment of connections. Moreover an evaluation of a plan based on the sequence of actions, evaluates the assembly sequence. This will be explained in Chapter 5.

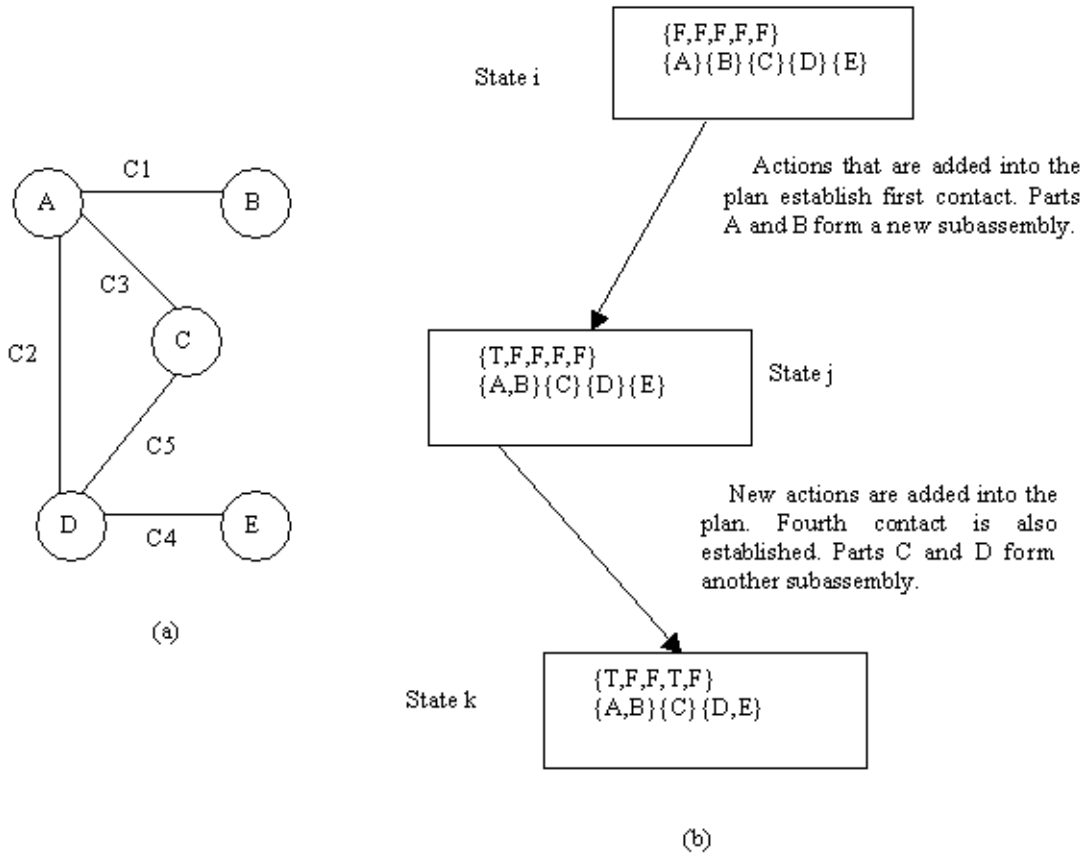

Figure 4.1 Graph of connections (a), Configurations of the product (b)

There is a difference between the representation of assembly states in assembly sequence planning and our product configuration representation. Establishment of each contact causes a new subassembly to form in assembly sequence planning. Therefore, whenever a value in the contact vector changes, that is becomes true or

false, subsets of the parts also change, two sets unite or a set is divided into two, as in Figure 4.1. However, when there are contacts from and onto the same part or subassembly in our product specification, establishment of these contacts will not make two sets of parts unite. These contacts stand for the use of tools on a single part, changing the nature of it, such as cleaning and painting.

We have seen how the search space is explored by the planner and how the configuration of the product and the current world change with state changes. Similarities and differences of the states between assembly planning studies and our study are also discussed in this chapter. Next chapter describes how plans are evaluated in our study.

CHAPTER 5

SEARCH CONTROL AND EVALUATION OF THE PLANS

Search is controlled by modal checking of temporal formulas, relating to the states in the timeline starting from the current state [11]. We use first-order linear temporal logic, in which temporal formulas are obtained by adding temporal modalities to first-order formulas.

Motion economy knowledge is used to determine the quality of a plan. A plan is better than another plan if it obeys the principles of motion economy more. One method to select good plans would be to evaluate plans with a plan checker after generating all possible plans. This method is computationally inefficient if it is not infeasible. Number of possible plans exponentially grows with every part, contact, location and tool. Another method would be modifying a plan, which is obtained by a conventional search method, by plan rewriting rules, replacing certain action sequences according to motion economy principles. The approach we use is to modify the plans while they are being generated. The use of search control is extended to evaluate and select plans.

Precedence relations developed by *"can not be established if"* and *"can not be established if not"* contact sets do not impose one and only sequence for establishing contacts. Sometimes there can be two or more feasible contacts. Choice of which contact will be established next, at each level, leads different plans which all complete the product. This difference of the sequence of contact establishments is reflected to the order of actions and conformity of the plan with motion economy. The reason why these contact sets are defined instead of running the planner with one order of contact establishment, is to keep this diversity of plans in the search

space and not to miss opportunity of improvement of plans through the selection of contact establishment sequence. In fact evaluation of selection of assembly sequences is one of the major research areas of assembly sequence planning. The evaluation of assembly sequences is done using transition matrices of costs or such useful information of contacts. Although motion economy domain has a dissimilar way of evaluation, the notion of good plans of motion economy has a correlation to the notion of a good assembly sequence.

Search control rules contain three sets of rules. First two sets of rules are used to prune misleading states and fasten the search. First rule set is called *common sense rules*; for these rules do not require the expertise of a motion economy specialist, an ordinary person will easily find them logical. Second set of rules exploit the information of the goal state to prune search space. This set is *called goal-related rules*. Third set of rules contains accumulated motion economy knowledge. These rules are called *motion economy rules*. The following to sections include the detailed explanation of search control rules that we have formulated and employed for this thesis.

5.1 Common sense rules

To prevent transporting an empty hand between the same locations is an example of rules from this group. A forward chaining planner using an admissible heuristic, where the heuristic function is the number of actions, will not probably generate plans having action sequences that are prevented by *common sense rules*. A plan with needless consecutive transports will be chosen over a plan, which has the same action sequence but the needless transports. *Common sense rules* are not useless anyway, for they eliminate exploration of these illogical plans. *Common sense rules* are devised in the course of this study and formulated as temporal formulas.

i. Do not make too many consecutive transports

(always (≤ (tr-made left) (cons-tr)) ∧ (≤ (tr-made right) (cons-tr)))

tr-made function counts the number of consecutive transports a hand makes. It is defined for each hand. Transport actions increase the value of *tr-made*, while other actions, such as *select-grasp* and *assemble*, initializes its value to 0. *Cons-tr* is a constant which is the number of consecutive transports permitted. This temporal formula states that hands may only make *cons-tr* number of transports, not being interrupted by other actions. *Cons-tr* value used in experimental runs is 1. 2 consecutive transports are not allowed. This rule prunes plans in which hands wander on the workspace, without making any progress on the accomplishment of goal, in terms of both product realization and workspace arrangement. Possibility of going back and forth between the same locations is also prevented with this rule.

ii. Do not grasp an object which has just been released (soon after releasing it)

((always ∀[h,p:(will release h p)]
(until ¬∃[h2,x:(will grasp h2 x)] (connected p x)
∃[c:(contact c)] (contact-est h c)) ∧
(always ∀[h,p:(will position h p)]
(until ¬∃[h2,x:(will grasp h2 x)] (connected p x)
∃[c:(contact c)] (contact-est h c)))

This rule forces hands to make one of the contact establishment actions before grasping the part, which they have just released or positioned. *(contact-est h c)* checks whether hand *h* performs an action, related to contact establishment or disestablishment. These actions are *assemble*, *disassemble*, *use* and *disuse*.

((will assemble1 h c) ∨ (will assemble2 h c) ∨
(will disassemble1 h c) ∨ (will disassemble2 h c) ∨
(will use_wf1 h c) ∨ (will use_wf2 h c) ∨
(will disuse_wf1 h c)) ∨ (will disuse_wf2 h c)) ∨

(will use_sc h c) → (contact-est h c)

If the part is not atomic and is a part of an assembly, grasping another part connected to it equals grasping the part. Grasping all parts that are connected to the part, which has been released, is checked by the rule. *(connected p x)* checks if part *x* and part *p* have established contacts. Hands may change locations, take hold of other parts, but may not grasp the same part in between contact establishment actions. This rule prunes plans in which a part is temporarily stored at a location after being grasped and taken back. This is a common sense rule as temporarily storing an object somewhere in the workspace is not related to any progress in accomplishment of any type of goal.

iii. Do not release an object which has just been grasped (soon after grasping it)

(always ∀[h,p:(will grasp h p)]
 (until ¬∃[x;(will release h x)] (connected p x)
  ∃[c:(contact c)] (contact-est h c))

This rule forces hands to make one of the contact establishment actions or transport actions before releasing the part, which they have just grasped. If the part is not atomic and is a part of an assembly, releasing another part connected to it equals releasing the part. Releasing all parts that are connected to the part, which has been grasped, is checked by the rule. Hands are forced to make a contact establishment action. This rule prunes plans in which a part is temporarily held at a certain location. This is a common sense rule since the world before grasping the part, and the world after releasing it have no difference. Nothing will be changed or achieved with this action pair.

iv. Do not consecutively establish and disestablish same contact

(always ∀[h,c:(hand h) ∧ (contact c)] (contact-est h c) ⇒
 (until (next (¬(contact-est h c)))

(next (∃[x:(contact x)] (¬(= x c) ∧ (contact-est h x))))))

This rule prohibits application of contact establishment actions on the same contact repeatedly, unless another contact is subject to one of these actions in between.

v. Do not transfer an object between the hands

(always ∀[h1,h2:(hand h1) ∧ (hand h2)]
¬(= h1 h2) ⇒ ¬∃[x:(will release h1 x)] (will grasp h2 x))

Transferring an object between the hands is not allowed. This means same part is being released by one of the hands while it is being grasped by the other. If the object is needed at one of the hands, it should be grasped by that hand in advance, rather than having other hand grasp it and transfer it to this hand.

5.2 Goal-related rules

Goal-related rules avoid exploring states, which will not lead to a progress for realizing the goal state. Planner will eventually backtrack from these states, although these are not pruned by *common sense rules*. We have developed *goal-related rules*, which can work with any kind of problem. Formulations of these rules are as follows.

i. Do not grasp a part whose contacts, which are in the goal, are all infeasible

(always ∀[p:(part p)] ∃[h:(hand h)] (will grasp h p)) ⇒
(∃[c:(contact c)] (cpart c p)) ⇒
(∃[c:(cpart c p)] (goal (established c)) ⇒ (feasible c))))

None of the contacts of a part, which are in the goal, are feasible, then do not grasp the part. Part has to be grasped when at least one of its contacts is feasible. That is when the contact can be established. This rule has no adverse effects on planning

decision, since if one of its contacts is yet to be established, grasping the part when its contacts are infeasible will eventually end up with releasing it, with no change in the world model. This rule always evaluates true for parts, which do not have any contacts. This rule does not avoid grasping assemblies. Contacts remain feasible soon after they are established, since meanwhile status of other contacts does not change. Thus, an assembly can still be grasped even if all contacts of all of its parts are established by grasping the parts, which are connected last.

ii. If an object has a feasible and disestablished contact which is in the goal, do not release the object until the contact is established

$$
\begin{aligned}
&(\text{always } \forall[\text{h,p,c:(holds h p)} \wedge \text{(cpart c p)}] \\
&\quad((\neg(\text{established c}) \wedge \text{(goal (established c))} \wedge \text{(feasible c)}) \vee \\
&\qquad((\text{established c}) \wedge \neg(\text{goal (established c)}) \wedge \text{(feasible c)})) \Rightarrow \\
&\qquad(\text{until } \neg\exists[\text{x:(will release h x)}]\ (\text{connected ?p ?x}) \\
&\qquad\qquad\exists[\text{c:(contact c)}]\ (\text{contact-est h c})))
\end{aligned}
$$

Releasing a part when its contact is in the goal and feasible will cost extra useless actions to the plan, for the part will be eventually grasped again to be assembled. If there are any other feasible contacts at the same time, and establishing one of them rather than this contact is in the favor of the plan because of another rule, the alternatives are still not pruned by this rule. The planner will then backtrack from the point where the part has not been grasped and a part of the other contact will be grasped. However, this rule shortens the length of this branch of the search before backtracking from the start of it.

iii. When a hand is holding a part and all contacts of the goal are established, do not release it until it is in the location specified in the goal state

$$
\begin{aligned}
&(\text{always } \forall[\text{p,l,h:(goal (at p l))} \wedge \text{(holds h p)}] \\
&\qquad\forall[\text{c:(contact c)}]\ ((\text{goal (established c)}) \wedge (\text{established c})) \Rightarrow \\
&\qquad(\text{until (holds h p) (at p l)})
\end{aligned}
$$

This rule matches a state where all of the required contact establishment actions are finished and a hand is holding the assembly. In the goal state a final location of the assembly is specified. Any other operation regarding to this assembly than taking it to the specified location will result in needless exploration of the search space. This rule prunes all states, which will not cause the part to be in that location.

iv. Do not establish a contact when it does not need to be established

(always
∀[c:(contact c)] (∃[h:(hand h)] (will assemble1 h c) ∨
∃[h:(hand h)] (will use_wf1 h c) ∨ ∃[h:(hand h)] (will use_sc1 h c))
⇒
((goal (established c)) ∨ ∃[c2:(cbein c c2)]))

A contact can be established by *assemble*, *use-with-fixture* or *use-self-contact*. If a contact is not required to be established in the goal state, or some other contact cannot be established if it is not established, there is no need to establish the contact. This rule is useful for plans in which an assembly is dismantled to its atomic parts. In those plans, contacts are established in the initial state. Parts are disassembled one by one. In the intermediate states, there are contacts, which are deformed by the plan. These contacts may be feasible for *assemble*. Assembling them should be avoided. This operation will bring the need to plan for disassembling them again, which has already been done. A contact can be temporarily established according to the requirements of the task even if it is not established in the goal state. This rule permits contact establishment in this condition. If another contact cannot be established if this contact is not established, the contact can be established.

5.3 Motion economy rules

Motion economy rules are representations of principles from the literature on motion economy expertise as temporal formulas [2], [3], [13], [14], [15], [16], [17]. These rules serve generating better plans from the point of view of motion economy.

Motion economy rules eliminate some of the possible next states. Choice of sequence and type of the actions for the plan is guided by the rules. This control also gives way to selection of the sequence of establishment of connections.

Motion economy rules prune states that will lead to plans, which are conflicting with the motion economy principles. For instance, one of the rules states to provide fixed places for the things worked with. This rule will prune plans in which a tool is left and picked up from several locations. This aspect, which may be ignored by many individuals, promotes one plan from many others. Note that a motion economy rule may prune plans having fewer actions for the sake of the principle it applies. Although common sense rules do not serve for plan evaluation, this group is beneficial for both plan evaluation and selection and speeding up the search.

As we have stated above, preparation of this rule set is done by a survey of motion economy studies. In literature related to motion economy, principles and checklists are given as guidance to conduct motion economy studies in real life. Principles summarize the expertise in motion economy. Motion economy principles are common in all of these references with slight differences. All of the rules described below can be found in literature with exact phrases. We provide at least one reference to each rule in the set.

i. Keep the work in the normal working area

The intersection of normal reach areas of the two hands is the suitable area for two-handed work to be done. Especially, maximum reach area *"constitutes a zone where which two handed work cannot be done without causing considerable disturbance of posture, accompanied by excessive fatigue"* [3], [14]. This area is called the working area in this study. We have formulated this principle in temporal logic as follows.

(always ∀[h:(hand h)] ∃[c:(contact c)] (contact-est h c))⇒ (at h WA))

What is meant by work here are the contact establishment operations. All of these operations are done with cooperation of the two hands. If a hand will do any of these actions, its location must be the working area.

ii. Do not grasp anything from the normal working area [14]

This rule is a complement of the first one. Working area must be free of containers for two-handed work can be efficiently done. Our formulation of the principle is given below.

(always ¬∃[x:(container x WA)])

This rule activates when an atomic part is grasped for the first time, for a container is only added in this case. Hands are allowed to put an assembly or an atomic part to the working area temporarily.

iii. Grasp a single part instead of parts, jumbled together

Time spent for *select-grasp* should be avoided whenever possible. When there is need for searching the articles in the workspace, time for selecting them for *select-grasp* increases. Parts must not be stored among others in the same container [3], [15], [16]. We converted this principle to the temporal formula given below.

(always ¬∃[l:(isloc l)] (> (numconts l) (max-cont)))

(numconts l) is a function defined over locations which stores the number of containers that are placed at the location l. *Max-cont* is the maximum number of containers that are allowed to be placed in a location. Maximum 2 containers are allowed in a location in the current model. Number of *select-grasp*s at a location

determines the number of containers at a location. This rule states that the number of containers must not exceed *max-cont*. Storing of different parts at the same container is neither allowed nor modeled. If there are many containers at one partition of the workspace, parts are assumed to be jumbled, and extensive care will be spent while searching and selecting them for *select-grasp*.

iv. Provide a fixed place for the things worked with

The operator must be relieved from the decision where to grasp the part, which is needed for the next operation. Movements of the operator must be completed with the *"least conscious mental direction"* [3], [14]. We have formulated this principle in temporal logic as follows.

$$(\text{always } (\forall[x,l:(\text{container } x\ l)]\ \neg\exists[l2:(\text{container } x\ l2)]\ \neg(=\ l\ l2) \wedge \neg\exists[t,l:(\text{tool } t) \wedge (\text{at } t\ l)]\ (\neg(=\ l\ WA)) \wedge \neg(\text{container } (\text{type } t)\ l))))$$

Any part or tool must not be stored more than one location in the workspace. For a certain type of object when there is an attempt to grasp it, the attempt will be stopped by this rule if there is a container elsewhere in the workspace. Moreover, if a tool has been grasped and a container of it is located, it cannot be elsewhere in any time. Recall that *select-grasp* has not a precondition checking the existence of a container. Objects can be grasped from anywhere and a container is added to the world while grasping.

v. Parts that are used most frequently must be located near the point of use

For an operation, parts that are used more than others should be located close to the position of the worker to permit the least possible distance traveled [3]. We converted this principle to the temporal formula given below.

$$(\text{always } \forall[x,y,l1,l2:(\text{container } x\ l1) \wedge (\text{container } y\ l2)]$$

$(\neg(= x\ y) \wedge (\geq (\text{num-of-type } x)\ (\text{num-of-type } y))) \Rightarrow$
$\neg((= (\text{dist } l1)\ \text{max}) \wedge\ (= (\text{dist } l2)\ \text{normal})))$

(num-of-type x) determines the number of parts with same type that take place in the product specification. Parts, which take place more than others in the product specification, will certainly be used more frequently in a plan. Parts, which are used more than others must be stored in closer regions to working area. *Reach distance* property of a location, given with the *dist* function, determines its distance to the working area. A location may be at the normal reach area or maximum reach area. This rule states if there are containers of two different parts and one is used more than the other part in the product, location of the container of that part cannot be at maximum reach area while other being at the normal reach area. As containers are being put to one of these two areas, parts that are used more frequently will always be at the nearest locations.

vi. Keep transport as short as possible

Transportation usually cannot be eliminated from the plans. However, the time spent for transportation can be reduced by reducing the distances to be moved. Movements should be arranged to avoid movement of the trunk of the body. Materials should be located as close as possible to the working area [3], [14], [15], [16]. We have formulated this principle in temporal logic as follows.

