

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.



University Microfilms International
A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
313/761-4700 800/521-0600

Order Number 9315446

Discrete models for robot path planning

Austin, Shermane Annette, Ph.D.

City University of New York, 1993

Copyright ©1993 by Austin, Shermane Annette. All rights reserved.

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

A

**DISCRETE MODELS FOR
ROBOT PATH PLANNING**

by

SHERMANE A. AUSTIN

**A dissertation submitted to the Graduate Faculty in Computer
Science in partial fulfillment of the requirements for the degree of
Doctor of Philosophy, The City University of New York.**

1993

©1993

SHERMANE A. AUSTIN

All Rights Reserved

This manuscript has been read and accepted for the Graduate Faculty in Computer Science in satisfaction of the dissertation requirement for the degree of Doctor of Philosophy.

2/15/93
Date

Michael Amich
Chair of Examining Committee

2/16/93
Date

Stanley Habib
Executive Officer

Professor Stefan Burr
Professor Izidor Gertner
Professor Stanley Habib

Supervisory Committee

The City University of New York

Abstract

DISCRETE MODELS FOR ROBOT PATH-PLANNING

by

Shermane A. Austin

Adviser: Professor Michael Anshel

Our purpose in this work is to investigate the applications of discrete models for a restricted class of robot-planning problems. A high-level planner is formulated that uses the structural characteristics of workcell configurations as a basis for path-planning. The planner also provides a logical framework for representing and reasoning about positional information with procedural interfaces for processing descriptions of objects and robots in the workcell, and the low-level details of motion control.

We begin with a discussion of domain-independent planners and explore some of the special requirements for robot planning. Following this discussion, we develop and formulate a poset-based model. Motivated by investigations undertaken by Thuk-Air et al of k -directional relations, we use restricted directional posets to capture the underlying properties of objects in workcell configurations. We extend Thuk-Air's formulation of bi-directional relations by showing how arbitrary finite posets can be expanded to obtain bi-directional relations. We then further define and characterize a restricted class of directional posets, k_2 and $quad_2$, to better

approximate motion-planning in a high-level planner. We also formulate a class of posets, *goal* posets, as a basis for goal-directed path-planning analysis. This analysis together with the directed relation induced by a given configuration allows us to determine whether or not the objects can be moved in a straight line to their goal locations.

Finally, a *directional* logic is defined, using the directional posets as a basis for control. We illustrate its implementation with separability problems involving restricted direction assignments. Concurrent planning is also discussed and explored in these examples.

Contents

Abstract		iv
Figures		ix
Chapter 1	INTRODUCTION	1
1.1	Overview	1
1.2	Planning	2
1.3	Robot-Planning	4
1.4	Proposed Problem	6
1.5	Summary	12
Chapter 2	PLANNING FUNDAMENTALS	14
2.1	Planning	14
2.2	Traditional Models	19
2.3	The Frame Problem	22
2.4	Concurrent Planning	23
Chapter 3	ROBOT INTERFACES FOR PLANNING	27
3.1	Overview	27
3.2	Components	27
3.3	Path-determination in the planar workcell	28
3.4	Additional constraints and considerations	29

Chapter 4	DIRECTIONAL POSETS AND PATH-PLANNING	33
4.1	Motivation	33
4.2	Background	34
4.3	Expansions of Bi-directional Posets	37
4.4	Directional Posets	40
4.5	k_2 -directional posets	43
4.6	Straight-line motion in the plane	47
Chapter 5	AUTOMATED REASONING AND POSET MANIPULATION	53
5.1	Introduction	53
5.2	Domain Induction	57
5.3	The Blocking Relation	59
5.4	Implementation	61
Chapter 6	SEPARABILITY EXAMPLES USING AUTOMATED REASONING	63
6.1	Introduction	63
6.2	Bi-direction separability for one robot	67
6.3	Bi-direction separability for two robots	78
6.4	Quad2-direction separability for one robot	84

6.5	Quad ₂ -direction separability for two robots	93
6.6	Poset-partitioning for robot clusters	95
6.7	Goal-directed motion	96
6.8	Summary	98
Chapter 7	CONCLUSION	100
7.1	Results	100
7.2	Limitations and Questions for Further Study	101
7.3	Summary	102
Appendix A	Clauses for separability problem for one robot	104
Appendix B	Two directional separability problem for two robot arms	112
Appendix C	Quad₂-directional separability for one arm	126
Appendix D	Quad₂-directional separability for two arms	141
References		168

Figures

Figure 1.4.1	Uncluttered workcell	7
Figure 1.4.2	Effect of cycles	8
Figure 2.1.1	Block's World example	15
Figure 2.1.2	Example of Sussman's anomaly	18
Figure 4.2.1	Relations, subdivisions, and directional assignments	36
Figure 4.3.2	Adding expansions to a poset	38
Figure 4.3.3	Direction assignments resulting from an expansion	40
Figure 4.5.4	Adding a point to the directional poset	46
Figure 4.6.5	Objects and their goal locations	49
Figure 4.6.6	Poset Induced by objects and locations	49
Figure 4.6.7	Goal destinations resulting in cycles	51
Figure 4.6.8	Cluttered workspace	51
Figure 6.1.1	Separability Illustration	64
Figure 6.2.2	Chain-induced deadlock	71
Figure 6.2.3	Stair-stepping motion	72
Figure 6.2.4	Boundary motion for two directions	74
Figure 6.4.5	Deadlock example with quad_2 motion	90
Figure 6.4.6	Movement around the boundaries	91

CHAPTER 1

INTRODUCTION

§ 1.1 Overview

The goal of this research is to investigate the applications of discrete models to path-planning for robot tasks. A range of paradigms exist for the solution of such problems including algorithmic optimization techniques, neural net implementations and genetic algorithms. Path-planning applications for use in actual robot workcells are complex because of the need to describe the workspace configuration, and the geometric relations of the robots and objects in the workcell. Traditional planning paradigms have approached robot problem-solving from the perspective of higher-level reasoning without attempting to integrate the low-order details required to accomplish the specified tasks.

This investigation situates itself between the two levels. It is an attempt to explore structural characterizations of path-planning requirements at an abstract level while also providing the basis for interfacing with the specific procedural descriptions of actual robots and the objects to be manipulated. Discrete models for a restricted class of robot-planning problems will be explored and implemented using “directional logic.” The overall goals are twofold: 1) to continue the investigation of the role of

posets and poset-generation strategies in path-planning; and 2) to provide a logical framework for representing and reasoning about positional information.

§ 1.2 Planning

1.2.1 Definition

Planning is a process in which given specific objects, actions are applied in order to obtain a given outcome. Traditional planning terminology describes domain objects in terms of their properties and relations among objects [ALLEN90]. For example, if x and y are blocks on a table, then $CLEAR(x)$ and $ON(x,y)$ describe respectively that block x has the property of being clear, and block x is on top of block y . The causal events, those which affect changes in the domain, are actions or operators. Typical operators for a robotic system might include $MOVE(x,y)$, $GRASP(y)$, $PUTDOWN(z)$, where x,y and z are variables representing locations or objects. The application of an action or operator results in a change in the relations among domain objects. Goals are specified representing specific behaviors, and the output of the process is a plan or sequence of actions to achieve the goal. Most planners reason about the actions and effects of a single causal agent, i.e., the robot or the system implementing the plan. The two basic components of planning models are knowledge representation and plan generation. Knowledge representation is the description of the world view. This includes the domain, its objects, their properties, and relationships among objects. It also includes representations of permissible

actions or events, and changes in the worldview [ALLEN90]. Plan generation is the process of formulating the plan.

1.2.2 Complexity of planning

Like most artificial intelligence (AI) methods, planning is computationally inefficient for domains of any scale. The determination as to which sequence of operators will result in a given plan is based on a search strategy for exploring possibilities. As the number of relations and actions/operators increases, the number of possibilities increases exponentially. As a result, much attention is paid to the nature of search, as well as the development of heuristics to expedite searches for domain-specific applications. For robot clusters, additional complexity is imposed by the need to partition the sub-tasks among the systems and reason about their effects as a function of time. Approaches to the complexity of planning that lie beyond traditional formal models include: 1) the development of procedural approaches to task-class specific problems (algorithmic solutions); 2) the integration of procedural approaches with standard AI methodologies; and 3) the use of concurrent and parallel strategies for non-deterministic searches.

1.2.3 Concurrency and Time in Planning

Most approaches to planning assume the existence of a single causal agent, the system using the operators to effect the plan. If a specific event or operator does not modify a property or relation, then it can be inferred (or otherwise implied) that the

property or relation persists. When actions occur simultaneously, such inferences are not applicable. The persistence of a property or relation is not a function of a single event, but a set of events. Similarly, the effect of an action is a function of that event together with all other events occurring simultaneously.

Planning for concurrent events, or multi-agent planning, requires an extension of traditional planning models. The effects of events have to be accounted for in the knowledge representation. Discrete states or points in time can no longer be used to describe the worldview because actions have varying durations. Various strategies exist for concurrent events and their effects including reasoning about chronicles and concurrent events [GEORG87], the use of interval-based calculi [MCDER82, ALLEN84] and reasoning about continuous state changes [VERE83]. While these embrace many of the generic features of high-level descriptions of concurrent activity, only those strategies based on continuous state changes, are useful for robot applications. Domain-independent planners such as these, however, do not include the ability to represent or reason about positional information.

§ 1.3 Robot-Planning

1.3.1 Requirements

Symbolic operators and objects such as *GRASP(A)* and *MOVE(B, TABLE(x,y,z))*, where *A* and *B* are objects, are very general and easily understandable. They lack, however, needed detail for robot-planning. For example, *MOVE(A, TABLE)* gives

no indication as to how the move is to be accomplished. In a cluttered space, a controlled move would be required in order to avoid obstacle collision. Under other circumstances, an unconstrained move would suffice. Abstract planners do not include needed domain-specific geometric and physical descriptions of the objects and of the robots manipulating them. Robot-planning, also known as task-planning, is a domain-specific application of planning for robotic and other mechanical systems. It incorporates the special descriptions and needs of the robots, the workcells and domain objects. More so than the abstract planners discussed previously, task-planners suffer from computationally infeasible methods. Computational complexity is added by the need to describe and reason about positional and geometric representations.

1.3.2 Robot clusters

The use of robot clusters has implications for planners because of the added requirement of partitioning the tasks or portions thereof among several robotic systems. These systems will be satisfying subgoals or tasks concurrently. Unlike the domain-independent planners described above, the required partitions must be subject to work-cell dependent constraints such as collision-avoidance and arm-reachability for the robots involved. Optimization criteria, such as speed and manipulator motion minimization may also be specified.

Both the constraints and the optimization criteria add levels of complexity to the plan generation process. The dynamic nature of sub-task completion, the need to anticipate and avoid arm collisions, and the need to reason using positional

information for causal agents and objects in the problem space, are features that must be embedded in the planner. Additionally, robot clusters have the capacity for successful task-completion even in the face of error conditions. A desirable feature of planners for such systems would be the incorporation of some error detection/correction capability and the provision of built-in mechanisms such as plan modification or reformulation for recovery. The goal of the proposed research is to explore strategies which are situated between the abstract planning approach and the robot-planning level demands. The question posed is to what extent can path-planning requirements be generalized.

§ 1.4 Proposed Problem

1.4.1 Posets and Planning

The output of a traditional planner is typically an ordered list of required operations on objects to execute the plan. Consider, for example, the problem of moving a few objects in the plane, as in figure 1.4.1.

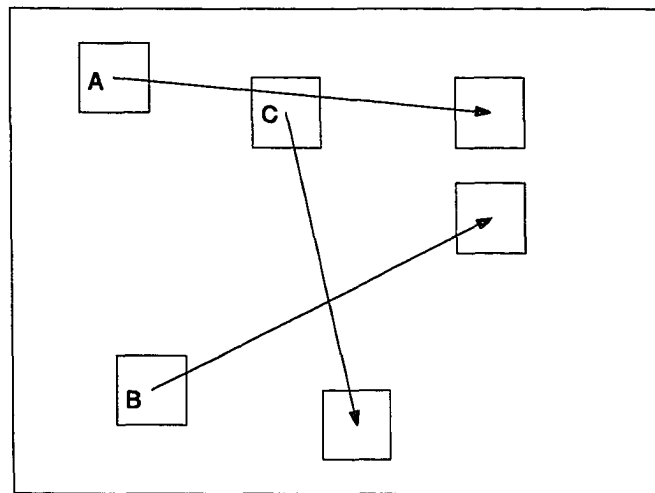


Figure 1.4.1 Uncluttered workcell

In this example, there is a straight-line unbroken-path between all of the objects and their destinations, and a corresponding poset that induces the ordering of the *MOVE* operations. No path-planning is occurring at this level as it involves a relatively uncluttered workspace. Generating the appropriate precedence relations guides the motion for a single robot and also provides a basis for a cluster of robots to manipulate the objects subject to the specific robot-dependent requirements of reachability and arm collision. The resulting poset is partitioned among the participating arms.

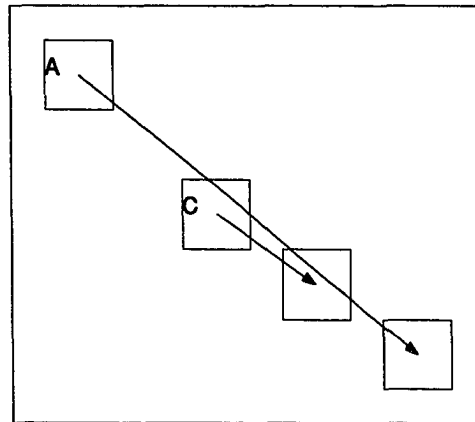


Figure 1.4.2 Effect of cycles

The above strategy, however, may fail in many instances. It cannot be applied to a cluttered workspace. In other workspaces, the relation may result in cycles. In fact, a simple modification of the planar objects yields a configuration in which the path from an object to its destination is a sequence of vectors. A simple case occurs in figure 1.4.2, the destination for object *C* lies on the path between object *A* and its destination. *A* is blocked by *C*, and the resulting path from *A* to its destination is a sequence of line segments if *C* is moved first. The task at this level requires determining the path that *A* should take to reach its destination. Formulating motion with respect to a planar grid-based workcell results in an infinite number of directions in which an object can travel. We will explore an approach that restricts possible directions along which an object can travel, if it is assumed that it cannot move in a straight-line towards its goal. The intent of the planner is to formulate a poset based on the initial, possible directional assignments for each object and

motivate path-planning with dynamically, updated posets. To distinguish these posets from the goal-oriented ones described earlier, they will be referred to as *directed* or *directional* posets, an extension of continuing investigation of directed relations [THUK91, RIVAL87]. At the same time, given a path-planning task, decisions must be made in terms of the appropriate strategy by the relationship among the objects and their goals. An uncluttered configuration, as defined previously, should be recognized as one in which the goal poset is sufficient to motivate planning.

1.4.2 Domain and Task specifications

A robot arm or cluster of arms is given the task of moving a group of objects through the plane to their destinations, as in figure 1.4.1. The domain of objects will be simple squares with descriptions of geometric dimensions and locations. The task will be specified with the set of primitive assertions, $GOAL(z,x,y)$ where z is an object, x,y its destination in terms of planar coordinates. The input to the planner will be the initial locations of the objects. The output will be a sequence of vectors specifying the path that an arm travels as it moves each object to its goal location.

1.4.3 Approximating Motion in the Plane

The planar workcell will be subdivided into a grid-based system of cells. The positional reference for objects will in terms of grid values (x_i,y_i) . The coordinates for a particular object correspond to the cell in which the tool-tip reference for that object is located. The specific tool-tip coordinates (x_i,y_i,z_0) and subsequent joint-

variable vector translations will be determined by a lower-level, robot-dependent interface. To simplify their descriptions, objects may only be contained in grid cells.

At the tool-tip level, movement along a straight-line is accomplished by interpolating points in joint-vector variable space. The time, t , required for an arm to move along a straight line is a function of the speed of the individual joints and of the interpolated motion. It is necessarily arm-dependent. On the other hand, a trajectory along a straight-line can be planned so that it is done in some time, T . For this reason, without loss of flexibility, the planner will assume that given an arbitrary tool-tip position, the time required to move to an adjacent cell is a constant with respect to a given arm. The motivation for the constant assumption is that the arm and path-collision constraint-handling is procedurally implemented. As a result, variable forms of path-time determinations are possible, but they are not integral to the planner. Instead they can be handled by external procedural calls. This permits the development of a planner that is arm-independent at the abstract level, and permits external flexibility in the handling and specification of paths and trajectories chosen.

1.4.4 Single-robot and k -robot models

Single-robot planning models will be explored to determine the appropriate precedence requirements for initial and final configuration clusters in uncluttered workcells. Strategies for approximating motion using directional posets induced by m -directed ordered sets in cluttered workspaces will also be developed.

The results of the one-robot models will be extended to create a planner for two robots. This planner will be an application of the separability problem with restricted directional motion. In our previous discussion, constant time was assumed for each stage. In practice, however, this would result in idle time for some of the robots involved. The planner will, however, only admit idle time, when no other appropriate motion is possible for additional arm(s). The robot cluster is assumed to consist of homogeneous arms in order to simplify the computations involved.

For cluttered workcells, the planner will provide a strategy for arm-motion assignments based on the directional poset induced by the m -directed ordered sets in the single-robot model. The consequences of concurrent activity will be investigated but will be not implemented in a general way. Arm-motion assignments will be generated based on the current directional poset, and subject to collision testing.

Structural characterizations of cluttered and uncluttered workcells will be explored. These properties form the basis of deciding, given a description of the workspace and the objects therein, which strategies to use. The strategies for uncluttered motion planning are inherently less complex than those for cluttered workcells and should be used whenever possible. This also implicitly provides for partitioning the objects themselves into cluttered/uncluttered regions. This type of partition, however, will not be implemented in these models.

§ 1.5 Summary

The goal of this research is to contribute to the ongoing investigation of robot control by posing an alternate, conceptual framework for path-planning applications. The central question is to what extent can the structural properties of a workcell configuration be used as the basis for motivating planning. This framework has three components: 1) the separation of the robot-dependent requirements from the abstract planning or problem-solving process; 2) the enforcement of a unified, homogeneous domain independent of predefined specific task actions and based on positional representations; and 3) the combination of the expressive capacity of automated reasoning methodologies with requisite procedural attachments capable of handling the decisions and translations motivated by the requirements of low-level robot control.

The proposed model to be constructed will restrict goal-oriented motion to the plane and will assume the use of stationary, homogeneous, cylindrical robots. The model will be implemented in an actual work cell in order to analyze its dynamic behavior. In spite of the restrictions, the proposed model is intended to be sufficiently extensible in order to deal with varying types of tasks, stationary one-armed robots, and constraint and optimization criteria.

In Chapter 2 we give a general background on planning and in Chapter 3 relate robot-specific issues to the task of planning. In Chapter 4, we provide the motivation

for the use of directional posets and provide definitions and results in our theoretical investigation of restricted classes of such posets. Chapter 5 provides a discussion of automated reasoning and the technique we reference in manipulating posets. Chapter 6 provides examples illustrating the use of directional posets for separability and discusses poset-partition strategies and as well as the application of such posets to generalized goal-directed planning problems. In Chapter 7 we summarize our work, discuss its limitations and propose questions for further research.

CHAPTER 2

PLANNING FUNDAMENTALS

§ 2.1 *Planning*

2.1.1 Overview

Planning is concerned with motivating the behavior of intelligent systems. The ability to develop automated planners for such systems remains an elusive and much sought-after goal in AI research. The domain of research spans mathematical modeling, philosophical inquiry and psychological analysis. This chapter is not an attempt to capture the broad-based issues related to the nature of planning. Instead, it emphasizes the relevant and fundamental concerns arising in the development of robot planners. The first part of the discussion surveys the "classic" approaches to planning. The next part of the discussion looks at those issues beyond the domains of these classic approaches with respect to the development of a reasoning system for cooperating robot clusters. The latter part of the chapter provides a discussion of a robot interface for the proposed model.

2.1.2 Block's World

Planning involves providing the processor with a knowledge base and the ability to apply a set of predefined rules to produce an executable plan for a robot system. An example can be illustrated using the simple Block's World Model [WINST84]. A few blocks are placed on a table and the robot has the task of moving them to a certain configuration or goal state, as in figure 2.1.1 The start configuration could be described as:

ON(A, Table), ON(B, Table), ON(C, Table), ON(D, Table).

The goal state could be described as

ON(A, B), ON(B, C), ON(D, Table), ON(C, Table).

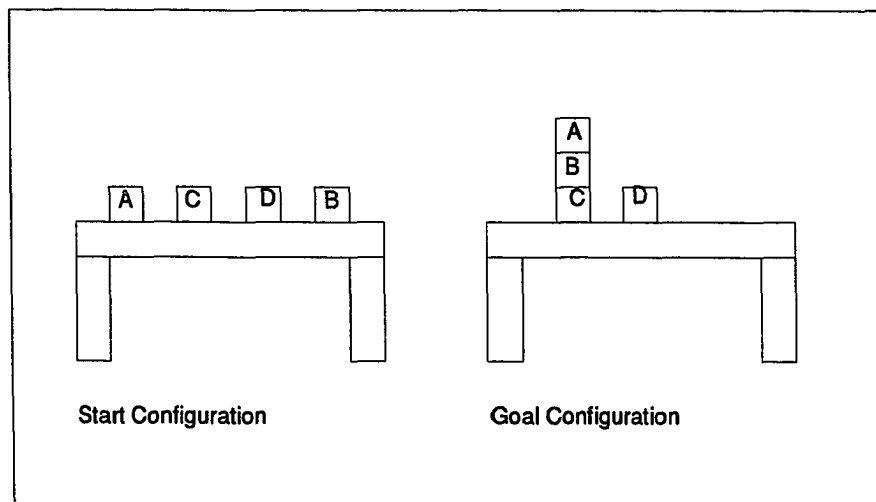


Figure 2.1.1 Block's World example

In both cases, the relations among objects are used to describe various states. The goal state represents a conjunction of relations to be achieved, where each relation may be considered a subgoal. The solution for the goal state requires applying operators that first achieve $ON(B, C)$ and then $ON(A, B)$. Such operations may include terms such as $GRASP(B)$ and $PUTON(B, C)$. In contrast to relations, they express actions to be taken by the robot arm.

Even for the simple Block's World problem, however, a number of problems emerge in the development of a planner to solve the problem in a general way. The *frame* problem is based on the need to update the global environment, by indicating what has *not* been changed after a given plan or subplan has been executed. In the example given, the knowledge base would be presumably updated by the robot's action, but what is also needed is the explicit recognition that relations concerning blocks not manipulated are unchanged [MCCAR69].

The *execution* problem is based on the need to distinguish between proving a goal is realizable (e.g. provable), and actually an executable plan. Waldinger cites the "monkey, banana and bomb" problem as an example of the execution problem [WALD87]. A monkey is in a room with two boxes. Beneath one of the boxes is a bunch of bananas and beneath the other is a bomb. A reasonable plan would be for the monkey to examine one box and if the bananas are not found, examine the other. The resulting plan, however, is not executable.

The *anomaly* problem is based on the recognition that in a robust plan, indepen-

dent causal agents may induce changes to the knowledge base [NILSS71]. These changes may either result in plan failure or the need for plan remodification. For example, if block D fell off the table, it would not alter the execution of the plan, however, if block A fell off the table, failure would occur.

The *search-space* problem is based on the complexity of planning. Planners are characterized by exponential-time solutions, and must rely on heuristic modeling or strategic processing to restrict the search as much as possible. In the example above, the goal state is the conjunction of the subgoals $ON(A, B)$ and $ON(B, C)$. If $ON(A, B)$ is the first subgoal chosen to be satisfied, then the second goal will be unrealizable unless the system backtracks and changes the order of the subgoals. If there are n subgoals then the number of possible orderings is $n!$. One response to the search-space dilemma is the development of parallel and concurrent planning systems based on appropriate architectures.

In addition to the problems cited above, given a planner there must be assurance that the planning mechanism can handle any variations within a given domain. Consider the anomaly, illustrated in figure 2.1.3 [SACER75b]. The solution becomes difficult for an automated system processing subgoals sequentially. The subgoals are $ON(A, B)$ and $ON(B, C)$. If $ON(A, B)$ is chosen first, then B is no longer clear and $ON(B, C)$ cannot be achieved. On the other hand, if $ON(B, C)$ is chosen first then $ON(A, B)$ cannot be achieved since A is beneath C .

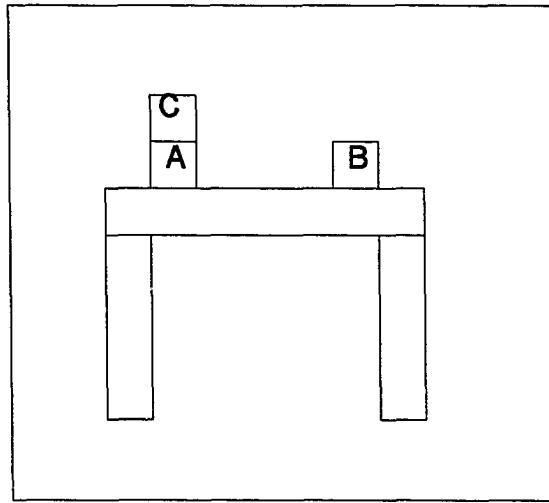


Figure 2.1.2 Example of Sussman's anomaly

2.1.3 Robot Planning

The above descriptions of relations among objects such as $ON(X,Y)$ and operations on objects, e.g., $PICKUP(X)$ are very intuitive. For robot systems, however, more information is needed to convey descriptions of objects, their relations, and operations that can be performed. The task of moving a simple block from one place to another, requires an adequate description of the physical dimensions of the blocks, their location with respect to the robot arm, their relative mass, etc. In addition, these descriptions do not indicate how the arm should move. Depending on the configurations, directed motion might be required to prevent collision with other blocks which might possibly result in plan failure.

§ 2.2 Traditional Models

2.2.1 Situation Logic

Pre-1975 planners were characterized by situation logic approaches. They involved the formulation of the problem or task in the predicate calculus and then applying mechanical theorem-proving to show that the goal is realizable. The situational logic framework in which these planners were developed was motivated by the realization that relations among objects change as they are manipulated. To maintain the logical distinction among the various stages of the database, state variables are introduced. For example, $ON(s_0, A, B)$ represents the assertion that in state s_0 , block A is on block B .

These models required frame assertions to reflect the global view at each state, i.e., that those relations unchanged by an action in state s_{i-1} would be the same in state s_i . The frame problem was addressed in two distinct ways. In early models [GREEN69, MCCAR69], separate frame assertions were required for each predicate, e.g., using Green's representations,

if $CLEAR(z, s)$ *and* $DIFF(z, x)$ *and* $DIFF(z, y)$ *then*
 $CLEAR(z, DO(PUTON(x, y), s))$

This expresses the fact that if z is clear in state s and the $PUTON(x, y)$ action was taken in state s involving two other distinct objects, then z will be clear in the new state $DO(\dots)$ that results from the action. It was observed that this was not particularly efficient if there are a large number of predicates, and this repre-

sensation could be improved by using terms to specify relations between states and relations for those states [KOWAL85]. Using the predicates and terms described by [KOWAL85], we can illustrate how this improvement is implemented. For example, $HOLDS(ON(A,B),s_0)$ is the assertion that the relation $ON(A,B)$ holds in state s_0 . States were also represented as the composition of terms, e.g., $RESULT(w,(PUTON(x,y)))$ is the new state that results after putting x on y in state w . Frame assertions were stated using these terms [KOWAL85],

$$HOLDS(p, RESULT(q, PUTON(x,y)) \text{ if } HOLDS(p,q) \text{ and } DIFF(p, CLEAR(y)).$$

The frame assertion also includes what can be deleted from the relations, i.e., $CLEAR(y)$. The control strategy, i.e., how the search is directed, was approached in several different ways. Top-down strategies began with the goal state and reasoned towards the start state. Bottom-up strategies begin with the start state and reasoned forward to the goal state. Both were computationally infeasible for problems of any significant size because all possible operations are considered at each stage until a sequence of operations produces the desired goal.

A derivative approach incorporating both assertional representation with a procedurally-oriented search mechanism evolved as the STRIPS family of planners [NILSS71]. Most of the *post-logic* generation of planners have evolved from these models [ALLEN90].

The STRIPS model used preconditions and postconditions to handle frame assertions. Its basic framework includes a knowledge base and rules represented as op-

erations each having three parts: precondition, add-to-list, delete-from-list. The precondition specifies what has to hold before the operation can be performed, whereas the latter two comprise the postconditions, indicating changes in the knowledge base resulting from the operation. No state variables are required, and the operations themselves provide updated information in lieu of frame assertions. For example, the operation, *MOVE(A,B)* would be defined as follows:

MOVE(A,B)

preconditions: CLEAR(A), CLEAR(B)

add-to-list: ON(A,B)

delete-from-list: CLEAR(B)

Given a start configuration and a goal state, this model attempts theorem-proving, and whenever failure ensues, selects the application of the rules so as to reduce the difference between each state and the goal state. STRIPS also included the ability to generalize rules, adding learning albeit limited. With the use of macro-actions developed in subsequent models, the generalization becomes more robust.

2.2.2 Limitations of classic approaches and their derivatives

The STRIPS family of planners and their extensions handle the ordering of subgoals more efficiently than the traditional logic-based approaches. On the other hand, the intuitive formalism and deductive capacity of the logic-based approach is lost in the procedural processing of subgoals. In addition, neither approach has the inherent expressive capability to represent or reason about positional information.

§ 2.3 The Frame Problem

2.3.1 Definition and examples

The frame problem is probably the most central feature in the development of planners. It represents the difficulty associated with reasoning with confidence about the effects of actions. While the description of relations among objects that are affected by an action are simple enough to relate, the problem emerges in the recognition and ability of a planner to reason about the indirect effects of an action. Researchers disagree as to a precise definition of the frame problem. Initially the frame problem was described as the ability to formulate the effects of actions over all relations in a computationally efficient way [MCCAR69]. Subsequent investigation has broadened the scope of the frame problem as more than just a "combinatorics" problem. Extensions include the perception of the frame problem as one of persistence – the period for which a given property or relation persists [MCDER82].