$(\text{always } \forall[l:(\text{isloc } l)]\ (= (\text{dist } l)\ \text{max}) \Rightarrow$
$(\exists[h:(\text{will transport\_empty } h\ l)] \vee \exists[h:(\text{will transport\_loaded } h\ l)]) \Rightarrow$
$(\neg\exists[x:(\text{isloc } x)]\ ((= (\text{dist } x)\ \text{normal}) \wedge (< (\text{numconts } x)\ (\text{max-cont}))) \vee$
$\exists[h,p:(\text{holds } h\ p)]\ (\text{goal } (\text{at } p\ l)))))$

There is a correspondence between the locations where objects are being grasped and the distance traveled by the hands. As contact establishments will be carried out at the working area, a hand may go to other locations to grasp an object. Grasping an object means addition of a container to the location. If there exists a location in the

normal reach area, which has not filled its capacity for containers, transports to maximum reach area are not allowed. The idea is to use normal reach area as long as it is possible. One exception is when a part is to be transported to its goal location.

vii. Changes in direction of motion should be kept minimum

Movements with sharp and abrupt changes of motions are fatiguing to the operator than smooth continuous curved motions [3]. Amount of time saved by making curved motions can best be tested by comparing the moving of the hand in a rectangular pattern with moving it in a circular pattern. Changing direction with high degree angles requires deceleration and acceleration of the hand. Straight lined motions are performed slower for this reason [16], [17]. We converted this principle to the temporal formula given below.

```
(always
 (∀[h,l1: (at h l1)] ((= (axis l1) horizontal) ∧
        (next ∃[l2:(will transport_empty h l2)] (= (axis l2) vertical)))) ⇒
                (until ¬(∃[l3:(will transport_loaded h l3)]
                        (= (axis l3) horizontal) ∨
   ∃[l3:(will transport_empty h l3) (= (axis l3) horizontal))
   ((at h WA) ∨ ∃[c:(contact c)] (contact-est h c)))) ∧
 (∀[h,l1:(at h l1)] ((= (axis l1) vertical) ∧
        (next ∃[l2:(will transport_empty h l2)] (= (axis l2) horizontal)))) ⇒
                (until ¬(∃[l3:(will transport_loaded h l3)]
                        (= (axis l3) vertical) ∨
   ∃[l3:(will transport_empty h l3) (= (axis l3) vertical))
   ((at h WA) ∨ ∃[c:(contact c)] (contact-est h c)))) ∧
 (∀[h,l1:(at h l1)] ((= (axis l1) horizontal) ∧
        (next ∃[l2:(will transport_loaded h l2)] (= (axis l2) vertical)))) ⇒
                (until ¬(∃[l3:(will transport_loaded h l3)]
                        (= (axis l3) horizontal) ∨
          ∃[l3:(will transport_empty h l3) (= (axis l3) horizontal))
   ((at h WA) ∨ ∃[c:(contact c)] (contact-est h c)))) ∧
        (∀[h,l1:(at h l1)] ((= (axis l1) vertical) ∧
        (next ∃[l2:(will transport_loaded h l2)] (= (axis l2) horizontal)))) ⇒
                (until ¬(∃[l3:(will transport_loaded h l3)]
                        (= (axis l3) vertical) ∨
```

∃[l3:(will transport_empty h l3) (= (axis l3) vertical))
((at h WA) ∨ ∃[c:(contact c)] (contact-est h c)))))

A numeric measure of changes in direction is not calculated for the formulation of this rule. A sort of minimization of changes in direction is applied by permitting some kind of changes in location for all plans. Transports between the normal and maximum reach areas on a straight perpendicular line from the operator are not considered to cause big changes in direction. These movements keep the angle between the arm and body constant. Likewise, sideway transports without changing the reach distance of the hands, are not also considered to cause big changes in direction. These movements change the angle between the arm and the body but keep the distance by which the arm is stretched constant. Major changes in direction of motion will occur when hands travel back and forth between vertical and horizontal locations, for this causes a change both in reach distances and the angle between the arm and the body. Thus, changes in the axis property of hands' locations are used to catch and prune big changes in direction. Track of the last three locations of a hand is kept and states, which have alternating axis properties for hands' locations are pruned.

This rule is designed to activate when a hand is in the middle of changing locations. That is when instant effects of transport actions are realized and delayed effects are in the event queue. At this point, the current location of the hand and its location in the next world are known. If this two locations have different values for the axis property, and the value of this property changes again when the hand visits a third location, no successor states of the last state will be explored. The *until* temporal formula is used to initialize the log of transports when the hand moves to the working area or performs a contact establishment operation. The transport log of this rule is not affected by *release* and *select-grasp*.

This rule is composed of four parts that are connected with ∧. First formula is written to catch horizontal-vertical-horizontal pattern for *transport_empty*. Second formula is

for vertical-horizontal-vertical pattern for *transport_empty*. Last two formulas are the versions of first two formulas for *transport_loaded*.

viii. Eliminate abnormal motions

Evidence show that forearm is the most desirable member of the body to be used in repetitive work. Use of the upper arm and shoulders should be avoided [3], [14]. Our formulation of the principle is given below.

(always (∀[h,l:(will transport_empty h l)]
(if-then-else (= h right) ¬(= (side l) left) (= (side l) right))) ∧
(∀[h,l:(will transport_loaded h l)]
(if-then-else (= h right) ¬(= (side l) left) ¬(= (side l) right))))

The notion of abnormal motion is ambiguous. The observer of the operation has to notice and modify the operation. Although kinds of these motions can be extended, one sort of transport operation is considered abnormal and is forbidden with this formula. Transport actions of one hand directing to the opposite side of workspace are excluded from plans. If the hand is right hand, it cannot move to left side of the workspace. If it is left hand, it cannot move to right side of the workspace.

ix. Motion of the arms should be simultaneous, in opposite and symmetrical directions

The hands should begin and complete the work at the same time. Simultaneous motions reduce the makespan of the tasks as well as the total idle time of the hands. Symmetrical motions tend to *"balance each other"* and *"reduce the shock and jar on the body"* [3]. Symmetric motions *"induce a rhythm in the operator's performance"*. Awkward motions without symmetry tend to be slower [16]. We converted this principle to the temporal formula given below.

(always ∀[lr1,ll1,lr2,ll2: (at right lr1) ∧ (at left ll1) ∧ (isloc lr2) ∧ (isloc ll2)]
(((next (will transport_loaded right lr2)) ∨
(next (will transport_empty right lr2)) ∨
(next (next (will transport_loaded right lr2))) ∨
(next (next (will transport_empty right lr2)))) ∧
((next (will transport_loaded left ll2)) ∨
(next (will transport_empty left ll2)) ∨
(next (next (will transport_loaded left ll2))) ∨
(next (next (will transport_empty left ll2))))) ⇒
(∃[h,p:(holds h p)] (goal (at p ll2) ∨ ∃[h,p:(holds h p)] (goal (at p lr2)) ∨
(((= lr2 WA) ∧ (= ll2 WA) ∧ (= (side lr1) right) ∧ (= (side ll1) left) ∧
((= (dist lr1) max) ⇒ (= (dist ll1) max)) ∧
((= (dist lr1) normal) ⇒ (= (dist ll1) normal)) ∧
((= (axis lr1) vertical) ⇒ (= (axis ll1) vertical)) ∧
((= (axis lr1) horizontal) ⇒ (= (axis ll1) horizontal))) ∨
((= lr1 WA) ∧ (= ll1 WA) ∧ (= (side lr2) right) ∧ (= (side ll2) left) ∧
((= (dist lr2) max) ⇒ (= (dist ll2) max)) ∧
((= (dist lr2) normal) ⇒ (= (dist ll2) normal)) ∧
((= (axis lr2) vertical) ⇒ (= (axis ll2) vertical)) ∧
((= (axis lr2) horizontal) ⇒ (= (axis ll2) horizontal))) ∨
(((= (side lr1) right) ∧ (= (side ll1) left) ∧
((= (dist lr1) max) ⇒ (= (dist ll1) max)) ∧
((= (dist lr1) normal) ⇒ (= (dist ll1) normal)) ∧
((= (axis lr1) vertical) ⇒ (= (axis ll1) vertical)) ∧
((= (axis lr1) horizontal) ⇒ (= (axis ll1) horizontal)) ∧
(= (side lr2) right) ∧ (= (side ll2) left) ∧
((= (dist lr2) max) ⇒ (= (dist ll2) max)) ∧
((= (dist lr2) normal) ⇒ (= (dist ll2) normal)) ∧
((= (axis lr2) vertical) ⇒ (= (axis ll2) vertical)) ∧
((= (axis lr2) horizontal) ⇒ (= (axis ll2) horizontal))))

(always ∀[h1,h2:(hand h1) ∧ (hand h2)] (¬(= h1 h2) ∧
(∃[l:(will transport_loaded h1 l)] ∨ ∃[l:(will transport_empty h1 l)]) ⇒
(∃[l:(will transport_loaded h2 l)] ∨ ∃[l:(will transport_empty h2 l)] ∨
¬(busy h2)))

(always ∀[h1,h2:(hand h1) ∧ (hand h2)] ¬(= h1 h2) ⇒
¬((≥ (rel-actions h1) 1) ∧ (≥ (rel-actions h2) 1) ∧
¬(busy h1) ∧ ¬(busy h2)))

This rule is composed of three formulas. First one activates when two hands are making simultaneous transports. Simultaneous transports are organized by imposing

constraints on the values of properties of destination locations and current locations of the hands. Transports must be from and to locations having same axis and reach distance properties, and transports must be in opposite sides of the workspace.

Second formula ensures that while one hand is making transports, the other hand makes no action other than a transport action. It may be idle.

Third formula prunes plans in which one hand acts while the other is idle at a certain step, and the other hand acts while this one is idle in the next step. These consecutive actions must be rearranged to be concurrent whenever possible.

x. A constantly extended position of the arms should be avoided

A constantly extended position of the hand leads to fatigue and diminishes precision and skill of the hand [17]. We have formulated this principle in temporal logic as follows.

(always
 ∀[h1,h2,l:(hand h1) ∧ (hand h2) ∧ (isloc l)]
 (¬(= h1 h2) ∧ (at h1 l) ∧ ¬(= l WA)) ⇒
  (until ((¬∃[h:(hand h)] (busy h)) ⇒
  ((> (rel-actions h2) (rel-actions h1)) ⇒
  (≤ (abs (- (rel-actions h2) (rel-actions h1))) (steps-to-extend))))
   ¬(at h1 l)))

(rel-actions h) is used to keep track of number of actions performed by a hand when the other hand is idle. It stands for the number of relative actions. Its value is incremented at every action performed by a single hand and it is initialized to zero when hands make simultaneous actions or when they act together. This rule activates when a hand is at some location other than the working area. Value of relative actions of the other and may not exceed a certain value, *(steps-to-extend)*, while the

hand stands at the same location. Value used for *(steps-to-extend)* is 1. This means a hand can remain outstretched for a single step.

xi. Eliminate hand as the holding device

Hold is an ineffective *therblig*. The hand should not be considered as a holding device. Hold can usually be eliminated with the use of jigs, friction, magnetic devices, fixtures and containers. Capacity for productive handwork is reduced by 50 % when hand is used for holding objects [3], [14], [16]. Our formulation of the principle is given below.

$$
\begin{aligned}
&(\text{always}\\
&\quad \forall[h1,h2,l,p:(\text{hand } h1) \wedge (\text{at } h1\ l) \wedge (\text{hand } h2) \wedge (\text{holds } h1\ p)]\\
&\qquad (\neg(= h1\ h2) \wedge \neg(\text{at } h1\ WA)) \Rightarrow\\
&\ (\text{until } (\neg\exists[h:(\text{hand } h)]\ (\text{busy } h) \Rightarrow\\
&\ ((> (\text{rel-actions } h2)\ (\text{rel-actions } h1)) \Rightarrow\\
&\ \ (\leq (\text{abs } (- (\text{rel-actions } h2)\ (\text{rel-actions } h1)))\ (\text{steps-to-hold}))))\\
&\ \ (\neg(\text{at } h1\ l) \vee \neg(\text{holds } h1\ p))))
\end{aligned}
$$

(rel-actions h) is used in a similar manner with the previous rule. While a hand is holding a part, number of actions made by the other hand is checked until hand releases or transports the part. *(steps-to-hold)* is a constant, which determines the number of relative actions allowed. Value used for *(steps-to-hold)* is 1. Holding an assembly at the working area is allowed. A hand may hold an assembly in the working area while the other hand make transports and carries another part to be assembled.

xii. Distribute work evenly between hands

Effort should be equally distributed to the hands [14], [17]. We converted this principle to the temporal formula given below.

(always
 ∀[h1,h2:(hand h1) ∧ (hand h2)]
 ¬(= h1 h2) ⇒ (≤ (abs (- (fatigue h1) (fatigue h2))) (rel-fat)))

Every action increases the *fatigue* value of a hand by one. *Fatigue* of the hands is balanced by keeping the difference between them below some constant value. *(rel-fat)*, stands for relative fatigue, determines the limit up to which a hand may be employed more than the other hand. The value of *(rel-fat)* used in the experiments is 4. This value is chosen to permit transport-grasp-transport triplets for one hand when the other is idle.

Motion economy rules, which are described above, do not represent a complete set of rules for motion economy related expertise. Considering the diversity and quantity of related studies, it is clear that claiming to have gathered a complete set of principles is unrealistic. Besides, the rule set, which we have formulated, does not represent all the principles and checklists that were available. These rules represent the most generic ones of those principles. To define some of these principles one must extend the model to describe some specific types of containers, parts and fixtures, and decide on the availability of these, while to define some others the knowledge representation must be extended to include details about the work place such as lighting. Some examples of motion economy principles, which are not formulated here, are given below. These are taken from [3].

- *Eliminate overcoming of momentum.*
- *Can objects be slid instead of carried?*
- *Are the eye movements properly coordinated with the hand motions?*
- *Are parts and materials properly labeled?*
- *Is the lighting satisfactory?*
- *Can color be used to facilitate selecting parts?*
- *Can a vacuum, magnet, rubber fingertip, or other device be used to advantage in grasp?*
- *The height of the work place and the chair should be arranged so that alternate sitting and standing at work are easily possible.*

Not a single rule of this set has been devised in the course of this study. We interpreted and represented the principles of motion economy in first order temporal logic. The principles or checklists were not designed to aid for symbolic representation of the domain. They may suggest experts' methods of improvement in many different ways. *Motion economy rules*, which we have formulated, are abstractions of the principles. *Motion economy rules* as temporal formulas, may have narrower horizons of influence than the interpretation of the principles by a motion economy specialist. We had to take initiative to adapt the principles of motion economy to our model. Rules may have loss of impact compared to the way they are presented in the referred studies. It is clear that a principle has a more potential when it is used by a motion economy expert, than the way it is symbolically defined here.

Principles of motion economy are analyzed and explained by examples in motion economy literature. General way of justification of these principles is to compare two methods, where an improved method is generated from the old one with the application of a principle. These methods are timed with trained operators, and a quantitative justification is made. Here, a justification of why a plan, which is in conformity with these rules, is preferred to others will not further be made. This is to judge why a principle is a principle and this judgment is made in the references provided. However, the way these rules guide the search and change the plans will be examined.

Three sets of search control rules are given in this chapter, common sense rules, goal-related rules and motion economy rules. We have developed first two rule sets from scratch to use for fastening the search. We have analyzed and interpreted some core motion economy principles to search control rules. We have developed third rule set to evaluate the plans. We have formulated all of the search control rules as temporal formulas to use them with TLPLAN.

CHAPTER 6

EXPERIMENTAL RESULTS AND DISCUSSION

TLPLAN is used to implement the model and generate plans for experimentation and discussion. TLPLAN is run on Mandrake Linux 10, on P4 1.4 GHz. Model of the domain written in TLPLAN input syntax is given in Appendix A. Choice of TLPLAN is related to our design of model. TLPLAN supports the following requirements of motion economy domain on top of a standard planner:

- Concurrency
- Actions with resources
- Actions with durations
- Defining search control strategy

Complexity of the problems depends on the number and type of parts, tools and contacts in a product specification. Number of parts and number of contacts are not completely dependent to each other, any configuration of them is possible, but type of actions required for a certain goal determines their number in a product representation. Major types of operations that determine the complexity are contact establishment operations. Number and variety of these actions affect search complexity, for they increase the size of the final plan. Complexity of the problem also increases with the number of feasible assembly sequences. Number of feasible assembly sequences depends on the tightness of the precedence relationships of the contacts. If the precedence relations are loose, there are many alternatives at each decision for contact establishment, and there are many possible plans which solve the problem. Representation of the workspace affects complexity too. If the workspace is divided into more locations, problem complexity increases with the increasing opportunities of transports and workspace arrangements.

Depth-first search strategy is used for experimental runs, and it is advised to be used for applications. The preference of depth-first strategy over breadth-first search or heuristic search stems from following reasons.

i. We do not want to find the shortest path to the solution. We do not either aim to define a cost function and find any least cost path to the solution. Our main motivation is to let motion economy rules to guide the construction of a "good" plan. We claim to produce the best plans without using any other search strategy than the principles we have formulated. Depth-first search allows us to evaluate the solutions only by the *motion economy rules*, for it is the least informed search strategy.
ii. A state in our model consumes much more memory than ordinary planning problems as states keep track of temporal formulas as well. Temporal formulas are evaluated over sequences of states, and formulas, whose terminating conditions are not satisfied, are transferred to successor states. Besides, each generation of a state starts new series of states and new instantiations of temporal formulas are added to the path. For the states at level n, number of temporal formulas being evaluated at that state, is the sum of numbers from 1 to n times the total number of temporal formulas in the worst case. This causes a huge consumption of memory with every state generated. Breadth-first search uses more space than depth-first search since all the nodes at a level are stacked to be expanded. Number of nodes at a level is exponential by the depth of the search. Experiments show that large consumption of memory slows down processing of the planner even in small size problems.
iii. We intend to maintain search control as a separate module by which we can experiment on sample problems. We want actions' preconditions to be free of search control to see how raw plans will be. This causes a very large branching factor. Branching factor is depens on the problem size for problem parameters match with the arguments of an action and cause more than one instantiation of the action.

For a problem with *m* parts and *n* contacts, number of instantiations of the major actions at any state is given in Table 6.1. We ignore the usage of tools for the sake of simplicity. In columns 2 to 5, factors increasing the number of instantiations for the actions are given. The workspace has nine distinct locations and hands are at one of them at a certain time. Eight other locations are possible destinations for transportation. Actions can match both of the hands. We assume that all contacts are always feasible. We do not include feasibility checks to this computation. *Assemble* can be instantiated for all of the contacts.

Table 6.1 Number of instantiations of actions at a state

	m parts	*n* contacts	locations	hands	Instantiations
transport_empty	-	-	8	2	16
transport_loaded	*m*	-	8	2	16*m*
select-grasp	*m*	-	-	2	2*m*
release	*m*	-	-	2	2*m*
assemble	-	*n*	-	1	*n*

Sum of the number of instantiations will give a rough estimate of the branching factor of the problem. We assume that the hands are empty half of the time, and add average of number of instantiations for *transport_empty* and *transport_loaded*, and *select-grasp* and *release*. Branching factor is estimated as 8+10*m*+*n*. In a simple plan, if we assume hands will do transport-grasp-transport triplets for parts and assemble for contacts, average plan length is 3*m*+*n*. Number of nodes at a certain level is Table 6.2 shows explosion of expected number of nodes at the solution depth for breadth-first search.

Table 6.2 Explosion of nodes in the solution level for breadth-first search

Contacts	**Parts**	**Estimated branching factor**	**Estimated length of the solution path**	**Number of nodes in the solution level**
1	2	29	7	17249876309
2	3	40	11	4.1943E+17
4	5	62	19	1.13617E+34
9	10	117	39	4.56298E+80

Table 6.3 shows the values for elapsed times and number of worlds generated for simple assembly problems. The simplest problem is to assemble two parts together. Parts have a single contact. The problem size is increased by adding a part and a contact at a time to the simpler problems. As two containers at maximum are allowed for a location by a motion economy rule, the problem with 15 contacts and 16 parts is the problem with largest size that can be solved unless this constraint is changed. All problems are run with all the search control rules. TLPLAN could not find a solution to the problem with 3 contacts and 4 parts within 18 hours. Planner probably cannot allocate enough memory to process further with the explosion of states and temporal formulas, with the reasons, which are explained above. Figure 6.1 and Figure 6.2 depict the performance of depth-first search and breadth-first search with increasing problem complexity according to the figures in Table 6.3.

Table 6.3 Plan statistics of two search strategies

Contacts	Depth-first search		Breadth-first search	
	Elapsed time (sec)	Worlds generated	Elapsed time (sec)	Worlds generated
1	0.08	149	1.28	2341
2	0.19	399	8.86	7571
3	0.85	1430		
4	1.33	1944		
7	13.96	8268		
9	42.11	18873		
11	81.08	32950		
15	216.86	73926		

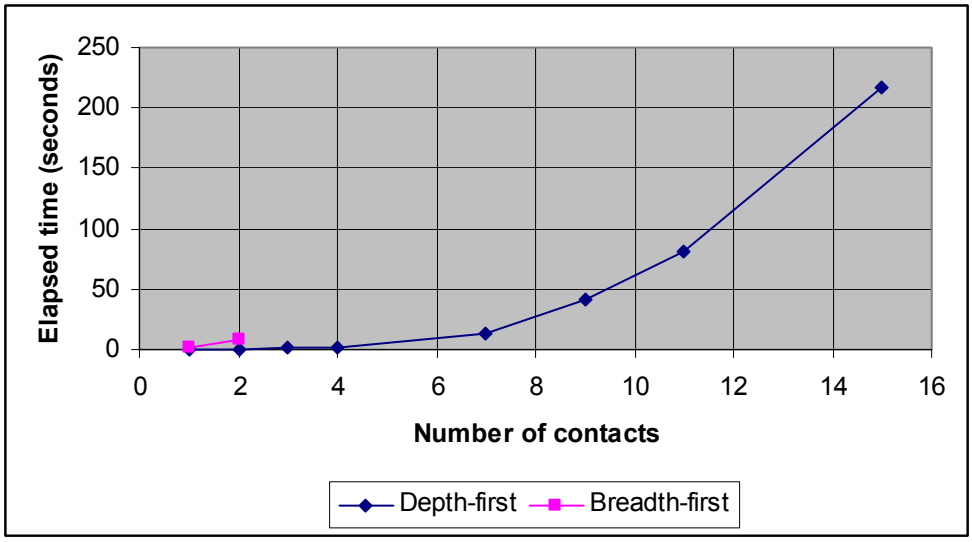

Figure 6.1 Elapsed time with respect to problem size

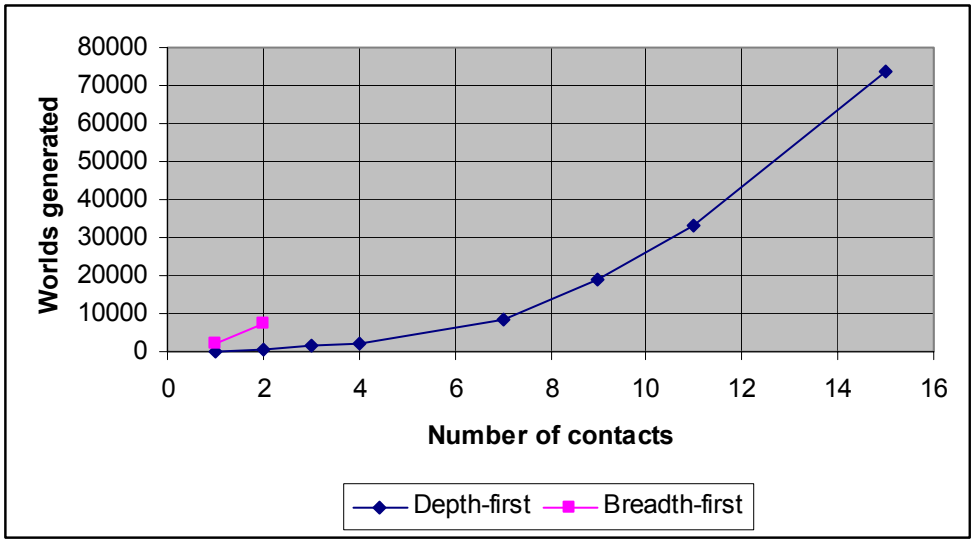

Figure 6.2 Worlds generated with respect to problem size

In the following sections of this chapter, we formulated and solved sample problems from the motion economy domain in our model. Representative problems are chosen to reflect the expressiveness of the representation scheme. In each problem, effects of the *motion economy rules* will be examined.

6.1 Sample problem 1

First problem we are going to represent and solve is assembling a ballpoint pen assembly from its atomic parts. Atomic parts of a ballpoint assembly are body (Bo), button (Bu), head (H), ink tube (I) and cap (C). Parts of the assembly are given in Figure 6.3. Graph of contacts of the ballpoint assembly is given in Figure 6.4. Head and ink tube must be assembled together. Button, cap and head must be assembled to the body. Head must be assembled from the front of the body. Precedence relations of the product are given in Table 6.4.

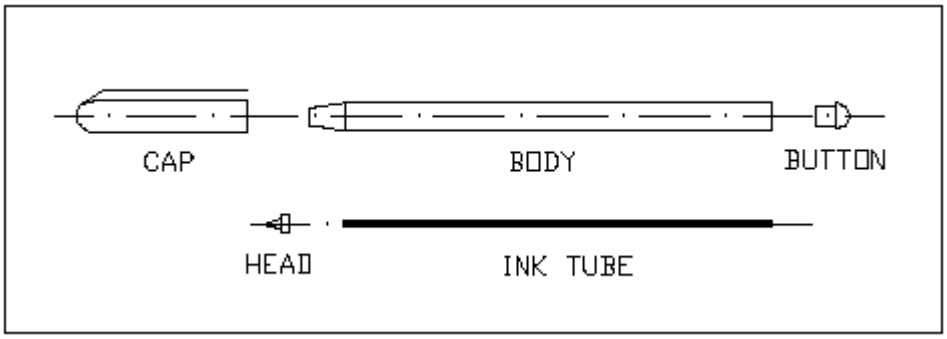

Figure 6.3 Ballpoint pen assembly

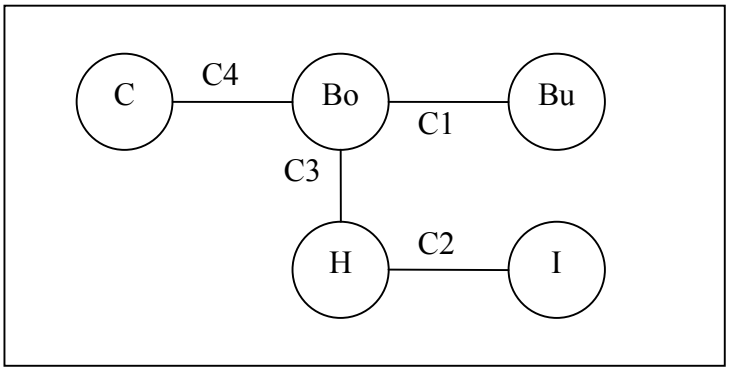

Figure 6.4 Graph of contacts of ballpoint pen assembly

Table 6.4 Precedence relations of the ballpoint pen assembly

Predicate	**Explanation**
(cbei C3 C4)	Body and head cannot be assembled if cap and body are assembled.
(cbein C3 C2)	Body and head cannot be assembled if head and ink tube are not assembled.
(cbein C4 C3)	Body and cap cannot be assembled if body and head are not assembled.
(cbein C4 C2)	Body and cap cannot be assembled if head and ink tube are not assembled.

This problem is first solved with merely the *common sense* and *goal-related rules*. All the motion economy rules are turned off. Figure 6.5 shows the arrangement of the workspace and sketches of the movements of the hands. Dashed lines show the movement pattern of right hand. Right hand will be shown with dashed lines throughout this chapter. In fact, movement patterns of both hands are alike although they are not simultaneous. The plan is given in Figure 6.6.

In this plan, body and button are assembled first. Body and button have been grasped from MLH location, which is on the maximum reach area, on the left-hand side and near to the vertical axis. Initial locations of body and button are set to MLH and containers are placed on the location. After *select-grasp*, parts are carried to MLV (maximum reach area, left-hand side, vertical) and they are assembled there. After *assemble*, right hand holds the assembly and carries to it MLH. Right hand releases the body-button assembly in MLH and moves back to MLV and grasps ink tube (time 8).

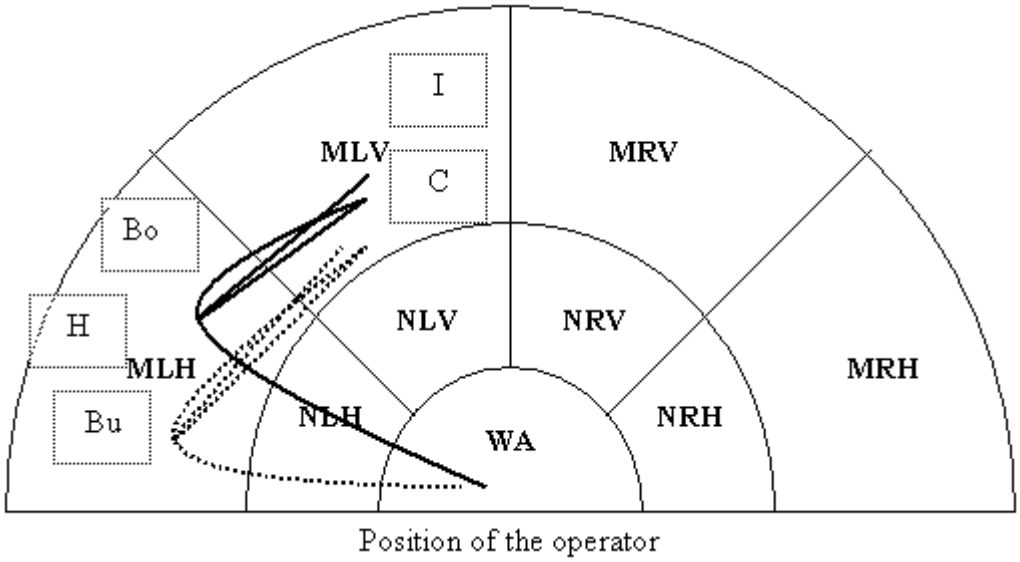

Figure 6.5 Sketch of the workspace for the plan in Figure 6.6

Container of ink tube is put at MLV. Meanwhile left hand moves to MLH, grasps head and returns to MLV. Container of head is set to MLH. Hands assemble head and ink tube at time 9. After *assemble*, right hand holds the assembly and carries it to MLH. Left hand moves to MLH and grasps body-button assembly. Body-button assembly and head-ink tube assembly are assembled in MLH. After *assemble*, right hand holds the assembly and carries it to MLV. Left hand meets right hand at MLV and grasps cap. Cap's initial location is MLV and a container is put to that location. Cap is assembled last in MLV, and ballpoint pen assembly is completed.

Time	Left Hand	Right Hand
1	(transport_empty left wa mlh)	(transport_empty right wa mlh)
2	(select-grasp-part left body)	(select-grasp-part right button)
3	(transport_loaded left mlh mlv)	(transport_loaded right mlh mlv)
4	(assemble left right c1)	(assemble left right c1)
5	(transport_empty left mlv mlh)	(transport_loaded right mlv mlh)
6	(select-grasp-part left head)	(release right body)
7	(transport_loaded left mlh mlv)	(transport_empty right mlh mlv)
8		(select-grasp-part right ink_tube)
9	(assemble left right c2)	(assemble left right c2)
10	(transport_empty left mlv mlh)	(transport_loaded right mlv mlh)
11	(select-grasp-part left body)	
12	(assemble left right c3)	(assemble left right c3)
13	(transport_empty left mlh mlv)	(transport_loaded right mlh mlv)
14	(select-grasp-part left cap)	
15	(assemble left right c4)	(assemble left right c4)

Figure 6.6 Plan of ballpoint pen assembly, without *motion economy rules*

This plan is found by turning all of the *motion economy rules* off. This means the planner selected this plan among many feasible ones with no evaluation mechanism. We will find a better plan by evaluating the feasible plans during plan generation process. In our model, plan generation and plan improvement processes are embedded. However, for the sake of demonstration, we will carry out an incremental addition of some of the rules to this problem, to show the improvement each rule will make on the raw plan, which is found in the first step.

Effect of motion economy rule 7 will be shown first on this plan. Conditions in which this rule is effective are seldom observed. This rule states "*changes in direction of motion should be kept minimum"*. This rule is effective when the hands move sideways without establishing a contact. Hands make zigzag motions in the first plan between times 3 and 5. The problem is solved by turning this rule on. Figure 6.7 shows the arrangement of the workspace and sketches of the movements of the hands.

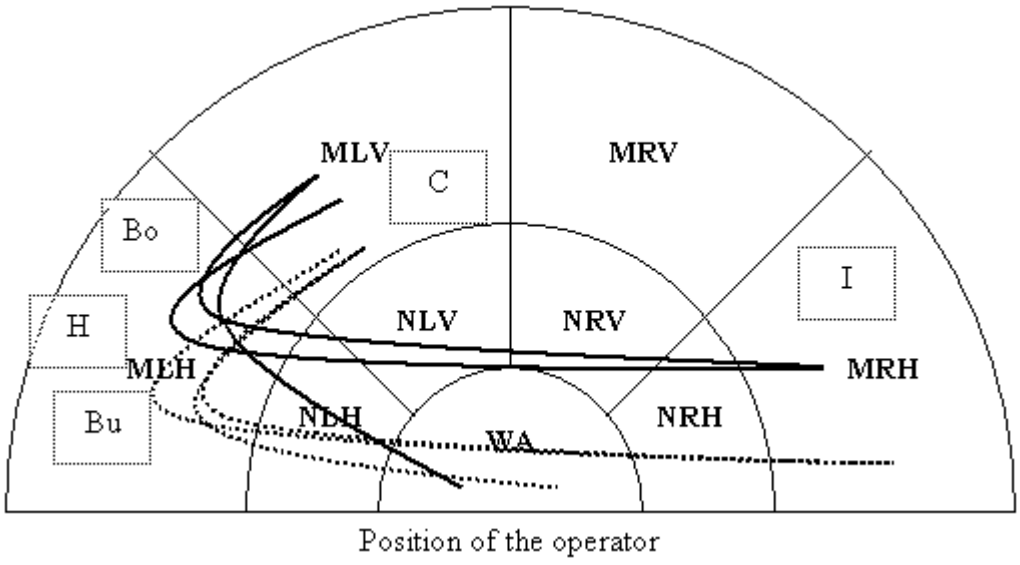

Figure 6.7 Sketch of the workspace for the plan in Figure 6.8

Plan generated with the control of this rule is given in Figure 6.8. To avoid moving between locations MLH and MLV, Ink tube is now grasped at MRH, and second contact is established there. Straight line motions and sharp changes in direction are prevented in this plan. However, movements seem to be more awkward. This plan is generated to show the effect of rule 7. Employment of the other motion economy rules will solve this anomaly of movements. We will continue by applying *motion economy rules* in order.

First two *motion economy rules* are "*keep the work in the normal working area"* and "*do not grasp anything from the normal working area"*. In the first plan given above assemblies are established outside WA (working area). Nothing was grasped from the working area by coincidence, because of the uninformed search strategy. Now, we obtain a plan with the inclusion of these rules. In the plan given in Figure 6.9, all the assemblies are established in the working area and transports are arranged accordingly. After grasping button and body, and assembling them in WA, hands move to MLH where right hand releases body-button assembly, and left hand grasps head at the same time. Then left hand returns to WA. Right hand moves to MLV, grasps ink tube and returns to WA. Head and ink tube are assembled. Last two contacts are established as right hand holds each assembly at WA and left hand brings body-button assembly and cap one by one to WA.

Time	Left Hand	Right Hand
1	(transport_empty left wa mlh)	(transport_empty right wa mlh)
2	(select-grasp-part left body)	(select-grasp-part right button)
3	(transport_loaded left mlh mlv)	(transport_loaded right mlh mlv)
4	(assemble left right c1)	(assemble left right c1)
5	(transport_empty left mlv mlh)	(transport_loaded right mlv mlh)
6	(select-grasp-part left head)	(release right body)
7	(transport_loaded left mlh mrh)	(transport_empty right mlh mrh)
8		(select-grasp-part right ink_tube)
9	(assemble left right c2)	(assemble left right c2)
10	(transport_empty left mrh mlh)	(transport_loaded right mrh mlh)
11	(select-grasp-part left body)	
12	(assemble left right c3)	(assemble left right c3)
13	(transport_empty left mlh mlv)	(transport_loaded right mlh mlv)
14	(select-grasp-part left cap)	
15	(assemble left right c4)	(assemble left right c4)

Figure 6.8 Plan of ballpoint pen assembly, with *motion economy rule* 7

Time	Left Hand	Right Hand
1	(transport_empty left wa mlh)	(transport_empty right wa mlh)
2	(select-grasp-part left body)	(select-grasp-part right button)
3	(transport_loaded left mlh wa)	(transport_loaded right mlh wa)
4	(assemble left right c1)	(assemble left right c1)
5	(transport_empty left wa mlh)	(transport_loaded right wa mlh)
6	(select-grasp-part left head)	(release right body)
7	(transport_loaded left mlh wa)	(transport_empty right mlh mlv)
8		(select-grasp-part right ink_tube)
9		(transport_loaded right mlv wa)
10	(assemble left right c2)	(assemble left right c2)
11	(transport_empty left wa mlh)	
12	(select-grasp-part left body)	
13	(transport_loaded left mlh wa)	
14	(assemble left right c3)	(assemble left right c3)
15	(transport_empty left wa mlh)	
16	(select-grasp-part left cap)	
17	(transport_loaded left mlh wa)	
18	(assemble left right c4)	(assemble left right c4)

Figure 6.9 Plan of ballpoint pen assembly, with *motion economy rules* 1 and 2

This plan is better than the first one in terms of motion economy since working area is the suitable area for two-handed work to happen. Two-handed work when the arms are stretched out is at the expense of the ease of hands. Containers of the parts are still set to MLH and MLV with slight differences. Container of ink tube is at MLV and other containers are at MLH. Note that plan length is increased. Plan length is not a measure of goodness of plans for us. Plans with larger steps may be preferred in light of the motion economy principles. When timed, plans with larger steps can last shorter due to the changing conditions in which operations are performed. Sketch of the workspace is given in Figure 6.10.

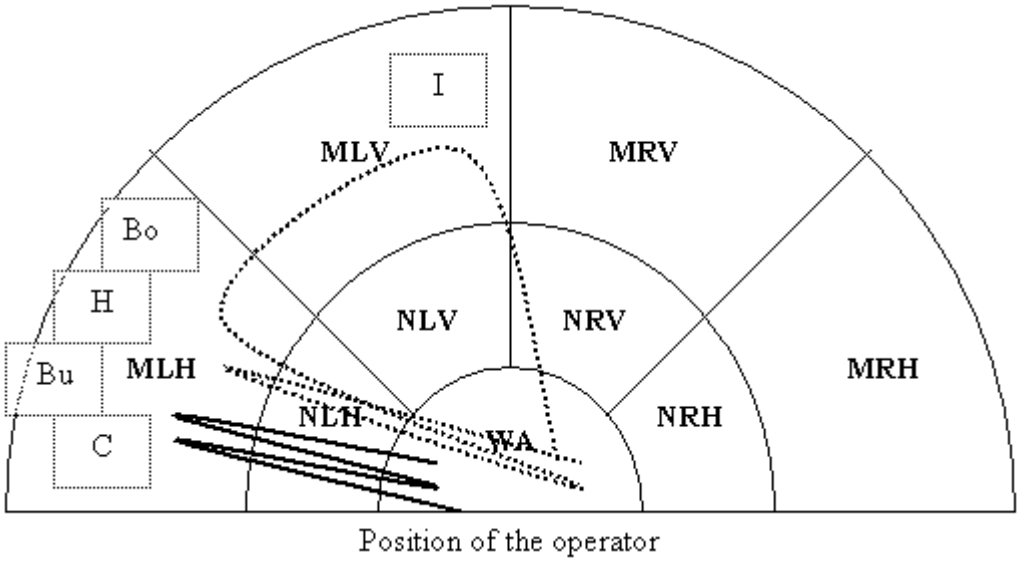

Figure 6.10 Sketch of the workspace for the plan in Figure 6.9

Next, we turn on *motion economy rule* 6. We pass rules 4 and 5. They concern problems where same part is used more than once, and ballpoint pen problem does not have such parts. Rule 3 will be applied later on. This rule states "*keep transport as short as possible"*. Resulting plan is given in Figure 6.11. This plan is very similar to the last one. Only difference is the place of containers. The containers are now brought close to the working area. Locations of containers are in the working area having the same side and axis properties. Container of ink tube is at NLV (normal reach area, left-hand side, vertical) and other containers are at NLH (normal reach area, left-hand side, horizontal). The plan is even better. Transport distances of the hands are reduced when the containers are closed to the working position. The plan became less tiring for the hands.