In the context of the proposed planning model, the significance of the frame problem stems from the ability to reason about the effects of continuous activity. The implications for concurrent activity also are relevant. Additionally when work-cell error occurs, the handling of the frame problem provides the basis for plan reformulation. It provides a description of the knowledge base at some time t when plan modification is needed. Such inferences cannot be drawn based on discrete states and under these circumstances, the combinatorics approach is unrealistic. The

persistence property is closer to the dynamics of motion-planning. While a direct application of this approach is difficult in the context of continuous state changes, the proposed planner implicitly uses persistence properties with respect to descriptions of some of the underlying relations among the objects and the arms in the workcell with dynamically updated posets. At the same time, the automated reasoning context requires explicit frame axioms to describe the effects of motion in terms of position, location, direction and “existence” of objects in the knowledge base.

§ 2.4 Concurrent Planning

2.4.1 Description

In the most general sense, planning for concurrent activity admits the existence of uncertainty to account for unknown causal forces. A distinction must be made between independent agents functioning concurrently and cooperating agents driven by the same reasoning program. Independent agents give rise to problems associated with external causal activities and introduce uncertainty with respect to the frame problem. Planning based on closed world assumptions assumes that 1) only agents can act; and 2) all possible actions of the agents can be inferred or deduced from the planner[GEORG87].

2.4.2 General Strategies

Concurrent planning involves strategies that permit the exploration of more than one activity at a time. The output is not necessarily partitioned among multiple agents

[SACER75b, WALD77]. Planners that are successful with the anomaly described previously require the ability to explore the parallel generation of potential tasks which can then be output in a linear sequence as in NOAH [SACER75b]. Since the resulting plan is for a single robot, concurrent activity is not necessarily possible among the potential tasks at a given point for these planners. Common strategies for concurrent planners include network-based models, posets, PERT diagrams or more general directed graphs. These structures are useful in preserving the underlying relationships among subgoals or tasks. Non-comparability among the tasks or subgoals describe potential concurrent activity.

A problem associated with many concurrent planners is the treatment of time, and its effects on concurrent activity. Such planners are based on discrete units, states, or snapshots, the set of which fully describes changes to the knowledge base. They do not include the ability to make inferences based on intermediate times between states. Some planners overcome these limitations by introducing interval-based or duration-based descriptions and strategies as the basis for describing concurrent activity and its effects [MCDER87, ALLEN84]. Actions are assigned implicit or explicit durations, and intervals are associated with the actions of the cooperating agents. Most are inadequate for application to complex domains. The formal systems described are inadequate for robot applications in part due to their inability to make use of positional representations of the robots and objects in the workcell.

One approach for robot-planning applications is based on continuous state changes as a basis for concurrent activity [VERE83, SALTER83]. Few models, however, are applicable for implementation in actual workcells. Vere developed an imbedded planner based on durations and windows for concurrent activity applicable to a wide class of problems, however, the planner lacks the ability to reason about positional representations and constraints in that context [VERE83]. One example of a plan model using continuous domains that is applicable to robot-planning problems is [NAGAT88]. The model, designed for factory assembly workcells, consists of two subsystems, PLAMAT which generates candidate plans and SYDAMUC which subjects those plans to collision constraints. The overall efficiency of the model is hampered by excluding collision detection and avoidance constraints from the primary planner [NAGAT88].

The proposed planning model is intended to explore task-oriented motion for arbitrary workcell configurations. Unlike the Nagata model, the actions need not fully express goal-oriented behavior. Instead the intent is to explore some of the special considerations of arbitrary clusters such as generating paths in cluttered areas and describing and generating relations based on regional boundaries associated with objects in the workcell. The underlying relations will implicitly exclude object/object collisions, and testing for other forms of collisions will be embedded at the planning level.

Finally the assignment of tasks among multiple robots can be done in several

ways which inform the planning process. In one approach, all robots in the cluster begin a new task at the same time and no robot begins another task until all robots have completed their current task. This approach is used in the Nagata model and simplifies the planning process. While it also results in idle time for some of the robots, it may be a desirable approach if optimization criteria other than minimum time are specified. It permits the identification of a unique poset for each task stage. The second approach, in which each robot begins a new task upon completion of the current one is more complex at the planning level. The dynamically generated posets then must reflect underlying relations resulting from task "ranges" or intervals associated with each arm. A third approach is assignment-dependent. Based on the partitioning constraints, arms may be idled if there is no given task that can be performed at that stage.

CHAPTER 3

ROBOT INTERFACES FOR PLANNING

§ 3.1 Overview

The proposed model is intended for use with actual robot systems. Implementation of such planners require specifications for the arms, workcell and objects to be manipulated. There are also various levels at which these specifications are required. The intent of the proposed planner is to be as generic as possible with respect to typical robot arms, recognizing the need for, and incorporating procedural interfaces for use with specific systems. This section addresses some of the basic needs at the robot planning level for the path-planning model. Specific requirements include: 1) the translation from the gross grid-based coordinates to the continuous, reference coordinates for the robot system; 2) path-determination in the plane; and 3) constraint considerations.

§ 3.2 Components

A robot system typically consists of a robot arm, a controller, a teach pendant and a communications port for a host computer. A robot arm consists of a series of rigid links connected by joints [FU87]. The joints may be revolute, i.e., rotate about

an axis, or prismatic, i.e., translate along an axis. The location of the tool or end effector may be described in terms of a joint variable vector, $\langle q_1, q_2, \dots, q_6 \rangle^T$, in which q_i represents the relative rotation or translation of joint i along its axis. For many practical uses, however, this representation is insufficient. It is frequently desirable to 1) determine the location of the end effector in Cartesian coordinates with respect to a fixed coordinate reference system; and 2) determine the joint variable vector required to move to a given point, (x_i, y_i, z_i) . These problems are known respectively as the direct kinematics problem (DKP) and the indirect kinematics problem (IKP) [FU87].

§ 3.3 Path-determination in the planar workcell

Path-planning is a part of the overall task-planning. In this investigation, the planar workcell used by the planner will be in terms of a grid-based coordinate system. If a single arm is engaged in path-planning, its base coordinate system serves as the reference point for objects in the plane as well as their motion. If more than one robot is involved in a task, one approach is to use a global coordinate system for the workcell, since each arm has a different base coordinate system. Positional information for each object may then be recorded in terms of global coordinates. These coordinates represent the center of mass of the object. When a given arm has the task of moving a specific object, the location and destination of the object are translated from the global coordinates to the reference coordinates for that specific arm. The center of the planar workcell is the origin of the global coordinate

system as well as the grid-based system. While such a representation simplifies the implementation of the model, it restricts the distance an object travels in the plane to nd where n is a positive integer and d is the length of a cell.

The simplest approach to straight-line path-planning is interpolating over cartesian coordinates. This is computationally intensive, however, and may result in jerky motion for the arm. Robots with less than six degrees of freedom are kinematically "incomplete" and cannot be made to follow arbitrary paths. Bounded linear deviation is a method that can be applied in such cases. The resulting straight-line path is a series of interpolated points in joint space. The approach used for testing is based on an adaptation of Taylor's original algorithm for bounded linear deviation [SCHILL90].

§ 3.4 Additional constraints and considerations

Additional considerations at the robot-planning level include the definition and determination of collisions, geometric representations of the robot arms and objects, and the actual workcell. There are basically three types of collisions that may occur: 1) tool-tip/object collisions; 2) tool-tip/tool-tip collisions and 3) arm/arm collisions. Tool-tip/object collisions occur when the path travelled by the tool-tip contains a set of points where a stationary object is located. Our simulation of motion in the plane, however, actually positions the tool-tip slightly above the objects. This restriction effectively avoids tool-tip/object collisions. The potential paths generated are still, however, subject to the latter two types of collisions.

For tool-tip/tool-tip collisions, the principal concern is whether the paths under consideration, represented as vectors, intersect at some time, t_i . A secondary consideration is the avoidance of object/object collision as two objects are being moved through the plane, which is based on the spatial dimensions and orientations of the objects being moved. In both cases, the planner together with its requisite procedural interfaces handles these constraints.

Arm collisions are distinct from path-collisions. The proposed problem assumes that no arm travels directly above or below another. This is a reasonable constraint, since arm failure in such an instance may result in arm collision and possible damage. At the planning level, for the robot cluster involved, the arm is described by its rectangular projection on the plane. The rectangles formed by the arms as a function of time are used for the determination of arm collisions.

In task-planning, geometric representations of objects within the workcell, are an acknowledged area of difficulty. While the location of an object with respect to the tool tip (for grasping) is relatively simple to determine, the geometric descriptions are vital to enabling the movement of objects throughout the workcell. Real world workcell situations are comprised of irregular objects whose physical descriptions require complex mapping. The extraction of this information in a robot planner is non-trivial. Since the emphasis of the proposed planner is on path-planning, regular polygons, e.g. squares, triangles, rectangles, etc. are used in the workcell in order to simplify their description at the planning level. The actual planner was tested using

squares but is sufficiently extensible to handle other simple polygons. In addition, for this restricted model, irregular objects can be described in terms of their rectangular dimensions. The position of an object is described in terms of the real workcell coordinates and the associated mapping in the grid. The referential position is the center of mass of the object.

The actual workcell is a rectangular region in which the arms and the objects are located. The bases for the robot arms are located on the boundaries of the region. This is done to prevent possible entanglements of the cables connecting each arm to its controller. Because the location of an object is its center of mass, object motion through the plane is simulated in the workcell. Each object has, located at its center, a grasping part, which its coordinates reference. A *MOVE* operation at the workcell level has several components - *MOVETO*, *LOWER*, *GRASP*, and *RAISE*. For example, to move an object whose location is (x_i, y_i, z_i) , the arm travels first to the position $(x_i, y_i, z_i + c)$, where c places the tool-tip slightly above the grasping portion and is constant. This is typically done in order to prevent tool-tip collision with the grasping portion if the tool-tip is not properly oriented. *LOWER* is an operation that permits the tool-tip to move to the desired location with the proper orientation, a translation along the z -axis. In the case of this workcell, *LOWER* has no arguments since c is constant and implied. *GRASP* closes the tool-tip at the grasping portion, and *RAISE* is used for frictionless motion in the plane and again the implied argument is c . The actual object and tool-tip motion in the workcell as described, is not reflected

in the proposed planner. This was done principally so as to formulate a path-planning model that is sufficiently general for eventual extension to 3-D path-planning.

CHAPTER 4

DIRECTIONAL POSETS AND PATH-PLANNING

§ 4.1 Motivation

Underlying the problem of path-planning is the need for a control strategy in which the structural properties of the configuration motivates the search. As noted in Chapter 2, the results of a planner are typically a linear order of actions. This linear ordering is usually derived from a poset of possible actions with ordering constraints. The poset indicates what actions must be performed sequentially and which may be performed in parallel. The requirement, however, is to formulate suitable poset models applicable to path-planning problems. Research in m -directional orders in general and bi-directional orders in particular provided the basis for the directional poset model of this investigation [RIVAL87, THUK91]. The sections that follow provide background and definitions of previous research, some extensions of these results, and a characterization of directional posets and how they can be used as a basis for path-planning.

§ 4.2 Background

4.2.1 Definitions

The following definitions are from [RIVAL87, THUK91]. Given a relation, R , each element in the underlying ground set is assigned a motion, or one of m possible directions along which it will move. A blocking relation is defined as follows: if point A is blocked by point B , then B lies along the direction assigned to A . An m -directional representation of a relation, R , is a ground set whose blocking relation is consistent with the ordering of R . The blocking relation is defined as reflexive and transitive. Given a finite relation, R , a subdivision with an m -directional point representation may be induced [RIVAL87, THUK91]. The resulting m -directional representation has a ground set consisting of the original elements of P together with additional points joined to edges in R . These added points are the subdivisions and represent changes in the direction of a given point.

4.2.2 Bi-directional orders — definitions and results

The following definitions and results are adapted from [THUK91]. A bi-directional ordered set is a relation in which one of two possible directions is assigned to each element. Since an ordered set with a bi-directional point representation has one when the directions are orthogonal, the following assume that the possible directions are north, (N) and east, (E), as in [THUK91].

Any tree with at most two maximal lower covers has a bi-directional point representation. This implies that a relation with either a directed cycle or an element with more than two lower covers does not have a bi-directional point representation. As an example, consider the relation,

$$\{a < b, b < c, c < a\}$$

consisting of a directed cycle. Assume that a is assigned the direction, $d1$ then b can be assigned either of two directions, $d1$ or $d2$. If b is assigned $d1$, then c lies along the same line as a , and there is no possible directional assignment for c such that $c < a$ since $a < c$. If b is assigned $d2$, then c does not share an x or y coordinate in common with a and no directional assignment is possible such that $c < a$.

A poset with a simple cycle has no bi-directional point representation. (A *simple* cycle refers to the underlying *undirected* graph.) Given at most one subdivision along some of its edges, a bi-directional point representation can be obtained based on the initial ordered set and the additional subdivisions.

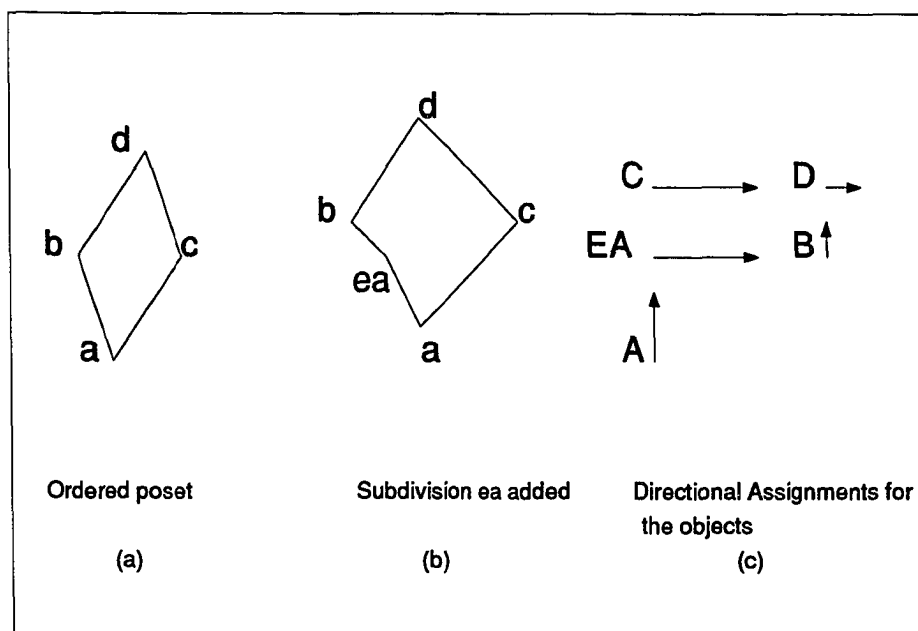


Figure 4.2.1 Relations, subdivisions, and directional assignments

As an example, consider the poset shown in figure 4.2.1a, in which the underlying simple graph has a cycle. There is no bi-directional point representation for this set. Since b and c are lower covers of d and are non-comparable, they do not have a common x or y coordinate, otherwise comparability would be implied. On the other hand, since a is a lower cover of both b and c , this implies that b and c lie along the same path as a which implies that they do share a common x or y coordinate. A contradiction results. By adding a single subdivision, ea , the resulting poset has a bi-directional point representation, as shown in figure 4.2.1b.

Finally, for some ordered sets in which each element has at most two lower covers, [THUK91] shows that bi-directional point representations may be obtained

by adding subdivisions to almost of the edges. In addition, for some ordered sets if every edge has a subdivision, no bi-directional point representation is possible.

§ 4.3 Expansions of Bi-directional Posets

We can provide a simple extension to Thuk-Air's results in which he shows that bi-directional relations are posets. Given a poset, P , it can be shown that even when the maximal lower cover of an element is greater than two, *expansions* can be used in order to generate a bi-directional poset consisting of P joined with E , where E is set of expansions. The expansions are "psuedo"-points in much the same way as the subdivisions characterized by Rival et al. As an example, consider the ordered set,

$$\{a < d, b < d, c < d\}.$$

As given, a two-directional assignment is not possible. Two nodes in the lower cover of d which are non-comparable would have the same directional assignment. Since they are both blocked by d , this implies one would be blocked by the other and therefore they must be comparable. It is possible, however, to generate a set S , which is a superset of P which has a two-directional point representation. Assume that P contains no simple cycles. The expansions can be generated as follows:

1. For a given P , let e_j be a node with more than two lower covers.
2. Let k be the total number of lower covers of e_j .
3. Partition the lower covers of e_j into the sets S_2 and S_i where

$$|S_2| = 1$$

$$|S_i| = k-1.$$

4. Add a new element , e_i , so that

$$e_i < e_j,$$

$$(S_2) < e_j$$

$$(S_i) < e_i,$$

where $(S_2) < e_j$ means that each of the elements in S_2 is a lower cover of e_j .

5. Select an element in S_i to be associated with e_i .
6. Recursively apply the process on e_i until an element, e_j , is reached whose maximal cover is less than two.

Figure 4.3.2 illustrates the process for a given node a , and figure 4.3.3 illustrates the resulting two-directional representation.

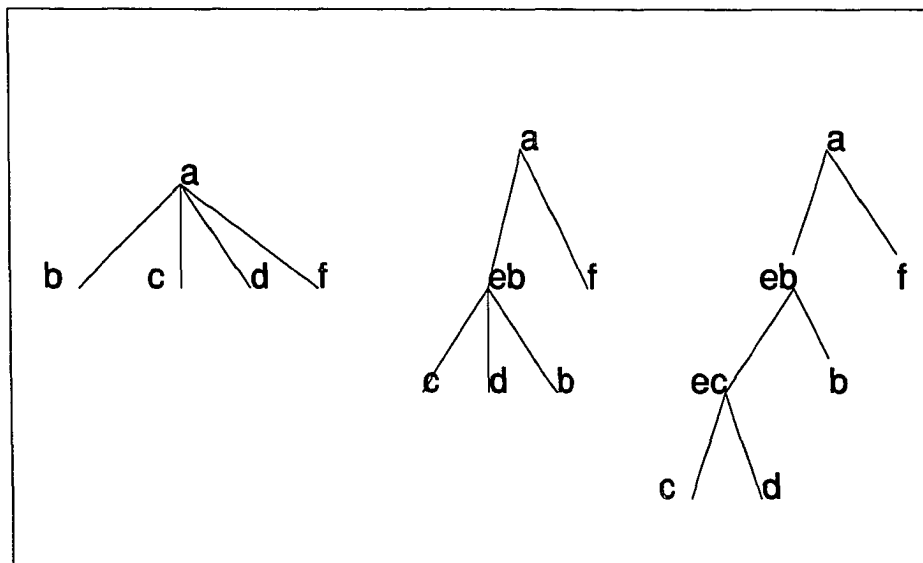


Figure 4.3.2 Adding expansions to a poset

The effect of the expansions are similar to those of subdivisions in that they represent a change in direction for an element and the underlying ordering is maintained, although the formulation is different and restricted to posets. In addition, unlike subdivisions, there is a precise method for generating expansions and the number of required expansions can be determined for a given m and a given poset. If $n(e)$ is the number of maximal lower covers of an element, e , and the nodes whose lower covers are greater than two are represented as $e_1, e_2, \dots, e_k$, where k is the number of such nodes, then the number of required expansions will be equal to:

$$\sum_{i=1}^k (n(e_i) - 2)$$

This expansion process has implications for k -directional assignments for posets for any finite k , and provides a basis for investigating the formulation of m -directional posets, where $m < k$.

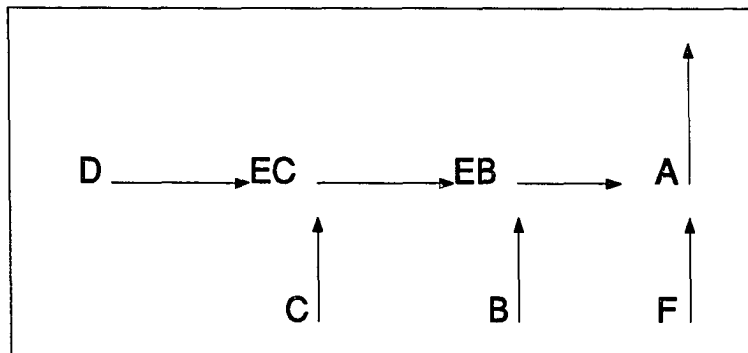


Figure 4.3.3 Direction assignments resulting from an expansion

§ 4.4 Directional Posets

4.4.1 Definition

While [RIVAL87, THUK91] provide the motivation for modeling the structural character of configurations, we offer a more refined definition and distinctions in terms of the base relations associated with direction assignments. The characterization of directional posets in this section is intended to provide a firmer theoretical basis for a high-level path-planning problems. A directional poset is defined as a poset in for which the underlying elements in the ground set have been assigned directions along which they will travel.

We define, however, *two different sets of relations on the elements*. One is the blocking relation and the second is the relation that characterizes the poset. As we shall see in Chapter 4, this duality is necessary in order to manipulate directional posets for control purposes. The blocking relation is defined by the following two rules:

1. If element B lies along the direction in which element A is assigned, then A is blocked by B ,

$$A <_d B$$

where d is the direction assigned to A

2. If $A <_{d1} B$ and $B <_{d2} C$ and $d1 \neq d2$ then $\neg(A <_{d1} C)$

In other words, transitivity can only be implied by common directional assignments, otherwise non-transitivity is explicitly implied.

From the blocking relations defined above, the relations for the directional poset are given as follows:

$$1. A <_d B \longrightarrow A < B$$

$$2. A <_{d1} B \text{ and } B <_{d2} C \longrightarrow A < C$$

Consistent with the definition of a poset, these relations are transitive. Clearly, the ordered pairs for the blocking relation are a subset of the ordered pairs for the directional poset relation. They are reflexive and anti-symmetric as well.

We can show other properties as well, in terms of the relationship between the blocking relation and the directional relations. They are essential because we will use only the blocking relation (as we have defined it) to characterize completely the associated directional poset.

4.4.2 Properties

While the blocking relation as described in the previous section forms a subset of the relations in the directional poset, it can be used to describe properties associated

with the directional poset. This is not in general true for posets, but flows from the relationship between the blocking precedences and the direction precedences. Let B be the relation formed by the blocking precedences and let P be the relation formed by the directional precedences. The following properties can be shown:

1. if y is a maximal (minimal) element in B then y is a maximal (minimal) element in P .
2. if y is a maximal lower cover (minimal upper cover) of z in B then y is a maximal lower cover (minimal upper cover) of z in P .

Lemma: Given a blocking relation, B , and its corresponding direction poset, P , if y is a maximal element in B then y is a maximal element in P .

Proof: Assume that y is a maximal element in B and not a maximal element in P , then

$$\forall z \in S$$

either $z <_d y$ where d is in $\{d1, \dots, dk\}$

or z and y are non-comparable.

if y is not a maximal element in P , then there is a relation in P ,

$$y < z$$

which implies a sequence of relations in B ,

$$y <_{d1} \dots <_{d1} z$$

but y is a maximal element in B , a contradiction.

Lemma: Given a blocking relation, B , and its corresponding direction poset, P ,

let x and y be distinct members of the underlying ground set. If y is a maximal lower cover of z in B , then y is a maximal lower cover of z in P .

Proof: Assume that y is a maximal lower cover of z in B and not a maximal lower cover of z in P , then

$y <_d z$ is in B and there is no x such that $x <_d z$ and $y <_d x$

If y is not a lower cover of z in P , then the following relations must exist in P :

$y < a \dots a < z$ where $a \neq y$

This implies that there is a relation in B ,

$y <_{d1} a$

case1: $d1 \neq d$,

$y <_d z$ and $y <_{d1} a$, but y is assigned a unique direction so $d = d1$, a contradiction.

case2: $d1 = d$

$y <_d a$, $y <_d z$ and $a <_d z$ or $a <_{d2} z$ where $d \neq d2$

if $a <_d z$ and $y <_d a$ then $a <_d z$ is in B and y is not a maximal lower cover of z in B , a contradiction.

if $a <_{d2} z$ and $y <_d a$ since $y <_d z$, a lies on the line adjoining y and z . If a is blocked by z , then the only possible directional assignment is d , in which case y is again not a maximal lower cover of z in B , a contradiction.

§ 4.5 k_2 -directional posets

Our investigation concerns itself with a particular class of k -directional posets

which we call k_2 -directional posets. These are directional posets in which k is finite and there are $k/2$ distinct directions. The remaining $k/2$ directions are all inverse direction vectors of the original directions. These posets have particular properties. The maximum number of maximal lower covers of an element (a point) is $k/2$ since $k/2 + 1$ lower covers would violate the poset condition. In particular our investigation focuses on $quad_2$ -directional posets in which the directions are the planar unit vectors. Given a set of points in the plane, the goal is to assign one of these four directions while maintaining a poset. As points are moved through the plane, the goal is to generate dynamic posets in which relations reflect the motion, and new directions are assigned to points at rest. This is the basis for initial poset construction as well as regenerating the poset after multiple points have moved.

Theorem: *Given a $quad_2$ -directional poset consisting of k points. If a $k+1$ point is added to the ground set, there exists at least one directional assignment for that element such that the directional poset property is maintained.*

Definition: Given a poset, P , and an element, x , in the underlying ground set, the lower cover, y , of x is defined as:

$$\{y | y < x \text{ and } \neg \exists z (z < x \text{ and } y < z)\}$$

Observation: If a cycle results when an element, x , is assigned a direction, then the cycle consists of:

$$y_1 <_{d1} \dots <_{dk} x <_{dk2} \dots <_{dk3} y_1 \text{ which implies that } y_1 < x \text{ and } x < y_1$$

which violates the antisymmetry property.

Proof: Assume that there is no directional assignment for the element. Since this is a $quad_2$ poset, then the maximal lower cover of an element is two.

case 1: x has no lower cover. Then

$$\neg \exists y (y < x)$$

Therefore no directional assignment can make x part of a cycle and any direction may be assigned.

case 2: x has one lower cover, y . Let d be the direction assigned to y . Then $y <_d x$. Assign the same direction to x , then if there is a cycle:

$$y <_d x <_d y_1 <_{d_2} \dots <_{d_3} y$$

Since y and y_1 are in the original poset, this implies $y <_d y_1 <_{d_2} \dots <_{d_3} y$ or $y <_{y_1} y_1$ and $y_1 < y$ which is a contradiction. Therefore x can be assigned the direction, d , of its lower cover.

case 3: x has two lower covers, y_1 and y_2 with directional assignments d_1 and d_2 respectively. By case 2, if we assign d_1 to x , it cannot produce a cycle with y_1 , so it must produce a cycle with y_2 ..:

$$y_2 <_{d_2} x <_{d_1} a <_{d_i} \dots <_{d_k} y_2$$

Since $y_1 <_{d_1} x$ then $y_1 <_{d_1} a$ since the blocking relation is transitive for the same direction. Then we have:

$y_1 <_{d1} a$ and $a <_{d2} y_2$ which implies $y_1 < y_2$, but y_1 and y_2 are non-comparable, so this is a contradiction. A similar argument holds if d_2 is assigned to x , therefore x can be assigned either d_1 or d_2 .

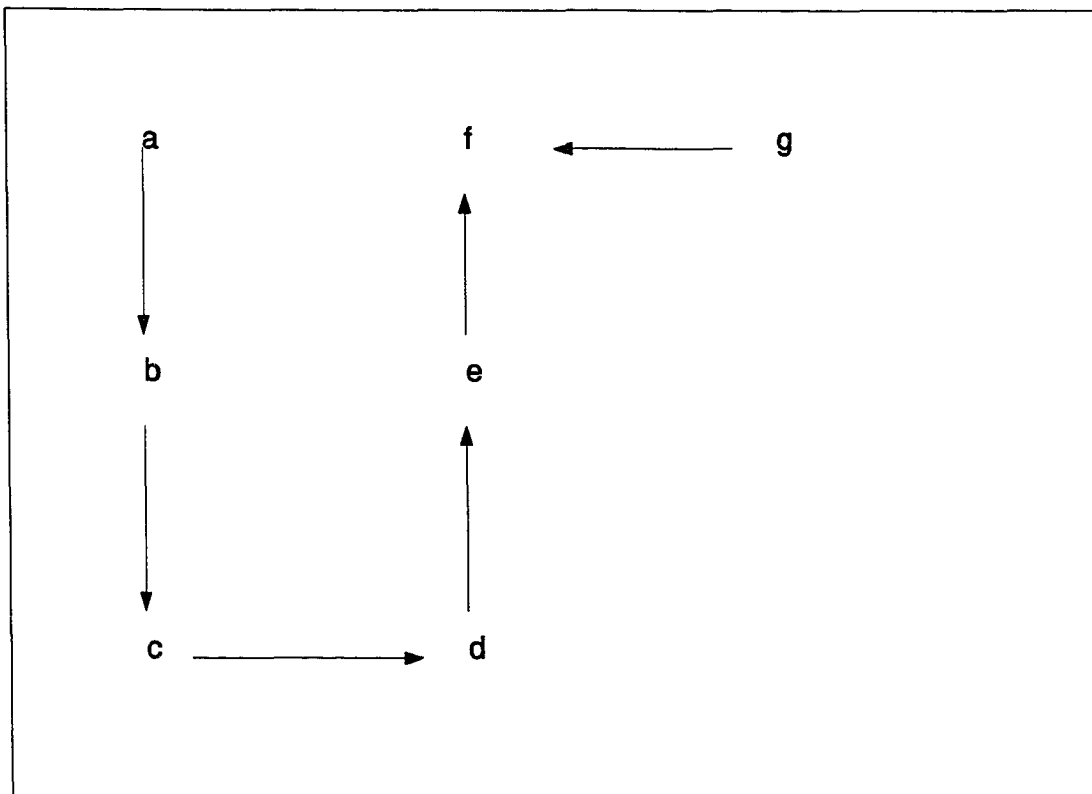


Figure 4.5.4 Adding a point to the directional poset

Figure 4.5.4 illustrates this theorem. Point f is the element whose direction is to be assigned. Note that its only maximal lower cover is e . (In the directional poset, $g < e$.) f is then assigned the same direction as e .