Time	Left Hand	Right Hand
1	(transport_empty left wa nlh)	(transport_empty right wa nlh)
2	(select-grasp-part left body)	(select-grasp-part right button)
3	(transport_loaded left nlh wa)	(transport_loaded right nlh wa)
4	(assemble left right c1)	(assemble left right c1)
5	(transport_empty left wa nlh)	(transport_loaded right wa nlh)
6	(select-grasp-part left head)	(release right body)
7	(transport_loaded left nlh wa)	(transport_empty right nlh nlv)
8		(select-grasp-part right ink_tube)
9		(transport_loaded right nlv wa)
10	(assemble left right c2)	(assemble left right c2)
11	(transport_empty left wa nlh)	
12	(select-grasp-part left body)	
13	(transport_loaded left nlh wa)	
14	(assemble left right c3)	(assemble left right c3)
15	(transport_empty left wa nlh)	
16	(select-grasp-part left cap)	
17	(transport_loaded left nlh wa)	
18	(assemble left right c4)	(assemble left right c4)

Figure 6.11 Plan of ballpoint pen assembly, with *motion economy rules* 1, 2 and 6

Next, *motion economy rule* 8 is turned on. This rule is "e*liminate abnormal motions"*. The notion of abnormal motions is ambiguous. The temporal formula we have used avoids movement of one hand to the other side of the workspace. The plan is given in Figure 6.12. Up to now all transports were to and from the left-hand side of the workspace. Current plan have the same sequences of actions except transports for the right hand changed location. Right hand moves to the right-hand side of the workspace. It grasps button from NRH (normal reach area, right-hand side, horizontal), ink tube from NRV (normal reach area, right-hand side, vertical). Initial positions and containers of these parts are changed accordingly. The temporary location which right hand releases body-button assembly is changed to NRH. This automatically caused a change in the plan. Now right hand picks up body-button assembly second time for *assemble* because left hand cannot move to that location. Left hand was the operator hand in last two contact establishments. It was empty after *assemble* and it carried parts to WA. In order to make right hand to grasp body-button assembly from NRH plan switched the operator hand in establishment of C2. After the operation left hand holds the assembly and right hand is empty. Notice the difference in representation of *assemble* at time 10 between Figure 6.11 and Figure 6.12. Operator hand is written first. Current plan uses *(assemble right left c2)* instead of *(assemble left right c2)*. The plan is improved since right hand makes no more moves with reverse angle. Less tiring motions are preferred in terms of direction and distance with this plan. Sketch of the workspace for this plan is given in Figure 6.13.

Time	Left Hand	Right Hand
1	(transport_empty left wa nlh)	(transport_empty right wa nrh)
2	(select-grasp-part left body)	(select-grasp-part right button)
3	(transport_loaded left nlh wa)	(transport_loaded right nrh wa)
4	(assemble left right c1)	(assemble left right c1)
5	(transport_empty left wa nlh)	(transport_loaded right wa nrh)
6	(select-grasp-part left head)	(release right body)
7	(transport_loaded left nlh wa)	(transport_empty right nrh nrv)
8		(select-grasp-part right ink_tube)
9		(transport_loaded right nrv wa)
10	(assemble right left c2)	(assemble right left c2)
11		(transport_empty right wa nrh)
12		(select-grasp-part right body)
13		(transport_loaded right nrh wa)
14	(assemble left right c3)	(assemble left right c3)
15	(transport_empty left wa nlh)	
16	(select-grasp-part left cap)	
17	(transport_loaded left nlh wa)	
18	(assemble left right c4)	(assemble left right c4)

Figure 6.12 Plan of ballpoint pen assembly, with *motion economy rules* 1, 2, 6 and 8

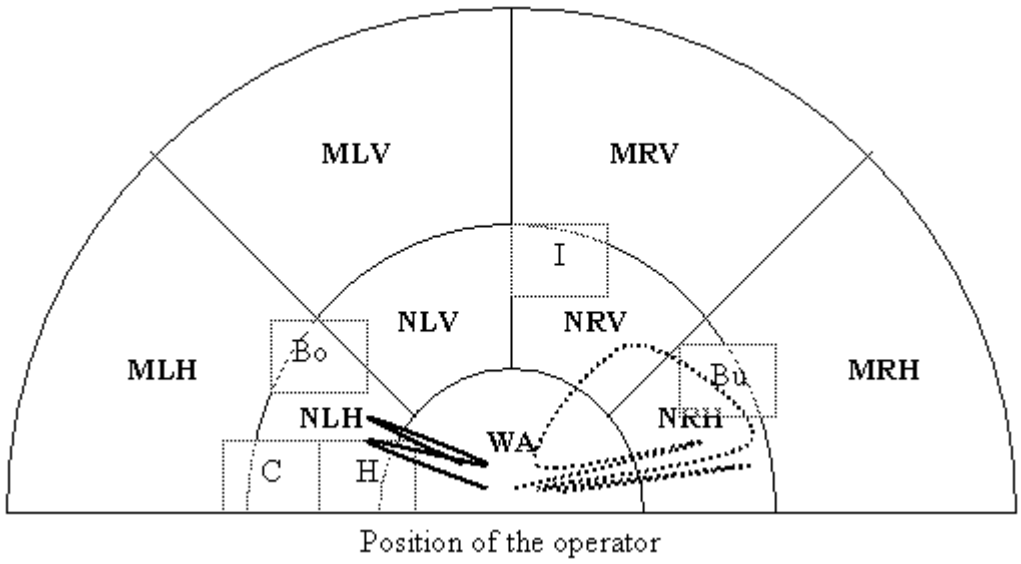

Figure 6.13 Sketch of the workspace for the plan in Figure 6.12

Now, ninth of *motion economy rules* is added to the rule set, which guides the search. This rule states "m*otion of the arms should be simultaneous, in opposite and symmetrical directions"*. Our current plan is the one given in Figure 6.14. This rule has changed the part of the plan between time 7 and 9. Before, while left hand is returning to WA, right hand makes 2 more actions to grasp ink tube from NRV. Movement of right hand back to WA is done while left hand is idle. Current plan changes the location where ink tube is grasped. Container of the tube is placed in NRH. Left hand is at NRH while ink tube is grasped. Both hands move simultaneously back to WA.

Time	Left Hand	Right Hand
1	(transport_empty left wa nlh)	(transport_empty right wa nrh)
2	(select-grasp-part left body)	(select-grasp-part right button)
3	(transport_loaded left nlh wa)	(transport_loaded right nrh wa)
4	(assemble left right c1)	(assemble left right c1)
5	(transport_empty left wa nlh)	(transport_loaded right wa nrh)
6	(select-grasp-part left head)	(release right body)
7		(select-grasp-part right ink_tube)
8	(transport_loaded left nlh wa)	(transport_loaded right nrh wa)
9	(assemble right left c2)	(assemble right left c2)
10		(transport_empty right wa nrh)
11		(select-grasp-part right body)
12		(transport_loaded right nrh wa)
13	(assemble left right c3)	(assemble left right c3)
14	(transport_empty left wa nlh)	
15	(select-grasp-part left cap)	
16	(transport_loaded left nlh wa)	
17	(assemble left right c4)	(assemble left right c4)

Figure 6.14 Plan of ballpoint pen assembly, with *motion economy rules* 1, 2, 6, 8 and 9

This plan is better from the previous one since an unnecessary transportation is avoided and time is saved. Moreover, the concurrent transports at time 7 of the previous plan are changed with concurrent transports at time 8. Now the transports are symmetrical, left hand moves from NLH to WA while right hand moves from NRH to WA. Symmetric motions are easier to automate and reduce workload of the mind. If this rule was applied one step before, it also would prohibit right hand to move to the left-hand side as motion economy rule 8.

Motion economy rule 10, *"a constantly extended position of the arms should be avoided"*, is included next. The plan given in Figure 6.15 is found. Again, how the head and ink tube are assembled after body-button assembly is changed. In the previous plan right hand released body-button assembly and grasped ink tube from NRH at time 6 and 7. Left hand spent 2 units of time at the normal reach area. *Motion economy rule* 10 avoids this time spent while the arms are reaching away from the working area. Plan is modified to temporarily store body-button assembly at WA. Hands now perform a symmetric and simultaneous transport-grasp-transport triplet, after right hand releases the assembly. Transportation to bring back the body-button assembly is not required when the assembly is stored at WA. These are improvements to the last version of the plan. Not also, time is saved but the hands are relieved from the fatigue caused by the time spent at the normal reach area. Even if nothing is done, stretching out the arms requires force.

Time	Left Hand	Right Hand
1	(transport_empty left wa nlh)	(transport_empty right wa nrh)
2	(select-grasp-part left body)	(select-grasp-part right button)
3	(transport_loaded left nlh wa)	(transport_loaded right nrh wa)
4	(assemble left right c1)	(assemble left right c1)
5		(release right body)
6	(transport_empty left wa nlh)	(transport_empty right wa nrh)
7	(select-grasp-part left head)	(select-grasp-part right ink_tube)
8	(transport_loaded left nlh wa)	(transport_loaded right nrh wa)
9	(assemble left right c2)	(assemble left right c2)
10	(select-grasp-part left body)	
11	(assemble left right c3)	(assemble left right c3)
12	(transport_empty left wa nlh)	
13	(select-grasp-part left cap)	
14	(transport_loaded left nlh wa)	
15	(assemble left right c4)	(assemble left right c4)

Figure 6.15 Plan of ballpoint pen assembly, with *motion economy rules* 1, 2, 6, 8, 9 and 10

Our plan is very close to its final form. Left hand grasps body, head and cap from NLH. Last rule we are going to turn on is rule number 3. Motion economy rule 3 is "g*rasp a single part instead of parts, jumbled together"*. With the addition of this rule, search control prunes plans where more than two containers are placed at the same location. New plan is shown in Figure 6.16. Last part, cap, is now grasped from NLV. Container of cap is set to NLV. This plan is accepted as a better plan because grasping too many parts from the same location increases the time spent for the selection of the part that the hand searches. When the search and decision before select-grasp gets harder, fatigue of mind and body increases.

In fact, this is the same plan with the version in which we use all the *motion economy rules*. The plan is already formed in a way other rules do not conflict with it. Last two rules, rules 11 and 12, do not improve it further. We say this is the best plan we could find with our model because it is in conformity with all of the principles we have formulated. It may not be the only plan since we did not explore all of the search space. There may be other best plans. These plans may have differences with this plan in the sequence of establishment of contacts or the roles of the hands. Figure 6.17 shows the arrangement of the workspace and sketches of movements for the final plan. Dashed lines show the movement path of right hand.

Table 6.5 lists the plans, which are analyzed for the first sample problem. Plan statistics are also given. It is not possible to find a correlation between speed of the search and set of rules that are on. In general, we expect faster solutions with the increase in the number of the rules applied, since rules will restrict the search space. However searching for a better plan may take longer, since paths, which lead to the goal state and are closer to the initial state, may be pruned by *motion economy rules.*

Time	Left Hand	Right Hand
1	(transport_empty left wa nlh)	(transport_empty right wa nrh)
2	(select-grasp-part left body)	(select-grasp-part right button)
3	(transport_loaded left nlh wa)	(transport_loaded right nrh wa)
4	(assemble left right c1)	(assemble left right c1)
5		(release right body)
6	(transport_empty left wa nlh)	(transport_empty right wa nrh)
7	(select-grasp-part left head)	(select-grasp-part right ink_tube)
8	(transport_loaded left nlh wa)	(transport_loaded right nrh wa)
9	(assemble left right c2)	(assemble left right c2)
10	(select-grasp-part left body)	
11	(assemble left right c3)	(assemble left right c3)
12	(transport_empty left wa nlv)	
13	(select-grasp-part left cap)	
14	(transport_loaded left nlv wa)	
15	(assemble left right c4)	(assemble left right c4)

Figure 6.16 Plan of ballpoint pen assembly, with *motion economy rules* 1, 2, 3, 6, 8, 9 and 10

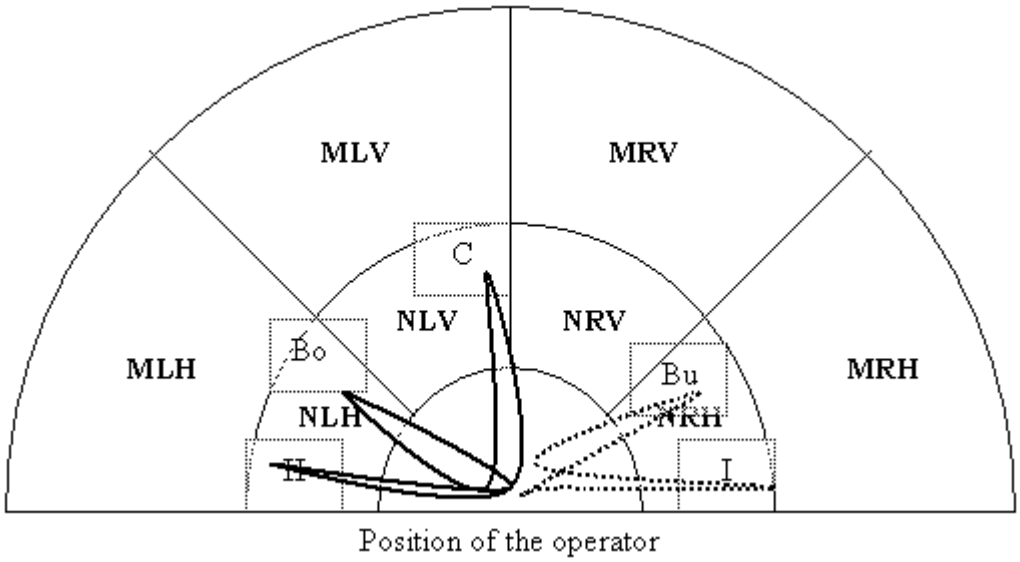

Figure 6.17 Sketch of the workspace for the plan in Figure 6.15

Table 6.5 Plan statistics for sample problem 1

Plan	**Motion economy rules**	**Number of worlds generated**	**Elapsed CPU time (seconds)**
Figure 6.6	None	16315	24.3
Figure 6.8	7	16059	25.7
Figure 6.9	1, 2	40288	149.7
Figure 6.10	1, 2, 6	11540	15.9
Figure 6.11	1, 2, 6, 7, 8	4517	3
Figure 6.13	1, 2, 6, 7, 8, 9	2208	1.2
Figure 6.14	1, 2, 6, 7, 8, 9, 10	16441	33.9
Figure 6.15	All	18705	30

6.2 Sample Problem 2

This sample problem is chosen to analyze tasks in which same parts are used more than once in the product. Figure 6.18 depicts the assembly of two plates, three bolts and three nuts [18]. Contact graph of the assembly is given in Figure 6.19. Precedence relations will not be given in detail. Two plates must be connected first. Bolts must be assembled to the plates before nuts are assembled to bolts. Bolts can be assembled to plates and can be assembled to bolts in any order.

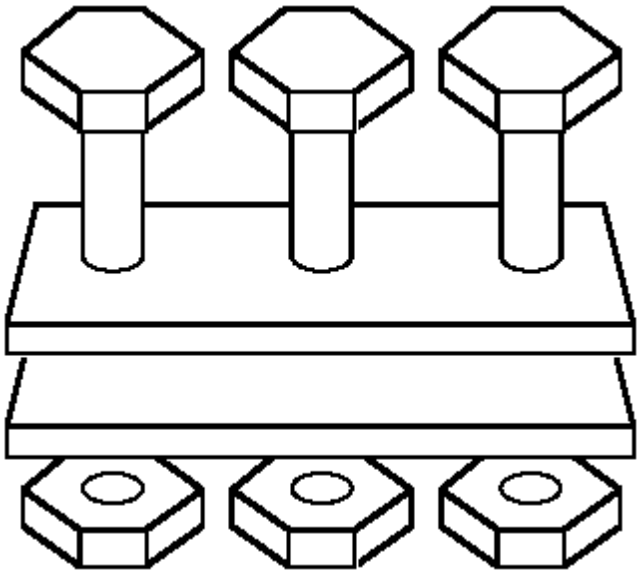

Figure 6.18 Plates, nuts and bolts assembly

Identical plates, nuts and bolts are used. We represent each part as a separate entity in our model. Parts, which have the same characteristics, are assigned to same *type*. *Type* attribute of parts helps us to decide for storage locations of the parts. When parts are grouped under same type, they can be planned to be in one container. As we will see in the plans generated, forcing the parts to be grasped from one location also

changes the way the actions are sequenced. Table 6.6 gives the type information we used for the assembly shown in Figure 6.18.

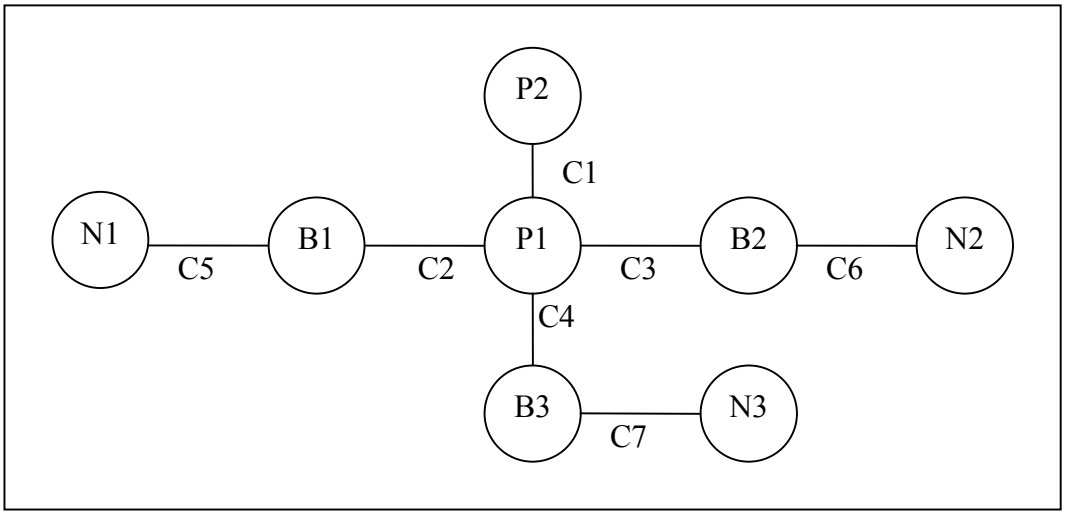

Figure 6.19 Contact graph of plates, nuts and bolts assembly

Identical plates, nuts and bolts are used. We represent each part as a separate entity in our model. Parts, which have the same characteristics, are assigned to same *type*. *Type* attribute of parts helps us to decide for storage locations of the parts. When parts are grouped under same type, they can be planned to be in one container. As we will see in the plans generated, forcing the parts to be grasped from one location also changes the way the actions are sequenced. Table 6.6 gives the type information we used for the assembly shown in Figure 6.18.

All the bolts in the assembly are of the same type. Nuts also have the same type. However, we distinguish between plate1 and plate2, and give each plate a different type. The way we want the operation plan to be has affected our choice for types of plates. Plates must be assembled first. We have seen that, if we force the plates to be in one container, hands cannot simultaneously grasp the plates to assemble them because of *motion economy rule* 8, which forbids hands to move to the opposite side

of the workspace. In fact, *goal-related rule* 2, which is *"if an object has a feasible and disestablished contact which is in the goal, do not release the object until the contact is established"*, makes the problem infeasible in the case where a hand is forced to grasp two parts of a contact. Moreover, since symmetric motions are preferred it is wise to have two containers located in symmetric locations for the same part [3]. This decision can be given for other parts as well. Parts can be divided into groups of types to guide the planner to use symmetric and simultaneous motions. This shows that representation of the product, or the task, is affecting the feasibility and quality of the solutions. Besides, rules may narrow the search space such that there are no feasible solutions, or a few feasible solutions exist and speed of the planner decreases. Tuning of the search control rules by changing the conditions in which they are effective may be necessary prior to different types of tasks to be planned.

Table 6.6 Type attributes of the parts of plates, bolts and nuts assembly

Part	**Type**
plate1	plate1
plate2	plate2
bolt1	Bolts
bolt2	Bolts
bolt3	Bolts
nut1	Nuts
nut2	Nuts
nut3	Nuts

The plan generated by turning only the *motion economy rule* 4 off is given in Figure 6.20. This rule is "p*rovide a fixed place for the things worked with*". Although the parts have the same type left hand grasps one of the bolts from NLV and others from

NLH. Similarly, right hand grasps one of the nuts from NRV and others from NRH. There are two containers for bolts and two containers for nuts on the workspace. These containers are located on the same side of the workspace and are not used for simultaneous actions. This means that, before each select-grasp the operator is confronted with the decision where to get the next bolt or nut. The operator must be relieved from all conscious actions such as these. Time for select-grasp is increased and these pauses affect the rhythm of the operator, decreasing productivity and causing fatigue. Note that in the first four steps, two plates are assembled with symmetric and simultaneous motions. These motions, which would not be planned this way if the plates had the same type value, are favorable in terms of motion economy. The plan given in Figure 6.21 is obtained with the *motion economy rule* 4. All the bolts and all the nuts are grasped from the same location. There exists one container for each type of these parts.

Motion economy rule 5 is *"parts that are used most frequently must be located near the point of use"*. In sample problem 2, bolts and nuts are used three times and plates are used once, since they are not defined to be same type. We have four types of atomic parts at total. Four locations in the normal reach area are at the same distance from the working area. These locations suffice to store all the parts as can be seen in Figures 7.20 and 7.21. 8 containers can be located in normal reach area as there are 4 of them and 2 containers are allowed at a location. Meaning that in order this rule to be effective more than 8 types must be present in a product description. We had to decrease the number of locations on the workspace to observe the effect of this rule in this problem. We discard the locations close to vertical axis and decrease the number of locations to 5. We also have to decrease the maximum containers allowed to 1. Only then, it is observed that with the inclusion of rule 5, containers of plates are placed in maximum reach area, MLH and MRH, and containers of bolts and nuts are placed closer to the working area, NLH and NRH. With the absence of this rule, hands grasp plates from normal reach area and grasp bolts and nuts from maximum reach area.

Time	Left Hand	Right Hand
1	(transport_empty left wa nlh)	(transport_empty right wa nrh)
2	(select-grasp-part left plate1)	(select-grasp-part right plate2)
3	(transport_loaded left nlh wa)	(transport_loaded right nrh wa)
4	(assemble left right c1)	(assemble left right c1)
5	(transport_empty left wa nlh)	
6	(select-grasp-part left bolt1)	
7	(transport_loaded left nlh wa)	
8	(assemble right left c2)	(assemble right left c2)
9		(transport_empty right wa nrh)
10		(select-grasp-part right bolt2)
11		(transport_loaded right nrh wa)
12	(assemble left right c3)	(assemble left right c3)
13	(transport_empty left wa nlh)	
14	(select-grasp-part left bolt3)	
15	(transport_loaded left nlh wa)	
16	(assemble right left c4)	(assemble right left c4)
17		(transport_empty right wa nrv)
18		(select-grasp-part right nut1)
19		(transport_loaded right nrv wa)
20	(assemble left right c5)	(assemble left right c5)
21	(transport_empty left wa nlv)	
22	(select-grasp-part left nut2)	
23	(transport_loaded left nlv wa)	
24	(assemble right left c6)	(assemble right left c6)
25		(transport_empty right wa nrv)
26		(select-grasp-part right nut3)
27		(transport_loaded right nrv wa)
28	(assemble left right c7)	(assemble left right c7)

Figure 6.20 Plan of plates, nuts and bolts assembly, without *motion economy rule* 4

Time	Left Hand	Right Hand
1	(transport_empty left wa nlh)	(transport_empty right wa nrh)
2	(select-grasp-part left plate1)	(select-grasp-part right plate2)
3	(transport_loaded left nlh wa)	(transport_loaded right nrh wa)
4	(assemble left right c1)	(assemble left right c1)
5	(transport_empty left wa nlh)	
6	(select-grasp-part left bolt1)	
7	(transport_loaded left nlh wa)	
8	(assemble right left c2)	(assemble right left c2)
9		(transport_empty right wa nrh)
10		(select-grasp-part right nut1)
11		(transport_loaded right nrh wa)
12	(assemble left right c5)	(assemble left right c5)
13	(transport_empty left wa nlh)	
14	(select-grasp-part left bolt2)	
15	(transport_loaded left nlh wa)	
16	(assemble right left c3)	(assemble right left c3)
17		(transport_empty right wa nrh)
18		(select-grasp-part right nut2)
19		(transport_loaded right nrh wa)
20	(assemble left right c6)	(assemble left right c6)
21	(transport_empty left wa nlh)	
22	(select-grasp-part left bolt3)	
23	(transport_loaded left nlh wa)	
24	(assemble right left c4)	(assemble right left c4)
25		(transport_empty right wa nrh)
26		(select-grasp-part right nut3)
27		(transport_loaded right nrh wa)
28	(assemble left right c7)	(assemble left right c7)

Figure 6.21 Plan of plates, nuts and bolts assembly, with *motion economy rule* 4

6.3 Sample Problem 3

Sample problem 3 is producing a soldered cable. A picture of parts and tools used in soldered cable is given in Figure 6.22. To produce a soldered cable, first, isolation material is cut off the cable by a blade to make an open head of the cable where the wire can be assembled to the pin. Pin is assembled to the wire. Connection of the pin and the wire is strengthened by soldering. Soldering is done by heating and melting the solder into the pin by a soldering iron.

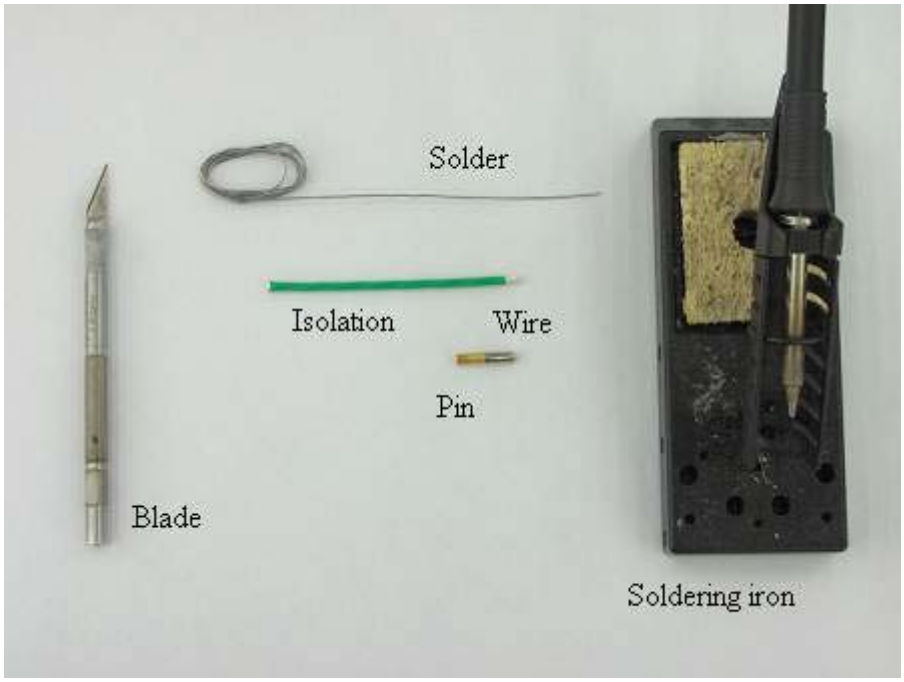

Figure 6.22 Soldered cable

Soldered cable involves disassembling two parts with the help of a tool, which is separating the isolation and wire from each other with a blade. Soldering is an example of assembling two parts with a tool where parts are solder and pin, and tool

is the soldering iron. *Disuse-with-fixture* and *use-with-fixture* must be used in plans to accomplish the goal. Other examples of *disuse-with-fixture* are slicing a bread, using a screwdriver to remove screws and opening a can with a can opener. *Use-with-fixture* may also represent gluing two papers together and hammering a nail.

Soldering iron is represented with four parts, which are isolation (I), wire (W), solder (S) and pin (P). Soldering iron (SI) and blade (B) are the tools. Contact graph of soldered iron is given in Figure 6.23. C1 is established in the initial state. Precedence relations are organized the way the task is described.

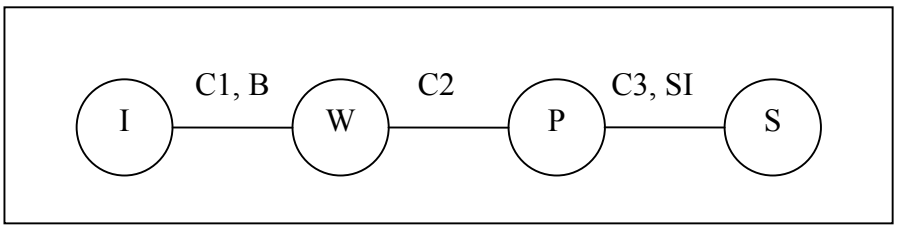

Figure 6.23 Contact graph of soldered cable

This problem lets us analyze motion economy rules 11 and 12, which were not analyzed so far. These rules are *"eliminate hand as the holding device"* and *"distribute work evenly between hands"* respectively. First plan, which is given in Figure 6.24, is generated with all of the *motion economy rules* except rules 10 and 11. Rules 10, which is *"a constantly extended position of the arms should be avoided"*, and 11, activate when there is an action sequence, in which one hand is stretched aside from the working area or holds an object, while the other hand is busy. Holding is only allowed when the hand is at the working area. Number of actions, which the other hand can make during this period, can be adjusted. In experiments, we allow only one action. Since these rules interact, we generated the first plan with the absence of these two. Next, we will turn on rules one by one starting with rule 11.

At the beginning of the plan in Figure 6.24, isolation-wire assembly, which is in fact the raw cable, and blade are grasped from symmetric locations. Grasping of the cable is realized as *(select-grasp-part left isolation)* as if only isolation is grasped. However, since C1 is established initially, wire, which is connected to isolation with C1, is also grasped with isolation. As explained in Chapter 3, although actions instantiate with atomic parts, they affect the assembly, to which parts belong, if parts are not atomic. Right hand uses blade to disestablish the contact between isolation and wire, while left hand holds the assembly. After this operation left hand holds the isolation, which is atomic now, and the wire is in WA. Right hand is holding the blade. Hands release objects, which they hold, to their initial locations. Then left hand moves to WA and grasps wire, right hand grasps pin from NRH, and returns to WA. Container of the pin is set to NRH. Wire and pin are assembled. After *assemble*, right hand positions the wire-pin assembly at WA. A fixture is located at WA as a consequence of positioning. Left hand grasps solder and right hand grasps soldering iron to establish the last contact. Last action of the plan is *use-with-fixture*. Right hand uses soldering iron, while left hand holds the solder and the pin is positioned at WA. Solder and pin are assembled together with the help of soldering iron.

Time	Left Hand	Right Hand
1	(transport_empty left wa nlh)	(transport_empty right wa nrh)
2	(select-grasp-part left isolation)	(select-grasp-tool right blade)
3	(transport_loaded left nlh wa)	(transport_loaded right nrh wa)
4	(disuse_wf right left c1)	(disuse_wf right left c1)
5	(transport_loaded left wa nlh)	(transport_loaded right wa nrh)
6	(release left isolation)	(release right blade)
7	(transport_empty left nlh wa)	
8	(select-grasp-part left wire)	(select-grasp-part right pin)
9		(transport_loaded right nrh wa)
10	(assemble left right c2)	(assemble left right c2)
11	(transport_empty left wa nlv)	
12	(select-grasp-part left solder)	(position right pin)
13		(transport_empty right wa nrv)
14		(select-grasp-tool right sol_iron)
15	(transport_loaded left nlv wa)	(transport_loaded right nrv wa)
16	(use_wf right left c3)	(use_wf right left c3)

Figure 6.24 Plan of soldered cable, without *motion economy rules* 10 and 11

The plan given in Figure 6.24 conflicts with rule 11 because left hand holds solder for 3 steps after grasping it from NLV. Meanwhile right hand reaches for soldering iron to NRV and grasps it. This is not desired, since left hand could be utilized elsewhere at the period it holds the solder, motions of the arms are not simultaneous and in rhythm, and left arm is stretched out for a while which increases its fatigue. Inclusion of motion economy rule 11 to the search control rules arranges a better sequence of the actions in this period, solving this undesired situation. Generated plan is given in Figure 6.25. Now left hand waits at WA for right hand while it positions the pin. Then hands simultaneously move to symmetric locations, NRV and NLV, grasp solder and soldering iron and carry them to WA.

Action sequence between time 4 and 10, in the plan in Figure 6.25 is inconsistent with rule 10. Right hand is in an extended position at NRH after it releases the blade until it grasps the pin. With the inclusion of rule 10, right hand releases blade to WA and grasps the wire, which was at WA after *disuse*. Left hand grasps pin from NRV, which was grasped by right hand from NRH before. The plan is given in Figure 6.26. This plan is generated by employing all of the *motion economy rules*. We prefer this plan to all other feasible plans, two of which were given above, since it conforms all of the rules.

This plan is also consistent with *motion economy rule* 12, which is *"distribute work evenly between hands"*, since this rule is active. This rule does not permit plans, in which a hand performs five or more actions than the other. Utilizing both hands at maximum is always a good practice. This shortens the duration of the tasks and prevents tiring one hand while the other is idle most of the time. Figure 6.27 shows the plan generated when rule 12 is turned off. We see that one but all of the *select-grasps* are done with the left hand. Right hand is poorly utilized. Plan lasts longer and it lacks rhythm. Left hand has a very high fatigue compared to right hand.

Time	Left Hand	Right Hand
1	(transport_empty left wa nlh)	(transport_empty right wa nrh)
2	(select-grasp-part left isolation)	(select-grasp-tool right blade)
3	(transport_loaded left nlh wa)	(transport_loaded right nrh wa)
4	(disuse_wf right left c1)	(disuse_wf right left c1)
5	(transport_loaded left wa nlh)	(transport_loaded right wa nrh)
6	(release left isolation)	(release right blade)
7	(transport_empty left nlh wa)	
8	(select-grasp-part left wire)	(select-grasp-part right pin)
9		(transport_loaded right nrh wa)
10	(assemble left right c2)	(assemble left right c2)
11		(position right pin)
12	(transport_empty left wa nlv)	(transport_empty right wa nrv)
13	(select-grasp-part left solder)	(select-grasp-tool right sol_iron)
14	(transport_loaded left nlv wa)	(transport_loaded right nrv wa)
15	(use_wf right left c3)	(use_wf right left c3)

Figure 6.25 Plan of soldered cable, without *motion economy rule* 11

Time	Left Hand	Right Hand
1	(transport_empty left wa nlh)	(transport_empty right wa nrh)
2	(select-grasp-part left isolation)	(select-grasp-tool right blade)
3	(transport_loaded left nlh wa)	(transport_loaded right nrh wa)
4	(disuse_wf right left c1)	(disuse_wf right left c1)
5	(transport_loaded left wa nlh)	
6	(release left isolation)	(release right blade)
7	(transport_empty left nlh nlv)	
8	(select-grasp-part left pin)	(select-grasp-part right wire)
9	(transport_loaded left nlv wa)	
10	(assemble left right c2)	(assemble left right c2)
11		(position right pin)
12	(transport_empty left wa nlh)	(transport_empty right wa nrh)
13	(select-grasp-part left solder)	(select-grasp-tool right sol_iron)
14	(transport_loaded left nlh wa)	(transport_loaded right nrh wa)
15	(use_wf right left c3)	(use_wf right left c3)

Figure 6.26 Plan for soldered cable, with all of the *motion economy rules*

Time	Left Hand	Right Hand
1	(transport_empty left wa nlh)	(transport_empty right wa nrh)
2	(select-grasp-part left isolation)	(select-grasp-tool right blade)
3	(transport_loaded left nlh wa)	(transport_loaded right nrh wa)
4	(disuse_wf right left c1)	(disuse_wf right left c1)
5	(transport_loaded left wa nlh)	
6	(release left isolation)	(release right blade)
7	(transport_empty left nlh nlv)	
8	(select-grasp-part left pin)	(select-grasp-part right wire)
9	(transport_loaded left nlv wa)	
10	(assemble left right c2)	(assemble left right c2)
11	(transport_empty left wa nlh)	
12	(select-grasp-part left solder)	
13	(transport_loaded left nlh wa)	
14	(position left solder)	
15	(transport_empty left wa nlh)	
16	(select-grasp-tool left sol_iron)	
17	(transport_loaded left nlh wa)	
18	(use_wf left right c3)	(use_wf left right c3)

Figure 6.27 Plan of soldered iron, without *motion economy rule* 12

6.4 Sample Problem 4

Sample Problem 4 is to prepare a brochure. The brochure consists of three sheets. Three sheets are stapled from the middle of the sheets to make a six pages brochure. The task also involves placing the brochure inside of an envelope. Figure 6.28 shows the brochure and envelope for sample problem 4. This problem is an example where a tool is used on a part not to assemble it to another, but to change the nature of the part. After a pile is made from the sheets, stapler is used on the pile to turn it into a six pages brochure. Plans must be composed by using *use-self-contact*. There are many interpretations of a self-contact, such as cleaning a part with a brush, painting, drilling a hole in something and writing.

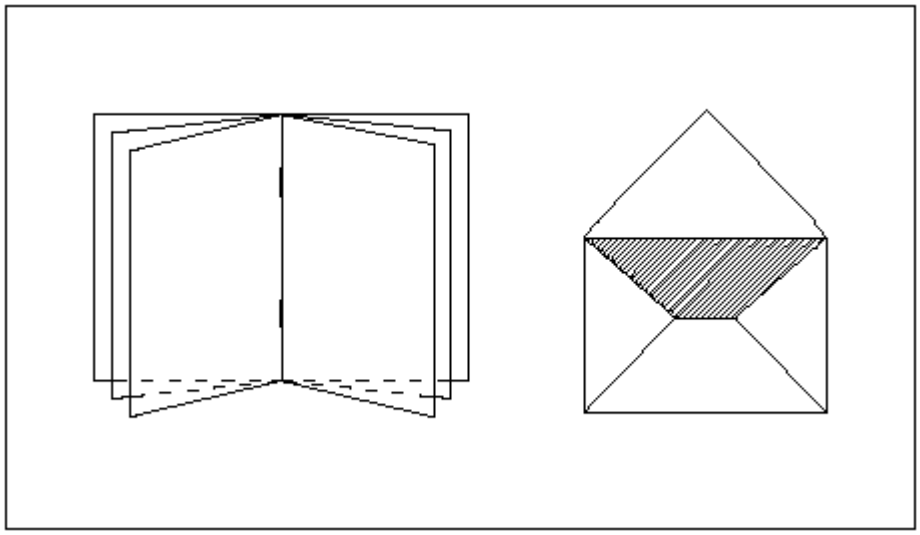

Figure 6.28 Brochure with three sheets and envelope

There are four parts, sheet1 (S1), sheet2 (S2), sheet3 (S3) and envelope (E), and a tool, stapler (S), in the representation of this task. Contact graph of the brochure is given in Figure 6.29. Stapling operation is represented by an arc from and onto one

of the sheets. The contacts between the sheets must be established first. Stapling can only be done when the sheets are formed a pile. Choice of the sheet to represent stapling does not make a difference, as precedence rules are determined to make the stapling after all sheets are assembled. Brochure is prepared with two staples, so there are two self-contacts. In the same manner, envelope is shown to connect only one of the sheets.

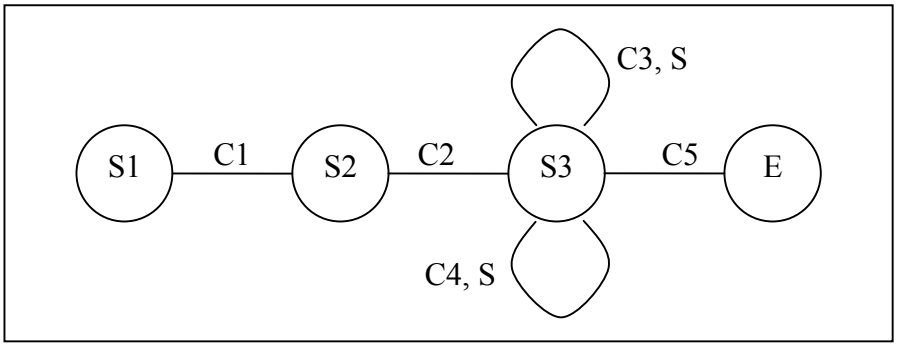

Figure 6.29 Contact graph of brochure

Unlike the problems discussed so far, we want the final product to be at a specific location. Goal states has *(at envelope NLV)* predicate. We also want hands to be at WA when the plan is completed. Since all the contacts must be established at goal state, envelope and all the parts that are connected to it, will be placed at NLV after all the contact establishment operations are completed. Think of this final location as the storage location for finished goods. The plan represents one cycle of a production run to produce a certain number of brochures.