Finally, given a quad₂-directional poset in which the two distinct directions are orthogonal and the remaining two are inverse directions of these, such a poset can be represented as a two-directional poset as well. As a result, there are no underlying simple cycles in this quad₂-directional poset.

§ 4.6 *Straight-line motion in the plane*

4.6.1 Straight-line motion and posets

In relatively uncluttered configurations, objects may be moved to their goal locations in straight-line paths, i.e., the path is the direction vector beginning at the object's location and terminating at its destination. Typically, even these configurations require some ordering constraints. Not every object may have an obstacle-free path to its location, however, as other objects are moved, such a path may become available.

In this investigation, the ordering constraints are defined with respect to the goal locations for straight-line motion. A primary concern is that some objects must be placed before others, otherwise there would be no clear straight-line path to their destinations. In some cases, there may not be a path at all to an object's destination. For example, in figure 4.6.5, representing objects and their goal locations, object *C* must be moved to its destination before object *B*.

Various strategies exist for determining the order of placement of the objects. We define a set of *goal* relations for object placement. An object, *A*, precedes an

object B , denoted $A < B$, if B 's *goal* location lies on the direction vector from A to its goal. For example, in figure 4.6.5, the set of goal relations would be:

$$\{ C < B, D < A, B < A \}$$

These goal relations are reflexive and transitive. If there are no cycles, they form a poset. Figure 4.6.6 shows the poset induced by the goal relations of the objects in figure 4.6.5. It is readily apparent that a necessary condition to achieve straight-line motion for all objects in a given configuration is that the relations form a poset. If there is a cycle, at least one object will not be able to move to its goal destination along a straight-line path, as in figure 4.6.7.

The goal poset condition, however, is not a sufficient one. We have omitted thus far any discussion of obstacle precedences, i.e. an object is blocked by other obstacles in its straight-line path, as in figure 4.6.8. A second relation may be induced by considering the direction assignments in terms of our previously discussed blocking relation. This indicates which objects block other objects. In this case, the set of possible directions is infinite. Now we can assert that if the goal relations form a poset, then straight-line motion is possible,

iff $\forall x, y \in S$,

if $x < y$ is in the goal poset, then

in the direction relations, either:

$y < x$ or x and y are non-comparable.

This provides a basis for selecting suitable motion-planning strategies for given

configurations based on the properties of the properties of the goal and direction relations.

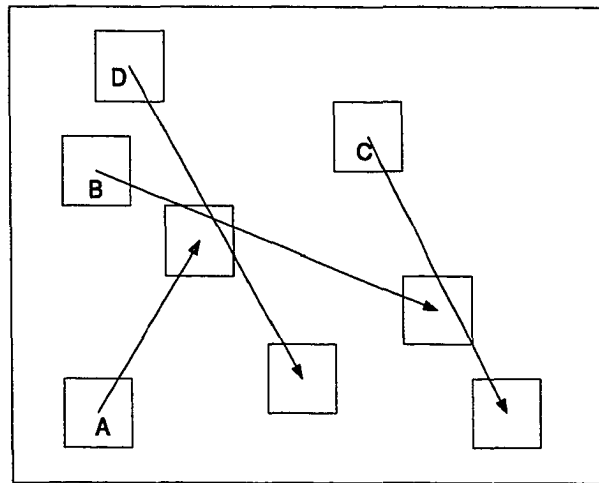


Figure 4.6.5 Objects and their goal locations

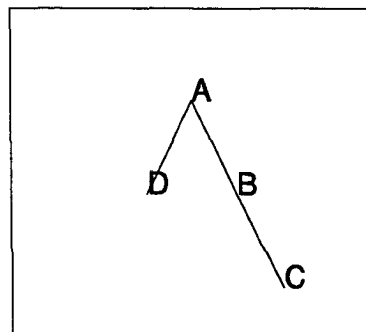


Figure 4.6.6 Poset Induced by objects and locations

4.6.2 *Using the goal poset*

Once the goal poset has been generated, objects can be moved by selecting minimal elements in turn and moving the first one for which a straight-line is clear. For more than one robot, all minimal elements for a given stage can be generated and assigned to the arms subject to collision and obstacle-avoidance constraints. If there are k arms and 1) less than k minimal elements in the goal poset or 2) less than k minimal elements with a obstacle-free or collision-free path, then idle the remaining arms.

Additionally, for straight-line planning, some consideration must be given to the dimensions of the objects being moved. In our case, simple objects of cell dimensions are being transferred. However, when they are moved, they form a rectangular path through the grid. As a result, the precedences that are generated for straight-line motion must consider adjacent cells as needed.

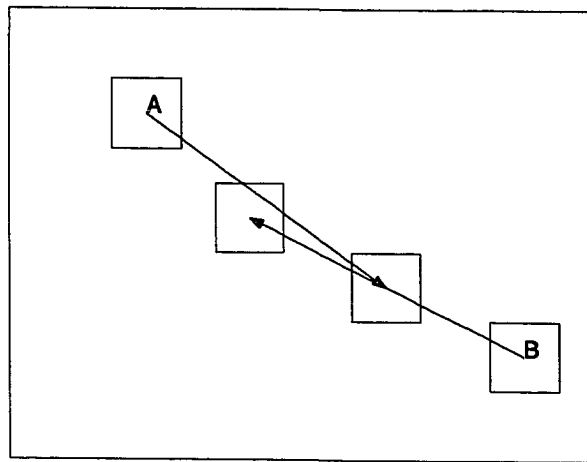


Figure 4.6.7 Goal destinations resulting in cycles

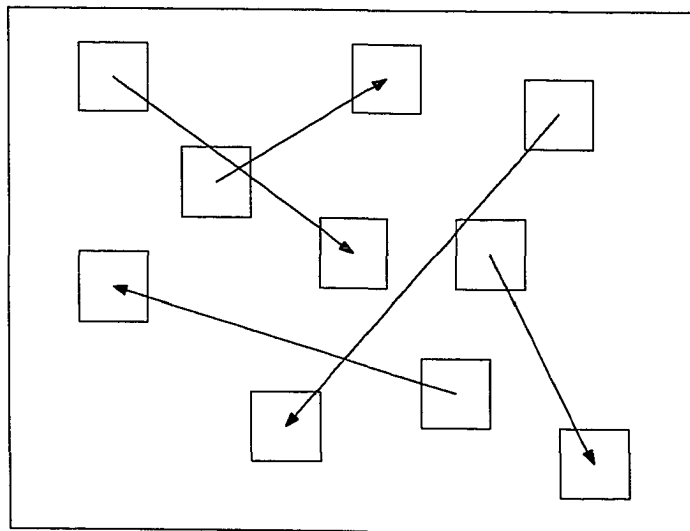


Figure 4.6.8 Cluttered workspace

4.6.3 Summary

In this chapter, we have formulated a structural mechanism, direction posets, for path-planning in the plane with applications for both cluttered and uncluttered configurations. The motivation for exploring direction posets was to use them as a basis for both selection strategies in planning and exploring concurrent activity among robot clusters.

Implicit in this discussion is the analysis with respect to elements that are points. When applied to possible objects in the plane, certain properties are no longer valid. While basic properties of quad₂-direction posets still hold, varying geometric dimensions of objects will have implications in terms of the maximal lower covers of objects. In addition, rotation and orientation strategies should play a role in the path-planning on this level. In this investigation we choose to restrict the objects in the plane, to those whose dimensions are no greater than a single grid-cell, in an attempt to explore more fully the application of direction to path-planning. At the the same time, all objects in the plane are assumed to be moveable, we have not explored the possibility of fixed obstacles in the plane.

While our initial interest in direction posets is with respect to goal-oriented motion, we choose instead to implement the separability problem cited by [THUK91] for both bi-direction and quad₂-direction posets.

CHAPTER 5

AUTOMATED REASONING AND POSET MANIPULATION

§ 5.1 Introduction

The path-planning models demonstrated have been implemented using clause-based reasoning. Before discussing the models in detail, some background on logic-programming will be provided. In addition, there are persistent similarities among the models and techniques used, also in the explicit and implicit representations of posets, and the overall strategy in terms of how the programs were driven. These will be discussed in this chapter as well.

Clause-based reasoning is based on the logic programming paradigm. It differs from other types of programming in that it is declarative in nature and non-procedural. A set of assumptions is given as true, and queries can be proven based on these assumptions. Using logic programming requires first of all the ability to map a given problem and its domain onto first-order predicate calculus. This process can be non-trivial for problems based on higher-order formulations, such as sets, graphs and matroids.

All statements are translated into clausal form or conjunctive normal form (CNF). A clause consists of literals separated by disjunctions and CNF is the

conjunction of clauses. All variables in the resulting clauses are implicitly universally quantified. Existential quantification is handled by using Skolem constants or functions [AMBLE87].

Once the problem and its conditions have been converted to CNF, the problem can be solved using resolution theorem-proving, a two-fold process consisting of unification and resolution. Unification is the process to making two expression identical by identifier substitution [AMBLE87]. A simple example would be the following:

All people are mortal $Person(x) \rightarrow Mortal(x)$

Socrates is a person $Person(Socrates)$

In order to show that Socrates is mortal, the variable x in the first statement is instantiated to Socrates. Often unification requires substitutions in both expressions. The resolution step combines unification with the repeated application of a simple rule of inference which eliminates clauses. The rule of inference, generally stated, is as follows:

$\neg P(x) \quad Q(x)$

$P(x)$

conclude $Q(x)$

Resolution theorem-proving is refutation-based. The goal is negated, and resolution is repeatedly applied until the empty clause is derived. If the conclusion is true, then its negation will make the clause set inconsistent. The completeness of the

refutation-based nature of resolution theorem-proving was established by Herbrand [NILSS88].

Given the simplicity of the resolution rule, mechanical theorem-proving is less an intuitive proof technique and more just plain pattern-matching. Only the search procedure and strategy dictate how many steps are required to deduce the empty clause for a given problem. The process in NP-complete, and mechanical theorem-provers become grossly inefficient as the number of clauses and literals increases.

The simple example of resolution given earlier is actually a specific form of resolution - hyperresolution. In general, a resolvent may be derived by hyperresolution from a set of clauses, only one of which may contain negative literals, and those are 'clashed' with positive literals in the other clauses to yield a clause consisting of one or more positive literals. For example, the following set of clauses:

$$\neg A(x,y) \mid \neg R(x) \mid R(y)$$

$$A(\text{con1}, \text{con2})$$

$$R(\text{con2})$$

where x and y are variables and con1 and con2 are constants, would produce the resolvent $R(\text{con2})$ by hyperresolution. A related inference rule, negative hyperresolution is similar to hyperresolution except the roles of positive and negative literals are reversed, and the result is a negative literal.

Other forms of resolution include binary resolution, UR-resolution and unit resolution [WOS92]. Binary resolution requires that only two clauses be involved in

the resolution, with no restrictions on positive and negative literals. UR-resolution requires that the resolvent of a set of clauses must be a unit clause. Unit resolution is a restriction of binary resolution in which one of the two clauses must be a unit clause.

Two additional, powerful rules of inference involve substitution over the equality predicate. They are paramodulation and demodulation [WOS92]. Paramodulation occurs between two clauses, from one clause containing an equality literal into another. Resulting clauses are added to the clause space. Demodulation is a form of equality substitution that is more restrictive than paramodulation. It is actually a form of rewriting and requires a unit-equality clause. Clauses generated by demodulation replace existing clauses. Various options are permitted for both demodulation and paramodulation. The direction may be specified as either left-to-right or right-to-left. Lex-dependent demodulators can be used. Some mechanical theorem-provers support backward demodulation, i.e., rewrite every previously generated clause as a result of an equality substitution.

Because mechanical theorem-provers lack the inherent capability to apply resolution rules intuitively, models based on them exploit a variety of features in order to control the search. These features include directedness and approach. Directedness is determined by specific search strategies which control the direction of inference rules. Examples of these include forward-reasoning models, backtracking models, etc. Several common search strategies include top-down, bottom-up and mixed.

The importance of considering possible inference rules and strategies stems from the need to develop appropriate models, capable of exploiting features of a given clause-based system.

§ 5.2 Domain Induction

The model relies heavily upon a very efficient domain induction technique which permits iteration over a linear ordering of elements. This technique was used in the investigation of automated reasoning and job-shop scheduling [WOS86]. Since a modification of it occurs in all of the examples presented, we will explain it in some detail. The following clauses are taken directly from [WOS86] and only predicate names have been changed for clarity.

Selecting a maximal element from a poset can be time-consuming as unnecessary clauses are generated and added to the clause space. The following technique reduces this overhead. Given a set of N elements, $e_1, \dots, e_n$, and relations on these elements, start with e_1 . If e_1 is not blocked by an element, e_2 , then e_1 is maximal so far, otherwise e_2 is maximal. Compare the maximal element to e_3 and continue until all the elements have been checked. This requires keeping track of the trial maximal elements as well as generating successive elements to test. The following clauses illustrate this technique:

if MAX_SO_FAR(x, z) & EQUAL(N, z) then MAXIMAL(x)

if MAX_SO_FAR(x,z) & GEN(z,y) & PREC(x,y)
then MAX_SO_FAR(y,z+1)
if MAX_SO_FAR(x,z) & GEN(z,y) & -PREC(x,y)
then MAX_SO_FAR(x,z+1)
if GET_MAX & P(p(z)) then GEN(z-1,p(z))
if GET_MAX & P(p(1)) then MAX_SO_FAR(p(1),1)

The *GEN* predicate generates every element in the poset and is used to iterate through the elements. The use of this technique requires a linear ordering of the elements in the ground set. The existence of elements, $P(p(1))$ through $P(p(n))$ was assumed in the previous discussion.

The previous clauses assume that every element remains in the poset. If, however, elements are deleted from the poset, the generators will not function properly. For example, if $P(p(1))$ is deleted from the set because it has been moved to its goal location, then the selection of the next maximal element will not occur. An additional set of clauses is included to reflect those elements no longer in the poset. The assertion that an element is in the poset is represented by $P(p(x))$, and similarly $\neg P(p(x))$ means that an element is no longer in the set. Instead of *GEN* being used to generate these elements, *NOT_GEN* is used to generate these to reflect that no precedences or *PREC* relations hold for them. The following are the additional clauses:

if NOT_MAX_SO_FAR(z) & EQUAL(N,z) then -MAXIMAL(x)

if GET_MAX & -P(p(1)) then NOT_MAX_SO_FAR(1);
if MAX_SO_FAR(x,z) & NOT_GEN(z,y) then MAX_SO_FAR(x,z+1)
if NOT_MAX_SO_FAR(x) & GEN(z,y) then MAX_SO_FAR(y,z+1)
if NOT_MAX_SO_FAR(x) & NOT_GEN(z,y) then NOT_MAX_SO_FAR(z+1)
if GET_MAX & -P(p(z)) then NOT_GEN(z-1, p(z))

One final addition to the above clauses is the need to reflect that precedences may be changing from one poset stage to another. An additional variable, analogous to the state variable, is required to indicate the current poset.

§ 5.3 The Blocking Relation

In order to use the domain induction technique in the previous section, we are required to explicitly generate *-BLOCKED_BY* relations. In a poset, $\neg(x < y)$ can mean either: 1) $y < x$ or 2) x and y are non-comparable. In a reasoning program, one way to derive this is by recursively descending through the lower covers of y until minimal elements are reached. If x is not seen then we can assert that $\neg(x < y)$. Alternatively we could also ascend through the upper covers of y . In either case, the process is time-consuming and many unnecessary clauses are added to the clause space for *each* such test. The induction technique requires that for every pair of elements in the ground set, i and j , there is either the assertion $i < j$ or $\neg i < j$. The recursive tests required would overshadow the gains obtained by using this induction technique.

As a result, throughout our examples, the blocking relation for directional posets is used instead of the directional relation because it allows us to infer the negated relations more efficiently. We have already shown that key properties for the poset can still be verified by using the more restricted blocking relation.

The induction technique, however, assumes that the underlying poset is completely described or inferred. This poses problems, however, since by our definition, $\neg BLOCKED_BY(A,B)$ means only that object A is not blocked by object B in the direction it is assigned. However, since transitivity holds in the associated poset, this does not necessarily imply $\neg PREC(A,B)$. To overcome this problem, we use a slightly less efficient version of domain induction. To test for the maximality of an object, $P(p(i),w)$, we check the $BLOCKED_BY$ and $\neg BLOCKED_BY$ relations of the objects, $P(p(1),w)$ through $P(p(n),w)$ where n is the number of objects. If there is another object, $P(p(j),w)$ which blocks $P(p(i),w)$ then that object becomes the new tentative choice. The clauses used are as follows:

if GETMAX(w) & P(p(1),w) then MAXSOFAR(p(1),1,w)
if GETMAX(w) & -P(p(1),w) then NOT_MAXSOFAR(1,w)
if MAXSOFAR(z,x,w) & GEN(x,y,w) & BLOCKED_BY(z,y,w) then NOTMAX(z,w)
if MAXSOFAR(z,x,w) & GEN(x,y,w) & -BLOCKED_BY(z,y,w) then MAXSOFAR(z,x+1,w)
if MAXSOFAR(z,x,w) & GEN(x,y,w) & (z=y) then MAXSOFAR(z,x+1,w)
if MAXSOFAR(z,x,w) & NOT_GEN(x,y,w) then MAXSOFAR(z,x+1,w)

if MAXSOFAR(z,x,w) & EQUAL(N,x,w) then MAXIMAL(z,w)
if NOT_MAXSOFAR(x,w) & GEN(x,y,w) then MAXSOFAR(y,x+1,w)
if NOT_MAXSOFAR(x,w) & NOT_GEN(x,y,w) then NOT_MAXSOFAR(x+1,w)
if NOT_MAXSOFAR(x,w) & EQUAL(N,x,w) then -MAXIMAL(x,w)
if NOTMAX(p(z),w) & GEN(z,y,w) then MAXSOFAR(y,1,w)
if NOTMAX(p(z),w) & NOT_GEN(z,y,w) then NOT_MAXSOFAR(z+1,w)
if GETMAX(w) & P(p(z),w) then GEN(z-1,p(z),w)
if GETMAX(w) & -P(p(z),w) then NOT_GEN(z-1,p(z),w)

§ 5.4 Implementation

The automated reasoning program selected for the implementation of the examples in the next chapter is the Interactive Theorem-Prover (ITP) [LUSK84]. ITP has been tested in a variety of applications including group theory, number theory, logic circuit design and real-time systems [WOS92].

ITP is interactive and permits the user control over the processing of newly-generated clauses. This interactive feature was instrumental in the development of our examples. ITP has many option families and includes the substitution options of paramodulation and demodulation. It also includes a wide range of resolution options such as UR, binary, hyperresolution, etc. Other available options include subsumption and weighting. The principal differences between ITP and Prolog, the more widely used tool for logic programming, are: 1) Prolog uses just one rule of

inference — negative hyperresolution, while ITP permits the user to select the rules of inference to be used; and 2) Prolog uses a depth-first search against the set of support, while ITP uses a breadth-first search. ITP proofs consider each clause in the set of support (by order of weight which is also user-customizable) and generate any new clauses from it based on the axioms, demodulators and rules of inference selected. The increased flexibility of ITP has a price. The clause space can become unwieldy and proofs are frequently much slower than their Prolog counterparts. ITP was selected for the examples because the basic path-planning model is in a germinal state and its exploratory features assisted in refining the basic model.

CHAPTER 6

SEPARABILITY EXAMPLES USING AUTOMATED REASONING

§ 6.1 Introduction

The investigation of bi-directional posets and motion-planning is based on finding solutions to the separability problem, i.e., separating objects in the plane by moving them along their assigned directions [THUK91]. This section describes a basic model for this problem by placing it in the context of goal-directed motion. Each element is associated with a specific destination. The destination is described in terms of the outer boundaries or limits of a pre-defined configuration, i.e., the objects are to be separated with respect to a given area in the plane. For example, given a cluster of points in a defined area, illustrated in figure 6.1.1a, the goal can be expressed in terms of moving elements along their assigned directions until they reach the limits of the area, shown in figure 6.1.1b. At that point, they are considered separated. The resulting separation would be along the lines of figure 6.1.1c.

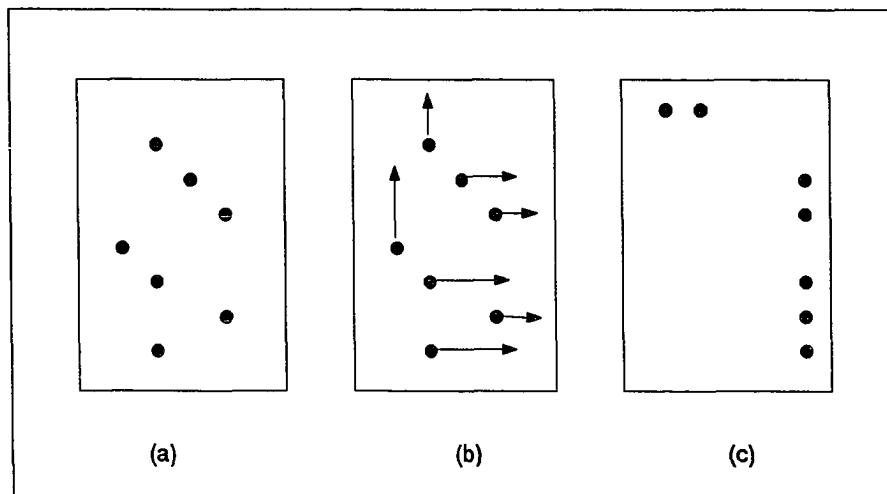


Figure 6.1.1 Separability Illustration

This formulation of the separability problem constitutes one approach to the solution of general goal-directed path-planning. If the objects can be separated so that they lie on the outer boundaries of the work-cell configuration, this will increase the probability that straight-line paths exist between each object and its destination. Only the initial invariant ordering is required.

The initial assumptions for the simple model are that the input is a set of points from which an initial poset will be generated. For the current poset, a maximal element is selected and the point associated with that element is moved along its assigned direction until either 1) it reaches its final location (a boundary) if possible, or 2) it reaches a point where it is blocked by another element in which case it stops and changes direction. After the element is moved, the current poset is updated to reflect the resulting relations and changes in direction or membership. Two sets are

maintained throughout the process, the current poset and its associated underlying ground set. The current poset consists of the current blocking relations based on locations and directional assignments. The current ground set consists of those elements that have not been moved to their goal locations. This set is needed in order to prevent costly searching in the selection of a maximal element. Given a poset, P_i , if an element, e , a member of the underlying ground set, S , and a maximal element, is moved to its goal location, then

$$S = S - \{e\}$$

and the resulting poset,

$$P_{i+1} = P_i - \{(x,y) \mid x = e \text{ or } y = e\}.$$

When all elements have been moved to their final locations, the goal poset, P_g , and its underlying ground set are null. The goal condition can then be expressed in terms of membership for the current poset.

Each point is represented by a label and a location, (x,y) . The initial poset is generated by first assigning each point either of the allowable directions. A point will be assigned the direction associated with its closest boundary. Once the directional assignments have been made, the blocking relations for the initial poset are derived from the locations and directions of the elements.

Decisions for moving the selected points will be based only upon a strategy for achieving a positive solution. No concern will be given to obtaining shortest path sequences. At the same time, in order to limit the search space, the strategy should

not be dependent on knowledge about the location of any other elements, except the one possible element moved to its goal location which blocks the present element.

For a given poset, P_i , a maximal element which has been moved, will either be deleted from P_{i+1} or will have its position and direction changed. P_{i+1} then must be updated to reflect new relations resulting from these changes. All other elements in P_i , however, will have the same position and direction in P_{i+1} . This formulation is suggestive of the situational logic paradigm described in Chapter 2. There are, however, important distinctions. Posets are not states. States are intended to capture snapshots of the knowledge base — the current world view. Posets, on the other hand, form discrete snapshots of the changing relations as objects are moved through the plane. These snapshots overlay a continuous domain. The only properties that persist from one poset to another are the directional assignments of the elements. As elements are moved through the plane, the underlying relations are changing. To achieve a given state, as in the situational logic approach, is to define a specific world view. To achieve a given poset, however, requires only achieving a set of relations that have no unique world view. The situational logic formulation is based on deriving the invariant poset (although expressed as a linear order). This poset is implicit in this simple path-planning model, and is defined by the selection process which in this example is the maximal element, but more generally, may be an already existing invariant order. To reflect these distinctions, our path-planning approach is referred to as directional logic. In the following

sections we present examples of the separability problem using bi-directional and quad₂-directional posets and include discussions of poset-partitioning strategies and general goal-directed motion-planning.

§ 6.2 *Bi-direction separability for one robot*

6.2.1 *Knowledge Representation*

This discussion has two distinct parts. The first stage is an initial model concerned only with the initial representation and poset generation, as well as the selection of maximal elements and the determination of initial movement. The second stage expands the initial model to include representations for dynamic posets. This model is suitable for robot arm implementation and provides the basis for subsequent examples.

Initially, the only input consists of the points and their locations. For example, one descriptive approach might be predicates *POINT* and *LOC* in which *POINT(A)*, describes an element and *LOC(A,I,I)* its location. From our discussion of domain induction techniques in automated reasoning, we need instead to define an enumeration over the elements. e.g. *POINT(p(I))*, and *LOC(p(I))*. Given these descriptions, the first task is the generation of initial directional assignments for each point. As indicated previously, these initial assignments will be based on an element's proximity to the outer boundaries. If an element is closest to the Northern boundary, then its direction will be N, otherwise it will be E. We can refine these simple directional

specifications by describing the directional vectors associated with an object. If an object is at $loc(x1,y1)$ and its direction is North, then $f(ax,x1)$ is the representation we will use to indicate that the object is traveling along the line $x = x1$. East is represented similarly. The clauses to assign directions are:

if $LOC(z,x,y) \ \& \ ((MAXX-x) > (MAXY-y))$ *then* $DIR(z,f(ay,y))$

if $LOC(z,x,y) \ \& \ (MAXX-x, MAXY-y)$ *then* $DIR(z,f(ax,x))$.

The next task is the derivation of the initial poset. An element, $z1$, is blocked by an element, $z2$, if the line joining $z1$ and $z2$ lies along the directional assignment of $z1$. This can be determined by checking the x and y -coordinates of the elements:

if $LOC(z1,x1,y1) \ \& \ LOC(z2,x2,y2) \ \& \ DIR(z1,f(ay,y1)) \ \& \ (y1=y2)$

& (x2>x1) *then* $BLOCKED_BY(z1,z2)$

if $LOC(z1,x1,y1) \ \& \ LOC(z2,x2,y2) \ \& \ DIR(z1,f(ax,x1)) \ \& \ (x1=x2)$

& (y2>y1) *then* $BLOCKED_BY(z1,z2)$

We also exploit the dual precedence feature by omitting the explicit derivation of transitive $BLOCKED_BY$ predicates, and define $-BLOCKED_BY(x,y)$ as meaning that y does not lie on the directional vector assigned to x . These additional relations are defined as follows:

if $LOC(z1,x1,y1) \ \& \ LOC(z2,x2,y2) \ \& \ -(x1=x2) \ \& \ -(y1=y2)$

then $-BLOCKED_BY(z1,z2)$

if $LOC(z1,x1,y1) \ \& \ LOC(z2,x2,y2) \ \& \ DIR(z1,f(ay,y1)) \ \& \ (y1=y2)$

& (x2<=x1) *then* $-BLOCKED_BY(z1,z2)$

$if\ LOC(z1,x1,y1) \ \&\ LOC(z2,x2,y2) \ \&\ DIR(z1\ f(ax,x1)) \ \&\ (x1=x2)$
 $\&\ (y2 \leq y1) \ then \ -BLOCKED_BY(z1,z2)$
 $if\ LOC(z1,x1,y1) \ \&\ LOC(z2,x2,y2) \ \&\ DIR(z1\ f(ay,y1)) \ \&\ (x1=x2)$
 $then \ -BLOCKED_BY(z1,z2)$
 $if\ LOC(z1,x1,y1) \ \&\ LOC(z2,x2,y2) \ \&\ DIR(z1\ f(ax,x1)) \ \&\ (y1=y2)$
 $then \ -BLOCKED_BY(z1,z2)$

We make one further expansion on our representation. As objects are moving through the plane, their locations, directional assignments and relations are changing. Dynamic directional posets represent snapshots of the changing relations. As a result, a variable much like the state variable must be added to reflect the assertion that a given location, directional assignment and relations are true for a given poset.

An object's location then is represented by $LOC(z,x,y,w)$ where z is the object, x,y is the position, and w is the poset for which that location is true. Objects themselves are represented by $POINT(p(i),w)$ where $1 \leq i \leq n$ and w reflects the poset(s) for which $p(i)$ does or does not exist. If an object, i , is moved to its goal destination at $poset(w)$, for the next poset, $w+1$, and subsequent posets, the object is removed from the ground set for the current poset, $-POINT(p(i),wz)$.

6.2.2 Planning Strategy

Once an object is selected as the maximal element for the poset, then a decision has to be made as to where and how to move the object. This, of course, is dependent on its location, direction, and knowledge of existing free space. The placement

rules include provisions, actually simple heuristics, for avoiding the chaining effect mentioned earlier.

There are three considerations for determining the placement of an object depending on whether or not the goal location along its direction vector is free and where the object is located. In addition to movement, if an object is not moved to its destination, then changes in direction assignment must also be considered.

There are three distinct cases:

1. an object is not on a boundary and the goal location along its direction vector is free,
2. an object is in the interior of the space and the goal location along its direction is not free,
3. an object is on the East or North boundary, i.e. one of its coordinates is *MAXX* or *MAXY*.

The first case is the simplest. If an object is *not* on a boundary and the goal location is free then move it. The clauses to cover this case are for both the North and East boundaries:

if MOVE(z,w) & loc(z,x,y,w) & dir(z,f(ay,y),w)
& -(y=NORTH_BOUNDARY) & FREE(MAXX,y,w)
then PLACE(z,MAXX,y,w);

if MOVE(z,w) & loc(z,x,y,w) & dir(z,f(ax,x),w)
& -(x=EAST_BOUNDARY) & FREE(x,MAXY,w)

then PLACE($z,x,MAXY,w$);

The first clause states that if an object is headed East and it is not on the North boundary, and the goal location is free then place the object. The second clause is similar for placement in the North direction.