Figure 6.30 shows the plan generated for preparing the brochure. All of the *motion economy rules* are used. Containers of sheet1 and sheet3 are at NLH. Container of sheet2 is at NRH. Stapler is located at NRH. Container of the envelope is located at NRV. Sheets 1 and 2 are assembled at the beginning of the plan with simultaneous

motions of the hands. Sheet3 is carried to WA by the left hand. After assembling all the three sheets, right hand takes the stapler from NRH and uses it on the pile of sheets, which is held by the left hand, twice. Right hand grasps the envelope and brochure is put into the envelope. Finally, left hand carries the enclosed brochure to its goal location, NLV, and returns back to WA.

When no *motion economy rules* guide the planner, the plan in Figure 6.31 is generated. Contacts are established in the same order in this plan. However, all of the parts and the stapler are stored at maximum reach area, two-handed work is not performed at WA and there is no definite place for two-handed work. Hands wander around the workspace and make unnecessary transports. Left hand carries out most of the grasping and releasing, while right hand spends most of the time idle, extended away from WA and holding a part. Only left-hand side of the workspace is used, therefore simultaneous motions are not symmetric. At times 15 and 17 left-hand moves back and forth between two axes of the horizontal plane. All of the above actions and sequences of actions that we point out are inconsistent with one or more *motion economy rules*. These facts make second plan we have shown worse than the one we generated with the help of *motion economy rules*. In the first plan, rules have avoided all of these situations, which require more effort and cause more fatigue as we have explained in Chapter 5 and shown in previous sample problems. Without *motion economy rules*, it took TLPLAN 180.5 seconds to arrive at a solution after generating 59859 worlds. With *motion economy rules*, plan is found in 5.5 seconds and the planner generated 4818 worlds. For this problem, we can conclude that the better solution is cheaper.

Time	Left Hand	Right Hand
1	(transport_empty left wa nlh)	(transport_empty right wa nrh)
2	(select-grasp-part left sheet1)	(select-grasp-part right sheet2)
3	(transport_loaded left nlh wa)	(transport_loaded right nrh wa)
4	(assemble left right c1)	(assemble left right c1)
5	(transport_empty left wa nlh)	
6	(select-grasp-part left sheet3)	
7	(transport_loaded left nlh wa)	
8	(assemble right left c2)	(assemble right left c2)
9		(transport_empty right wa nrh)
10		(select-grasp-tool right stapler)
11		(transport_loaded right nrh wa)
12	(use_sc right left c3)	(use_sc right left c3)
13	(use_sc right left c4)	(use_sc right left c4)
14		(release right stapler)
15		(transport_empty right wa nrv)
16		(select-grasp-part right envelope)
17		(transport_loaded right nrv wa)
18	(assemble right left c5)	(assemble right left c5)
19	(transport_loaded left wa nlv)	
20	(release left envelope)	
21	(transport_empty left nlv wa)	

Figure 6.30 Plan of preparing the brochure, with *motion economy rules*

Time	Left Hand	Right Hand
1	(transport_empty left wa mlh)	(transport_empty right wa mlh)
2	(select-grasp-part left sheet1)	(select-grasp-part right sheet2)
3	(transport_loaded left mlh mlv)	(transport_loaded right mlh mlv)
4	(assemble left right c1)	(assemble left right c1)
5	(transport_empty left mlv mlh)	(transport_loaded right mlv mlh)
6	(select-grasp-part left sheet3)	
7	(assemble left right c2)	(assemble left right c2)
8	(transport_empty left mlh mlv)	(transport_loaded right mlh mlv)
9	(select-grasp-tool left stapler)	
10	(use_sc left right c3)	(use_sc left right c3)
11	(transport_loaded left mlv mlh)	(transport_loaded right mlv mlh)
12	(use_sc left right c4)	(use_sc left right c4)
13	(transport_loaded left mlh mlv)	(transport_loaded right mlh mlv)
14	(release left stapler)	
15	(transport_empty left mlv mlh)	
16	(select-grasp-part left envelope)	
17	(transport_loaded left mlh mlv)	
18	(assemble left right c5)	(assemble left right c5)
19	(transport_empty left mlv wa)	(transport_loaded right mlv nlv)
20		(release right envelope)
21		(transport_empty right nlv wa)

Figure 6.31 Plan of preparing the brochure, without *motion economy rules*

6.5 Sample Problem 5

This problem is chosen to show the type of problems in which the reverse of an ordinary assembling problem is carried out. The task is to prepare a table with a fast food meal kit. Figure 6.32 shows a typical meal kit, which includes a fork (F), a knife (K), a napkin (N) and a bag of salt (S). These materials are enclosed in a nylon bag (B). The graph of contacts for the meal kit is given in Figure 6.33.

Figure 6.32 The fast food meal kit

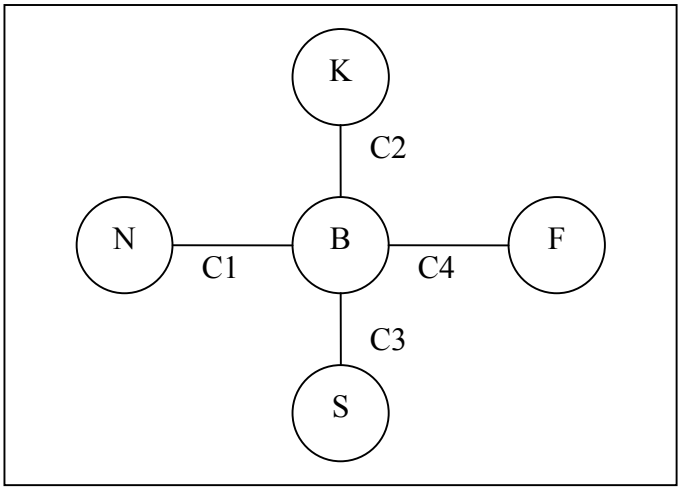

Figure 6.33 Graph of contacts of meal kit

At the initial state, all of the contacts are established. The assembly is at a certain location in the workspace. Goal state defines the desired final locations of the parts. Figure 6.34 shows the desired arrangement of the workspace for meal kit problem, which in this case stands for a meal table. To achieve this final arrangement, contacts must be disestablished in the plan with *disassemble*. In our problem, the initial location of the meal kit assembly is MLV, and final locations of fork, knife, napkin, salt and bag are NLH, NRH, NRH, NRV and NLV respectively.

Figure 6.34 Desired arrangement of the meal kit problem

In contrast to making a product from its atomic parts, deciding the initial locations of the containers of parts is not a part of the problem in this kind of tasks. Parts cannot be grasped from any location since they exist initially on the workspace. Arrangement of the workspace must be decided in advance and must be given to the planner.

The plan generated with all of the motion economy rules is given in Figure 6.35. Bag of meal kit is grasped from MLV and brought to WA by the left hand. Napkin and knife are disassembled and carried to their final locations by the right hand. Then, fork is disassembled from the bag and released to WA. Final *disassemble* separates the bag and salt from each other. Hands simultaneously carry bag and salt to their final locations, which are symmetric on the workspace. Finally, fork, which was left at WA, is grasped by the left hand, and is carried to NLH.

TLPLAN was unable to solve this problem within 8 hours when the *motion economy rules* were not employed. It appears that *goal-related rules* are not helpful enough to fasten search for this kind of problems. Although *goal-related rules* do not prune feasible paths for disassembling tasks, as the plan in Figure 6.35 is generated while they are active, they seem to be oriented to work better with assembling problems.

We have presented five sample problems in chapter 6. Representation of the products is analyzed at the beginning of each sample problem. Sample problems are chosen to show a different type of tasks, which can be solved by our model. Effects of search control rules are analyzed by solving the problems with different combinations of the rules.

Time	Left Hand	Right Hand
1	(transport_empty left wa mlv)	
2	(select-grasp-part left bag)	
3	(transport_loaded left mlv wa)	
4	(disassemble left right c1)	(disassemble left right c1)
5		(transport_loaded right wa nrh)
6		(release right napkin)
7		(transport_empty right nrh wa)
8	(disassemble left right c2)	(disassemble left right c2)
9		(transport_loaded right wa nrh)
10		(release right knife)
11		(transport_empty right nrh wa)
12	(disassemble left right c4)	(disassemble left right c4)
13		(release right fork)
14	(disassemble left right c3)	(disassemble left right c3)
15	(transport_loaded left wa nlv)	(transport_loaded right wa nrv)
16	(release left bag)	(release right salt)
17	(transport_empty left nlv wa)	(transport_empty right nrv wa)
18	(select-grasp-part left fork)	
19	(transport_loaded left wa nlh)	

Figure 6.35 Plan of meal kit, with *motion economy rules*

CHAPTER 7

POSSIBLE EXTENSIONS AND FUTURE WORK

Implementation of the domain, which was employed in the experimental runs, does not include some features that were designed. This is mainly because observing the changes in these design parameters was not the main motivation behind this study. These features were not also explained in the representation divisions of the thesis. It is better to exclude these features completely in the thesis than to present and do not analyze them in experimentation, even though their effects on the experiments would be harmless to our main motivation. The only harm they would cause to the experimental runs is to increase the complexity of the problem. On the other hand, this may decrease the run times, which is indeed favorable, for these additions narrow the search space by imposing new constraints on the problem. Number of applicable actions at a state will fall. However, they would crowd the model and make it harder for a stranger to understand it. In both the representation sections and the experimental results, they would distract the readers' attention, although they offer no additional outcomes. First two headlines of this chapter describe excluded features of the design. Following other sections, describe what may possible extensions be.

Preference of hands: Humans cannot use both of their hands with the same precision, automaticty and power. From the early childhood, one of the hands is frequently used, thus it is trained to perform operations, which require relatively much precision or force than others. Humans are classified with the choice of this hand. For all we know, these are called left-handedness and right-handedness. The current model of motion economy domain does not contain this knowledge. It is assumed that hands are identical, the operator has no preference of any them, and all sorts of operations can be performed by any of them.

The representation of left or right-handedness can easily be done by defining this property in the hand object and testing this condition in the actions. For operations, which require force and which require precision, the property of preference can be split into two, although in reality same hand is trained and chosen for these two kinds of operations. These attributes, say *preference-for–precision* and *preference-for-force*, may take Boolean values and availability of hand for some operations can be turned on and off. They might take numeric values to represent ranges of operation that a hand is effective. This way, some operations can be done with any of the two hands while some can only be done with one of them.

A general idea of force or precision requirement for therbligs is not considered. Therbligs are generic descriptions of operations, which may have various conditions. Assemble therblig may stand for different parts of different sizes and shapes and different contacts of different types. How much force or precision is required by an operation, must be given in the product specification, for the contact, part and tool specifies these criteria. Two additional properties, one for *force-requirement* and one for *precision-requirement*, may be defined for each of these entities. In the preconditions of the actions, values of these attributes will be compared with *preference-for-force* and *preference-for-precision* values of the hands. When these are Boolean values, if preference and requirement values are true then the operation is feasible. When numeric values are used, preference values must be greater than or equal to requirement values.

List of conditions, which may appear and corresponding checks for numeric values are given in the Table 7.1. Thin and flat materials usually require precision grasps. Transporting heavy parts or tools may be much more tiring for the less trained hand of the worker. Preferred hand may be forced to carry these parts. Similarly, although no transportation is involved, holding these objects for a long time may result in an undesirable fatigue of the hand. The preferred hand must be the operator hand in

contact establishment operations, especially where a tool is needed. Some of these operations require precision, such as writing, some of these operations, such as hammering, drilling, require force, and some require both. Specifying the value of requirement in contacts is satisfactory, for this value will be an overall judgment about the operation considering the parts and the tools involved.

Table 7.1 Conditions regarding the preference of hands

Operation	**Condition**
Transporting an object	hand(preference-for-force) ≥ part(force-requirement) hand(preference-for-force) ≥ tool(force-requirement)
Keeping hold of an object for a period of time	hand(preference-for-force) ≥ part(force-requirement) hand(preference-for-force) ≥ tool(force-requirement)
Contact establishment (assemble, disassemble, use, disuse)	hand(preference-for-force) ≥ contact(force-requirement) hand(preference-for-precision) ≥ contact(precision-requirement)
Grasping an object	hand(preference-for-precision) ≥ part(precision-requirement) hand(preference-for-precision) ≥ tool(precision-requirement)

Hands as resources with finite capacity: In the preconditions of *select-grasp*, it is stated that the hand must be empty. A hand is empty if it holds nothing. When the hand grasps a part, an assembly or a tool, it is not empty anymore and it can not grasp another object unless it releases the one it holds. This process models hands as unit capacity storages. However, a hand may grasp another material when it is already holding one or even more materials. Moreover, at times when a hand can not grasp another object due to an object it holds, the number of objects it holds can still be increased if other hand releases objects onto it. Recall when counting coins we

release the ones we counted to the other hand. It is hard to grasp more coins when you are holding some, but holding much more of them is easy, if someone puts the coins on your hand for you.

Hands can be modeled as multi-capacity resources. Each hand must be assigned a capacity attribute. Tools and parts will have capacity-consumption values, which define how much capacity is required to carry and store them, in product specification. As long as total capacity consumption of the objects do not exceed hand's capacity, it may grasp and transport more than one object.

Variable fatigue: In the current implementation of the domain, every action increases the fatigue value of the hands by one. The type and parameters of the actions is not considered. However actions, such as *assemble*, *transport* and *select-grasp*, have different characteristics. They will not tire hands with same amount. Even for the same action, parameters of actions will directly affect the resulting increase in fatigue. Fatigue caused by transporting a part between two locations depends on the characteristics of the part and the locations, while the fatigue caused by assembling a contact depends on the type of the contact and parts. A function, which will take attributes of the actions as input and return the fatigue value, must be defined. However, this requires analysis of the effects of each parameter to fatigue. New features of contacts, such as type, insertion or planar, and parts, such as weight and size, can be defined to use for computation of fatigue. Fatigue value is used in the twelfth motion economy rule. This rule controls the fairness of distribution of actions to the hands by monitoring the number of actions hands perform. With the improvement of the measure of fatigue, the decision variable of this rule will no longer be number of actions. This rule can be more effective with sound fatigue computation.

Variable durations: In representation of motion economy domain, it is assumed that actions last unit time despite their differences. In fact, durations of actions must vary depending on the types and parameters of actions. TLPLAN supports actions with

variable durations. Each action can be assigned a function, which computes the duration of the action with the values of its present instantiation.

An alternative approach is to maintain a database having predetermined times for actions with certain values of parameters. Standard times must be developed for grasping parts' having certain thickness and weight, transportation times for certain distances and such. The research of predetermined time systems is a field of work and time study.

A predetermined time system consists of a set of time data and a systematic procedure which analyses and subdivides any manual operation of human task into motions, body movements, or other elements or human performance, and assigns to each the appropriate value [3] .

Without observation and timing of the process, the analyst can work on timed charts in the design phase of the process. One can determine in advance, how long it will take to perform an operation in the shop, merely by examining a drawing of the work place layout and the description of a method. Similarly, an accurate evaluation can be made of several different work method designs or different tool designs. Two of well-known and widely used predetermined time systems are the work-factor system and methods-time-measurement (MTM) system. An MTM table for grasp is given in Table 7.2 [3]. Motion economy domain can be integrated with a predetermined time system for a particular industry or family of products. Planner can work on restricted group of operations with such system, since developing and maintaining standard times for every operation, the model can represent, is impossible. Application of such a system must be conducted by specialists in the field [3].

Table 7.2 MTM table for grasp

Type of Grasp	Case	Time TMU	Description	
Pick-up	1A	2.0	Any size object by itself, easily grasped	
	1B	3.5	Object very small or lying close against a flat surface	
	1C1	7.3	Diameter larger than ½"	Interference with Grasp on bottom and one side of nearly cylindrical object.
	1C2	8.7	Diameter ¼" to ½"	
	1C3	10.8	Diameter less than ¼"	
Regrasp	2	5.6	Change grasp without relinquishing control	
Transfer	3	5.6	Control transferred from one hand to the other.	
Select	4A	7.3	Larger than 1" x 1" x 1"	Object jumbled with other objects so that search and select occur.
	4B	9.1	¼" x ¼" x ⅛" to 1" x 1" x 1"	
	4C	12.9	Smaller than ¼" x ¼" x ¼"	
Contact	5	0	Contact, Sliding, or Hook Grasp.	

Either by functions of durations or by predetermined time systems, use of variable durations bring a more complex notion of concurrency. Simultaneous actions of left and right hand will not necessarily start and end at the same time, some portion of them will overlap and some will fall before or after the other action. Generating plans conforming the principles of motion economy will become more important but more difficult than before.

Bimanual coordination: Humans are incapable of coordinating the operations of left and right hands to the same extent. There are operations, or couples of operations, which hands cannot simultaneous carry out. This is an issue studied by cognitive scientist, which is entitled as bimanual coordination. The sets of rules in the current implementation do no account for bimanual coordination. Plans may force left and right hand to perform certain series of actions, which may not be

coordinated well. If principles of bimanual coordination can be represented as temporal formulas such as principles of motion economy, a fourth set of search control rules may be developed. However, these studies are not oriented to generate rules or other forms of instructions that can be transformed into rules, which may govern automatic generation of manual operation plans. Usually experiments are conducted, which analyze the performances of the subjects in repetitive tasks. Formulation of bimanual coordination rules may be done with the supervision of a cognitive scientist who works in this area. This may be an addition of great value to the current model of the domain.

Meta-rules: We have seen that the planner tends to generate plans quicker for tasks, in which a product is formed from its atomic parts. This is because *goal-related rules* narrow the search space better in these problems. When one wants to increase these rules' efficiency for all kinds of tasks, contrasting conditions of tasks having reverse flow of actions must be considered. Otherwise, planner may find no plan in some tasks even if there are feasible solutions. For instance, additional rules, which are specific for disassembling tasks, may be developed to stop explosion of the search space in these problems. However, for assembly tasks, these rules may forbid generation of feasible plans. We have developed g*oal-related rules* such that they never forbid feasible plans for any type of task. If these rules are examined, it will be seen that type of the goal is always checked for the application of the rules. There is trade-off between fastening the search and not forbidding the generation of feasible plans for all types of tasks. To avoid this problem, rather than developing search control rules for whole class of manual operations, different sets of rules for tasks having different characteristics may be constructed and activation of these rule sets may be controlled by meta-rules. Meta-rules can decide the use of a rule set by checking the initial and goal states of a problem. When there is an established assembly in the initial state, and parts are atomic in the goal state, special rules to fasten planning for disassembling will be activated. When contacts are defined to have tools, special rules concerning the use of tools will be activated. It is clear that

one extreme case of specializing search control rules is to develop rule sets for specific domains.

Two extensions to our model, which was given in previous chapters, are explained at the beginning of the chapter. One of these extensions account for preference of one of the hands for some operations. The other extension is to model hands as resources with finite capacity. We have discussed other possible extensions as future work in the rest of the chapter.

CHAPTER 8

CONCLUSION

Main motivation behind this study was to employ motion economy expertise to evaluate and improve plans for manual operations. This domain was never a subject of any AI study before. No representation scheme for operation plans was present. We developed and formalized a model for representing actions, environment and products of the domain. We formulated the problem as an AI planning problem in the scope of this study. We have exploited new approaches in AI planning to plan with actions with resources, actions with durations and concurrency. In general, representation of a product and definition of the goal determines the tasks to be done. We represent products with a graph of contacts, which is developed for assembly sequence planning research. Due to the diversity of the operations in our domain, we extended the characteristics of the graph of contacts. We increased the expressivity of graph of contacts for operations, which require tools. We also represented the workspace and the properties defining the current world for the first time, to conduct motion economy research with AI tools.

Evaluating and selecting the plans with the help of motion economy is performed during plan generation. We interpreted and represented expertise of motion economy specialists as search control rules by using temporal formulas. We selected 12 core principles of motion economy, to form the set of rules, which decides whether a plan is “good” or not. In addition, a notion of common sense of manual operations is conceptualized during the course of this study. A set of common sense search control rules are devised and formulated to avoid plans having sequences of actions, which are considered unreasonable by an ordinary person.

Sample problems that we have formulated show the expressiveness of the representation scheme. Many sorts of tasks can be represented and solved by planner.

Solutions contain extra features, which are not supported by standard forward chaining planners, such as concurrency and resources. Plans satisfy the requirements of the task specifications well.

Effectiveness of motion economy rules are shown by comparing the plans generated with and without the help of the rules. Rules are examined one by one to show the effects of each. Sample problems show not only the rules are well formulated but also the rules guide the search to produce “good” plans. Justification of why a plan is evaluated to be good is given by describing the improvement, motion economy rules make, compared to the plans, which are obtained only by applying common sense search control. It is observed that motion economy offers a great deal of improvement to these raw plans, which seem reasonable for an ordinary person.

REFERENCES

[1] Abrantes M. J., Hill S. D.: Computer Aided Planning of Mechanical Assembly Sequences. Technical Report, Department of Robotics and Digital Technology, Monash University (1995)

[2] Gilbreth F. B., L. M.: Applied Motion Study. Sturgis & Walton Co. (1917)

[3] Barnes R. M.: Motion and Time Study, Design and Measurement of Work. John Wiley & Sons (1958)

[4] Bourjault A.: Contribution a une Approche Methodologique de l'Assemblage Automatise: Eloboration Automatique des Sequencres Operatories. PhD Thesis, Universite de Franche-Comte, Besancon, France (1984)

[5] De Fazio T. L., Whitney D. E.: Simplified Generation of all Mechanical Assembly Sequences. IEEE Journal of Robotics and Automation (1987) 640-658

[6] De Fazio T., Whitney D.: Generation and Consideration of all Assembly Sequences for Assembly System Design. International Workshop on Engineering Design and Manufacturing Management (1988) 43-56

[7] Homem De Mello L., Sanderson A.: A Correct and Complete Algorithm for the Generation of Mechanical Assembly Sequences. IEEE Transactions on Robotics and Automation (1990)

[8] Russel S., Norvig P.: Artificial Intelligence, a Modern Approach. Prentice-Hall, 1995.

[9] Bacchus F.: TLPlan. http://www.cs.toronto.edu/~fbacchus/tlplan.html (2001)

[10] Bacchus F., Ady M.: Planning with Resources and Concurrency, A Forward Chaining Approach (2001)

[11] Bacchus F., Kabanza F.: Using Temporal Logics to Express Search Control Knowledge for Planning (2000)

[12] Huth M., Ryan M.: Logic in Computer Science: Modeling and Reasoning about Systems. Cambridge University Press (2000)

[13] Gazdar G., Klein E. H., Pullum G. K., Sag I. A.: Generalized Phrase Structure Grammar. Oxford: Blackwell, and Cambridge Ma. Harvard University Press (1985)

[14] Mundel M. E.: Motion and Time Study, Improving Productivity, Sixth Edition. Prentice Hall (1985)

[15] Karger D. W., Bayha F. H.: Engineered Work Measurement. Industrial Press Inc. (1977)

[16] Niebel B. W.: Motion and Time Study, Eighth Edition. Irwin (1988)

[17] Grandjean E.: Fitting the Task to the Man, An Ergonomic Approach. Taylor & Francis Ltd. (1969)

[18] Wolter J. D.: On the Automatic Generation of Assembly Plans (1989)

APPENDIX A

PREDICATES USED IN THE WORLD MODEL

Table A.1 Predicates used in the world model

Predicate	**Description**
(isloc L)	L is a location.
(pos P)	Part P is positioned.
(established C)	Contact C is established.
(hand H)	H is a hand.
(holds H X)	Hands H holds part / tool X.
(at X L)	Object X is at location L.
(container C L)	There is a container of P at L.
(fixture F L)	There is a fixture of P at L.
(contact C)	C is a contact.
(part P)	P is an atomic part of the product.
(ctool C T)	Contact C requires tool T to be established.
(cbei C1 C2)	Contact C1 cannot be established if contact C2 is established.
(cbein C1 C2)	Contact C1 cannot be established if contact C2 is NOT established.
(cpart C P)	Part P has contact C.
(tool T)	T is a tool.