We make a distinction between objects in the interior and objects on the boundary because we want to avoid clustering along the boundaries. Such situations can lead to failure even if a solution exists. For example, consider figure 6.2.2, in which a number of objects have already placed at east goal locations. The dashed lines represent the boundary. Both objects are headed east, yet if they move as far as they can, there is no way to move them to goal locations. On the other hand, there are free spaces available for the objects in the north direction.

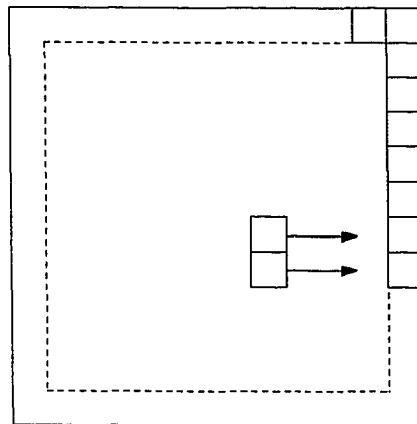


Figure 6.2.2 Chain-induced deadlock

The strategy for objects in the interior is to permit objects to “stairstep” through the space searching for free goal locations by alternating direction assignments. This

guarantees that if there is space available for an object, it will indeed be placed as in figure 6.2.3.

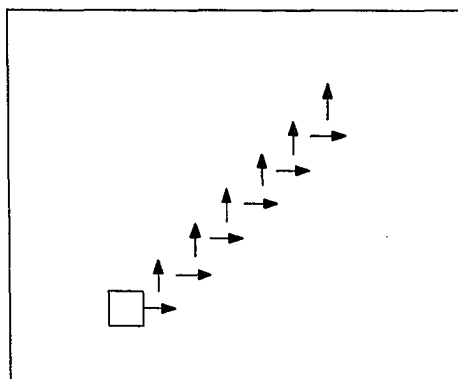


Figure 6.2.3 Stair-stepping motion

The question might be asked whether the object should first search in the north direction before moving east. The restrictions of our poset require that an interior object can only move in the direction that it is assigned because that is the only direction in which we are assured maximality. If an object is not maximal in an alternate direction for a given stage, that goal location will not be free when it is. We attempt to provide a basic strategy induced by the characteristics of the poset and the relations that characterize it.

This stairstepping motion has two parts. In the first case, if the goal location is not free, the object moves one unit in the direction it is assigned. That space is available because the object is maximal. In the second case, the object also changes

its directional assignment to an orthogonal one. The clauses for this interior motion are as follows:

**pointed east;*

if MOVE(z,w) & loc(z,x,y,w) & dir(z,f(ay,y),w)

& -(y=NORTH_BOUNDARY) & -(x=EAST_BOUNDARY)

& NOTFREE(MAXX,y,w) then PLACE(z,x+1,y,w);

if MOVE(z,w) & loc(z,x,y,w) & dir(z,f(ay,y),w)

& -(y=NORTH_BOUNDARY) & -(x=EAST_BOUNDARY)

& NOTFREE(MAXX,y,w) then dir(z,f(ax,x+1),w+1);

**pointed north;*

if MOVE(z,w) & loc(z,x,y,w) & dir(z,f(ax,x),w)

& -(y=NORTH_BOUNDARY) & -(x=EAST_BOUNDARY)

& NOTFREE(x,MAXY,w) then PLACE(z,x,y+1,w);

if MOVE(z,w) & loc(z,x,y,w) & dir(z,f(ax,x),w)

& -(y=NORTH_BOUNDARY) & -(x=EAST_BOUNDARY)

& NOTFREE(x,MAXY,w) then dir(z,f(ay,y+1),w+1);

The final case is when objects are located along the boundaries. These are the only objects that may move orthogonal to their directional assignments. Unlike objects located in the interior, objects along boundaries may not unconditionally go to goal locations along their directional assignments. For example, if an object is along the northern boundary and its directional assignment is east, even if that goal

location is free, to prevent clustering, it is prevented from moving there. Instead, a check is made to determine if the goal location orthogonal to the object's directional assignment is free. If so, then the object moves there. If not, then it moves one unit along its direction vector, and maintains the same directional assignment as in figure 6.2.4 where the solid lines represent the motion and directional assignment of an object along the boundary and the dashed lines represent the positions it checks.

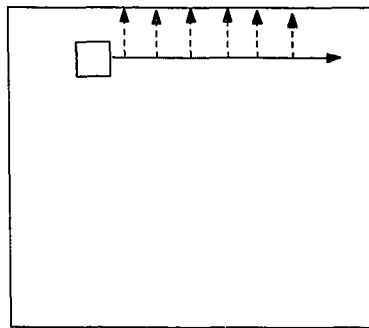


Figure 6.2.4 Boundary motion for two directions

Movements to free goal locations for boundaries differ from those of interior objects. Either of two cases will be permitted:

1. The object is moving along a boundary and a free goal location exists which is orthogonal to its directional assignment
2. The object is located on a boundary and its direction assignment is in the same direction as the boundary, and a free goal location exists. For example, an object is on the North boundary, and is pointed north. This might appear to be a redundant rule since the object is a maximal element, however, its arrival on the boundary

and corresponding direction assignment may have resulted from the stairstepping described above.

The following clauses handle these two cases:

**boundary - same direction;*

```

    if MOVE(z,w) & loc(z,EAST_BOUNDARY,y,w)
& dir(z,f(ay,y),w) & FREE(MAXX,y,w)
then PLACE(z,MAXX,y,w);

    if MOVE(z,w) & loc(z,x,NORTH_BOUNDARY,w)
& dir(z,f(ax,x),w) & FREE(x,MAXY,w)
then PLACE(z,x,MAXY,w);

```

**boundary - orthogonal direction;*

```

    if MOVE(z,w) & loc(z,EAST_BOUNDARY,y,w)
& dir(z,f(ax,EAST_BOUNDARY),w) & FREE(MAXX,y,w)
then PLACE(z,MAXX,y,w);

    if MOVE(z,w) & loc(z,x,NORTH_BOUNDARY,w)
& dir(z,f(ay,NORTH_BOUNDARY),w) & FREE(x,MAXY,w)
then PLACE(z,x,MAXY,w);

```

For objects moving along the boundaries, clauses are needed to consider all possible directional assignments and conditions. There are two general cases to consider:

1. an object is on a boundary, its direction assignment is the same and the goal

location is not free, and

2. an object is on a boundary, its direction assignment is orthogonal to the boundary, and the goal location is not free.

In the first case, since the object is maximal with respect to its direction assignment, we cannot with confidence move it in another direction. As a result, the object does not move, it is essentially given a null move — placement is in the same location and its direction assignment is changed to an orthogonal one.

**eastern boundary;*

*if MOVE(z,w) & loc(z,EAST_BOUNDARY,y,w) & dir(z,f(ay,y),w)
& NOTFREE(MAXX,y,w) then PLACE(z,EAST_BOUNDARY,y,w);*

*if MOVE(z,w) & loc(z,EAST_BOUNDARY,y,w) & dir(z,f(ay,y),w)
& NOTFREE(MAXX,y,w) then dir(z,f(ax,EAST_BOUNDARY),w+1);*

**northern boundary;*

*if MOVE(z,w) & loc(z,x,NORTH_BOUNDARY,w) & dir(z,f(ax,x),w)
& NOTFREE(x,MAXY,w) then PLACE(z,x,NORTH_BOUNDARY,w);*

*if MOVE(z,w) & loc(z,x,NORTH_BOUNDARY,w) & dir(z,f(ax,x),w)
& NOTFREE(x,MAXY,w) then dir(z,f(ay,NORTH_BOUNDARY),w+1);*

In the second case, the object moves one step along its direction assignment but its directional assignment is not changed.

**eastern boundary;*

if MOVE(z,w) & loc(z,EAST_BOUNDARY,y,w)

```

& dir(z,f(ax,EAST_BOUNDARY),w) & NOTFREE(MAXX,y,w)
then PLACE(z,EAST_BOUNDARY,y+1,w);
    if MOVE(z,w) & loc(z,EAST_BOUNDARY,y,w)
& dir(z,f(ax,EAST_BOUNDARY),w) & NOTFREE(MAXX,y,w)
then dir(z,f(ax,EAST_BOUNDARY),w+1);
    *northern boundary;
    if MOVE(z,w) & loc(z,x,NORTH_BOUNDARY,w)
& dir(z,f(ay,NORTH_BOUNDARY),w) & NOTFREE(x,MAXY,w)
then PLACE(z,x+1,NORTH_BOUNDARY,w);
    if MOVE(z,w) & loc(z,x,NORTH_BOUNDARY,w)
& dir(z,f(ay,NORTH_BOUNDARY),w) & NOTFREE(x,MAXY,w)
then dir(z,f(ay,NORTH_BOUNDARY),w+1);

```

Since each object selected for motion is a maximal element the only checking required is whether or not a given goal location is free. As a result, the only information required as to free locations are those specifying the configuration goal locations. These are all initialized as free:

FREE(MAXX,y,0);

FREE(x,MAXY,0);

As objects are placed, these are updated to indicate those that are still free, and those which are no longer free.

After the object is moved, what remains are essentially the "frame axioms"

indicating locations, directional assignments and existence. They are not unlike the classic formulations of frame axioms. For example, if an object was not moved during poset w , then its location, direction and "existence" remain the same.

Poset updates are handled fairly easily. At each stage, only one maximal object is generated, and there are no other possibilities to consider. As a result, only one object is moved and the only decisions to make concern where the object is to be moved. This is one of the features of the domain induction technique, effectively preventing the generation of clauses that represent other possible options. As a result, the poset occurring after poset w is simply $w+1$.

The complete set of clauses for the bi-directional separability for one arm are given in Appendix A.

§ 6.3 Bi-direction separability for two robots

6.3.1 Extensions for k -robots

At each point in the previous model, a single robot selects the next object to move based on the current maximal element. Since this is a poset, we are assured that there is at least one maximal element. The selection process is somewhat more complex when more than one agent is involved.

For the following model, the arm-geometry, location, and return motion of the agents will be ignored. Given the integer domain, the objects can be thought of as squares of unit-cell dimension.

Assuming that there are k -agents, several possibilities emerge. In the first case, if there are m maximal elements and $m \geq k$, then each agent is assigned an element for motion. This trial motion, however, must be subjected to collision tests, since two points may collide even if they are maximal.

In the second case, there are m maximal elements and $m < k$. There are several possible strategies. In the first, only m agents move points while the others remain idle. Idle time may be reduced by permitting the selection of elements that are not maximal. This involves, of course, a modification of the overall strategy. One strategy might be to select those elements whose upper covers are maximal elements that have been "trial"-assigned to another agent.

The solution implemented in this simple model, "trial" assigns a distinct maximal element to each arm. If there are not enough maximal elements, then some arms are idle. Even in those cases where each arm has been assigned a maximal element, if collisions result, they are resolved in favor of idling arms.

In all cases, possible motion is subject to collision constraints, i.e. the vectors do not intersect at a given time t . In addition, some consideration must be given to the determination of t , where t_i represents the total time required by an agent i to move a point to its destination or until it has to change direction. An associated question is at what point is the underlying directional poset updated to reflect the changing relations? There are two possibilities here, selection of either $\max(t_1, \dots, t_k)$ or $\min(t_1, \dots, t_k)$, where t_i is the time required for agent i to move its assigned point to

the determined destination. Selecting the maximum time results in idle time for the other agents, i.e. they are not assigned another point until the last agent completes its task. A more efficient selection is minimum time. When t_{min} occurs, each agent stops and the poset update occurs. The implications of this strategy are that not all of the k agents will complete possible motion along the assigned direction, however, since the output is a plan it will result in fairly continuous motion for all k agents and minimize idle time.

The key questions are how to generate trial assignments, and what to do if path collisions result. If the number of possible assignments is k , then if any collisions occur over t_{min} , one choice is to idle the affected arms. A second choice is to select an agent to change the direction of one or more points. The strategy must be as efficient as possible in order to avoid the combinatorial explosion associated with trying different possibilities. In this simple bidirectional case, the collision determination is simplified since it only occurs for one case, i.e., the directions are different.

6.3.2 Two-robot model

The basic representation of points, their locations and directions remains unchanged in the two-robot model. Required extensions include 1) the need to attempt to generate a second maximal element; 2) the need to test for collisions if both arms are moving objects and 3) an expansion of the representation of object placement.

The domain induction technique can be extended to select a maximal element for a particular arm. We could introduce additional clauses to insure that *arm 2* does not select the same maximal element as *arm 1*.

The *GETMAX* clauses which initialize the search for a maximal element may include an arm variable. If the first arm is searching for a maximal element, the two initial *GETMAX* clauses for a one-arm solution are not substantially different:

```

if GETMAX(w,arm(w2)) & P(p(1),w) & (w2=1)
then MAX_SO_FAR(p(1),1,w,arm(w2));

if GETMAX(w,arm(w2)) & -P(p(1),w)
then NOT_MAX_SO_FAR(1,w,arm(w2));

```

For the remaining arm, however, additional clauses are needed to prevent the selection of the first object, *p(1)*, as maximal so far, if it has already been selected by *arm 1*.

```

if GETMAX(w,arm(w2)) & (w2>1) & MAXIMAL(p(1),w,arm(1))
then NOT_MAX_SO_FAR(1,w,arm(w2));

if GETMAX(w,arm(w2)) & (w2>1) & MAXIMAL(x,w,arm(1))
& -(x=p(1)) then MAX_SO_FAR(p(1),1,w,arm(w2));

```

Other than the inclusion of the additional tests to insure that the two arms do not select the same object, these clauses are essentially the same as in the one-arm model. The above solution presents problems, however. It is not easily extendable to *k* arms. There would be additional clauses to check for arm *k* to ensure that

a candidate maximal element was not selected for each of the previous $k-1$ arms. In addition, in the event of a collision, there is no simple way to select another maximal element.

A better solution is to have a simple way to generate the next maximal element. Given $p(j)$ is a maximal element, we need to be able to determine the next maximal element. The induction technique used would have to be modified significantly to permit a search for the next element which may be an element before j in the linear ordering. One possibility, implemented here, is to generate a linear ordering of all maximal elements for a given stage. Then the total number of such elements is quickly determined, for use by additional arms. The following set of clauses generates all maximal elements of the current poset,

```

if CHKMAX(wz,w) & GEN(y,wz,w) then MAXSOFAR(wz,0,wz,w);
if CHKMAX(wz,w) & NOT_GEN(y,wz,w) then NOTMAX(wz,w);
if MAXSOFAR(wz,z,wz,w) & GEN(z,y,w) & (y=wz)
then MAXSOFAR(wz,z+1,wz,w);

if MAXSOFAR(wz,z,wz,w) & GEN(z,y,w) & -(y=wz) & prec(wz,y,w)
then NOTMAX(wz,w);

if MAXSOFAR(wz,z,wz,w) & GEN(z,y,w) & -(y=wz) & -prec(wz,y,w)
then MAXSOFAR(wz,z+1,wz,w);

if MAXSOFAR(wz,z,wz,w) & NOT_GEN(z,y,w) then MAXSOFAR(wz,z+1,wz,w);
if MAXSOFAR(wz,z,wz,w) & EQUAL(N,z,w) then NEWMAX(wz,w);

```

if GETALLMAX(z,wk,w) then CHKMAX(z,w);
if GETALLMAX(wz,wk,w) & NEWMAX(wz,w) then MAX(wz,wk+1,w);
if GETALLMAX(p(wz),wk,w) & NEWMAX(p(wz),w) & EQUAL(N,wn,w) &
(wz < wn) then GETALLMAX(p(wz+1),wk+1,w);
if GETALLMAX(p(wz),wk,w) & NEWMAX(p(wz),w) & EQUAL(N,wz,w) then
ALLMAX(wk+1,w);
if GETALLMAX(p(wz),wk,w) & NEWMAX(p(wz),w) & EQUAL(N,wz,w) then
MEMMAX(1,w);
if GETALLMAX(p(wz),wk,w) & NOTMAX(p(wz),w) & EQUAL(N,wz,w) then
MEMMAX(1,w);
if GETALLMAX(p(wz),wk,w) & NOTMAX(p(wz),w) & EQUAL(N,wn,w) &
\$LT(wz,wn) then GETALLMAX(p(wz+1),wk,w);
if GETALLMAX(p(wz),wk,w) & NOTMAX(p(wz),w) & EQUAL(N,wz,w)
then ALLMAX(wk,w);

The generation process determines for every element in the current poset whether it is a maximal element, and orders the maximal elements generated. For example, $MAX(z,i,w)$ means that z is the i th maximal element for poset w .

If there are at least two maximal elements then each arm will be assigned an object for the determination of a tentative move. For example, the following clause assigns the first maximal element to the first robot,

if MAX(wz,1,w) then TMOVE(wz,arm(1),w);

The placement rules determine where the element should be moved. These rules are exactly the same as in the one-robot model except that they are trial moves until collision detection and resolution occurs. The next step is to test for collisions. If a collision occurs then a new, tentative move is generated for the second arm. It will only be idled if there are no additional maximal elements. If the paths selected for placement, however, are collision-free, the minimum distance is determined and assigned to the length of each move.

The set of clauses for this model is substantially larger than in the one-arm case because of the additional checking involved. These tests, however, are fairly simple and this model also is driven in much the same way as the previous model in that only one pair of possible moves is considered at a time. The ability to achieve this almost procedurally-driven strategy hinges on the domain-induction technique. At the same time, we should observe that this strategy produces a “first” choice in terms of object selection and makes no attempt to determine an optimum one.

§ 6.4 Quad2–direction separability for one robot

6.4.1 Knowledge Representation

Similar to the bi-directional model, for each object there are predicates reflecting:

1. membership of the ground set for the current, $P(p(k),w)$,
2. present location, $loc(p(k),x,y,w)$

The possible directional assignments are extended to reflect the inverse vectors. For example, if an object is headed south along the direction vector $y = 3$, its direction is described as:

$$\text{dir}(p(k), y, 3, -1)$$

Before direction assignments are generated for each element, direction weights are determined. The four possible directions that an object can have are weighted based on its proximity to the corresponding boundary. These are used in conjunction with the poset construction theorem to determine the direction assignment for a given object.

6.4.2 Poset construction

The theorem in Chapter 4 is used as the basis for constructing the poset. The first element, $p(1)$ is assigned its optimum direction and is a poset. Subsequent elements in the ground set are added to the poset by first determining the maximal lower covers of an element and then assigning a direction using the poset construction theorem and the previously generated weights.

The following set of clauses generates the partial maximal lower covers for each element.

*_____;

*get lower covers for an element ;

*partial only from 1 to $i-1$;

*_____;

$*\text{---}z = z1 \text{ and } z2$
 ; if $GTLC(p(z),p(z),p(z),wy,wz,w) \ \& \ LGEN(wy,p(y),w) \ \& \ (y \leq wz)$
 $\& \ prec(p(y),p(z),w)$ then $GTLC(p(y),p(y),p(z),wy+1,wz,w);$
 $*\text{---}z = z1, \ z \neq z2;$
 if $GTLC(p(z),p(z2),p(z),wy,wz,w) \ \& \ LGEN(wy,p(y),w) \ \& \ (y \leq wz)$
 $\& \ -(z=z2) \ \& \ prec(p(y),p(z),w) \ \& \ -prec(p(z2),p(y),w)$
 then $GTLC(p(y),p(z2),p(z),wy+1,wz,w);$
 if $GTLC(p(z),p(z2),p(z),wy,wz,w) \ \& \ LGEN(wy,p(y),w) \ \& \ (y \leq wz)$
 $\& \ -(z=z2) \ \& \ prec(p(y),p(z),w) \ \& \ prec(p(z2),p(y),w)$
 then $GTLC(p(y),p(y),p(z),wy+1,wz,w);$
 $*\text{---}z = z2, \ z \neq z1;$
 if $GTLC(p(z1),p(z),p(z),wy,wz,w) \ \& \ LGEN(wy,p(y),w) \ \& \ (y \leq wz)$
 $\& \ -(z=z1) \ \& \ prec(p(y),p(z),w) \ \& \ -prec(p(z1),p(y),w)$
 then $GTLC(p(z1),p(y),p(z),wy+1,wz,w);$
 if $GTLC(p(z1),p(z),p(z),wy,wz,w) \ \& \ LGEN(wy,p(y),w) \ \& \ (y \leq wz)$
 $\& \ -(z=z1) \ \& \ prec(p(y),p(z),w) \ \& \ prec(p(z1),p(y),w)$
 then $GTLC(p(y),p(y),p(z),wy+1,wz,w);$
 $*\text{---}z \neq z1, \ z \neq z2;$
 if $GTLC(p(z1),p(z2),p(z),wy,wz,w) \ \& \ LGEN(wy,p(y),w) \ \& \ (y \leq wz) \ \& \ -(z=z1)$
 $\& \ -(z=z2) \ \& \ prec(p(y),p(z),w) \ \& \ prec(p(z1),p(y),w) \ \& \ -prec(p(z2),p(y),w)$
 then $GTLC(p(y),z2,p(z),wy+1,wz,w);$

if $GTLC(p(z1),p(z2),p(z),wy,wz,w) \ \& \ LGEN(wy,p(y),w) \ \& \ (y \leq wz) \ \& \ -(z=z1)$
 $\ \& \ -(z=z2) \ \& \ prec(p(y),p(z),w) \ \& \ prec(p(z2),p(y),w) \ \& \ -prec(p(z1),p(y),w)$
 then $GTLC(p(z1),p(y),p(z),wy+1,wz,w);$
 if $GTLC(p(z1),p(z2),p(z),wy,wz,w) \ \& \ LGEN(wy,p(y),w) \ \& \ (y \leq wz) \ \& \ -(z=z1)$
 $\ \& \ -(z=z2) \ \& \ prec(p(y),p(z),w) \ \& \ prec(p(z1),p(y),w) \ \& \ prec(p(z2),p(y),w)$
 then $GTLC(p(y),p(y),p(z),wy+1,wz,w);$
 if $GTLC(p(z1),p(z2),p(z),wy,wz,w) \ \& \ LGEN(wy,p(y),w) \ \& \ (y \leq wz)$
 $\ \& \ -prec(p(y),p(z),w)$ then $GTLC(p(z1),p(z2),p(z),wy+1,wz,w);$
 if $GTLC(p(z1),p(z2),p(z),wy,wz,w) \ \& \ NLGEN(wy,p(y),w) \ \& \ (y \leq wz)$
 then $GTLC(p(z1),p(z2),p(z),wy+1,wz,w);$
 if $GTLC(p(z1),p(z2),p(z),wy,wz,w) \ \& \ LGEN(wy,p(y),w) \ \& \ -(w=0) \ \& \ (y=z)$
 then $GTLC(p(z1),p(z2),p(z),wy+1,wz,w);$
 *————done rule —————;
 if $GTLC(p(z1),p(z2),p(z),wy,wz,0) \ \& \ GTDIR(p(z),wy,0) \ \& \ (z=wy+1)$
 then $LCOV(p(z1),p(z2),p(z),0);$
 if $GTLC(p(z1),p(z2),p(z),wy,wz,w) \ \& \ GTDIR(p(z),wy,w)$
 $\ \& \ -(w=0) \ \& \ EQUAL(N,wy,w)$ then $LCOV(p(z1),p(z2),p(z),w);$

In the above set of clauses, the predicate *prec* refers to the blocking relation not the direction. In addition, for each element, $p(i)$, the only existing blocking relations are of the form $prec(p(k), p(z))$ where $k < i$. They are not yet defined for $p(j)$ where $k \leq j$. Since this is a quad₂-direction poset, the number of maximal lower covers

of an element is not more than two. Therefore in *GTLC* only two arguments are required to keep track of the current maximal lower cover. This can be extended more generally by storing the lower covers in a list.

After generating the maximal lower covers for each element, then the direction is assignment according to our theorem:

```

*_____ ;

*case 1: no lower covers ;

*_____ ;

if LCOV(p(z),p(z),p(z),w) & dirwt(p(z),1,f(xy,xyk,xy1),w) & START(w)
then dir(p(z),f(xy,xyk,xy1),w);

*_____ ;

*case 2: one lower cover ;

*_____ ;

if LCOV(p(y),p(y),p(z),w) & dir(p(y),f(xy,xyk,xy1),w) & START(w)
then dir(p(z),f(xy,xyk,xy1),w);

*_____ ;

*case 3: two lower covers - get best weight ;

*_____ ;

if LCOV(p(z1),p(z2),p(z),w) & -(z1=z2) & dir(p(z1),f(xy1,xyk1,xy11),w)
& dir(p(z2),f(xy2,xyk2,xy22),w) & dirwt(p(z),wk1,f(xy1,xyk1,xy11),w)
& dirwt(p(z),wk2,f(xy2,xyk2,xy22),w) & (wk1>wk2) & START(w)

```

```

then dir(p(z),f(xy1,xyk1,xy11),w);

    if LCOV(p(z1),p(z2),p(z),w) & -(z1=z2) & dir(p(z1),f(xy1,xyk1,xy11),w)
    & dir(p(z2),f(xy2,xyk2,xy22),w) & dirwt(p(z),wk1,f(xy1,xyk1,xy11),w)
    & dirwt(p(z),wk2,f(xy2,xyk2,xy22),w) & (wk2>wk1) & START(w)
then dir(p(z),f(xy2,xyk2,xy22),w);

```

6.4.3 Planning Strategy

Our principal concern with the model is maintaining dynamic direction posets. Unlike the bi-directional model, this is not guaranteed as the objects move through the plane. In particular, as objects move around corners, as in figure 6.4.5, the possibility of cycles results. One way to preserve the poset property is to reformulate the poset after each move. While this is possible, it is overly slow and can still give rise to deadlocks in the corners. For example, in figure 6.4.5, where objects have already been placed in the northwest corner of the configuration space, object A is a maximal element. If object A is moved along the direction it is assigned, the only possible reassignments of direction are those facing non-free goal locations or those inducing cycles.

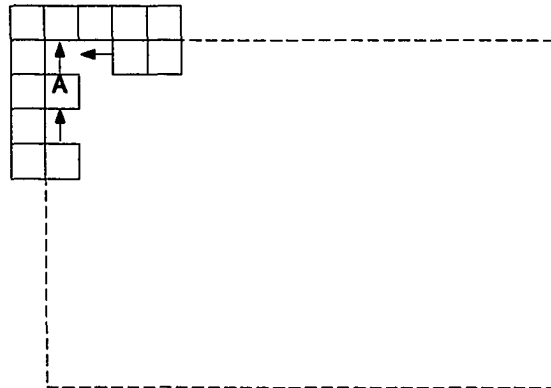


Figure 6.4.5 Deadlock example with $quad_2$ motion

Another strategy is to simplify the way in which the objects are placed at the configuration edges by allowing them to cycle the configuration space in a search for free space. Objects may move along the boundary in predetermined directions based on rules for boundary motion as in the bi-direction model. In this case, objects on the boundary move counterclockwise as in figure 6.4.6. Again like the bi-direction model, such elements move one unit at a time searching for free locations in the orthogonal direction. For example, in figure 6.4.6, object *A* would be placed in location *C*, not location *D*. As a heuristic, this works fairly well even in cluttered areas. Additional clauses are introduced to “fix” those elements that are circling the boundary. No maximal elements are selected when an object is fixed, only when it is placed in a goal location. Those elements moving along the boundary are no longer members of the underlying ground set. As a result, a maximal element is selected only if there is no current previously selected element moving along the

boundary. If the location of a maximal element is on the boundary then it will move in accordance with the boundary rules.

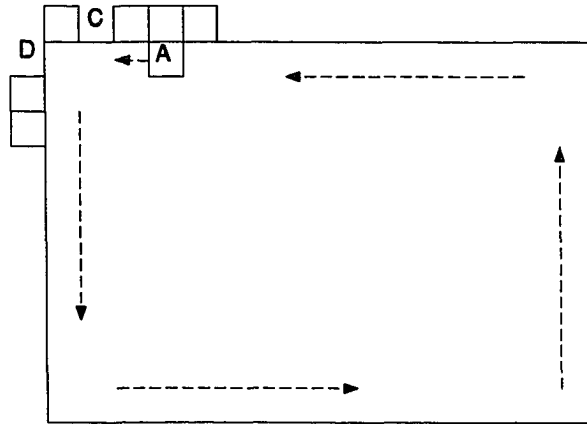


Figure 6.4.6 Movement around the boundaries

The basic strategy is then:

1. generate the direction weights,
2. generate the poset and then assign directions for the objects,
3. if an element is fixed, i.e. on the boundary, then move it in a predetermined order until it reaches a goal destination,
4. if no elements are fixed for the current poset, then get the maximal element and move it to either a goal destination if one is available or to a boundary.

Because of this “fix” strategy, and the need to avoid deadlock we divide the possible actions an object can take into several cases:

1. If an object is on a boundary, but not a corner and its goal location is not free, then the object moves along that boundary — much like the elements in the bi-directional model. For example, if an object is on the northern boundary, then it moves one unit west to search for the next available goal location, e.g.,

if MOVE(z,w) & loc($z,x,NORTH_BOUNDARY,w$) & ($x>WEST_BOUNDARY$)
& ($x<EAST_BOUNDARY$) & NOTFREE($x,MAXY,w$)
then PLACE($z,x-1,NORTH_BOUNDARY,w$);

2. If an object is on a boundary in a corner and neither of the two possible goal locations are free, then it will move one unit in accordance with the clockwise rule. For example if an element is in the northwest corner, then it moves one unit south, e.g.,

if MOVE(z,w) & loc($z,WEST_BOUNDARY,NORTH_BOUNDARY,w$)
& NOTFREE(MINX, $NORTH_BOUNDARY,w$)
& NOTFREE(WEST_ $BOUNDARY,MAXY,w$)
then PLACE($z,WEST_BOUNDARY,NORTH_BOUNDARY-1,w$);

3. Elements whose goal locations are free, based either on their direction assignments or on their occupation of a boundary, move directly to the goal location. For example, in the first clause that follows an element's goal location is free so it moves directly there. In the second clause, the element is on a boundary, but its adjacent goal location is free and it moves directly there, e.g.,

if MOVE(z,w) & loc(z,x,y,w) & dir($z,f(ax,x,1),w$) & FREE($x,MAXY,w$) *then*

PLACE(z,x,MAXY,w);

*if MOVE(z,w) & loc(z,x,NORTH_BOUNDARY,w) & FREE(x,MAXY,w) then
PLACE(z,x,MAXY,w);*

Finally after the object moves, updates are made for the next motion. If an element, z , moves to a boundary, then it is *FIXED*, e.g.,

if PLACE(z,x,NORTH_BOUNDARY,w) then FIX(z,w+1);

If an element, z , moves to its goal location, then *NOFIX* is concluded for the next poset, e.g.,

if PLACE(z,x,MAXY,w) then NOFIX(w+1);

The remaining updates are similar to those in the bi-directional model. For each element, frame axioms indicate its status in the next poset in terms of existence in the ground set, direction assignment, and location. In addition, updates are required to reflect the current *FREE* and *NOTFREE* spaces. A complete set of clauses can be found in Appendix B.

§ 6.5 Quad₂—direction separability for two robots

6.5.1 Extensions

The quad₂—direction separability model for two robots is in some senses a merger of the extensions of the bi-directional model for two robots with the one robot version of quad₂. It is, however, complicated by the need to consider combinations of *FIX/NOFIX* possibilities in the case of collisions.

The initial generation of the poset, directional assignments and precedences are the same as in quad_2 . Instead of generating just one maximal element, however, all maximal elements are generated. The motivation is to have a means of selecting the next maximal element for assignment to an arm.

The main extensions are those for detecting and handling collisions. For the given directions, a specification of possible collisions must be provided. If the directions are the same or they are inverse directions, then no object collision is possible. In the first case, if the directions are the same and there is a possibility of collision, then one of the elements is not maximal and would not have been selected. In the second case, if the directions are different and collision is possible then the poset condition has been violated.

The only potential collisions then are like the bi-directional model, those in which the directions are different. The descriptions of these collisions includes the recognition that a fixed element may be on a collision course with a maximal element. Once collisions have been detected a decision must be made in terms of possible reassignments. There are just two basic cases for two arms. If the collision involves two maximal elements, let the second arm retry tentative moves if any exist. If the collision involves a maximal element and a fixed element, then the arm assigned to the maximal element retries, with a new tentative move if one exists. If two fixed objects are in motion, no collision is possible.

§ 6.6 Poset-partitioning for robot clusters

The aforementioned strategies for separability problems do not fully exploit the structural properties of the configuration for manipulation by robot clusters. The mechanism for testing for collisions in the case of object/object collisions is largely combinatorial at best and requires procedural calls to test for arm/arm collisions. One or both of these tests are needed for every move. An alternate strategy that requires less testing would be to divide the workcell area into disjoint regions and assign each arm to a given region. Then the direction poset can be reformulated in terms of these disjoint regions. If an arm is moving an object in its region to a destination within the region, no collision test is required. Given the initial direction poset, P , and a set of regions, $R_1, \dots, R_k$ corresponding to each of the arms, the subdivision is as follows:

if $x < y$ in P and x, y are in R_i then $x < y$ is in P_i where P_i is the poset partition corresponding to R_i .

If an element is maximal in P , then it will be maximal in P_i . The basic strategy for motion-planning would be for each arm to select a maximal element with respect to its poset. If the goal location for the element is in the same poset for each arm, then no collision tests are required. On the other hand, if the goal location for an element is in another region, then the arm would move this element just across the border if possible. The handling of region “crossings” requires the use of the global maximal poset in conjunction with the regional posets. This accomplishes two things:

1) fewer tests for collisions and 2) built-in constraints for preventing arm collisions.

§ 6.7 Goal-directed motion

In the previous sections, we provided a tentative directional logic and poset-manipulation strategy. The concern is the extensibility of that strategy to path-planning for arbitrary cluttered configurations. In this section a discussion of possible strategies is presented.

One solution based on the *quad2* example is to separate objects in the plane along the outer edges of the configuration area, and then employ the straight-line motion planning strategy. While overhead is generated in the separation process, this can also be improved by resolving any possible straight-line first, then separating the remaining objects within a given area. This approach is useful only if the goal sub-configuration space and the object sub-configuration space do not overlap.

Another approach is to use the *quad2* directional assignments to move objects to their goal locations using similar strategies as in the bi-directional model. In this case the objects' directional assignments were weighted towards the direction vector describing their optimal straight-line path. There were several problems with this approach, however. First, the strategy is heuristically-based. For an arbitrary configuration, we cannot determine if a solution does or does not exist. Second, the directional models explored assumed that when an object was placed at its goal location, it was effectively removed from the set of moveable objects. As a result,

if an object was inadvertently moved into a region that became bounded by “goal” objects, there is no way to recover from this error. Nor does this representation implicitly provide information as to how to avoid those “danger” zones. Finally, for a relatively uncluttered obstacle configuration, the paths generated were much more involved than those using straight-line motion strategies. To a large extent, many of the problems with this approach and the basis for planning, stem from the need to continue to refine the characterization of these directional posets with respect to both object and goal configurations.

A third approach is both more restrictive and (often) more efficient than the previous two approaches. The configurations are restricted to those in which there is little or no overlap between the object region and the goal region. A dual strategy can be used in which:

- 1) at every stage, attempt to resolve any possible straight-line motion, and
- 2) if there is no possible straight-line motion, then disperse or separate the objects within their region to achieve straight-line motion at a later stage.

Unlike the separability examples described in previous sections, this strategy uses the minimal elements in the goal poset as a basis for motion. The goal poset is also dynamically updated. The effectiveness of this approach depends on the initial configuration of the objects and their goal locations. If the two regions are disjoint or the overlap is partial then a solution is possible. However, containment and major overlap may result in deadlocked situations.

Of the strategies discussed, only the first using separability then straight-line motion can be applied without attempting to describe the exhaustive set of rules appropriate for arbitrary configurations. Our investigation of directional posets and goal posets is not yet sufficiently robust to contend with arbitrary configurations.

§ 6.8 Summary

In this section, we have presented solutions to restricted separability problems based on the use of directional posets and discussed additional strategies.

While there are various other strategies that could have been applied in the solution of both separability problems, we instead chose relatively reasonable ones, to permit the global structure of the configuration to serve as the sole basis for path-planning. Some of the immediate concerns are:

- a. the need to investigate more fully the role of poset-partitioning as a basis for motivating movement among clusters. While this can be accomplished without partitions, the resulting additional clauses required to test for collisions is inefficient.
- b. we did not fully explore goal-directed motion, and more investigation is required to determine to what extent our directional posets and possibly goal posets can motivate such motion-planning in a general way. Separability can be used for goal-directed motion-planning only in the restricted cases in which the object and goal sub-configurations do not overlap.

c. Fixed obstacles in the configuration space have not been explored. Their inclusion, like that of general goal-direction motion, requires more insight into the structural characteristics of these configurations.

d. we observe that procedural calls are implicitly required for both straight-line motion and for those computations involving arm motion (as a rectangular projection on the plane) as opposed to object motion.

Clausal implementation is presented to indicate that such problems can be solved in this context. The distinction is that this implementation was used almost procedurally. Rather than generating all possible motions for a given poset and exploring the resulting tree, methods were used to specify the object(s) to be moved during a given phase. This was done to make the search more efficient. This is however based on a given linear ordering of elements and excludes any optimization criteria.

CHAPTER 7

CONCLUSION

§ 7.1 Results

Some results have been achieved in extending the investigation of ordered, directed sets as a basis for motion-planning. They include the formulation of poset expansions, the characterization of k_2 and $quad_2$ -directed posets and some of their properties. We have also demonstrated the use of this framework within a formal automated reasoning context. Finally, as a consequence of the investigation, a characterization was obtained for deciding if straight-line motion in the plane is achievable for a given configuration.

Poset expansions complement the original bi-directional investigation [THUK91] and may have applications for k -directional posets. Expansions have not been implemented in the planning examples illustrated here primarily because their use is more general. Given a poset with an initial ordering (based on some previous directional specification), expansions permit a restriction of the order to a bi-directional order while maintaining the poset. Essentially any k -directional poset (where k is finite) can be reduced to a bi-directional poset. The more general

question, still to be explored is whether an m -directional poset can be formed from a k -directional poset where $k > m$.

k_2 and $quad_2$ directional posets were defined and characterized based on the need to describe typical arm motion. The restriction to posets was used to provide selection criteria in a high-level planner. We also showed in the $quad_2$ -poset construction theorem that such a poset can be constructed and how the directional assignments can be made.

We have also investigated the manipulation of these posets with automated reasoning using the duality of relations defined earlier. Such a strategy permits a faster search of particular poset characteristics as well as the ability to characterize the poset in general.

Finally, by applying and modifying the concept of ordered, directed sets we were able to formulate goal posets as a basis for goal-directed path-planning analysis. This analysis together with the actual directed relation induced by a given configuration allows us to determine whether or not straight-line motion is possible.

§ 7.2 Limitations and Questions for Further Study

While we have suggested partial strategies for path-planning, the $quad_2$ -directional posets have not been sufficiently characterized for arbitrary goal-directed motion. Although separability strategies can be used for subclasses of these problem spaces, they may not always be optimal and further research is needed.

In addition, the relationship between goal posets and dynamic directional posets requires more investigation.

The path-planning strategies have emphasized fast searches for a solution over optimization techniques. More investigation is required to implement optimization criteria with respect to time, distance, and possible partitions. Strategies for an arbitrary number of arms need to be considered as well.

The point-representation model [THUK91] has been implemented here using simple polygons. The poset characterization and manipulation, however, needs to be extended to represent arbitrary objects and their geometric descriptions. In addition, further investigation is needed to extend the structural representation to account for obstacles or fixed objects in the plane.

Additional avenues of investigation for these directional posets include: 1) the possibility of poset achievability as a basis for learning and/or error recovery; 2) the extension of the structural representation to account for obstacles or fixed objects in the plane, and 3) the effect of orientation in the characterization of straight-line motion.

§ 7.3 Summary

The use of directional posets to model path-planning strategies has applications in gross motion-planning for both robot clusters and autonomous mobile robots. Such planning is situated between an abstract problem-solver and the low-level procedures

required for computation and effectuation of actual motion. In addition, the accuracy and precision of motion and placement is a function of the low-level procedural attachments, the mapping that exists between the arm, objects and workcell to the abstract grid, and the capabilities of the specific robots. Nevertheless this may prove to be a promising paradigm, in the context of a hybrid system in which the higher-level framework of path-planning is isolated from the lower level details of robot manipulation.

Appendix A

Clauses for separability problem for one robot

```

*_____ ;

*initialize directional assignments ;

*_____ ;

if poset(0) & P(z,0) & loc(z,x,y,0) & ((MAXX-x) <= (MAXY-y))
then dir(z,f(ay,y),0);

if poset(0) & P(z,0) & loc(z,x,y,0) & ((MAXX-x) > (MAXY-y))
then dir(z,f(ax,x),0);

*_____ ;

*Generate directional precedences ;

*_____ ;

if poset(w) & P(z1,w) & P(z2,w) & loc(z1,x1,y1,w) & loc(z2,x2,y1,w)
& dir(z1,f(ay,y1),w) & (x2>x1) then prec(z1,z2,w);

if poset(w) & P(z1,w) & P(z2,w) & loc(z1,x1,y1,w) & loc(z2,x1,y2,w)
& dir(z1,f(ax,x1),w) & (y2>y1) then prec(z1,z2,w);

if poset(w) & P(z1,w) & P(z2,w) & loc(z1,x1,y1,w) & loc(z2,x2,y2,w)
& -(x1=x2) & -(y1=y2) then -prec(z1,z2,w);

if poset(w) & P(z1,w) & P(z2,w) & loc(z1,x1,y1,w) & loc(z2,x2,y1,w)
& dir(z1,f(ay,y1),w) & (x2<x1) then -prec(z1,z2,w);

```

if poset(w) & P(z1,w) & P(z2,w) & loc(z1,x1,y1,w) & loc(z2,x2,y2,w)
& dir(z1,f(ax,x1),w) & (y2 < y1) then -prec(z1,z2,w);
if poset(w) & loc(z1,x1,y1,w) & loc(z2,x1,y2,w) & dir(z1,f(ay,y1),w)
then -prec(z1,z2,w);
if poset(w) & loc(z1,x1,y1,w) & loc(z2,x2,y1,w) & dir(z1,f(ax,x1),w)
then -prec(z1,z2,w);
if poset(w) & P(z1,w) & -P(z2,w) then -prec(z1,z2,w);
if INITIAL(w) & P(p(1),w) then MAXSOFAR(p(1),1,w);
if INITIAL(w) & -P(p(1),w) then NOT_MAXSOFAR(1,w);
if MAXSOFAR(z,x,w) & GEN(x,y,w) & prec(z,y,w) then NOTMAX(z,w);
if MAXSOFAR(z,x,w) & GEN(x,y,w) & -prec(z,y,w) then MAXSOFAR(z,x+1,w);
if MAXSOFAR(z,x,w) & GEN(x,y,w) & (z = y) then MAXSOFAR(z,x+1,w);
if MAXSOFAR(z,x,w) & NOT_GEN(x,y,w) then MAXSOFAR(z,x+1,w);
if MAXSOFAR(z,x,w) & EQUAL(N,x,w) then MAXIMAL(z,w);
if NOT_MAXSOFAR(x,w) & GEN(x,y,w) then MAXSOFAR(y,x+1,w);
if NOT_MAXSOFAR(x,w) & NOT_GEN(x,y,w) then NOT_MAXSOFAR(x+1,w);
if NOT_MAXSOFAR(x,w) & EQUAL(N,x,w) then -MAXIMAL(x,w);
if NOTMAX(p(z),w) & GEN(z,y,w) then MAXSOFAR(y,1,w);
if NOTMAX(p(z),w) & NOT_GEN(z,y,w) then NOT_MAXSOFAR(z+1,w);
if INITIAL(w) & P(p(z),w) then GEN(z-1,p(z),w);
if INITIAL(w) & -P(p(z),w) then NOT_GEN(z-1,p(z),w);

if poset(w) & MAXIMAL(x,w) then MOVE(x,w);
**_____;*
**Moves for elements not at boundaries;*
**_____;*
**pointed east;*
if MOVE(z,w) & loc(z,x,y,w) & dir(z,f(ay,y),w)
& -(y=NORTH_BOUNDARY) & -(x=EAST_BOUNDARY)
& NOTFREE(MAXX,y,w) then PLACE(z,x+1,y,w);
if MOVE(z,w) & loc(z,x,y,w) & dir(z,f(ay,y),w)
& -(y=NORTH_BOUNDARY) & -(x=EAST_BOUNDARY)
& NOTFREE(MAXX,y,w) then dir(z,f(ax,x+1),w+1);
**pointed north;*
if MOVE(z,w) & loc(z,x,y,w) & dir(z,f(ax,x),w)
& -(y=NORTH_BOUNDARY) & -(x=EAST_BOUNDARY)
& NOTFREE(x,MAXY,w) then PLACE(z,x,y+1,w);
if MOVE(z,w) & loc(z,x,y,w) & dir(z,f(ax,x),w)
& -(y=NORTH_BOUNDARY) & -(x=EAST_BOUNDARY)
& NOTFREE(x,MAXY,w) then dir(z,f(ay,y+1),w+1);
**_____;*
**Moves for elements points to free goals;*
**_____;*

**non-boundary rules;*

*if MOVE(z,w) & loc(z,x,y,w) & dir(z,f(ay,y),w)
& -(y=NORTH_BOUNDARY) & FREE(MAXX,y,w)
then PLACE(z,MAXX,y,w);*

*if MOVE(z,w) & loc(z,x,y,w) & dir(z,f(ax,x),w)
& -(x=EAST_BOUNDARY) & FREE(x,MAXY,w)
then PLACE(z,x,MAXY,w);*

**boundary - same direction;*

*if MOVE(z,w) & loc(z,EAST_BOUNDARY,y,w)
& dir(z,f(ay,y),w) & FREE(MAXX,y,w)
then PLACE(z,MAXX,y,w);*

*if MOVE(z,w) & loc(z,x,NORTH_BOUNDARY,w)
& dir(z,f(ax,x),w) & FREE(x,MAXY,w)
then PLACE(z,x,MAXY,w);*

**boundary - orthogonal direction;*

*if MOVE(z,w) & loc(z,EAST_BOUNDARY,y,w)
& dir(z,f(ax,EAST_BOUNDARY),w) & FREE(MAXX,y,w)
then PLACE(z,MAXX,y,w);*

*if MOVE(z,w) & loc(z,x,NORTH_BOUNDARY,w)
& dir(z,f(ay,NORTH_BOUNDARY),w) & FREE(x,MAXY,w)
then PLACE(z,x,MAXY,w);*

*_____;

**Moves for elements at boundaries ;*

*_____;

**pointed at not free goal, then change dir, null move;*

**eastern boundary;*

*if MOVE(z,w) & loc(z,EAST_BOUNDARY,y,w) & dir(z,f(ay,y),w)
& NOTFREE(MAXX,y,w) then PLACE(z,EAST_BOUNDARY,y,w);*

*if MOVE(z,w) & loc(z,EAST_BOUNDARY,y,w) & dir(z,f(ay,y),w)
& NOTFREE(MAXX,y,w) then dir(z,f(ax,EAST_BOUNDARY),w+1);*

**northern boundary;*

*if MOVE(z,w) & loc(z,x,NORTH_BOUNDARY,w) & dir(z,f(ax,x),w)
& NOTFREE(x,MAXY,w) then PLACE(z,x,NORTH_BOUNDARY,w);*

*if MOVE(z,w) & loc(z,x,NORTH_BOUNDARY,w) & dir(z,f(ax,x),w)
& NOTFREE(x,MAXY,w) then dir(z,f(ay,NORTH_BOUNDARY),w+1);*

**pointed orthogonal to non free goal, move one, keep same dir;*

**eastern boundary;*

*if MOVE(z,w) & loc(z,EAST_BOUNDARY,y,w)
& dir(z,f(ax,EAST_BOUNDARY),w) & NOTFREE(MAXX,y,w)
then PLACE(z,EAST_BOUNDARY,y+1,w);*

*if MOVE(z,w) & loc(z,EAST_BOUNDARY,y,w)
& dir(z,f(ax,EAST_BOUNDARY),w) & NOTFREE(MAXX,y,w)*

then dir(z,f(ax,EAST_BOUNDARY),w+1);
**northern boundary;*
if MOVE(z,w) & loc(z,x,NORTH_BOUNDARY,w)
& dir(z,f(ay,NORTH_BOUNDARY),w) & NOTFREE(x,MAXY,w)
then PLACE(z,x+1,NORTH_BOUNDARY,w);
if MOVE(z,w) & loc(z,x,NORTH_BOUNDARY,w)
& dir(z,f(ay,NORTH_BOUNDARY),w) & NOTFREE(x,MAXY,w)
then dir(z,f(ay,NORTH_BOUNDARY),w);
**_____;*
**UPDATES ;*
**_____;*
**object moved; if PLACE(z,x,y,w) then loc(z,x,y,w+1);*
if PLACE(z,MAXX,y,w) then NOTFREE(MAXX,y,w+1);
if PLACE(z,MAXX,y,w) then -P(z,w+1);
if PLACE(z,x,MAXY,w) then NOTFREE(x,MAXY,w+1);
if PLACE(z,x,MAXY,w) then -P(z,w+1);
if PLACE(z,x,y,w) & (x<MAXX) & (y<MAXY) then P(z,w+1);
**objects not moved;*
if PLACE(z,x,y,w) & loc(z1,x1,y1,w) & -(z=z1) then loc(z1,x1,y1,w+1);
if PLACE(z,x,y,w) & P(z1,w) & dir(z1,f(xy,xy1),w) & -(z=z1)
then dir(z1,f(xy,xy1),w+1);

```

if PLACE(z,x,y,w) & P(z1,w) & (z=z1) then P(z1,w+1);

*free/notfree updates

if PLACE(z,x,y,w) & NOTFREE(x1,y1,w) then NOTFREE(x,y,w+1);

if PLACE(z,x,y,w) & -(x=MAXX) & -(y=MAXY) & FREE(x1,y1,w)
then FREE(x1,y1,w+1);

if PLACE(z,x,y,w) & FREE(x1,y1,w) & -(x=x1) then FREE(x1,y1,w+1);

if PLACE(z,x,y,w) & FREE(x1,y1,w) & -(y=y1) then FREE(x1,y1,w+1);

*general updates;

if PLACE(z,x,y,w) then poset(w+1);

if PLACE(z,x,y,w) then INITIAL(w+1);

if loc(z,MAXX,y,w) then -P(z,w);

if loc(z,x,MAXY,w) then -P(z,w);

* _____;

*Set of Support ;

* _____;

EQUAL(N,k,w);

FREE(x,MAXY,0);

FREE(MAXX,y,0);

;

P(p(1),0);

...

```

loc(p(1),...);

...

poset(0);

INITIAL(0)

; ; ; ;

Appendix B

Two directional separability problem for two robot arms

```

*Bidirection separability for two arms;

*_____;;

*SET DIRECTION;

*_____;;

if poset(0) & P(z,0) & loc(z,x,y,0) & ((10-x) <= (10-y))
then dir(zf(ay,y),0);

if poset(0) & P(z,0) & loc(z,x,y,0) & ((10-x) > (10-y))
then dir(zf(ax,x),0);

*_____;;

*GET PRECEDENCES;

*_____;;

if poset(w) & loc(z1,x1,y1,w) & loc(z2,x2,y1,w) & dir(z1f(ay,y1),w)
& (x2 > x1) & P(z1,w) & P(z2,w) then prec(z1,z2,w);

if poset(w) & loc(z1,x1,y1,w) & loc(z2,x1,y2,w) & dir(z1f(ax,x1),w)
& (y2 > y1) & P(z1,w) & P(z2,w) then prec(z1,z2,w);

if poset(w) & loc(z1,x1,y1,w) & loc(z2,x2,y2,w) & -(x1=x2)
& -(y1=y2) then -prec(z1,z2,w);

```

if poset(w) & loc(z1,x1,y1,w) & loc(z2,x2,y1,w) & dir(z1,f(ay,y1),w)
& (x2<x1) then -prec(z1,z2,w);
if poset(w) & loc(z1,x1,y1,w) & loc(z2,x2,y2,w) & dir(z1,f(ax,x1),w)
& (y2<y1) then -prec(z1,z2,w);
if poset(w) & loc(z1,x1,y1,w) & loc(z2,x1,y2,w) & dir(z1,f(ay,y1),w)
then -prec(z1,z2,w);
if poset(w) & loc(z1,x1,y1,w) & loc(z2,x2,y1,w) & dir(z1,f(ax,x1),w)
then -prec(z1,z2,w);
if poset(w) & P(z1,w) & -P(z2,w) then -prec(z1,z2,w);
if INITIAL(w) & P(p(z),w) then PG(z-1,p(z),w);
if INITIAL(w) & -P(p(z),w) then NPG(z-1,p(z),w);
** _____;*
**clauses to generate maximal elements ;*
** _____*
if CHKMAX(wz,w) & PG(y,wz,w) then MAXSOFAR(wz,0,wz,w);
if CHKMAX(wz,w) & NPG(y,wz,w) then NOTMAX(wz,w);
if MAXSOFAR(wz,z,wz,w) & PG(z,y,w) & (y=wz)
then MAXSOFAR(wz,z+1,wz,w);
if MAXSOFAR(wz,z,wz,w) & PG(z,y,w) & -(y=wz) & prec(wz,y,w)
then NOTMAX(wz,w);
if MAXSOFAR(wz,z,wz,w) & PG(z,y,w) & -(y=wz) & -prec(wz,y,w)

then MAXSOFAR(wz,z+1,wz,w);
if MAXSOFAR(wz,z,wz,w) & NPG(z,y,w) then MAXSOFAR(wz,z+1,wz,w);
if MAXSOFAR(wz,z,wz,w) & EQUAL(N,z,w) then NEWMAX(wz,w);
if GETALLMAX(z,wk,w) then CHKMAX(z,w);
if GETALLMAX(wz,wk,w) & NEWMAX(wz,w) then MAX(wz,wk+1,w);
if GETALLMAX(p(wz),wk,w) & NEWMAX(p(wz),w) & EQUAL(N,wn,w) &
(wz<wn) then GETALLMAX(p(wz+1),wk+1,w);
if GETALLMAX(p(wz),wk,w) & NEWMAX(p(wz),w) & EQUAL(N,wz,w) then
ALLMAX(wk+1,w);
if GETALLMAX(p(wz),wk,w) & NEWMAX(p(wz),w) & EQUAL(N,wz,w) then
MEMMAX(1,w);
if GETALLMAX(p(wz),wk,w) & NOTMAX(p(wz),w) & EQUAL(N,wz,w) then
MEMMAX(1,w);
if GETALLMAX(p(wz),wk,w) & NOTMAX(p(wz),w) & EQUAL(N,wn,w) &
\$LT(wz,wn) then GETALLMAX(p(wz+1),wk,w);
if GETALLMAX(p(wz),wk,w) & NOTMAX(p(wz),w) & EQUAL(N,wz,w) then ALL-
MAX(wk,w);
 *_____;
 *Generate initial moves ;
 *_____;
if MAX(wz,1,w) then TMOVE(wz,arm(1),w);

if TMOVE(wz,arm(1),w) & MAX(wz,wk,w) & MAX(wz2,wk1,w)
& SEQ(wk1,wk+1) then TMOVE(wz2,arm(2),w);
if TMOVE(wz,arm(1),w) & MAX(wz,wk,w) & ALLMAX(wk,w)
then NOMOVE(arm(2),w);
if NEWTRIAL(arm(2),w) & TMOVE(wz2,arm(2),w) & MAX(wz2,wk,w) &
MAX(wz3,wk+1,w) then TMOVE(wz3,arm(2),w);
if NEWTRIAL(arm(2),w) & TMOVE(wz2,arm(2),w) & MAX(wz2,wk,w) & ALL-
MAX(wk,w) then NOMOVE(arm(2),w);

**_____;*
**MOVE rules same as for single arm ;*

**_____;*

**_____;*

**Moves for elements not at boundaries;*

**_____;*

**pointed east, move one unit to the east;*

if TMOVE(z,arm(wz),w) & loc(z,x,y,w) & dir(z,f(ay,y),w)
& -(y=NORTH_BOUNDARY) & -(x=EAST_BOUNDARY)
& NOTFREE(MAXX,y,w) then TRIAL(z,x+1,y,w,arm(wz));

**if pointed east, and moved one unit to the east, change direction*
to north;

if PLACE(z,x1,y,w,arm(wz)) & loc(z,x,y,w) & (x1=x+1)

& dir(zf(ay,y),w) & -(y=NORTH_BOUNDARY) & -(x=EAST_BOUNDARY)

& NOTFREE(MAXX,y,w) then dir(zf(ax,x+1),w+1);

**pointed north, move one unit to the north;*

if TMOVE(z,arm(wz),w) & loc(z,x,y,w) & dir(zf(ax,x),w)

& -(y=NORTH_BOUNDARY) & -(x=EAST_BOUNDARY)

& NOTFREE(x,MAXY,w) then TRIAL(z,x,y+1,w,arm(wz));

**if pointed north, and moved one unit to the north, change direction
to east;*

if PLACE(z,x,y1,w,arm(wz)) & loc(z,x,y,w) & (y1=y+1)

& dir(zf(ax,x),w) & -(y=NORTH_BOUNDARY) & -(x=EAST_BOUNDARY)

& NOTFREE(x,MAXY,w) then dir(zf(ay,y+1),w+1);

**_____;*

**Moves for elements pointed to free goals;*

**_____;*

**non-boundary rules;*

if TMOVE(z,arm(wz),w) & loc(z,x,y,w) & dir(zf(ay,y),w)

& -(y=NORTH_BOUNDARY) & FREE(MAXX,y,w)

then TRIAL(z,MAXX,y,w,arm(wz));

if TMOVE(z,arm(wz),w) & loc(z,x,y,w) & dir(zf(ay,y),w)

& -(y=NORTH_BOUNDARY) & FREE(MAXX,y,w)

then GOALPATH(z,w,arm(wz));

```

    if TMOVE(z,arm(wz),w) & loc(z,x,y,w) & dir(z,f(ax,x),w)
    & -(x=EAST_BOUNDARY) & FREE(x,MAXY,w)
    then TRIAL(z,x,MAXY,w,arm(wz));

    if TMOVE(z,arm(wz),w) & loc(z,x,y,w) & dir(z,f(ax,x),w)
    & -(x=EAST_BOUNDARY) & FREE(x,MAXY,w)
    then GOALPATH(z,w,arm(wz));

    *boundary - same direction;

    if TMOVE(z,arm(wz),w) & loc(z,EAST_BOUNDARY,y,w)
    & dir(z,f(ay,y),w) & FREE(MAXX,y,w)
    then TRIAL(z,MAXX,y,w,arm(wz));

    if TMOVE(z,arm(wz),w) & loc(z,EAST_BOUNDARY,y,w)
    & dir(z,f(ay,y),w) & FREE(MAXX,y,w)
    then GOALPATH(z,w,arm(wz));

    if TMOVE(z,arm(wz),w) & loc(z,x,NORTH_BOUNDARY,w)
    & dir(z,f(ax,x),w) & FREE(x,MAXY,w)
    then TRIAL(z,x,MAXY,w,arm(wz));

    if TMOVE(z,arm(wz),w) & loc(z,x,NORTH_BOUNDARY,w)
    & dir(z,f(ax,x),w) & FREE(x,MAXY,w)
    then GOALPATH(z,w,arm(wz));

    *boundary - orthogonal direction;

    if TMOVE(z,arm(wz),w) & loc(z,EAST_BOUNDARY,y,w)