APPENDIX B

MODEL IN TLPLAN INPUT SYNTAX

```
(declare-described-symbols
 (predicate isloc 1)
 (predicate pos 1)
 (predicate contact 1)
 (predicate cpart 2)
 (predicate established 1)
 (predicate cbei 2)
 (predicate cbein 2)
 (predicate ctool 2)
 (predicate part 1)
 (predicate hand 1)
 (predicate holds 2)
 (predicate tool 1)
 (predicate container 2)
 (predicate fixture 2)
 (predicate at 2)
 (function fatigue 1)
 (predicate will 3)
 (predicate busy 1)
 (function tr-made 1)
 (function rel-actions 1)
 (function type 1)
 (function side 1)
 (function dist 1)
 (function axis 1)
 (function cons-tr 0)
 (function steps-to-extend 0)
 (function steps-to-hold 0)
 (function rel-fat 0)
 (function max-cont 0)
)

(declare-defined-symbols
 (predicate feasible 1)
 (predicate empty 1)
 (predicate transport-all-holding 3)
 (predicate check-holding-contact 3)
 (predicate check-sim-grasp 1)
 (predicate check-sim-grasp2 3)
 (predicate connected 2)
 (predicate connected2 4)
 (predicate grasp-all-contact 2)
 (predicate grasp-all-contact2 4)
 (predicate release-all-contact 2)
 (predicate release-all-contact2 4)
 (predicate isatomic 1)
 (predicate init-action-counter 1)
 (predicate init-action-counter2 0)
 (predicate contact-est 2)
 (function held 3)
 (function numconts 1)
 (function num-of-type 1)
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;
;;;;;;;Defined functions and predicates
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;

(def-defined-predicate (contact-est ?h ?c)
 (or
  (will assemble1 ?h ?c)
  (will assemble2 ?h ?c)
  (will disassemble1 ?h ?c)
  (will disassemble2 ?h ?c)
  (will use_sc1 ?h ?c)
  (will use_sc2 ?h ?c)
  (will use_wf1 ?h ?c)
  (will use_wf2 ?h ?c)
  (will disuse_wf1 ?h ?c)
  (will disuse_wf2 ?h ?c)
 )
)

(def-defined-function (num-of-type ?x)
 (local-vars ?i)
 (and
  (:= ?i 0)
  (forall (?p) (part ?p)
   (implies
    (= (type ?p) ?x)
    (:= ?i (+ ?i 1))
   )
  )
  (:= num-of-type ?i)
 )
)

(def-defined-predicate (init-action-counter ?h)
 (implies
  (exists (?h2) (hand ?h2)
   (and
    (not (= ?h ?h2))
    (busy ?h2)
   )
  )
  (and
   (add (= (rel-actions right) 0))
   (add (= (rel-actions left) 0))
  )
 )
)

(def-defined-predicate (init-action-counter2)
 (and
  (add (= (rel-actions right) 0))
  (add (= (rel-actions left) 0))
 )
)

(def-defined-function (numconts ?l)
 (local-vars ?i)
 (and
  (:= ?i 0)
  (forall (?x) (container ?x ?l) (:= ?i (+ ?i 1)))
  (:= numconts ?i)
 )
)

(def-defined-predicate (isatomic ?p)
 (forall (?c) (cpart ?c ?p) (not (established ?c)))
)

(def-defined-predicate (empty ?h)
 (not (exists (?x) (holds ?h ?x)))
```

```
)

(def-defined-predicate (feasible ?c)
 (and
  (forall (?x) (cbei ?c ?x) (not (established ?x)))
  (forall (?x) (cbein ?c ?x) (established ?x))
 )
)

(def-defined-predicate (check-holding-contact ?c ?h1 ?h2)
 (and
  (exists (?x) (cpart ?c ?x) (?y) (cpart ?c ?y)
   (and
    (not (= ?x ?y))
    (holds ?h1 ?x)
    (holds ?h2 ?y)
   )
  )
 )
)

(def-defined-predicate (connected ?p ?p2)
 (local-vars ?num-held (?been-held 100))
 (and
  (:= ?num-held 1)
  (or
   (= ?p ?p2)
   (connected2 ?p ?p2 ?num-held ?been-held)
  )
 )

)

(def-defined-predicate (connected2 ?p ?p2 ?num-held ?been-held)
 (and
  (:= (?been-held ?num-held) ?p)
  (or
   (exists (?c) (cpart ?c ?p)
    (and
     (established ?c)
     (cpart ?c ?p2)
    )
   )
   (exists (?c) (cpart ?c ?p) (?x) (cpart ?c ?x)
    (and
     (= (held ?x ?num-held ?been-held) FALSE)
     (established ?c)
     (:= ?num-held (+ ?num-held 1))
     (connected2 ?x ?p2  ?num-held ?been-held)
    )
   )
  )
 )
)

(def-defined-predicate (check-sim-grasp ?p)
 (local-vars ?num-held (?been-held 100))
 (and
  (:= ?num-held 1)
  (check-sim-grasp2 ?p ?num-held ?been-held)
 )
)

(def-defined-predicate (check-sim-grasp2 ?p ?num-held ?been-held)
 (and
  (:= (?been-held ?num-held) ?p)
  (not (exists (?x) (hand ?x) (will grasp ?x ?p)))
  (forall (?c) (cpart ?c ?p) (?x) (cpart ?c ?x)
   (implies
```

```
     (and
      (= (held ?x ?num-held ?been-held) FALSE)
      (established ?c)
     )
     (and
      (:= ?num-held (+ ?num-held 1))
      (check-sim-grasp2 ?x ?num-held ?been-held)
     )
    )
   )
  )
)

(def-defined-function (held ?p ?num-held ?been-held)
 (if-then-else
  (exists (?i) (is-between ?i 1 ?num-held) (= ?p (?been-held ?i)))
  (:= held TRUE)
  (:= held FALSE)
 )
)

(def-defined-predicate (grasp-all-contact ?h ?p)
 (local-vars ?num-held (?been-held 100))
 (and
  (:= ?num-held 1)
  (grasp-all-contact2 ?h ?p ?num-held ?been-held)
 )
)

(def-defined-predicate (grasp-all-contact2 ?h ?p ?num-held ?been-held)
 (and
  (add (holds ?h ?p))
  (:= (?been-held ?num-held) ?p)
  (forall (?c) (cpart ?c ?p) (?x) (cpart ?c ?x)
   (implies
    (and
     (= (held ?x ?num-held ?been-held) FALSE)
     (established ?c)
    )
    (and
     (:= ?num-held (+ ?num-held 1))
     (grasp-all-contact2 ?h ?x ?num-held ?been-held)
    )
   )
  )
 )
)

(def-defined-predicate (release-all-contact ?h ?p)
 (local-vars ?num-held (?been-held 100))
 (and
  (:= ?num-held 1)
  (release-all-contact2 ?h ?p ?num-held ?been-held)
 )
)

(def-defined-predicate (release-all-contact2 ?h ?p ?num-held ?been-held)
 (and
  (del (holds ?h ?p))
  (:= (?been-held ?num-held) ?p)
  (forall (?c) (cpart ?c ?p) (?x) (cpart ?c ?x)
   (implies
    (and
     (= (held ?x ?num-held ?been-held) FALSE)
     (established ?c)
    )
    (and
     (:= ?num-held (+ ?num-held 1))
     (release-all-contact2 ?h ?x ?num-held ?been-held)
```

```
     )
    )
   )
  )
 )

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;
;;;;;;; Initialization
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;

(set-initialization-sequence
 (add (= (tr-made left) 0))
 (add (= (tr-made right) 0))
 (add (= (cons-tr) 1))
 (add (= (max-cont) 1))
 (add (= (rel-fat) 4))
 (add (= (steps-to-extend) 0))
 (add (= (steps-to-hold) 1))
 (add (= (rel-actions left) 0))
 (add (= (rel-actions right) 0))
 (add (= (fatigue left) 0))
 (add (= (fatigue right) 0))
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;
;;;;;;;Actions
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;

(def-adl-operator (transport_empty ?h ?from ?to)
 (pre
  (?h) (hand ?h)
  (?from) (isloc ?from)
  (?to) (isloc ?to)
  (and
   (not (busy ?h))
   (empty ?h)
   (not (= ?from ?to))
   (at ?h ?from)
  )
 )
 (add (will transport_empty ?h ?to))
 (add (busy ?h))
 (add (+= (tr-made ?h) 1))
 (add (+= (fatigue ?h) 1))
 (add (+= (rel-actions ?h) 1))
 (del (at ?h ?from))
 (delayed-action 1 (finished-transport-empty ?h ?from ?to)
  (and
   (init-action-counter ?h)
   (del (will transport_empty ?h ?to))
   (del (busy ?h))
   (add (at ?h ?to))
  )
 )
)

(def-adl-operator (transport_loaded ?h ?from ?to)
 (pre
  (?h) (hand ?h)
  (?from) (isloc ?from)
  (?to) (isloc ?to)
  (and
   (not (busy ?h))
   (not (empty ?h))
   (not (= ?from ?to))
```

```
    (at ?h ?from)
   )
  )
  (del (at ?h ?from))
  (add (will transport_loaded ?h ?to))
  (add (busy ?h))
  (add (+= (tr-made ?h) 1))
  (add (+= (rel-actions ?h) 1))
  (add (+= (fatigue ?h) 1))
  (forall (?x) (holds ?h ?x) (del (at ?x ?from)))
  (delayed-action 1 (finished-transport-loaded ?h ?from ?to)
   (and
    (add (at ?h ?to))
    (forall (?x) (holds ?h ?x) (add (at ?x ?to)))
    (init-action-counter ?h)
    (del (will transport_loaded ?h ?to))
    (del (busy ?h))
   )
  )
 )

 (def-adl-operator (disassemble ?h1 ?h2 ?c)
  (pre
   (?h1) (hand ?h1)
   (?h2) (hand ?h2)
   (?c) (contact ?c)
   (and
    (not (= ?h1 ?h2))
    (not (busy ?h1))
    (not (busy ?h2))
    (exists (?p) (cpart ?c ?p) (holds ?h1 ?p))
    (empty ?h2)
    (exists (?l) (isloc ?l) (and (at ?h1 ?l) (at ?h2 ?l)))
    (not (exists (?x) (ctool ?c ?x)))
    (established ?c)
    (feasible ?c)
   )
  )
  (add (will disassemble1 ?h1 ?c))
  (add (will disassemble2 ?h2 ?c))
  (add (busy ?h1))
  (add (busy ?h2))
  (add (= (tr-made ?h1) 0))
  (add (= (tr-made ?h2) 0))
  (del (established ?c))
  (delayed-action 1 (finished-disassemble ?h1 ?h2 ?c)
   (and
    (exists (?x) (cpart ?c ?x) (?y) (cpart ?c ?y)
     (and
      (not (= ?x ?y))
      (grasp-all-contact ?h2 ?y)
      (release-all-contact ?h1 ?y)
     )
    )
    (init-action-counter2)
    (del (will disassemble1 ?h1 ?c))
    (del (will disassemble2 ?h2 ?c))
    (del (busy ?h1))
    (del (busy ?h2))
   )
  )
 )

 (def-adl-operator (disuse_wf ?h1 ?h2 ?c)
  (pre
   (?h1) (hand ?h1)
   (?h2) (hand ?h2)
   (?c) (contact ?c)
   (?t) (ctool ?c ?t)
```

```
  (and
   (not (= ?h1 ?h2))
   (exists (?l) (isloc ?l) (and (at ?h1 ?l) (at ?h2 ?l)))
   (not (busy ?h1))
   (not (busy ?h2))
   (exists (?p) (cpart ?c ?p) (holds ?h2 ?p))
   (holds ?h1 ?t)
   (not (exists! (?p) (cpart ?c ?p)))
   (established ?c)
   (feasible ?c)
  )
 )
 (add (will disuse_wf1 ?h1 ?c))
 (add (will disuse_wf2 ?h2 ?c))
 (add (busy ?h1))
 (add (busy ?h2))
 (add (= (tr-made ?h1) 0))
 (add (= (tr-made ?h2) 0))
 (del (established ?c))
 (delayed-action 1 (finished-disuse_wf ?h1 ?h2 ?c)
  (and
   (exists (?x) (cpart ?c ?x) (?y) (cpart ?c ?y)
    (and
     (not (= ?x ?y))
     (release-all-contact ?h2 ?y)
    )
   )
   (init-action-counter2)
   (del (will disuse_wf1 ?h1 ?c))
   (del (will disuse_wf2 ?h2 ?c))
   (del (busy ?h1))
   (del (busy ?h2))
  )
 )
)

(def-adl-operator (assemble ?h1 ?h2 ?c)
 (pre
  (?h1) (hand ?h1)
  (?h2) (hand ?h2)
  (?c) (contact ?c)
  (and
   (not (busy ?h1))
   (not (busy ?h2))
   (not (= ?h1 ?h2))
   (exists (?l) (isloc ?l) (and (at ?h1 ?l) (at ?h2 ?l)))
   (not (exists (?x) (ctool ?c ?x)))
   (not (established ?c))
   (feasible ?c)
   (check-holding-contact ?c ?h1 ?h2)
  )
 )
 (add (will assemble1 ?h1 ?c))
 (add (will assemble2 ?h2 ?c))
 (add (busy ?h1))
 (add (busy ?h2))
 (add (= (tr-made ?h1) 0))
 (add (= (tr-made ?h2) 0))
 (delayed-action 1 (finished-assemble ?h1 ?h2 ?c)
  (and
   (init-action-counter2)
   (forall (?x) (holds ?h1 ?x) (and (add (holds ?h2 ?x)) (del (holds ?h1 ?x))))
   (del (will assemble1 ?h1 ?c))
   (del (will assemble2 ?h2 ?c))
   (del (busy ?h1))
   (del (busy ?h2))
   (add (established ?c))
  )
 )
```

```
)

(def-adl-operator (use_sc ?h1 ?h2 ?c)
 (pre
  (?h1) (hand ?h1)
  (?h2) (hand ?h2)
  (?c) (contact ?c)
  (?t) (ctool ?c ?t)
  (and
   (not (= ?h1 ?h2))
   (not (busy ?h1))
   (not (busy ?h2))
   (not (established ?c))
   (exists! (?p) (cpart ?c ?p))
   (exists (?l) (isloc ?l) (and (at ?h1 ?l) (at ?h2 ?l)))
   (feasible ?c)
   (holds ?h1 ?t)
   (exists (?p) (cpart ?c ?p) (holds ?h2 ?p))
  )
 )
 (add (will use_sc1 ?h1 ?c))
 (add (will use_sc2 ?h2 ?c))
 (add (busy ?h1))
 (add (busy ?h2))
 (add (= (tr-made ?h1) 0))
 (add (= (tr-made ?h2) 0))
 (delayed-action 1 (finished-use_sc ?h1 ?h2 ?c)
  (and
   (init-action-counter2)
   (del (will use_sc1 ?h1 ?c))
   (del (will use_sc2 ?h2 ?c))
   (del (busy ?h1))
   (del (busy ?h2))
   (add (established ?c))
  )
 )
)

(def-adl-operator (use_wf ?h1 ?h2 ?c)
 (pre
  (?h1) (hand ?h1)
  (?h2) (hand ?h2)
  (?c) (contact ?c)
  (?t) (ctool ?c ?t)
  (and
   (not (= ?h1 ?h2))
   (not (busy ?h1))
   (not (busy ?h2))
   (not (established ?c))
   (exists (?l) (isloc ?l) (and (at ?h1 ?l) (at ?h2 ?l)))
   (not (exists! (?p) (cpart ?c ?p)))
   (feasible ?c)
   (holds ?h1 ?t)
   (exists (?x) (cpart ?c ?x) (?y) (cpart ?c ?y)
    (and
     (not (= ?x ?y))
     (holds ?h2 ?y)
     (pos ?x)
     (exists (?l) (isloc ?l) (and (at ?h1 ?l) (at ?x ?l)))
    )
   )
  )
 )
 (add (will use_wf1 ?h1 ?c))
 (add (will use_wf2 ?h2 ?c))
 (add (busy ?h1))
 (add (busy ?h2))
 (add (= (tr-made ?h1) 0))
 (add (= (tr-made ?h2) 0))
```

```
 (delayed-action 1 (finished-use_wf ?h1 ?h2 ?c)
  (and
   (init-action-counter2)
   (exists (?x) (cpart ?c ?x) (?y) (cpart ?c ?y)
    (and
     (not (= ?x ?y))
     (holds ?h2 ?y)
     (del (pos ?x))
    )
   )
   (forall (?x) (holds ?h2 ?x) (del (holds ?h2 ?x)))
   (del (will use_wf1 ?h1 ?c))
   (del (will use_wf2 ?h2 ?c))
   (del (busy ?h1))
   (del (busy ?h2))
   (add (established ?c))
  )
 )
)

(def-adl-operator (select-grasp-part ?h ?p)
 (pre
  (?h) (hand ?h)
  (?p) (part ?p)
  (and
   (not (busy ?h))
   (empty ?h)
   (not (exists (?x) (hand ?x) (holds ?x ?p)))
   (not (exists (?l) (at ?p ?l) (not (at ?h ?l))))
   (check-sim-grasp ?p)
   (implies
    (not (isatomic ?p))
    (exists (?l) (isloc ?l) (and (at ?h ?l) (at ?p ?l)))
   )
  )
 )
 (add (will grasp ?h ?p))
 (add (busy ?h))
 (add (= (tr-made ?h) 0))
 (add (+= (rel-actions ?h) 1))
 (add (+= (fatigue ?h) 1))
 (delayed-action 1 (finished-select-grasp-part ?h ?p)
  (and
   (grasp-all-contact ?h ?p)
   (exists (?l) (at ?h ?l)
    (implies
     (and (isatomic ?p) (not (at ?p ?l)))
     (and
      (add (at ?p ?l))
      (implies
       (not (container (type ?p) ?l))
       (add (container (type ?p) ?l))
      )
     )
    )
   )
   (init-action-counter ?h)
   (del (will grasp ?h ?p))
   (del (busy ?h))
   (forall (?x) (part ?x)
    (implies
     (and  (connected ?p ?x) (pos ?x))
     (del (pos ?x))
    )
   )
  )
 )
)
```

```
(def-adl-operator (select-grasp-tool ?h ?t)
 (pre
  (?h) (hand ?h)
  (?t) (tool ?t)
  (and
   (not (busy ?h))
   (empty ?h)
   (not (exists (?x) (hand ?x) (holds ?x ?t)))
   (not (exists (?l) (at ?t ?l)))
   (not (exists (?x) (hand ?x) (will grasp ?x ?t)))
  )
 )
 (add (will grasp ?h ?t))
 (add (busy ?h))
 (add (= (tr-made ?h) 0))
 (add (+= (rel-actions ?h) 1))
 (add (+= (fatigue ?h) 1))
 (exists (?l) (at ?h ?l)
  (implies
   (not (at ?t ?l))
   (and
    (add (at ?t ?l))
    (implies
     (not (container (type ?t) ?l))
     (add (container (type ?t) ?l))
    )
   )
  )
 )
 (delayed-action 1 (finished-select-grasp-tool ?h ?t)
  (and
   (add (holds ?h ?t))
   (init-action-counter ?h)
   (del (will grasp ?h ?t))
   (del (busy ?h))
  )
 )
)

(def-adl-operator (select-grasp_fh ?h1 ?h2 ?o)
 (pre
  (?h1) (hand ?h1)
  (?h2 ?o) (holds ?h2 ?o)
  (and
   (not (busy ?h1))
   (not (busy ?h2))
   (empty ?h1)
   (exists (?l) (isloc ?l) (and (at ?h1 ?l) (at ?h2 ?l)))
  )
 )
 (add (will grasp ?h1 ?o))
 (add (will release ?h2 ?o))
 (add (busy ?h1))
 (add (busy ?h2))
 (add (= (tr-made ?h1) 0))
 (add (= (tr-made ?h2) 0))
 (add (+= (rel-actions ?h1) 1))
 (add (+= (rel-actions ?h2) 1))
 (delayed-action 1 (finished-select-grasp-fh ?h1 ?h2 ?o)
  (and
   (init-action-counter2)
   (grasp-all-contact ?h1 ?o)
   (release-all-contact ?h2 ?o)
   (del (will grasp ?h1 ?o))
   (del (will release ?h2 ?o))
   (del (busy ?h1))
   (del (busy ?h2))
  )
 )
```

```
)

(def-adl-operator (release ?h ?o)
 (pre (?h ?o) (holds ?h ?o)
  (not (busy ?h))
 )
 (add (will release ?h ?o))
 (add (busy ?h))
 (add (= (tr-made ?h) 0))
 (add (+= (rel-actions ?h) 1))
 (add (+= (fatigue ?h) 1))
 (delayed-action 1 (finished-release ?h ?o)
  (and
   (release-all-contact ?h ?o)
   (init-action-counter ?h)
   (del (will release ?h ?o))
   (del (busy ?h))
  )
 )
)

(def-adl-operator (release_th ?h1 ?h2 ?o)
 (pre
  (?h1 ?o) (holds ?h1 ?o)
  (?h2) (hand ?h2)
  (and
   (not (busy ?h1))
   (not (busy ?h2))
   (empty ?h2)
   (exists (?l) (isloc ?l) (and (at ?h1 ?l) (at ?h2 ?l)))
  )
 )
 (add (will release ?h1 ?o))
 (add (will grasp ?h2 ?o))
 (add (busy ?h1))
 (add (busy ?h2))
 (add (= (tr-made ?h1) 0))
 (add (= (tr-made ?h2) 0))
 (add (+= (rel-actions ?h1) 1))
 (add (+= (rel-actions ?h2) 1))
 (delayed-action 1 (finished-release ?h1 ?h2 ?o)
  (and
   (init-action-counter2)
   (release-all-contact ?h1 ?o)
   (grasp-all-contact ?h2 ?o)
   (del (will release ?h1 ?o))
   (del (will grasp ?h2 ?o))
   (del (busy ?h1))
   (del (busy ?h2))
  )
 )
)

(def-adl-operator (position ?h ?p)
 (pre (?h ?p) (holds ?h ?p)
  (and
   (part ?p)
   (not (busy ?h))
   (exists (?c) (cpart ?c ?p) (?t) (ctool ?c ?t)
    (and
     (not (exists! (?p) (cpart ?c ?p)))
     (feasible ?c)
     (not (established ?c))
    )
   )
  )
 )
 (add (will position ?h ?p))
 (add (busy ?h))
```

```
 (add (= (tr-made ?h) 0))
 (add (+= (rel-actions ?h) 1))
 (add (+= (fatigue ?h) 1))
 (delayed-action 1 (finished-position ?h ?p)
  (and
   (release-all-contact ?h ?p)
   (init-action-counter ?h)
   (exists (?l) (at ?p ?l)
    (implies
     (not (fixture (type ?p) ?l))
     (add (fixture (type ?p) ?l))
    )
   )
   (add (pos ?p))
   (del (will position ?h ?p))
   (del (busy ?h))
  )
 )
)

(def-adl-operator (event)
 (wait-for-next-event)
 (priority -1)
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;
;;;;;;;Common Sense Rules
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;

(define (csr1)
 (always
  (and
   (<= (tr-made left) (cons-tr))
   (<= (tr-made right) (cons-tr))
  )
 )
)

(define (csr2)
 (always
  (forall (?h ?p) (will release ?h ?p)
   (until
    (not (exists (?h2 ?x) (will grasp ?h2 ?x) (connected ?p ?x)))
    (exists (?c) (contact ?c) (contact-est ?h ?c))
   )
  )
 )
)

(define (csr2-1)
 (always
  (forall (?h ?p) (will position ?h ?p)
   (until
    (not (exists (?h2 ?x) (will grasp ?h2 ?x) (connected ?p ?x)))
    (exists (?c) (contact ?c) (contact-est ?h ?c))
   )
  )
 )
)

(define (csr3)
 (always
  (forall (?h ?p) (will grasp ?h ?p)
   (until
    (not (exists (?x) (will release ?h ?x) (connected ?p ?x)))
    (exists (?c) (contact ?c) (contact-est ?h ?c))
   )
```

```
  )
 )
)

(define (csr4)
 (always
  (forall (?h) (hand ?h) (?c) (contact ?c)
   (implies
    (contact-est ?h ?c)
    (until
     (next (not (contact-est ?h ?c)))
     (next (exists (?x) (contact ?x)
      (and
       (not (= ?x ?c))
       (contact-est ?h ?x)
      )
     ))
    )
   )
  )
 )
)

(define (csr5)
 (always
  (forall (?h1) (hand ?h1) (?h2) (hand ?h2)
   (implies
    (not (= ?h1 ?h2))
    (not (exists (?x) (will release ?h1 ?x) (will grasp ?h2 ?x)))
   )
  )
 )
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;
;;;;;;;Goal-related Rules
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;

(define (grr1)
 (always
  (forall (?p) (part ?p)
   (implies
    (exists (?h) (hand ?h) (will grasp ?h ?p))
    (implies
     (exists (?c) (cpart ?c ?p))
     (exists (?c) (cpart ?c ?p)
      (implies
       (goal (established ?c))
       (feasible ?c)
      )
     )
    )
   )
  )
 )
)

(define (grr2)
 (always
  (forall (?h ?p) (holds ?h ?p) (?c) (cpart ?c ?p)
   (implies
    (or
     (and
      (not (established ?c))
      (goal (established ?c))
      (feasible ?c)
     )
```

```
      (and
       (established ?c)
       (not (goal (established ?c)))
       (feasible ?c)
      )
     )
     (until
      (not (exists (?x) (will release ?h ?x) (connected ?p ?x)))
      (exists (?c) (contact ?c) (contact-est ?h ?c))
     )
    )
   )
  )
)

(define (grr3)
 (always
  (forall (?p ?l) (goal (at ?p ?l)) (?h) (holds ?h ?p)
   (implies
    (forall (?c) (contact ?c)
     (and
      (goal (established ?c))
      (established ?c)
     )
    )
    (until (holds ?h ?p) (at ?p ?l))
   )
  )
 )
)

(define (grr4)
 (always
  (forall (?c) (contact ?c)
   (implies
    (or
     (exists (?h) (hand ?h) (will assemble1 ?h ?c))
     (exists (?h) (hand ?h) (will use_wf1 ?h ?c))
     (exists (?h) (hand ?h) (will use_sc1 ?h ?c))
    )
    (or
     (goal (established ?c))
     (exists (?c2) (cbein ?c ?c2))
    )
   )
  )
 )
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;
;;;;;;; Motion Economy Rules
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;

(define (mer1)
 (always
  (forall (?h) (hand ?h)
   (implies
    (exists (?c) (contact ?c) (contact-est ?h ?c))
    (at ?h WA)
   )
  )
 )
)

(define (mer2)
 (always
  (not (exists (?x) (container ?x WA)))
```

```
 )
)

(define (mer3)
 (always
  (not (exists (?l) (isloc ?l) (> (numconts ?l) (max-cont)) ))
 )
)

(define (mer4)
 (always
  (and
   (forall (?x ?l)(container ?x ?l)
    (not (exists (?l2) (container ?x ?l2)
     (not (= ?l ?l2))
    ))
   )
   (not (exists (?t) (tool ?t) (?l) (at ?t ?l)
    (and
     (not (= ?l WA))
     (not (container (type ?t) ?l))
    )
   ))
  )
 )
)

(define (mer5)
 (always
  (forall (?x1 ?l1) (container ?x1 ?l1) (?x2 ?l2) (container ?x2 ?l2)
   (implies
    (and
     (not (= ?x1 ?x2))
     (>= (num-of-type ?x1) (num-of-type ?x2))
    )
    (not (and
     (= (dist ?l1) max)
     (= (dist ?l2) normal)
    ))
   )
  )
 )
)

(define (mer6)
 (always
  (forall (?l) (isloc ?l) (implies
   (= (dist ?l) max)
   (implies
    (or
     (exists (?h) (will transport_empty ?h ?l))
     (exists (?h) (will transport_loaded ?h ?l))
    )
    (or
     (not (exists (?x) (isloc ?x)
      (and
       (= (dist ?x) normal)
       (< (numconts ?x) (max-cont))
      )
     ))
     (exists (?h ?p) (holds ?h ?p) (goal (at ?p ?l)))
     (exists (?p) (at ?p ?l) (not (isatomic ?p)))
    )
   )
  ))
 )
)

(define (mer7)
```

```
(always (and
 (forall (?h ?l1) (at ?h ?l1)
  (implies
   (and
    (= (axis ?l1) horizontal)
    (next (exists (?l2) (will transport_empty ?h ?l2) (= (axis ?l2) vertical)))
   )
   (until
    (not (or
     (exists (?l3) (will transport_loaded ?h ?l3) (= (axis ?l3) horizontal))
     (exists (?l3) (will transport_empty ?h ?l3) (= (axis ?l3) horizontal))
    ))
    (or
     (at ?h WA)
     (exists (?c) (contact ?c) (contact-est ?h ?c))
    )
   )
  )
 )
 (forall (?h ?l1) (at ?h ?l1)
  (implies
   (and
    (= (axis ?l1) vertical)
    (next (exists (?l2) (will transport_empty ?h ?l2) (= (axis ?l2) horizontal)))
   )
   (until
    (not (or
     (exists (?l3) (will transport_loaded ?h ?l3) (= (axis ?l3) vertical))
     (exists (?l3) (will transport_empty ?h ?l3) (= (axis ?l3) vertical))
    ))
    (or
     (at ?h WA)
     (exists (?c) (contact ?c) (contact-est ?h ?c))
    )
   )
  )
 )
 (forall (?h ?l1) (at ?h ?l1)
  (implies
   (and
    (= (axis ?l1) horizontal)
    (next (exists (?l2) (will transport_loaded ?h ?l2) (= (axis ?l2) vertical)))
   )
   (until
    (not (or
     (exists (?l3) (will transport_loaded ?h ?l3) (= (axis ?l3) horizontal))
     (exists (?l3) (will transport_empty ?h ?l3) (= (axis ?l3) horizontal))
    ))
    (or
     (at ?h WA)
     (exists (?c) (contact ?c) (contact-est ?h ?c))
    )
   )
  )
 )
 (forall (?h ?l1) (at ?h ?l1)
  (implies
   (and
    (= (axis ?l1) vertical)
    (next (exists (?l2) (will transport_loaded ?h ?l2) (= (axis ?l2) horizontal)))
   )
   (until
    (not (or
     (exists (?l3) (will transport_loaded ?h ?l3) (= (axis ?l3) vertical))
     (exists (?l3) (will transport_empty ?h ?l3) (= (axis ?l3) vertical))
    ))
    (or
     (at ?h WA)
     (exists (?c) (contact ?c) (contact-est ?h ?c))
```

```
      )
     )
    )
   )
  ))
)

(define (mer8)
 (always (and
  (forall (?h ?l) (will transport_empty ?h ?l)
   (and
    (if-then-else
     (= ?h right)
     (not (= (side ?l) left))
     (not (= (side ?l) right))
    )
   )
  )
  (forall (?h ?l) (will transport_loaded ?h ?l)
   (and
    (if-then-else
     (= ?h right)
     (not (= (side ?l) left))
     (not (= (side ?l) right))
    )
   )
  )
 ))
)

(define (mer9)
 (always
  (forall (?lr1) (at right ?lr1) (?ll1) (at left ?ll1) (?lr2) (isloc ?lr2) (?ll2)
(isloc ?ll2)
   (implies
    (and
     (or
      (next (will transport_loaded right ?lr2))
      (next (will transport_empty right ?lr2))
      (next (next (will transport_loaded right ?lr2)))
      (next (next (will transport_empty right ?lr2)))
     )
     (or
      (next (will transport_loaded left ?ll2))
      (next (will transport_empty left ?ll2))
      (next (next (will transport_loaded left ?ll2)))
      (next (next (will transport_empty left ?ll2)))
     )
    )
    (or
     (or
      (exists (?h ?p) (holds ?h ?p) (goal (at ?p ?ll2)))
      (exists (?h ?p) (holds ?h ?p) (goal (at ?p ?lr2)))
     )
     (and
      (= ?lr2 WA)
      (= ?ll2 WA)
      (= (side ?lr1) right)
      (= (side ?ll1) left)
      (implies (= (dist ?lr1) max) (= (dist ?ll1) max))
      (implies (= (dist ?lr1) normal) (= (dist ?ll1) normal))
      (implies (= (axis ?lr1) vertical) (= (axis ?ll1) vertical))
      (implies (= (axis ?lr1) horizontal) (= (axis ?ll1) horizontal))
     )
     (and
      (= ?lr1 WA)
      (= ?ll1 WA)
      (= (side ?lr2) right)
      (= (side ?ll2) left)
```

```
       (implies (= (dist ?lr2) max) (= (dist ?ll2) max))
       (implies (= (dist ?lr2) normal) (= (dist ?ll2) normal))
       (implies (= (axis ?lr2) vertical) (= (axis ?ll2) vertical))
       (implies (= (axis ?lr2) horizontal) (= (axis ?ll2) horizontal))
      )
      (and
       (= (side ?lr1) right)
       (= (side ?ll1) left)
       (implies (= (dist ?lr1) max) (= (dist ?ll1) max))
       (implies (= (dist ?lr1) normal) (= (dist ?ll1) normal))
       (implies (= (axis ?lr1) vertical) (= (axis ?ll1) vertical))
       (implies (= (axis ?lr1) horizontal) (= (axis ?ll1) horizontal))
       (= (side ?lr2) right)
       (= (side ?ll2) left)
       (implies (= (dist ?lr2) max) (= (dist ?ll2) max))
       (implies (= (dist ?lr2) normal) (= (dist ?ll2) normal))
       (implies (= (axis ?lr2) vertical) (= (axis ?ll2) vertical))
       (implies (= (axis ?lr2) horizontal) (= (axis ?ll2) horizontal))
      )
     )
    )
   )
  )
)

(define (mer9-1)
 (always
  (forall (?h1) (hand ?h1) (?h2) (hand ?h2)
   (implies
    (and
     (not (= ?h1 ?h2))
     (or
      (exists (?l) (will transport_loaded ?h1 ?l))
      (exists (?l) (will transport_empty ?h1 ?l))
     )
    )
    (or
     (exists (?l) (will transport_loaded ?h2 ?l))
     (exists (?l) (will transport_empty ?h2 ?l))
     (not (busy ?h2))
    )
   )
  )
 )
)

(define (mer9-2)
 (always
  (forall (?h1) (hand ?h1) (?h2) (hand ?h2)
   (implies
    (not (= ?h1 ?h2))
    (not (and
     (>= (rel-actions ?h1) 1)
     (>= (rel-actions ?h2) 1)
     (not (busy ?h1))
     (not (busy ?h2))
    ))
   )
  )
 )
)

(define (mer10)
 (always
  (forall (?h1) (hand ?h1) (?h2) (hand ?h2) (?l) (isloc ?l)
   (implies
    (and
     (not (= ?h1 ?h2))
     (at ?h1 ?l)
```

```
    (not (= ?l WA))
   )
   (until
    (implies
     (not (exists (?h) (hand ?h) (busy ?h)))
     (implies
      (> (rel-actions ?h2) (rel-actions ?h1))
     (<= (abs (- (rel-actions ?h2) (rel-actions ?h1))) (steps-to-extend))
     )
    )
    (not (at ?h1 ?l))
   )
  )
 )
 )
)

(define (mer11)
 (always
  (forall (?h1) (hand ?h1) (?l) (at ?h1 ?l) (?h2) (hand ?h2) (?p) (holds ?h1 ?p)
   (implies
    (and
     (not (= ?h1 ?h2))
     (not (at ?h1 WA))
    )
    (until
     (implies
      (not (exists (?h) (hand ?h) (busy ?h)))
      (implies
       (> (rel-actions ?h2) (rel-actions ?h1))
      (<= (abs (- (rel-actions ?h2) (rel-actions ?h1))) (steps-to-hold))
      )
     )
     (or
      (not (at ?h1 ?l))
      (not (holds ?h1 ?p))
     )
    )
   )
  )
 )
)

(define (mer12)
 (always
  (forall (?h1) (hand ?h1) (?h2) (hand ?h2)
   (implies
    (not (= ?h1 ?h2))
    (<= (abs (- (fatigue ?h1) (fatigue ?h2))) (rel-fat))
   )
  )
 )
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;
;;;;;;;Search Control
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;

(define (cs-rules)
 (and
  (csr1)
  (csr2)
  (csr2-1)
  (csr3)
  (csr4)
  (csr5)
 )
```

```
)

(define (gr-rules)
 (and
  (grr1)
  (grr2)
  (grr3)
  (grr4)
 )
)

(define (me-rules)
 (and
  (mer1)
  (mer2)
  (mer3)
  (mer4)
  (mer5)
  (mer6)
  (mer7)
  (mer8)
  (mer9)
  (mer9-1)
  (mer9-2)
  (mer10)
  (mer11)
  (mer12)
 )
)

(define (search-control)
 (and
  (cs-rules)
  (gr-rules)
  (me-rules)
 )
)

(set-tl-control (search-control))
```