```

```

& dir(z,f(ax,EAST_BOUNDARY),w) & FREE(MAXX,y,w)
then TRIAL(z,MAXX,y,w,arm(wz));

    if TMOVE(z,arm(wz),w) & loc(z,EAST_BOUNDARY,y,w)
& dir(z,f(ax,EAST_BOUNDARY),w) & FREE(MAXX,y,w)
then GOALPATH(z,w,arm(wz));

    if TMOVE(z,arm(wz),w) & loc(z,x,NORTH_BOUNDARY,w)
& dir(z,f(ay,NORTH_BOUNDARY),w) & FREE(x,MAXY,w)
then TRIAL(z,x,MAXY,w,arm(wz));

    if TMOVE(z,arm(wz),w) & loc(z,x,NORTH_BOUNDARY,w)
& dir(z,f(ay,NORTH_BOUNDARY),w) & FREE(x,MAXY,w)
then GOALPATH(z,w,arm(wz));

*_____ ;

*Moves for elements at boundaries ;

*_____ ;

*not pointed at free goal, then change dir, null move;

*eastern boundary;

    if TMOVE(z,arm(wz),w) & loc(z,EAST_BOUNDARY,y,w)
& dir(z,f(ay,y),w) & NOTFREE(MAXX,y,w)
then TRIAL(z,EAST_BOUNDARY,y,w,arm(wz));

    if PLACE(z,EAST_BOUNDARY,y,w,arm(wz))
& loc(z,EAST_BOUNDARY,y,w) & dir(z,f(ay,y),w)

```

```

& NOTFREE(MAXX,y,w)
then dir(z,f(ax,EAST_BOUNDARY),w+1);
    *northern boundary;
    if TMOVE(z,arm(wz),w) & loc(z,x,NORTH_BOUNDARY,w)
& dir(z,f(ax,x),w) & NOTFREE(x,MAXY,w)
then TRIAL(z,x,MAXY,w,arm(wz));
    if PLACE(z,x,NORTH_BOUNDARY,w,arm(wz))
& loc(z,x,NORTH_BOUNDARY,w) & dir(z,f(ax,x),w)
& NOTFREE(x,MAXY,w)
then dir(z,f(ay,NORTH_BOUNDARY),w+1);
    *pointed orthogonal to non free goal, move one, keep same dir;
    *eastern boundary;
    if TMOVE(z,arm(wz),w) & loc(z,EAST_BOUNDARY,y,w)
& dir(z,f(ax,EAST_BOUNDARY),w) & NOTFREE(MAXX,y,w)
then TRIAL(z,EAST_BOUNDARY,y+1,w,arm(wz));
    if PLACE(z,EAST_BOUNDARY,y1,w,arm(wz))
& loc(z,EAST_BOUNDARY,y,w) & (y1=y+1)
& dir(z,f(ax,EAST_BOUNDARY),w) & NOTFREE(MAXX,y,w)
then dir(z,f(ax,EAST_BOUNDARY),w+1);
    *northern boundary;
    if TMOVE(z,arm(wz),w) & loc(z,x,NORTH_BOUNDARY,w)

```

```

& dir(z,f(ay,NORTH_BOUNDARY),w) & NOTFREE(x,MAXY,w)
then TRIAL(z,x+1,NORTH_BOUNDARY,w,arm(wz));

  if PLACE(z,x1,NORTH_BOUNDARY,w,arm(wz))
& loc(z,x,NORTH_BOUNDARY,w) & (x1=x+1)
& dir(z,f(ay,NORTH_BOUNDARY),w) & NOTFREE(x,MAXY,w)
then dir(z,f(ay,NORTH_BOUNDARY),w+1);

  *_____ ;

  *Collision check ;

  *_____ ;

  *directions are the same - no collision;

  if TRIAL(z1,x1,y1,w,arm(wz1)) & TRIAL(z2,x2,y2,w,arm(wz2))
& -(z1=z2) & dir(z1,f(ax,x1),w) & dir(z2,f(ax,x2),w)
then GETMIN(z1,z2,x1,y1,x2,y2,w);

  if TRIAL(z1,x1,y1,w,arm(wz1)) & TRIAL(z2,x2,y2,w,arm(wz2))
& -(z1=z2) & dir(z1,f(ay,y1),w) & dir(z2,f(ay,y2),w)
then GETMIN(z1,z2,x1,y1,x2,y2,w);

  *directions are opposite;

  *collision;

  if TRIAL(z1,x1,y1,w,arm(wz1)) & TRIAL(z2,x2,y2,w,arm(wz2))
& dir(z1,f(ax,x1),w) & dir(z2,f(ay,y2),w) & loc(z1,x1,y01,w)
& loc(z2,x02,y2,w) & (y2<=y1) & (y2>=y01) & (x1<=x2) & (x1>=x02)

```

```

then NEWTRIAL(arm(2),w);

    *all cases for no collision;

    if TRIAL(z1,x1,y1,w,arm(wz1)) & TRIAL(z2,x2,y2,w,arm(wz2))
    & dir(z1,f(ax,x1),w) & dir(z2,f(ay,y2),w) & loc(z1,x1,y01,w)
    & loc(z2,x02,y2,w) & (y2>y21) then GETMIN(z1,z2,x1,y1,x2,y2,w);

        if TRIAL(z1,x1,y1,w,arm(wz1)) & TRIAL(z2,x2,y2,w,arm(wz2))
        & dir(z1,f(ax,x1),w) & dir(z2,f(ay,y2),w) & loc(z1,x1,y01,w)
        & loc(z2,x02,y2,w) & (y2<y01) then GETMIN(z1,z2,x1,y1,x2,y2,w);

            if TRIAL(z1,x1,y1,w,arm(wz1)) & TRIAL(z2,x2,y2,w,arm(wz2))
            & dir(z1,f(ax,x1),w) & dir(z2,f(ay,y2),w) & loc(z1,x1,y01,w)
            & loc(z2,x02,y2,w) & (x1>x2) then GETMIN(z1,z2,x1,y1,x2,y2,w);

                if TRIAL(z1,x1,y1,w,arm(wz1)) & TRIAL(z2,x2,y2,w,arm(wz2))
                & dir(z1,f(ax,x1),w) & dir(z2,f(ay,y2),w) & loc(z1,x1,y01,w)
                & loc(z2,x02,y2,w) & (x1<x02) then GETMIN(z1,z2,x1,y1,x2,y2,w);

                    *_____ ;

                    *get minimum distance ;

                    *_____ ;

                    if GETMIN(z1,z2,x1,y1,x2,y2,w) & loc(z1,x01,y01,w) & loc(z2,x02,y02,w)
                    & ((x1-x01)+(y1-y01)>=(x2-x02)+(y2-y02))
                    then mindist(z1,z2,(x2-x02)+(y2-y02),w);

                        if GETMIN(z1,z2,x1,y1,x2,y2,w) & loc(z1,x01,y01,w) & loc(z2,x02,y02,w)

```

```

& ((x1-x01)+(y1-y01)<(x2-x02)+(y2-y02))
then mindist(z1,z2,(x1-x01)+(y1-y01),w);

    if TRIAL(z1,x1,y1,w,arm(1)) & loc(z1,x01,y01,w) & NOMOVE(arm(2),w)
then dist(z1,(x1-x01)+(y1-y01),w);

    *_____ ;

    *generate placements ;

    *_____ ;

    if TRIAL(z1,x1,y1,w,arm(1)) & TRIAL(z2,x2,y2,w,arm(2))
& mindist(z1,z2,wmin,w) then PUT(z1,arm(1),z2,arm(2),wmin,w);

        if TRIAL(z2,x2,y2,w,arm(2)) & TRIAL(z1,x1,y1,w,arm(1))
& mindist(z2,z1,wmin,w) then PUT(z1,arm(1),z2,arm(2),wmin,w);

            if TRIAL(z1,x1,y1,w,arm(1)) & NOMOVE(arm(2),w) & dist(z1,wmin,w)
then PUT(z1,arm(1),0,arm(2),wmin,w);

                if PUT(z1,arm(wz),z2,arm(wz2),wmin,w) & loc(z1,x,y,w) & dir(z1,f(ax,x),w)
then PLACE(z1,x,y+wmin,w,arm(wz));

                    if PUT(z1,arm(wz),z2,arm(wz2),wmin,w) & loc(z1,x,y,w) & dir(z1,f(ay,y),w)
then PLACE(z1,x+wmin,y,w,arm(wz));

                        if PUT(z1,arm(wz),z2,arm(wz2),wmin,w) & loc(z2,x,y,w) & dir(z2,f(ax,x),w)
then PLACE(z2,x,y+wmin,w,arm(wz2));

                            if PUT(z1,arm(wz),z2,arm(wz2),wmin,w) & loc(z2,x,y,w) & dir(z2,f(ay,y),w)
then PLACE(z2,x+wmin,y,w,arm(wz2));

```

```

if PUT(z1,arm(wz),0,arm(wz2),wmin,w) then PLACE(0,0,0,w,arm(wz2));
*
_____ ;
*update move - note placement for non-goal already sets direction;
*
_____ ;
*need to set direction for path to goal;
*move to goals;
if PLACE(z,x,MAXY,w,arm(wz)) then -P(z,w+1);
if PLACE(z,x,MAXY,w,arm(wz)) then NOTFREE(x,MAXY,w+1);
if PLACE(z,MAXX,y,w,arm(wz)) then -P(z,w+1);
if PLACE(z,MAXX,y,w,arm(wz)) then NOTFREE(MAXX,y,w+1);
if PLACE(z,x,y,w,arm(wz)) & (x<MAXX) & (y<MAXY) then P(z,w+1);
if PLACE(z,x,y,w,arm(wz)) & GOALPATH(z,w,arm(wz)) & dir(zf(xy,xy1),w)
then dir(zf(xy,xy1),w+1);
*move elsewhere;
if PLACE(z,x,y,w,arm(wz)) then loc(z,x,y,w+1);
*other objects;
if PUT(z1,arm(1),z2,arm(2),wmin,w) & P(z,w) & -(z=z1) & -(z=z2)
then P(z,w+1);
if PUT(z1,arm(1),z2,arm(2),wmin,w) & -P(p(z),w) then -P(p(z),w+1);
if PUT(z1,arm(1),z2,arm(2),wmin,w) & loc(z,x,y,w) & -(z=z1) & -(z=z2)
then loc(z,x,y,w+1);

```

```

    if PUT(z1,arm(1),z2,arm(2),wmin,w) & P(z,w) & dir(zf{xy,xy1},w)
    & -(z=z1) & -(z=z2) then dir(zf{xy,xy1},w+1);

    if loc(z,x,MAXY,w) then -P(z,w);

    if loc(z,MAXX,y,w) then -P(z,w);

    *free spaces;

    if PUT(z1,arm(1),z2,arm(2),wmin,w) & NOTFREE(x,y,w)
    then NOTFREE(x,y,w+1);

    if PLACE(z1,x1,y1,w,arm(1)) & PLACE(z2,x2,y2,w,arm(2))
    & FREE(x3,y3,w) & -(x1=x3) & -(x2=x3) then FREE(x3,y3,w+1);

    if PLACE(z1,x1,y1,w,arm(1)) & PLACE(z2,x2,y2,w,arm(2))
    & FREE(x3,y3,w) & -(y1=y3) & -(y2=y3) then FREE(x3,y3,w+1);

    if PLACE(z1,x1,y1,w,arm(1)) & PLACE(z2,x2,y2,w,arm(2))
    & FREE(x1,y2,w) & -(x1=x2) & -(y1=y2) then FREE(x1,y2,w+1);

    if PLACE(z1,x1,y1,w,arm(1)) & PLACE(z2,x2,y2,w,arm(2))
    & FREE(x2,y1,w) & -(x1=x2) & -(y1=y2) then FREE(x2,y1,w+1);

    *generate next phase;

    if PUT(z1,arm(1),z2,arm(2),wmin,w) then poset(w+1);

    if PUT(z1,arm(1),z2,arm(2),wmin,w) then GETALLMAX(p(1),0,w+1);

    if PUT(z1,arm(1),z2,arm(2),wmin,w) then INITIAL(w+1);

    * _____;

    *Set of Support;

```

```

* _____;
EQUAL(N,k,w);
P(p(1),0);
...
loc(p(1),1,1,0);
...
INITIAL(0);
GETALLMAX(p(1),0,0);
poset(0);
FREE(x,MAXY,0);
FREE(MAXX,y,0);

```

Appendix C

Quad2-directional separability for one arm

```

*four-directional separability problem;

*_____;;

*get direction weights ;

*_____;;

  if poset(w) & GTWT(z,w) & P(z,w) & loc(z,x,y,w) & (y >=MAXY/2)
& (x>=MAXY-y) & (y>=x) then dirwt(z,1 f(ax,x,1),w);

  if poset(w) & GTWT(z,w) & P(z,w) & loc(z,x,y,w) & (y<MAXY/2)
& (x>=y) & (MAXX-x>=y) then dirwt(z,1 f(ax,x,-1),w);

  if poset(w) & GTWT(z,w) & P(z,w) & loc(z,x,y,w) & (x>=MAXX/2)
& (y>MAXX-x) & (x>y) then dirwt(z,1 f(ay,y,1),w);

  if poset(w) & GTWT(z,w) & P(z,w) & loc(z,x,y,w) & (x<MAXX/2)
& (MAXY-y>x) & (y>x) then dirwt(z,1 f(ay,y,-1),w);

  if poset(w) & GTWT(z,w) & P(z,w) & loc(z,x,y,w) & dirwt(z,1 f(ax,x,xy),w)
& (y>MAXY/2) then dirwt(z,2 f(ay,y,1),w);

  if poset(w) & GTWT(z,w) & P(z,w) & loc(z,x,y,w) & dirwt(z,1 f(ax,x,xy),w)
& (y<=MAXY/2) then dirwt(z,2 f(ay,y,-1),w);

  if poset(w) & GTWT(z,w) & P(z,w) & loc(z,x,y,w) & dirwt(z,1 f(ay,y,xy),w)
& (x>MAXX/2) then dirwt(z,2 f(ax,x,1),w);

```

if $\text{poset}(w) \ \& \ \text{GTWT}(z,w) \ \& \ P(z,w) \ \& \ \text{loc}(z,x,y,w) \ \& \ \text{dirwt}(z,1f(ay,y,xy),w)$
 & $(x \leq \text{MAXX}/2)$ then $\text{dirwt}(z,2f(ax,x,-1),w);$
 if $\text{poset}(w) \ \& \ \text{GTWT}(z,w) \ \& \ P(z,w) \ \& \ \text{loc}(z,x,y,w) \ \& \ \text{dirwt}(z,2f(ay,y,xy),w)$
 then $\text{dirwt}(z,3f(ay,y,xy*(-1)),w);$
 if $\text{poset}(w) \ \& \ \text{GTWT}(z,w) \ \& \ P(z,w) \ \& \ \text{loc}(z,x,y,w) \ \& \ \text{dirwt}(z,2f(ax,x,xy),w)$
 then $\text{dirwt}(z,3f(ax,x,xy*(-1)),w);$
 if $\text{poset}(w) \ \& \ \text{GTWT}(z,w) \ \& \ P(z,w) \ \& \ \text{loc}(z,x,y,w) \ \& \ \text{dirwt}(z,1f(ay,y,xy),w)$
 then $\text{dirwt}(z,4f(ay,y,xy*(-1)),w);$
 if $\text{poset}(w) \ \& \ \text{GTWT}(z,w) \ \& \ P(z,w) \ \& \ \text{loc}(z,x,y,w) \ \& \ \text{dirwt}(z,1f(ax,x,xy),w)$
 then $\text{dirwt}(z,4f(ax,x,xy*(-1)),w);$
 *_____;
 *determine precedences ;
 *_____;
 *north/south;
 if $\text{poset}(w) \ \& \ P(z1,w) \ \& \ P(z2,w) \ \& \ \text{loc}(z1,x1,y1,w) \ \& \ \text{loc}(z2,x1,y2,w)$
 & $\text{dir}(z1f(ax,x1,xy1),w) \ \& \ ((y2-y1)*xy1 > 0)$ then $\text{prec}(z1,z2,w);$
 *east/west;
 if $\text{poset}(w) \ \& \ P(z1,w) \ \& \ P(z2,w) \ \& \ \text{loc}(z1,x1,y1,w) \ \& \ \text{loc}(z2,x2,y1,w)$
 & $\text{dir}(z1f(ay,y1,xy1),w) \ \& \ ((x2-x1)*xy1 > 0)$ then $\text{prec}(z1,z2,w);$
 *_____;
 *possible -prec;

```

*_____ ;

*north/south;

if poset(w) & P(z1,w) & P(z2,w) & loc(z1,x1,y1,w) & loc(z2,x1,y2,w)
& dir(z1 f(ax,x1,xy1),w) & ((y2-y1)*xy1<=0) then -prec(z1,z2,w);

if poset(w) & P(z1,w) & P(z2,w) & loc(z1,x1,y1,w) & loc(z2,x2,y2,w)
& dir(z1 f(ax,x1,xy1),w) & -(x1=x2) then -prec(z1,z2,w);

*east/west;

if poset(w) & P(z1,w) & P(z2,w) & loc(z1,x1,y1,w) & loc(z2,x2,y1,w)
& dir(z1 f(ay,y1,xy1),w) & ((x2-x1)*xy1<=0) then -prec(z1,z2,w);

if poset(w) & P(z1,w) & P(z2,w) & loc(z1,x1,y1,w) & loc(z2,x2,y2,w)
& dir(z1 f(ay,y1,xy1),w) & -(y1=y2) then -prec(z1,z2,w);

*_____ ;

*generate poset ;

*_____ ;

*set up p(1);

if P(p(1),0) & dirwt(p(1),1 f(xy,xyk,xy1),0) then dir(p(1),f(xy,xyk,xy1),0);

if dir(p(z),f(xy,xyk,xy1),0) & EQUAL(N,zk,0) & -(z=zk) & START(0)
then GTDIR(p(z+1),z,0);

*constant reference z = 5;

if dir(p(z),f(xy,xyk,xy1),0) & EQUAL(N,z,0) then INITIAL(0);

*_____ ;

```

*_____;

**get generators ;*

*_____;

if GTDIR($p(z), wz, w$) & $P(p(y), w)$ & $(y \leq wz)$ then LGEN($y-1, p(y), w$);

if GTDIR($p(z), wz, w$) & $\neg P(p(y), w)$ & $(y \leq wz)$ then NLGEN($y-1, p(y), w$);

if GTDIR($p(z), wz, w$) & $NPG(y-1, p(y), w)$ then NLGEN($y-1, p(y), w$);

if GTDIR($p(z), wz, w$) then GTLC($p(z), p(z), p(z), 0, wz, w$);

*_____;

**get lower covers for an element ;*

**partial only from 1 to $i-1$;*

*_____;

*_____ $z = z1$ and $z2$;

if GTLC($p(z), p(z), p(z), wy, wz, w$) & LGEN($wy, p(y), w$) & $(y \leq wz)$

& $\text{prec}(p(y), p(z), w)$ then GTLC($p(y), p(y), p(z), wy+1, wz, w$);

*_____ $z = z1, z \neq z2$;

if GTLC($p(z), p(z2), p(z), wy, wz, w$) & LGEN($wy, p(y), w$) & $(y \leq wz)$ & $\neg(z=z2)$

& $\text{prec}(p(y), p(z), w)$ & $\neg \text{prec}(p(z2), p(y), w)$ then GTLC($p(y), p(z2), p(z), wy+1, wz, w$);

if GTLC($p(z), p(z2), p(z), wy, wz, w$) & LGEN($wy, p(y), w$) & $(y \leq wz)$ & $\neg(z=z2)$

& $\text{prec}(p(y), p(z), w)$ & $\text{prec}(p(z2), p(y), w)$ then GTLC($p(y), p(y), p(z), wy+1, wz, w$);

*_____ $z = z2, z \neq z1$;

if GTLC($p(z1), p(z), p(z), wy, wz, w$) & LGEN($wy, p(y), w$) & $(y \leq wz)$ & $\neg(z=z1)$

$\& \text{prec}(p(y),p(z),w) \& \neg \text{prec}(p(z1),p(y),w) \text{ then } \text{GTLC}(p(z1),p(y),p(z),wy+1,wz,w);$
 $\quad \text{if } \text{GTLC}(p(z1),p(z),p(z),wy,wz,w) \& \text{LGEN}(wy,p(y),w) \& (y \leq wz) \& \neg(z=z1)$
 $\& \text{prec}(p(y),p(z),w) \& \text{prec}(p(z1),p(y),w) \text{ then } \text{GTLC}(p(y),p(y),p(z),wy+1,wz,w);$
 $\quad * \text{——} z \neq z1, z \neq z2;$
 $\quad \text{if } \text{GTLC}(p(z1),p(z2),p(z),wy,wz,w) \& \text{LGEN}(wy,p(y),w) \& (y \leq wz) \& \neg(z=z1)$
 $\& \neg(z=z2) \& \text{prec}(p(y),p(z),w) \& \text{prec}(p(z1),p(y),w) \& \neg \text{prec}(p(z2),p(y),w)$
 $\text{then } \text{GTLC}(p(y),z2,p(z),wy+1,wz,w);$
 $\quad \text{if } \text{GTLC}(p(z1),p(z2),p(z),wy,wz,w) \& \text{LGEN}(wy,p(y),w) \& (y \leq wz) \& \neg(z=z1)$
 $\& \neg(z=z2) \& \text{prec}(p(y),p(z),w) \& \text{prec}(p(z2),p(y),w) \& \neg \text{prec}(p(z1),p(y),w)$
 $\text{then } \text{GTLC}(p(z1),p(y),p(z),wy+1,wz,w);$
 $\quad \text{if } \text{GTLC}(p(z1),p(z2),p(z),wy,wz,w) \& \text{LGEN}(wy,p(y),w) \& (y \leq wz) \& \neg(z=z1)$
 $\& \neg(z=z2) \& \text{prec}(p(y),p(z),w) \& \text{prec}(p(z1),p(y),w)$
 $\& \text{prec}(p(z2),p(y),w) \text{ then } \text{GTLC}(p(y),p(y),p(z),wy+1,wz,w);$
 $\quad \text{if } \text{GTLC}(p(z1),p(z2),p(z),wy,wz,w) \& \text{LGEN}(wy,p(y),w) \& (y \leq wz)$
 $\& \neg \text{prec}(p(y),p(z),w) \text{ then } \text{GTLC}(p(z1),p(z2),p(z),wy+1,wz,w);$
 $\quad \text{if } \text{GTLC}(p(z1),p(z2),p(z),wy,wz,w) \& \text{NLGEN}(wy,p(y),w) \& (y \leq wz)$
 $\text{then } \text{GTLC}(p(z1),p(z2),p(z),wy+1,wz,w);$
 $\quad \text{if } \text{GTLC}(p(z1),p(z2),p(z),wy,wz,w) \& \text{LGEN}(wy,p(y),w) \& \neg(w=0) \& (y=z)$
 $\text{then } \text{GTLC}(p(z1),p(z2),p(z),wy+1,wz,w);$
 $\quad * \text{——} \text{done rule} \text{ ——};$
 $\quad \text{if } \text{GTLC}(p(z1),p(z2),p(z),wy,wz,0) \& \text{GTDIR}(p(z),wy,0) \& (z=wy+1)$

then LCOV(p(z1),p(z2),p(z),0);
 if GTLC(p(z1),p(z2),p(z),wy,wz,w) & GTDIR(p(z),wy,w) & -(w=0)
& EQUAL(N,wy,w) then LCOV(p(z1),p(z2),p(z),w);
 *_____;
 **set direction of z;*
 *_____;
 **case 1: no lower covers ;*
 *_____;
 if LCOV(p(z),p(z),p(z),w) & dirwt(p(z),1f(xy,xyk,xy1),w) & START(w) then
dir(p(z),f(xy,xyk,xy1),w);
 *_____;
 **case 2: one lower cover ;*
 *_____;
 if LCOV(p(y),p(y),p(z),w) & dir(p(y),f(xy,xyk,xy1),w) & START(w)
then dir(p(z),f(xy,xyk,xy1),w);
 *_____;
 **case 3: two lower covers - get best weight ;*
 *_____;
 if LCOV(p(z1),p(z2),p(z),w) & -(z1=z2) & dir(p(z1),f(xy1,xyk1,xy11),w)
& dir(p(z2),f(xy2,xyk2,xy22),w) & dirwt(p(z),wk1f(xy1,xyk1,xy11),w)
& dirwt(p(z),wk2f(xy2,xyk2,xy22),w) & (wk1>wk2) & START(w)

then dir(p(z),f(xy1,xyk1,xy11),w);
if LCOV(p(z1),p(z2),p(z),w) & -(z1=z2) & dir(p(z1),f(xy1,xyk1,xy11),w)
& dir(p(z2),f(xy2,xyk2,xy22),w) & dirwt(p(z),wk1,f(xy1,xyk1,xy11),w)
& dirwt(p(z),wk2,f(xy2,xyk2,xy22),w) & (wk2>wk1) & START(w)
then dir(p(z),f(xy2,xyk2,xy22),w);

**_____;*

**get maximal elements ;*

**_____;*

if INITIAL(w) & P(p(1),w) then MAXSOFAR(p(1),1,w);
if INITIAL(w) & -P(p(1),w) then NOT_MAXSOFAR(1,w);

if MAXSOFAR(z,x,w) & GEN(x,y,w) & prec(z,y,w) then NOTMAX(z,w);
if MAXSOFAR(z,x,w) & GEN(x,y,w) & -prec(z,y,w) then MAXSOFAR(z,x+1,w);
if MAXSOFAR(z,x,w) & GEN(x,y,w) & (z = y) then MAXSOFAR(z,x+1,w);
if MAXSOFAR(z,x,w) & NOT_GEN(x,y,w) then MAXSOFAR(z,x+1,w);

if MAXSOFAR(z,x,w) & EQUAL(N,x,w) then MAXIMAL(z,w);

if NOT_MAXSOFAR(x,w) & GEN(x,y,w) then MAXSOFAR(y,x+1,w);

if NOT_MAXSOFAR(x,w) & NOT_GEN(x,y,w) then NOT_MAXSOFAR(x+1,w);

if NOT_MAXSOFAR(x,w) & EQUAL(N,x,w) then -MAXIMAL(x,w);

if NOTMAX(p(z),w) & GEN(z,y,w) then MAXSOFAR(y,1,w);

if NOTMAX(p(z),w) & NOT_GEN(z,y,w) then NOT_MAXSOFAR(z+1,w);

if INITIAL(w) & P(p(z),w) then GEN(z-1,p(z),w);

```

if INITIAL(w) & -P(p(z),w) then NOT_GEN(z-1,p(z),w);

*_____ ;

*determine move - boundary vs. no boundary ;

*_____ ;

*basic strategy: maximal element is moved until placed at goal;

*until placed - FIX, when placed - NOFIX;

if MAXIMAL(z,w) then MOVE(z,w);

*_____ ;

*boundaries, no corners, not free;

*_____ ;

*north;

if MOVE(z,w) & loc(z,x,NORTH_BOUNDARY,w) & (x>WEST_BOUNDARY)
& (x<EAST_BOUNDARY) & NOTFREE(x,MAXY,w)
then PLACE(z,x-1,NORTH_BOUNDARY,w);

*south;

if MOVE(z,w) & loc(z,x,SOUTH_BOUNDARY,w) & (x>WEST_BOUNDARY)
& (x<EAST_BOUNDARY) & NOTFREE(x,MINY,w)
then PLACE(z,x+1,SOUTH_BOUNDARY,w);

*east;

if MOVE(z,w) & loc(z,EAST_BOUNDARY,y,w) & (y>SOUTH_BOUNDARY)
& (y<NORTH_BOUNDARY) & NOTFREE(MAXX,y,w)

```

```

then PLACE(z,EAST_BOUNDARY,y+1,w);

    *west;

    if MOVE(z,w) & loc(z,WEST_BOUNDARY,y,w) & (y>SOUTH_BOUNDARY)
& (y<NORTH_BOUNDARY) & NOTFREE(MINX,y,w)
then PLACE(z,WEST_BOUNDARY,y-1,w);

    *_____ ;

    *boundaries, corners, not free;

    *_____ ;

    if MOVE(z,w) & loc(z,WEST_BOUNDARY,NORTH_BOUNDARY,w)
& NOTFREE(MINX,NORTH_BOUNDARY,w)
& NOTFREE(WEST_BOUNDARY,MAXY,w)
then PLACE(z,WEST_BOUNDARY,NORTH_BOUNDARY-1,w);

    if MOVE(z,w) & loc(z,WEST_BOUNDARY,SOUTH_BOUNDARY,w)
& NOTFREE(WEST_BOUNDARY,MINY,w)
& NOTFREE(MINX,SOUTH_BOUNDARY,w)
then PLACE(z,WEST_BOUNDARY+1,SOUTH_BOUNDARY,w);

    if MOVE(z,w) & loc(z,EAST_BOUNDARY,SOUTH_BOUNDARY,w)
& NOTFREE(MAXX,SOUTH_BOUNDARY,w)
& NOTFREE(EAST_BOUNDARY,MINY,w)
then PLACE(z,EAST_BOUNDARY,SOUTH_BOUNDARY+1,w);

    if MOVE(z,w) & loc(z,EAST_BOUNDARY,NORTH_BOUNDARY,w)

```

```

& NOTFREE(MAXX,NORTH_BOUNDARY,w)
& NOTFREE(EAST_BOUNDARY,MAXY,w)
then PLACE(z,EAST_BOUNDARY-1,NORTH_BOUNDARY,w);
    *_____ ;
    *boundaries, free;
    *_____ ;
    if MOVE(z,w) & loc(z,x,NORTH_BOUNDARY,w) & FREE(x,MAXY,w)
then PLACE(z,x,MAXY,w);
    if MOVE(z,w) & loc(z,x,SOUTH_BOUNDARY,w) & FREE(x,MINY,w)
then PLACE(z,x,MINY,w);
    if MOVE(z,w) & loc(z,EAST_BOUNDARY,y,w) & FREE(MAXX,y,w)
then PLACE(z,MAXX,y,w);
    if MOVE(z,w) & loc(z,WEST_BOUNDARY,y,w) & FREE(MINX,y,w)
then PLACE(z,MINX,y,w);
    *_____ ;
    *non-boundaries, not free;
    *_____ ;
    *north;
    if MOVE(z,w) & loc(z,x,y,w) & -(y=NORTH_BOUNDARY) & dir(z,f(ax,x,1),w)
& NOTFREE(x,MAXY,w) then PLACE(z,x,NORTH_BOUNDARY,w);
    *south;

```

if MOVE(z,w) & loc(z,x,y,w) & -(y=SOUTH_BOUNDARY) & dir(z,f(ax,x,-1),w)
& NOTFREE(x,MINY,w) then PLACE(z,x,SOUTH_BOUNDARY,w);
**east;*
if MOVE(z,w) & loc(z,x,y,w) & -(x=EAST_BOUNDARY) & dir(z,f(ay,y,1),w)
& NOTFREE(MAXX,y,w) then PLACE(z,EAST_BOUNDARY,y,w);
**west;*
if MOVE(z,w) & loc(z,x,y,w) & -(x=WEST_BOUNDARY) & dir(z,f(ay,y,-1),w)
& NOTFREE(MINX,y,w) then PLACE(z,WEST_BOUNDARY,y,w);
**_____;*
**non-boundaries, free ;*
**_____;*
**north;*
if MOVE(z,w) & loc(z,x,y,w) & dir(z,f(ax,x,1),w) & FREE(x,MAXY,w)
then PLACE(z,x,MAXY,w);
**south;*
if MOVE(z,w) & loc(z,x,y,w) & dir(z,f(ax,x,-1),w) & FREE(x,MINY,w)
then PLACE(z,x,MINY,w);
**east;*
if MOVE(z,w) & loc(z,x,y,w) & dir(z,f(ay,y,1),w) & FREE(MAXX,y,w)
then PLACE(z,MAXX,y,w);
**west;*

```

    if MOVE(z,w) & loc(z,x,y,w) & dir(z,f(ay,y,-1),w) & FREE(MINX,y,w)
then PLACE(z,MINX,y,w);

    *_____ ;

    *updates after motion ;

    *_____ ;

    *boundary - fix;

    if PLACE(z,x,NORTH_BOUNDARY,w) then FIX(z,w+1);

    if PLACE(z,x,SOUTH_BOUNDARY,w) then FIX(z,w+1);

    if PLACE(z,EAST_BOUNDARY,y,w) then FIX(z,w+1);

    if PLACE(z,WEST_BOUNDARY,y,w) then FIX(z,w+1);

    *goal placement - no fix;

    if PLACE(z,x,MAXY,w) then NOFIX(w+1);

    if PLACE(z,x,MINY,w) then NOFIX(w+1);

    if PLACE(z,MAXX,y,w) then NOFIX(w+1);

    if PLACE(z,MINX,y,w) then NOFIX(w+1);

    *update for object moved;

    if PLACE(z,x,MAXY,w) then -P(z,w+1);

    if PLACE(z,x,MINY,w) then -P(z,w+1);

    if PLACE(z,MAXX,y,w) then -P(z,w+1);

    if PLACE(z,MINX,y,w) then -P(z,w+1);

    if PLACE(z,x,y,w) then loc(z,x,y,w+1);

```

**updates for other objects;*
 if $PLACE(z,x,y,w) \ \& \ P(zl,w) \ \& \ -(z=zl)$ then $P(zl,w+1)$;
 if $PLACE(z,x,y,w) \ \& \ dir(zl \ f(xy,xyk,xy1),w) \ \& \ -(z=zl)$
 then $dir(zl \ f(xy,xyk,xy1),w+1)$;
 if $PLACE(z,x,y,w) \ \& \ loc(zl,x1,y1,w) \ \& \ -(z=zl)$ then $loc(zl,x1,y1,w+1)$;
 if $PLACE(z,x,y,w) \ \& \ -P(zl,w)$ then $-P(zl,w+1)$;
**updates for free spaces;*
 if $PLACE(z,x,y,w) \ \& \ NOTFREE(x1,y1,w)$ then $NOTFREE(x1,y1,w+1)$;
 if $PLACE(z,x,y,w) \ \& \ FREE(x1,y1,w) \ \& \ -(x=x1) \ \& \ -(y=y1)$
 then $FREE(x1,y1,w+1)$;
 if $PLACE(z,x,y,w) \ \& \ FREE(x1,y1,w) \ \& \ -(x=x1)$ then $FREE(x1,y1,w+1)$;
 if $PLACE(z,x,y,w) \ \& \ FREE(x1,y1,w) \ \& \ -(y=y1)$ then $FREE(x1,y1,w+1)$;
**updates for goal placements/not free;*
 if $PLACE(z,x,MAXY,w)$ then $NOTFREE(x,MAXY,w+1)$;
 if $PLACE(z,x,MINY,w)$ then $NOTFREE(x,MINY,w+1)$;
 if $PLACE(z,MAXX,y,w)$ then $NOTFREE(MAXX,y,w+1)$;
 if $PLACE(z,MINX,y,w)$ then $NOTFREE(MINX,y,w+1)$;
 *;
 if $loc(z,x,MAXY,w)$ then $-P(z,w)$;
 if $loc(z,x,MINY,w)$ then $-P(z,w)$;
 if $loc(z,MAXX,y,w)$ then $-P(z,w)$;

```

if loc(z,MINX,y,w) then -P(z,w);

*next stage;

if PLACE(z,x,y,w) then poset(w+1);

if FIX(z,w) then MOVE(z,w);

if NOFIX(w) then INITIAL(w);

*set initial direction weights;

if P(p(z),0) then GTWT(p(z),0);

; *——;

*Set of Support;

*——;

EQUAL(N,k,w);

poset(0);

START(0);

P(p(1),0);

loc(p(1),...);

FREE(0MINX,X,0);

FREE(MAXX,X,0);

FREE(X,MAXY,0);

FREE(X,MINY,0);

;

;

```

;

Appendix D

Quad2-directional separability for two arms

```

*_____ ;
*get direction weights ;
*_____ ;

if poset(w) & GTWT(z,w) & P(z,w) & loc(z,x,y,w) & (y>=MAXY/2)
& (x>=MAXY-y) & (y>=x) then dirwt(z,1f(ax,x,1),w);

if poset(w) & GTWT(z,w) & P(z,w) & loc(z,x,y,w) & (y<MAXY/2)
& (x>=y) & (MAXX-x>=y) then dirwt(z,1f(ax,x,-1),w);

if poset(w) & GTWT(z,w) & P(z,w) & loc(z,x,y,w) & (x>=MAXX/2)
& (y>MAXX-x) & (x>y) then dirwt(z,1f(ay,y,1),w);

if poset(w) & GTWT(z,w) & P(z,w) & loc(z,x,y,w) & (x<MAXX/2)
& (MAXY-y>x) & (y>x) then dirwt(z,1f(ay,y,-1),w);

if poset(w) & GTWT(z,w) & P(z,w) & loc(z,x,y,w) & dirwt(z,1f(ax,x,xy),w)
& (y>MAXY/2) then dirwt(z,2f(ay,y,1),w);

if poset(w) & GTWT(z,w) & P(z,w) & loc(z,x,y,w) & dirwt(z,1f(ax,x,xy),w)
& (y<=MAXY/2) then dirwt(z,2f(ay,y,-1),w);

if poset(w) & GTWT(z,w) & P(z,w) & loc(z,x,y,w) & dirwt(z,1f(ay,y,xy),w)
& (x>MAXX/2) then dirwt(z,2f(ax,x,1),w);

```

if poset(w) & GTWT(z,w) & P(z,w) & loc(z,x,y,w) & dirwt(z,1 f(ay,y,xy),w)
 & (x<=MAXX/2) then dirwt(z,2 f(ax,x,-1),w);
 if poset(w) & GTWT(z,w) & P(z,w) & loc(z,x,y,w) & dirwt(z,2 f(ay,y,xy),w)
 then dirwt(z,3 f(ay,y,xy*(-1)),w);
 if poset(w) & GTWT(z,w) & P(z,w) & loc(z,x,y,w) & dirwt(z,2 f(ax,x,xy),w)
 then dirwt(z,3 f(ax,x,xy*(-1)),w);
 if poset(w) & GTWT(z,w) & P(z,w) & loc(z,x,y,w) & dirwt(z,1 f(ay,y,xy),w)
 then dirwt(z,4 f(ay,y,xy*(-1)),w);
 if poset(w) & GTWT(z,w) & P(z,w) & loc(z,x,y,w) & dirwt(z,1 f(ax,x,xy),w)
 then dirwt(z,4 f(ax,x,xy*(-1)),w);
 *_____;
 *determine precedences ;
 *_____;
 *north/south;
 if poset(w) & P(z1,w) & P(z2,w) & loc(z1,x1,y1,w) & loc(z2,x1,y2,w)
 & dir(z1 f(ax,x1,xy1),w) & (((y2-y1)*xy1) > 0) then prec(z1,z2,w);
 if poset(w) & P(z1,w) & FIX(z2,w,arm(wr)) & loc(z1,x1,y1,w) & loc(z2,x1,y2,w)
 & dir(z1 f(ax,x1,xy1),w) & ((y2-y1)*xy1) > 0) then prec(z1,z2,w);
 *east/west;
 if poset(w) & P(z1,w) & P(z2,w) & loc(z1,x1,y1,w) & loc(z2,x2,y1,w)
 & dir(z1 f(ay,y1,xy1),w) & (((x2-x1)*xy1) > 0) then prec(z1,z2,w);

```

    if poset(w) & P(z1,w) & FIX(z2,w,arm(wr)) & loc(z1,x1,y1,w) & loc(z2,x2,y1,w)
    & dir(z1,f(ay,y1,xy1),w) & (((x2-x1)*xy1) > 0) then prec(z1,z2,w);

    *_____ ;

    *possible -prec;

    *_____ ;

    *north/south;

    if poset(w) & P(z1,w) & P(z2,w) & loc(z1,x1,y1,w) & loc(z2,x1,y2,w)
    & dir(z1,f(ax,x1,xy1),w) & (((y2-y1)*xy1) <= 0) then -prec(z1,z2,w);

    if poset(w) & P(z1,w) & P(z2,w) & loc(z1,x1,y1,w) & loc(z2,x2,y2,w)
    & dir(z1,f(ax,x1,xy1),w) & -(x1=x2) then -prec(z1,z2,w);

    *east/west;

    if poset(w) & P(z1,w) & P(z2,w) & loc(z1,x1,y1,w) & loc(z2,x2,y1,w)
    & dir(z1,f(ay,y1,xy1),w) & (((x2-x1)*xy1) <= 0) then -prec(z1,z2,w);

    if poset(w) & P(z1,w) & P(z2,w) & loc(z1,x1,y1,w) & loc(z2,x2,y2,w)
    & dir(z1,f(ay,y1,xy1),w) & -(y1=y2) then -prec(z1,z2,w);

    *_____ ;

    *generate poset ;

    *_____ ;

    *set up p(1);

    if P(p(1),0) & dirwt(p(1),1,f(xy,xyk,xy1),0) then dir(p(1),f(xy,xyk,xy1),0);

    if dir(p(z),f(xy,xyk,xy1),0) & EQUAL(N,zk,0) & -(z=zk) & START(0)

```

then GTDIR(p(z+1),z,0);
if dir(p(z)f(xy,xyk,xy1),0) & EQUAL(N,z,0) then INITIAL(0);
**_____;*
**get generators ;*
**_____;*
if GTDIR(p(z),wz,w) & P(p(y),w) & (y<=wz) then LGEN(y-1,p(y),w);
if GTDIR(p(z),wz,w) & -P(p(y),w) & (y<=wz) then NLGEN(y-1,p(y),w);
if GTDIR(p(z),wz,w) & NPG(y-1,p(y),w) then NLGEN(y-1,p(y),w);
if GTDIR(p(z),wz,w) then GTLC(p(z),p(z),p(z),0,wz,w);
**_____;*
**get lower covers for an element ;*
**partial only from 1 to i-1 ;*
**_____;*
**_____z = z1 and z2;*
if GTLC(p(z),p(z),p(z),wy,wz,w) & LGEN(wy,p(y),w) & (y<=wz)
& prec(p(y),p(z),w) then GTLC(p(y),p(y),p(z),wy+1,wz,w);
**_____z = z1, z != z2;*
if GTLC(p(z),p(z2),p(z),wy,wz,w) & LGEN(wy,p(y),w) & (y<=wz) & -(z=z2) &
prec(p(y),p(z),w) & -prec(p(z2),p(y),w) then GTLC(p(y),p(z2),p(z),wy+1,wz,w);
if GTLC(p(z),p(z2),p(z),wy,wz,w) & LGEN(wy,p(y),w) & (y<=wz) & \$NE(z,z2) &
prec(p(y),p(z),w) & prec(p(z2),p(y),w) then GTLC(p(y),p(y),p(z),wy+1,wz,w);

*——z = z2, z != z1;

if GTLC(p(z1),p(z),p(z),wy,wz,w) & LGEN(wy,p(y),w) & (y<=wz) & -(z=z1) &
prec(p(y),p(z),w) & -prec(p(z1),p(y),w) then GTLC(p(z1),p(y),p(z),wy+1,wz,w);

if GTLC(p(z1),p(z),p(z),wy,wz,w) & LGEN(wy,p(y),w) & (y<=wz) & -(z=z1) &
prec(p(y),p(z),w) & prec(p(z1),p(y),w) then GTLC(p(y),p(y),p(z),wy+1,wz,w);

*——z != z1, z != z2;

if GTLC(p(z1),p(z2),p(z),wy,wz,w) & LGEN(wy,p(y),w) & (y<=wz) & -(z=z1) &
-(z=z2) & prec(p(y),p(z),w) & prec(p(z1),p(y),w) & -prec(p(z2),p(y),w)
then GTLC(p(y),z2,p(z),wy+1,wz,w);

if GTLC(p(z1),p(z2),p(z),wy,wz,w) & LGEN(wy,p(y),w) & (y<=wz) & -(z=z1) &
-(z=z2) & prec(p(y),p(z),w) & prec(p(z2),p(y),w) & -prec(p(z1),p(y),w)
then GTLC(p(z1),p(y),p(z),wy+1,wz,w);

if GTLC(p(z1),p(z2),p(z),wy,wz,w) & LGEN(wy,p(y),w) & (y<=wz) & -(z=z1) &
-(z=z2) & prec(p(y),p(z),w) & prec(p(z1),p(y),w) & prec(p(z2),p(y),w)
then GTLC(p(y),p(y),p(z),wy+1,wz,w);

if GTLC(p(z1),p(z2),p(z),wy,wz,w) & LGEN(wy,p(y),w) & (y<=wz)
& -prec(p(y),p(z),w) then GTLC(p(z1),p(z2),p(z),wy+1,wz,w);

if GTLC(p(z1),p(z2),p(z),wy,wz,w) & NLGEN(wy,p(y),w) & (y<=wz)
then GTLC(p(z1),p(z2),p(z),wy+1,wz,w);

if GTLC(p(z1),p(z2),p(z),wy,wz,w) & LGEN(wy,p(y),w) & -(w=0) & (y=z) then
GTLc(p(z1),p(z2),p(z),wy+1,wz,w);

```

*_____done rule _____;

if GTLC(p(z1),p(z2),p(z),wy,wz,0) & GTDIR(p(z),wy,0) & (z=(wy+1))
then LCOV(p(z1),p(z2),p(z),0);

    if GTLC(p(z1),p(z2),p(z),wy,wz,w) & GTDIR(p(z),wy,w) & -(w=0)
& EQUAL(N,wy,w) then LCOV(p(z1),p(z2),p(z),w);

*_____;

*set direction of z;

*_____;

*case 1: no lower covers ;

*_____;

    if LCOV(p(z),p(z),p(z),w) & dirwt(p(z),lf(xy,xyk,xy1),w) & START(w) then
dir(p(z),f(xy,xyk,xy1),w);

*_____;

*case 2: one lower cover ;

*_____;

    if LCOV(p(y),p(y),p(z),w) & dir(p(y),f(xy,xyk,xy1),w) & START(w)
then dir(p(z),f(xy,xyk,xy1),w);

*_____;

*case 3: two lower covers - get best weight ;

*_____;

    if LCOV(p(z1),p(z2),p(z),w) & -(z1=z2) & dir(p(z1),f(xy1,xyk1,xy11),w)

```

$\& \text{dir}(p(z2), f(xy2, xyk2, xy22), w) \& \text{dirwt}(p(z), wk1, f(xy1, xyk1, xy11), w)$
 $\& \text{dirwt}(p(z), wk2, f(xy2, xyk2, xy22), w) \& (wk1 > wk2) \& \text{START}(w)$
then $\text{dir}(p(z), f(xy1, xyk1, xy11), w);$
 if $\text{LCOV}(p(z1), p(z2), p(z), w) \& \neg(z1 = z2) \& \text{dir}(p(z1), f(xy1, xyk1, xy11), w)$
 $\& \text{dir}(p(z2), f(xy2, xyk2, xy22), w) \& \text{dirwt}(p(z), wk1, f(xy1, xyk1, xy11), w)$
 $\& \text{dirwt}(p(z), wk2, f(xy2, xyk2, xy22), w) \& (wk2 > wk1) \& \text{START}(w)$
then $\text{dir}(p(z), f(xy2, xyk2, xy22), w);$
 *_____;
 **clauses to generate maximal elements ;*
 *_____;
 if $\text{INITIAL}(w) \& P(p(z), w)$ *then* $\text{PG}(z-1, p(z), w);$
 if $\text{INITIAL}(w) \& \neg P(p(z), w)$ *then* $\text{NPG}(z-1, p(z), w);$
 if $\text{CHKMAX}(wz, w) \& \text{PG}(y, wz, w)$ *then* $\text{MAXSOFAR}(wz, 0, wz, w);$
 if $\text{CHKMAX}(wz, w) \& \text{NPG}(y, wz, w)$ *then* $\text{NOTMAX}(wz, w);$
 if $\text{MAXSOFAR}(wz, z, wz, w) \& \text{PG}(z, y, w) \& (y = wz)$
then $\text{MAXSOFAR}(wz, z+1, wz, w);$
 if $\text{MAXSOFAR}(wz, z, wz, w) \& \text{PG}(z, y, w) \& \neg(y = wz) \& \text{prec}(wz, y, w)$
then $\text{NOTMAX}(wz, w);$
 if $\text{MAXSOFAR}(wz, z, wz, w) \& \text{PG}(z, y, w) \& \neg(y = wz) \& \neg \text{prec}(wz, y, w)$
then $\text{MAXSOFAR}(wz, z+1, wz, w);$
 if $\text{MAXSOFAR}(wz, z, wz, w) \& \text{NPG}(z, y, w)$ *then* $\text{MAXSOFAR}(wz, z+1, wz, w);$

if MAXSOFAR(wz,z,wz,w) & EQUAL(N,z,w) then NEWMAX(wz,w);
 if GETALLMAX(z,wk,w) then CHKMAX(z,w);
 if GETALLMAX(wz,wk,w) & NEWMAX(wz,w) then MAX(wz,wk+1,w);
 if GETALLMAX(p(wz),wk,w) & NEWMAX(p(wz),w) & EQUAL(N,wn,w)
 & (wz < wn) then GETALLMAX(p(wz+1),wk+1,w);
 if GETALLMAX(p(wz),wk,w) & NEWMAX(p(wz),w) & EQUAL(N,wz,w)
 then ALLMAX(wk+1,w);
 if GETALLMAX(p(wz),wk,w) & NEWMAX(p(wz),w) & EQUAL(N,wz,w)
 then MEMMAX(1,w);
 if GETALLMAX(p(wz),wk,w) & NOTMAX(p(wz),w) & EQUAL(N,wz,w)
 then MEMMAX(1,w);
 if GETALLMAX(p(wz),wk,w) & NOTMAX(p(wz),w) & EQUAL(N,wn,w)
 & (wz < wn) then GETALLMAX(p(wz+1),wk,w);
 if GETALLMAX(p(wz),wk,w) & NOTMAX(p(wz),w) & EQUAL(N,wz,w)
 then ALLMAX(wk,w);
 *_____;
 *generate initial moves *adjust for FIX/NOFIX cases;
 *_____;
 if MAX(wz,1,0) then TMOVE(wz,arm(1),0);
 if MAX(wz,1,0) then NOFIX(0,arm(1));
 if MAX(wz,1,0) then NOFIX(0,arm(2));

if $MAX(wz,1,w) \ \& \ NOFIX(w,arm(1)) \ \& \ \$GT(w,0)$
then $TMOVE(wz,arm(1),w);$
if $MAX(wz,1,w) \ \& \ FIX(z,w,arm(1)) \ \& \ NOFIX(w,arm(2))$
then $TMOVE(wz,arm(2),w);$
if $TMOVE(wz,arm(1),w) \ \& \ NOFIX(w,arm(1)) \ \& \ NOFIX(w,arm(2))$
 $\ \& \ MAX(wz,wk,w) \ \& \ MAX(wz2,wk1,w) \ \& \ (wk1=(wk+1))$
then $TMOVE(wz2,arm(2),w);$
if $TMOVE(wz,arm(1),0) \ \& \ poset(0) \ \& \ MAX(wz,wk,0) \ \& \ MAX(wz2,wk1,0)$
 $\ \& \ (wk1=(wk+1))$ *then* $TMOVE(wz2,arm(2),0);$
if $TMOVE(wz,arm(wr1),w) \ \& \ FIX(wz,w,arm(wr1)) \ \& \ NOFIX(w,arm(wr2))$
 $\ \& \ ALLMAX(0,w)$ *then* $NOMOVE(arm(wr2),w);$
if $TMOVE(wz,arm(1),w) \ \& \ NOFIX(w,arm(1)) \ \& \ NOFIX(w,arm(2))$
 $\ \& \ MAX(wz,wk,w) \ \& \ ALLMAX(wk,w)$ *then* $NOMOVE(arm(2),w);$
if $TMOVE(wz,arm(1),w) \ \& \ FIX(wz,w,arm(1)) \ \& \ NOFIX(w,arm(2))$
 $\ \& \ MAX(wz,1,w) \ \& \ ALLMAX(1,w)$ *then* $NOMOVE(arm(2),w);$
if $TMOVE(wz,arm(2),w) \ \& \ FIX(wz,w,arm(2)) \ \& \ NOFIX(w,arm(1))$
 $\ \& \ MAX(wz,1,w) \ \& \ ALLMAX(1,w)$ *then* $NOMOVE(arm(1),w);$
if $NEWTRIAL(arm(wr),w) \ \& \ TMOVE(wz2,arm(wr),w) \ \& \ MAX(wz2,wk,w)$
 $\ \& \ MAX(wz3,wk+1,w)$ *then* $TMOVE(wz3,arm(wr),w);$
if $NEWTRIAL(arm(wr),w) \ \& \ TMOVE(wz2,arm(wr),w) \ \& \ MAX(wz2,wk,w)$
 $\ \& \ ALLMAX(wk,w)$ *then* $NOMOVE(arm(wr),w);$

```

*_____;
```

**determine move - boundary vs. no boundary ;*

```

*_____;
```

**basic strategy: maximal element is moved until placed at goal;*

**until placed - FIX, when placed - NOFIX;*

**if MAXIMAL(z,w) then MOVE(z,w);*

**if MAXIMAL(z,w) & NOFIX(w,arm(1)) then TMOVE(z,arm(1),w);*

**if MAXIMAL(z,w) & FIX(z,w,arm(1)) & NOFIX(w,arm(2))*
then TMOVE(z,arm(2),w);

```

*_____;
```

**boundaries, no corners, not free;*

```

*_____;
```

**north;*

if TMOVE(z,arm(wz),w) & loc(z,x,NORTH_BOUNDARY,w)
& (x>WEST_BOUNDARY) & (x<EAST_BOUNDARY)
& NOTFREE(x,MAXY,w)
then TRIAL(z,x-1,NORTH_BOUNDARY,w,arm(wz));

if TMOVE(z,arm(wz),w) & loc(z,x,NORTH_BOUNDARY,w)
& (x>WEST_BOUNDARY) & (x<EAST_BOUNDARY)
& NOTFREE(x,MAXY,w)
then IVEC(z,f(ay,NORTH_BOUNDARY,-1),w);

```

*south;

  if TMOVE(z,arm(wz),w) & loc(z,x,SOUTH_BOUNDARY,w)
& (x>WEST_BOUNDARY) & (x<EAST_BOUNDARY)
& NOTFREE(x,MINY,w)
then TRIAL(z,x+1,SOUTH_BOUNDARY,w,arm(wz));

    if TMOVE(z,arm(wz),w) & loc(z,x,SOUTH_BOUNDARY,w)
& (x>WEST_BOUNDARY) & (x<EAST_BOUNDARY)
& NOTFREE(x,MINY,w)
then IVEC(z,f(ay,SOUTH_BOUNDARY,1),w);

*east;

  if TMOVE(z,arm(wz),w) & loc(z,EAST_BOUNDARY,y,w)
& (y>SOUTH_BOUNDARY) & (y<NORTH_BOUNDARY)
& NOTFREE(MAXX,y,w)
then TRIAL(z,EAST_BOUNDARY,y+1,w,arm(wz));

    if TMOVE(z,arm(wz),w) & loc(z,EAST_BOUNDARY,y,w)
& (y>SOUTH_BOUNDARY) & (y<NORTH_BOUNDARY)
& NOTFREE(MAXX,y,w)
then IVEC(z,f(ax,EAST_BOUNDARY,1),w);

*west;

  if TMOVE(z,arm(wz),w) & loc(z,WEST_BOUNDARY,y,w)
& (y>SOUTH_BOUNDARY) & (y<NORTH_BOUNDARY)

```

```

& NOTFREE(MINX,y,w)
then TRIAL(z,MINX,y-1,w,arm(wz));
    if TMOVE(z,arm(wz),w) & loc(z,WEST_BOUNDARY,y,w)
    & (y>SOUTH_BOUNDARY) & (y<NORTH_BOUNDARY)
    & NOTFREE(MINX,y,w)
then IVEC(z,f(ax,MINX,-1),w);
    *_____ ;
    *boundaries, corners, not free;
    *_____ ;
    if TMOVE(z,arm(wz),w) & loc(z,WEST_BOUNDARY,NORTH_BOUNDARY,w)
    & NOTFREE(MINX,NORTH_BOUNDARY,w)
    & NOTFREE(WEST_BOUNDARY,MAXY,w)
then TRIAL(z,WEST_BOUNDARY,NORTH_BOUNDARY-1,w,arm(wz));
    if TMOVE(z,arm(wz),w) & loc(z,WEST_BOUNDARY,NORTH_BOUNDARY,w)
    & NOTFREE(MINX,NORTH_BOUNDARY,w)
    & NOTFREE(WEST_BOUNDARY,MAXY,w)
then IVEC(z,f(ax,WEST_BOUNDARY,-1),w);
    if TMOVE(z,arm(wz),w) & loc(z,WEST_BOUNDARY,SOUTH_BOUNDARY,w)
    & NOTFREE(WEST_BOUNDARY,MINY,w)
    & NOTFREE(MINX,SOUTH_BOUNDARY,w)
then TRIAL(z,WEST_BOUNDARY+1,SOUTH_BOUNDARY,w,arm(wz));

```

if TMOVE(z,arm(wz),w) & loc(z,WEST_BOUNDARY,SOUTH_BOUNDARY,w)
& NOTFREE(WEST_BOUNDARY,MINY,,w)
& NOTFREE(MINX,SOUTH_BOUNDARY,w)
then IVEC(z,f(ay,SOUTH_BOUNDARY,1),w);

if TMOVE(z,arm(wz),w) & loc(z,EAST_BOUNDARY,SOUTH_BOUNDARY,w)
& NOTFREE(MAXX,SOUTH_BOUNDARY,w)
& NOTFREE(EAST_BOUNDARY,MINY,w)
then TRIAL(z,EAST_BOUNDARY,SOUTH_BOUNDARY+1,w,arm(wz));

if TMOVE(z,arm(wz),w) & loc(z,EAST_BOUNDARY,SOUTH_BOUNDARY,w)
& NOTFREE(MAXX,SOUTH_BOUNDARY,w)
& NOTFREE(EAST_BOUNDARY,MINY,w)
then IVEC(z,f(ax,EAST_BOUNDARY,1),w);

if TMOVE(z,arm(wz),w) & loc(z,EAST_BOUNDARY,NORTH_BOUNDARY,w)
& NOTFREE(MAXX,NORTH_BOUNDARY,w)
& NOTFREE(EAST_BOUNDARY,MAXY,w)
then TRIAL(z,EAST_BOUNDARY-1,NORTH_BOUNDARY,w,arm(wz));

if TMOVE(z,arm(wz),w) & loc(z,EAST_BOUNDARY,NORTH_BOUNDARY,w)
& NOTFREE(MAXX,NORTH_BOUNDARY,w)
& NOTFREE(EAST_BOUNDARY,MAXY,w)
then IVEC(z,f(ay,NORTH_BOUNDARY,-1),w);

**_____;*

**boundaries, free;*
**_____;*
 if *TMOVE(z,arm(wz),w) & loc(z,x,NORTH_BOUNDARY,w) & FREE(x,MAXY,w)*
 then *TRIAL(z,x,MAXY,w,arm(wz));*
 if *TMOVE(z,arm(wz),w) & loc(z,x,NORTH_BOUNDARY,w) & FREE(x,MAXY,w)*
 then *IVEC(z,f(ax,x,1),w);*
 if *TMOVE(z,arm(wz),w) & loc(z,x,SOUTH_BOUNDARY,w) & FREE(x,MINY,w)*
 then *TRIAL(z,x,MINY,w,arm(wz));*
 if *TMOVE(z,arm(wz),w) & loc(z,x,SOUTH_BOUNDARY,w) & FREE(x,MINY,w)*
 then *IVEC(z,f(ax,x,-1),w);*
 if *TMOVE(z,arm(wz),w) & loc(z,EAST_BOUNDARY,y,w) & FREE(MAXX,y,w)*
 then *TRIAL(z,MAXX,y,w,arm(wz));*
 if *TMOVE(z,arm(wz),w) & loc(z,EAST_BOUNDARY,y,w) & FREE(MAXX,y,w)*
 then *IVEC(z,f(ay,y,1),w);*
 if *TMOVE(z,arm(wz),w) & loc(z,WEST_BOUNDARY,y,w) & FREE(MINX,y,w)*
 then *TRIAL(z,MINX,y,w,arm(wz));*
 if *TMOVE(z,arm(wz),w) & loc(z,WEST_BOUNDARY,y,w) & FREE(MINX,y,w)*
 then *IVEC(z,f(ay,y,-1),w);*
**_____;*
**non-boundaries, not free;*
**_____;*

```

*north;

  if TMOVE(z,arm(wz),w) & loc(z,x,y,w) & -(y=NORTH_BOUNDARY)
& dir(z,f(ax,x,1),w) & NOTFREE(x,MAXY,w)
then TRIAL(z,x,NORTH_BOUNDARY,w,arm(wz));

*south;

  if TMOVE(z,arm(wz),w) & loc(z,x,y,w) & -(y=SOUTH_BOUNDARY)
& dir(z,f(ax,x,-1),w) & NOTFREE(x,MINY,w)
then TRIAL(z,x,SOUTH_BOUNDARY,w,arm(wz));

*east;

  if TMOVE(z,arm(wz),w) & loc(z,x,y,w) & -(x=EAST_BOUNDARY)
& dir(z,f(ay,y,1),w) & NOTFREE(MAXX,y,w)
then TRIAL(z,EAST_BOUNDARY,y,w,arm(wz));

*west;

  if TMOVE(z,arm(wz),w) & loc(z,x,y,w) & -(x=WEST_BOUNDARY)
& dir(z,f(ay,y,-1),w) & NOTFREE(MINX,y,w)
then TRIAL(z,WEST_BOUNDARY,y,w,arm(wz));

  *_____ ;

  *non-boundaries, free ;

  *_____ ;

*north;

  if TMOVE(z,arm(wz),w) & loc(z,x,y,w) & dir(z,f(ax,x,1),w) & FREE(x,MAXY,w)

```

```

then TRIAL(z,x,MAXY,w,arm(wz));

    *south

    ; if TMOVE(z,arm(wz),w) & loc(z,x,y,w) & dir(z,f(ax,x,-1),w) & FREE(x,MINY,w)
then TRIAL(z,x,MINY,w,arm(wz));

    *east;

    if TMOVE(z,arm(wz),w) & loc(z,x,y,w) & dir(z,f(ay,y,1),w) & FREE(MAXX,y,w)
then TRIAL(z,MAXX,y,w,arm(wz));

    *west;

    if TMOVE(z,arm(wz),w) & loc(z,x,y,w) & dir(z,f(ay,y,-1),w) & FREE(MINX,y,w)
then TRIAL(z,MINX,y,w,arm(wz));

    *_____ ;

    *Collision check - N.B. different from bi-directional;

    *_____ ;

    *directions are the same - no collision;

    if TRIAL(z1,x1,y1,w,arm(wz1)) & TRIAL(z2,x2,y2,w,arm(wz2))
& -(z1=z2) & dir(z1,f(xy,xyl,xyk),w) & dir(z2,f(xy,xy2,xyk),w)
then GETMIN(z1,z2,x1,y1,x2,y2,w);

    *if TRIAL(z1,x1,y1,w,arm(wz1)) & TRIAL(z2,x2,y2,w,arm(wz2))
& -(z1=z2) & dir(z1,f(ay,y1),w) & dir(z2,f(ay,y2),w)
then GETMIN(z1,z2,x1,y1,x2,y2,w);

    *directions are different;

```

**opposite directions but different vectors;*

if TRIAL($z1, x1, y1, w, arm(wz1)$) & TRIAL($z2, x2, y2, w, arm(wz2)$)
& $-(z1=z2)$ & $dir(z1, f(xy, xy1, xyk), w)$ & $dir(z2, f(xy, xy2, xyk2), w)$
then GETMIN($z1, z2, x1, y1, x2, y2, w$);

if TRIAL($z1, x1, y1, w, arm(wz1)$) & TRIAL($z2, x2, y2, w, arm(wz2)$)
& $-(z1=z2)$ & $IVEC(z1, f(xy, xy1, xyk), w)$ & $dir(z2, f(xy, xy2, xyk2), w)$
then GETMIN($z1, z2, x1, y1, x2, y2, w$);

if TRIAL($z1, x1, y1, w, arm(wz1)$) & TRIAL($z2, x2, y2, w, arm(wz2)$)
& $-(z1=z2)$ & $dir(z1, f(xy, xy1, xyk), w)$ & $IVEC(z2, f(xy, xy2, xyk2), w)$
then GETMIN($z1, z2, x1, y1, x2, y2, w$);

if TRIAL($z1, x1, y1, w, arm(wz1)$) & TRIAL($z2, x2, y2, w, arm(wz2)$)
& $-(z1=z2)$ & $IVEC(z1, f(xy, xy1, xyk), w)$ & $IVEC(z2, f(xy, xy2, xyk2), w)$
then GETMIN($z1, z2, x1, y1, x2, y2, w$);

**collision ;*

**opposite directions but same vectors - not possible;*

**South/East;*

**explanation;*

**1. assume that $arm(wz1)$ is moving $z1$ from $x1, y1$ to $x1, y12$;*

**direction is north/south $x = x1$ ($ax, x1, xk$);*

**2. assume that $arm(wz2)$ is moving $z2$ from $x2, y2$ to $x22, y2$;*

**direction is east/west $y = y2$ ($ay, y2, yk$);*

**then using static region analysis, there will be a collision if;*
**a. x_1 is between x_2 and x_{22} adjust for $y = \text{direction} \pm y_k$;*
**b. y_2 is between y_1 and y_{12} adjust for $x = \text{direction} \pm x_k$;*
**what to do if there is a collision;*
** if $\text{FIX}(\text{arm}(1))$ and $\text{NOFIX}(\text{arm}(2))$ then $\text{NEWTRIAL}(\text{arm}(2))$;*
** if $\text{FIX}(\text{arm}(2))$ and $\text{NOFIX}(\text{arm}(1))$ then $\text{NEWTRIAL}(\text{arm}(1))$ - check above;*
** if $\text{FIX}(\text{arm}(1))$ and $\text{FIX}(\text{arm}(2))$ the idle $\text{arm}(2)$ and $\text{FIX}(\text{arm}(2), w+1)$;*
** resolve with last clauses;*
** if NOFIX either arm, then do default;*
if $\text{TRIAL}(z_1, x_1, y_{12}, w, \text{arm}(wz_1)) \& \text{TRIAL}(z_2, x_{22}, y_2, w, \text{arm}(wz_2)) \& \neg(z_1 = z_2) \&$
 $\text{dir}(z_1 f(ax, x_1, x_k), w) \& \text{dir}(z_2 f(ay, y_2, y_k), w) \& \text{loc}(z_1, x_1, y_1, w) \& \text{loc}(z_2, x_2, y_2, w) \&$
 $((y_2 - y_1) * x_k) \geq 0) \& ((y_{12} - y_2) * x_k) \geq 0) \& ((x_1 - x_2) * y_k) \geq 0)$
 $\& ((x_{22} - x_1) * y_k) \geq 0)$ then $\text{NEWTRIAL}(\text{arm}(2), w)$;
if $\text{TRIAL}(z_1, x_1, y_{12}, w, \text{arm}(wz_1)) \& \text{TRIAL}(z_2, x_{22}, y_2, w, \text{arm}(wz_2)) \& \neg(z_1 = z_2) \&$
 $\text{dir}(z_1 f(ax, x_1, x_k), w) \& \text{dir}(z_2 f(ay, y_2, y_k), w) \& \text{loc}(z_1, x_1, y_1, w) \& \text{loc}(z_2, x_2, y_2, w) \&$
 $((y_2 - y_1) * x_k) \geq 0) \& ((y_{12} - y_2) * x_k) \geq 0) \& ((x_1 - x_2) * y_k) \geq 0)$
 $\& ((x_{22} - x_1) * y_k) \geq 0)$ then $\text{CKFIX}(w)$;
if $\text{CKFIX}(w) \& \text{FIX}(z, w, \text{arm}(1)) \& \text{NOFIX}(w, \text{arm}(2))$
then $\text{NEWTRIAL}(\text{arm}(2), w)$;
if $\text{CKFIX}(w) \& \text{FIX}(z, w, \text{arm}(1)) \& \text{FIX}(z_2, w, \text{arm}(2))$
then $\text{NOMOVE}(\text{arm}(2), w)$;

```

    if CKFIX(w) & NOFIX(w,arm(1)) & FIX(z,w,arm(2))
then NEWTRIAL(arm(1),w);

    if CKFIX(w) & NOFIX(w,arm(1)) & NOFIX(w,arm(2))
then NEWTRIAL(arm(2),w);

    *no collision cases - negation of above;

    *four possibilities;

    if TRIAL(z1,x1,y12,w,arm(wz1)) & TRIAL(z2,x22,y2,w,arm(wz2)) & -(z1=z2) &
dir(z1 f(ax,x1,xk),w) & dir(z2 f(ay,y2,yk),w) & loc(z1,x1,y1,w) & loc(z2,x2,y2,w)
& ((y2-y1)*xk) < 0) then GETMIN(z1,z2,x1,y12,x22,y2,w);

    if TRIAL(z1,x1,y12,w,arm(wz1)) & TRIAL(z2,x22,y2,w,arm(wz2)) & -(z1=z2) &
dir(z1 f(ax,x1,xk),w) & dir(z2 f(ay,y2,yk),w) & loc(z1,x1,y1,w) & loc(z2,x2,y2,w)
& ((y12-y2)*xk) < 0) then GETMIN(z1,z2,x1,y12,x22,y2,w);

    if TRIAL(z1,x1,y12,w,arm(wz1)) & TRIAL(z2,x22,y2,w,arm(wz2)) & -(z1=z2) &
dir(z1 f(ax,x1,xk),w) & dir(z2 f(ay,y2,yk),w) & loc(z1,x1,y1,w) & loc(z2,x2,y2,w)
& ((x1-x2)*yk) < 0) then GETMIN(z1,z2,x1,y12,x22,y2,w);

    if TRIAL(z1,x1,y12,w,arm(wz1)) & TRIAL(z2,x22,y2,w,arm(wz2)) & -(z1=z2) &
dir(z1 f(ax,x1,xk),w) & dir(z2 f(ay,y2,yk),w) & loc(z1,x1,y1,w) & loc(z2,x2,y2,w)
& ((x22-x1)*yk) < 0) then GETMIN(z1,z2,x1,y12,x22,y2,w);

    *if an arm is fixed, then no collision;

    if TRIAL(z1,x1,y12,w,arm(wz1)) & TRIAL(z2,x22,y2,w,arm(wz2))
& FIX(z1,w,arm(wz1)) & NOFIX(w,arm(wz2))

```

```

then GETMIN(z1,z2,x1,y12,x22,y2,w);
    if TRIAL(z1,x1,y12,w,arm(wz1)) & TRIAL(z2,x22,y2,w,arm(wz2))
    & FIX(z1,w,arm(wz1)) & FIX(z2,w,arm(wz2)) & -(wz1=wz2)
then GETMIN(z1,z2,x1,y12,x22,y2,w);
    *_____ ;
    *get minimum distance ;
    *_____ ;
    if GETMIN(z1,z2,x1,y1,x2,y2,w) & loc(z1,x01,y01,w) & loc(z2,x02,y02,w)
    & dir(z1,f(xyl,xydl,xyk1),w) & dir(z2,f(xy2,xyd2,xyk2),w)
    & (((x1-x01)+(y1-y01))*xyk1) >= (((x2-x02)+(y2-y02))*xyk2))
then mindist(z1,z2,((x2-x02)+(y2-y02))*xyk2,w);
    if GETMIN(z1,z2,x1,y1,x2,y2,w) & loc(z1,x01,y01,w) & loc(z2,x02,y02,w)
    & IVEC(z1,f(xyl,xydl,xyk1),w) & dir(z2,f(xy2,xyd2,xyk2),w)
    & (((x1-x01)+(y1-y01))*xyk1) >= (((x2-x02)+(y2-y02))*xyk2))
then mindist(z1,z2,((x2-x02)+(y2-y02))*xyk2,w);
    if GETMIN(z1,z2,x1,y1,x2,y2,w) & loc(z1,x01,y01,w) & loc(z2,x02,y02,w)
    & dir(z1,f(xyl,xydl,xyk1),w) & IVEC(z2,f(xy2,xyd2,xyk2),w)
    & (((x1-x01)+(y1-y01))*xyk1) >= (((x2-x02)+(y2-y02))*xyk2))
then mindist(z1,z2,((x2-x02)+(y2-y02))*xyk2,w);
    if GETMIN(z1,z2,x1,y1,x2,y2,w) & loc(z1,x01,y01,w) & loc(z2,x02,y02,w)
    & dir(z1,f(xyl,xydl,xyk1),w) & dir(z2,f(xy2,xyd2,xyk2),w)

```

```

& (((x1-x01)+(y1-y01))*xyk1) < (((x2-x02)+(y2-y02))*xyk2))
then mindist(z1,z2,((x1-x01)+(y1-y01))*xyk1,w);

    if GETMIN(z1,z2,x1,y1,x2,y2,w) & loc(z1,x01,y01,w) & loc(z2,x02,y02,w)
    & IVEC(z1 f(xy1,xyd1,xyk1),w) & dir(z2 f(xy2,xyd2,xyk2),w)
    & (((x1-x01)+(y1-y01))*xyk1) < (((x2-x02)+(y2-y02))*xyk2))
    then mindist(z1,z2,((x1-x01)+(y1-y01))*xyk1,w);

        if GETMIN(z1,z2,x1,y1,x2,y2,w) & loc(z1,x01,y01,w) & loc(z2,x02,y02,w)
        & dir(z1 f(xy1,xyd1,xyk1),w) & IVEC(z2 f(xy2,xyd2,xyk2),w)
        & (((x1-x01)+(y1-y01))*xyk1) < (((x2-x02)+(y2-y02))*xyk2))
        then mindist(z1,z2,((x1-x01)+(y1-y01))*xyk1,w);

            *both arms are fixed;

            if GETMIN(z1,z2,x1,y1,x2,y2,w) & loc(z1,x01,y01,w) & loc(z2,x02,y02,w)
            & IVEC(z1 f(xy1,xyd1,xyk1),w) & IVEC(z2 f(xy2,xyd2,xyk2),w)
            & (((x1-x01)+(y1-y01))*xyk1) >= (((x2-x02)+(y2-y02))*xyk2))
            then mindist(z1,z2,((x2-x02)+(y2-y02))*xyk2,w);

                if GETMIN(z1,z2,x1,y1,x2,y2,w) & loc(z1,x01,y01,w) & loc(z2,x02,y02,w)
                & IVEC(z1 f(xy1,xyd1,xyk1),w) & IVEC(z2 f(xy2,xyd2,xyk2),w)
                & (((x1-x01)+(y1-y01))*xyk1) < (((x2-x02)+(y2-y02))*xyk2))
                then mindist(z1,z2,((x1-x01)+(y1-y01))*xyk1,w);

                    if TRIAL(z1,x1,y1,w,arm(1)) & loc(z1,x01,y01,w) & dir(z1 f(xy,xyd,xyk1),w)
                    & NOMOVE(arm(2),w) then dist(z1,((x1-x01)+(y1-y01))*xyk1,w);

```

```

    if TRIAL(z1,x1,y1,w,arm(1)) & loc(z1,x01,y01,w) & IVEC(z1,f(xy,xyd,xyk1),w)
& NOMOVE(arm(2),w) then dist(z1,((x1-x01)+(y1-y01))*xyk1,w);

    *allow NOMOVE possibility for both arms;

    if TRIAL(z1,x1,y1,w,arm(w1)) & loc(z1,x01,y01,w) & NOMOVE(arm(w2),w)
& -(w1=w2) & dir(z1,f(xy,xyd,xyk1),w)
then dist(z1,((x1-x01)+(y1-y01))*xyk1,w);

    if TRIAL(z1,x1,y1,w,arm(w1)) & loc(z1,x01,y01,w) & NOMOVE(arm(w2),w)
& -(w1=w2) & IVEC(z1,f(xy,xyd,xyk1),w)
then dist(z1,((x1-x01)+(y1-y01))*xyk1,w);

    *_____ ;

    *generate placements ;

    *_____ ;

    if TRIAL(z1,x1,y1,w,arm(1)) & TRIAL(z2,x2,y2,w,arm(2))
& mindist(z1,z2,wmin,w) then PUT(z1,arm(1),z2,arm(2),wmin,w);

    if TRIAL(z2,x2,y2,w,arm(2)) & TRIAL(z1,x1,y1,w,arm(1))
& mindist(z2,z1,wmin,w) then PUT(z1,arm(1),z2,arm(2),wmin,w);

    if TRIAL(z1,x1,y1,w,arm(1)) & NOMOVE(arm(2),w) & dist(z1,wmin,w)
then PUT(z1,arm(1),0,arm(2),wmin,w);

    if TRIAL(z1,x1,y1,w,arm(2)) & NOMOVE(arm(1),w) & dist(z1,wmin,w)
then PUT(0,arm(1),z1,arm(2),wmin,w);

    if PUT(z1,arm(wz),z2,arm(wz2),wmin,w) & loc(z1,x,y,w) & dir(z1,f(ax,x,xyk),w)

```

```

then PLACE(z1,x,y+(wmin*xyk),w,arm(wz));

    if PUT(z1,arm(wz),z2,arm(wz2),wmin,w) & loc(z1,x,y,w)
& IVEC(z1 f(ax,x,xyk),w) then PLACE(z1,x,y+(wmin*xyk),w,arm(wz));

    if PUT(z1,arm(wz),z2,arm(wz2),wmin,w) & loc(z1,x,y,w) & dir(z1 f(ay,y,xyk),w)
then PLACE(z1,x+(wmin*xyk),y,w,arm(wz));

    if PUT(z1,arm(wz),z2,arm(wz2),wmin,w) & loc(z1,x,y,w)
& IVEC(z1 f(ay,y,xyk),w) then PLACE(z1,x+(wmin*xyk),y,w,arm(wz));

    if PUT(z1,arm(wz),z2,arm(wz2),wmin,w) & loc(z2,x,y,w) & dir(z2 f(ax,x,xyk),w)
then PLACE(z2,x,y+(wmin*xyk),w,arm(wz2));

    if PUT(z1,arm(wz),z2,arm(wz2),wmin,w) & loc(z2,x,y,w)
& IVEC(z2 f(ax,x,xyk),w) then PLACE(z2,x,y+(wmin*xyk),w,arm(wz2));

    if PUT(z1,arm(wz),z2,arm(wz2),wmin,w) & loc(z2,x,y,w) & dir(z2 f(ay,y,xyk),w)
then PLACE(z2,x+(wmin*xyk),y,w,arm(wz2));

    if PUT(z1,arm(wz),z2,arm(wz2),wmin,w) & loc(z2,x,y,w)
& IVEC(z2 f(ay,y,xyk),w) then PLACE(z2,x+(wmin*xyk),y,w,arm(wz2));

    if PUT(z1,arm(wz),0,arm(wz2),wmin,w) then PLACE(0,0,0,w,arm(wz2));

    if PUT(0,arm(wz),z1,arm(wz2),wmin,w) then PLACE(0,0,0,w,arm(wz1));

    *_____ ;

    *updates after motion ;

    *_____ ;

    *boundary - fix;

```

```

if PLACE(z,x,NORTH_BOUNDARY,w,arm(wz)) then FIX(z,w+1,arm(wz));
if PLACE(z,x,SOUTH_BOUNDARY,w,arm(wz)) then FIX(z,w+1,arm(wz));
if PLACE(z,EAST_BOUNDARY,y,w,arm(wz)) then FIX(z,w+1,arm(wz));
if PLACE(z,WEST_BOUNDARY,y,w,arm(wz)) then FIX(z,w+1,arm(wz));

*goal placement - no fix;

if PLACE(z,x,MAXY,w,arm(wz)) then NOFIX(w+1,arm(wz));
if PLACE(z,x,MINY,w,arm(wz)) then NOFIX(w+1,arm(wz));
if PLACE(z,MAXX,y,w,arm(wz)) then NOFIX(w+1,arm(wz));
if PLACE(z,MINX,y,w,arm(wz)) then NOFIX(w+1,arm(wz));

*neither goal nor boundary - no fix;

if PLACE(z,x,y,w,arm(wz)) & (x<EAST_BOUNDARY)
& (x>WEST_BOUNDARY) & (y<NORTH_BOUNDARY)
& (y>SOUTH_BOUNDARY) then NOFIX(w+1,arm(wz));

if PLACE(z,x,y,w,arm(wz)) & (x<EAST_BOUNDARY)
& (x>WEST_BOUNDARY) & (y<NORTH_BOUNDARY)
& (y>SOUTH_BOUNDARY) & dir(z,f(xy,xyk,xyn),w)
then dir(z,f(xy,xyk,xyn),w+1);

if PLACE(z,x,y,w,arm(wz)) & (x<EAST_BOUNDARY)
& (x>WEST_BOUNDARY) & (y<NORTH_BOUNDARY)
& (y>SOUTH_BOUNDARY) & P(z,w) then P(z,w+1);

*idle - fix arm;

```

if PLACE(0,0,0,w,arm(wr)) & FIX(z,w,arm(wr)) then FIX(z,w+1,arm(wr));
**update for objects moved - should be able to use FIX/NOFIX;*
if PLACE(z,x,MAXY,w,arm(wz)) then -P(z,w+1);
if PLACE(z,x,MINY,w,arm(wz)) then -P(z,w+1);
if PLACE(z,MAXX,y,w,arm(wz)) then -P(z,w+1);
if PLACE(z,MINX,y,w,arm(wz)) then -P(z,w+1);
if PLACE(z,x,y,w,arm(wz)) then loc(z,x,y,w+1);
**updates for other objects;*
if PUT(z1,arm(1),z2,arm(2),wmin,w) & P(z,w) & -(z=z1) & -(z=z2)
then P(z,w+1);
if PUT(z1,arm(1),z2,arm(2),wmin,w) & P(z,w) & dir(z,f(xy,xyk,xy1),w)
& -(z=z1) & -(z=z2) then dir(z,f(xy,xyk,xy1),w+1);
if PUT(z1,arm(1),z2,arm(2),wmin,w) & loc(z,x1,y1,w) & -(z=z1) & -(z=z2) then
loc(z,x1,y1,w+1);
if PUT(z1,arm(1),z2,arm(2),wmin,w) & -P(z,w) & -(z=z1) & -(z=z2)
then -P(z,w+1);
**_____;*
**updates for free spaces;*
**adjusted for two arms;*
**_____;*
**notfree;*

```

    if PLACE(z,x,y,w,arm(wr)) & NOTFREE(x1,y1,w)
then NOTFREE(x1,y1,w+1);

    if PLACE(z,MINX,y,w,arm(wr)) then NOTFREE(MINX,y,w+1);
    if PLACE(z,MAXX,y,w,arm(wr)) then NOTFREE(MAXX,y,w+1);
    if PLACE(z,x,MINY,w,arm(wr)) then NOTFREE(x,MINY,w+1);
    if PLACE(z,x,MAXY,w,arm(wr)) then NOTFREE(x,MAXY,w+1);

    *free;

    if PUT(z1,arm(1),z2,arm(2),wmin,w) & NOTFREE(x,y,w)
then NOTFREE(x,y,w+1);

    if PLACE(z1,x1,y1,w,arm(1)) & PLACE(z2,x2,y2,w,arm(2))
& FREE(x3,y3,w) & -(x1=x3) & -(x2=x3) then FREE(x3,y3,w+1);

    if PLACE(z1,x1,y1,w,arm(1)) & PLACE(z2,x2,y2,w,arm(2))
& FREE(x3,y3,w) & -(y1=y3) & -(y2=y3) then FREE(x3,y3,w+1);

    if PLACE(z1,x1,y1,w,arm(1)) & PLACE(z2,x2,y2,w,arm(2))
& FREE(x1,y2,w) & -(x1=x2) & -(y1=y2) then FREE(x1,y2,w+1);

    if PLACE(z1,x1,y1,w,arm(1)) & PLACE(z2,x2,y2,w,arm(2))
& FREE(x2,y1,w) & -(x1=x2) & -(y1=y2) then FREE(x2,y1,w+1);

    if loc(z,x,MAXY,w) then -P(z,w);
    if loc(z,x,MINY,w) then -P(z,w);
    if loc(z,MAXX,y,w) then -P(z,w);
    if loc(z,MINX,y,w) then -P(z,w);

```

```

*_____ ;
*next stage;
*_____ ;
if PLACE(z,x,y,w,arm(wz)) then poset(w+1);
if FIX(z,w,arm(wz)) then TMOVE(z,arm(wz),w);
*_____ ;
if NOFIX(w,arm(wz)) then INITIAL(w);
if NOFIX(w,arm(wz)) then GETALLMAX(p(1),0,w);
if FIX(p(z),w,arm(wz)) then NPG(z-1,p(z),w);
*set initial direction weights;
if P(p(z),0) then GTWT(p(z),0);
*_____ ;
*SOS;

```

References

- ALLEN84 J. F. Allen, "Towards a general theory of time and action," *Artificial Intelligence*, 23, 1984, pp.123–154.
- ALLEN90 J. Allen, J. Hendler and A. Tate, ed., *Readings in Planning*, Morgan Kaufman, San Mateo, California, 1990.
- AMBLE87 T. Amble, *Logic Programming and Knowledge Engineering*, Addison-Wesley, 1987.
- CHAP87 D. Chapman, "Planning for Conjunctive Goals," *Artificial Intelligence*, 32, 1987, pp. 333–377.
- CRAIG89 J. Craig, *Introduction to Robotics, Mechanics and Control*, 2nd edition, Addison-Wesley, 1989.
- FU1987 K. S. Fu, R. C. Gonzalez, C. S. G. Lee, *Robotics: Control, Sensing, Vision and Intelligence*, McGraw Hill, 1987.
- GEORG87 M. Georgeff, "Planning", In J. Allen, J. Hendler and A. Tate, eds, *Readings in Planning*, Morgan Kaufman, San Mateo, California, 1990, pp. 5–25.
- GREEN69 C. Green, "Applications of theorem proving to problem solving," *IJCAI Proceedings*, 1969, pp. 219–240.
- HEND73 G. Hendrix, "Modeling simultaneous actions and continuous process," *Artificial Intelligence*, 4, 1973, pp. 145–180.
- KOWAL79a R. Kowalski, "Algorithm = Logic + Control," *Communications of the ACM*, Vol. 22, 1979, pp. 424–435.

- KOWAL79b R. Kowalski, *Logic for Problem Solving*, Artificial Intelligence Series, Computer Science Library, Elsevier Science Publishing Co., New York 1979.
- LUSK84 E. Lusk and R. Overbeek, *The Automated Reasoning System*, Argonne National Laboratory, Argonne, Illinois, 1984.
- MAIM90 Oded Maimon, "The robot task-sequencing planning problem," IEEE Transactions on Robotics and Automation, Vol. 6, No. 6, 1990, pp. 760–765.
- MART82 A. Martelli and U. Montanari, "An Efficient Unification Algorithm," ACM Transactions on Programming Languages and Systems, Vol. 4, pp. 258–282.
- MCCAR69 J. McCarthy and P. Hayes, "Some philosophical problems from the standpoint of artificial intelligence," Machine Intelligence, 4, 1969, pp. 463–502.
- MCDER82 D. McDermott, "A temporal logic for reasoning about processes and plans," Cognitive Science, 6, 1982, pp. 101–155.
- MCDER87 S. Hanks and D. McDermott, "Nonmonotonic logic and temporal projection," Artificial Intelligence, 33, 1987, pp. 379–412.
- MELLO90 L. Mello and A. Sanderson, "And/or graph representation of assembly plans," IEEE Transactions of Robotics and Automation, Vol. 6, No. 2, 1990, pp. 188–199.
- NAGAT88 T. Nagata, K. Honda and Y. Teramoto, "Multirobot plan generation in a continuous domain: planning by use of a plan graph and avoiding collisions among robots," IEEE Journal of Robotics and Automation, Vol. 4, No. 1, 1988, pp. 2–13.

- NILSS71 R.E.Fikes and N. Nilsson, "Strips: A new approach to the application of theorem proving to problem solving," *Artificial Intelligence*, Vol. 5, 1971, pp. 189–208.
- NILSS80 J. Nilsson, *Principles of Artificial Intelligence*, Morgan Kaufman, Los Altos, CA, 1986.
- NILSS88 J. Nilsson and M. Genesereth, *Logic Foundations of Artificial Intelligence*, Morgan Kaufman, Palo Alto, CA, 1988.
- POGUN86 W. Poguntke, "Order-theoretic aspects of scheduling," *Contemporary Mathematics*, 57, 1986, pp. 1–32.
- RIVAL87 I. Rival and J. Urrutia, "Representing orders by translating convex figures in the plane," *Order*, 4, 1987, pp. 319–339.
- SACER75a E. Sacerdoti, "Planning in a hierarchy of abstraction spaces," *Artificial Intelligence*, Vol. 5, 1974, pp. 115–135.
- SACER75b E. Sacerdoti, "The nonlinear nature of plans," *IJCAI Proceedings*, 1975, pp. 206–214.
- SALTER83 R.M. Salter, "Planning in a continuous domain — an introduction," *Robotica*, Vol. 1, 1983, pp. 85–93.
- SARID91 G. Saridis and K. Valavanis, "Probabilistic Modeling of Intelligent Robotic Systems," *IEEE Transactions on Robotics and Automation*, Vol. 7, No. 1, 1991, pp. 164–171.
- SCHILL90 D. Schilling, *Fundamentals of Robots, Analysis and Control*, Prentice-Hall, N.J. 1990.
- THUK91 I. R. Fawzi Al Thuk-Air, Andrzej Pelc and J. Urrutia, "Motion planning, two-directional point representations and ordered sets," *SIAM Journal of Discrete Math*, 1991, pp. 151–163.

- VERE83 S.A. Vere, "Planning in time: windows and durations for activities and goals," *Pattern Analysis and Machine Intelligence*, Vol 5, No. 3, 1983, pp. 246–247.
- WALD77 R. Waldinger, "Achieving several goals simultaneously," *Machine Intelligence*, 8, 1977, pp. 94–136.
- WALD87 Z. Manna and R. Waldinger, "How to clear a block: a theory of plans," *Journal of Automated Reasoning*, 3, 1987, pp. 343–377.
- WINST84 P. Winston, *Artificial Intelligence*, 2nd ed., Addison-Wesley, 1984.
- WOS84 L. Vos, R. Overbeek, E. Lusk, J. Boyle, *Automated Reasoning, Introduction and Applications*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1984.
- WOS86 B. Parrello, W. Kabat and L. Vos, "Job-shop scheduling using automated reasoning: a case study of the car-sequencing problem," *Journal of Automated Reasoning*, 2, 1986, pp. 1–42.