Synthesis of Operating Procedures for Complete Chemical Plants

by

Ramachandran Lakshmanan
B.Tech University of Madras, 1983

Submitted to the Department of Chemical Engineering in Partial Fulfillment of the Requirements of Doctor of Philosophy at the Massachusetts Institute of Technology

Signature of Author

Department of Chemical Engineering
December 4, 1989

Certified by

Professor George Stephanopoulos
Thesis Supervisor

Accepted by

Chairperson, Departmental Graduate Committee

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ARCHIVES
CONTENTS

ABSTRACT 0

INTRODUCTION 1

1 HIERARCHICAL, STRUCTURED MODELLING FOR NONLINEAR PLANNING 2

1.1 The General Planning Problem 3

1.1.1 Review of Domain Independent Planning Techniques 4

1.1.2 Review of Planning Research in Chemical Engineering 6

1.1.3 Outline of this Chapter 9

1.2 Impact of Modelling on Planning Methodologies 9

1.2.1 Basic Modelling Needs of a Planning Program 10

1.2.2 Effect of Operator Model on Complexity of Planning Methodology 11

1.3 Modelling Issues in the Planning of Process Operations 14

1.3.1 Planning Using Operator Models 14

1.3.2 Inadequacies of the Operator-based Approach 15

1.3.3 A Two-part Model for Planning Process Operations 18

1.4 A Hierarchical Structure of Models 19

1.4.1 Hierarchical Description of the Operating State of a System 20

1.4.2 Hierarchical Model of the Structure of a Chemical Process 24

1.5 Construction and Use of Hierarchical Models 27

1.5.1 Construction of an Example Hierarchical Model 27

1.5.2 Examples Utilizing the Hierarchical Structure 30

1.6 Summary 32

2 NONLINEAR PLANNING FOR TEMPORAL CONSTRAINTS 33

2.1 Introduction 34

2.1.1 Comparison of Linear and Nonlinear Planning Methodologies 35
2.1.2 Outline of this Chapter

2.2 Outline of the Planning Methodology

2.2.1 Phase I: Problem Formulation Phase

2.2.2 Phase II: Plan Generation

2.2.3 Case Study: Regeneration of a Catalytic Reactor

2.3 Problem Formulation Phase

2.3.1 Initial State Description

2.3.2 Goal State Description

2.3.3 Specification/Identification of Constraints

2.3.4 Specification/Identification of Planning Islands

2.4 Plan Generation Phase

2.4.1 Identification of Primitive Operations

2.4.2 Construction of a Partial Plan

2.4.3 Synthesis of Complete Plans

2.5 Application of the Methodology to the Case Study

3 PLANNING FOR QUALITATIVE MIXING CONSTRAINTS

3.1 Introduction

3.1.1 Binary, Qualitative Mixing Constraints

3.1.2 Outline of the Chapter

3.2 Previous Work

3.2.1 Synthesis of Failure-safe Operating Procedures

3.2.2 Planning of Purge Operations

3.3 Outline of the Methodology

3.4 Transformation of Mixing Constraints

3.4.1 Construction of Influence Graphs

3.4.2 Identification of Potential Constraint Violations
3.4.3 Generation of Temporal Constraints on Primitive Operations

3.5 Generation of Evacuation/Purge Planning Islands

3.6 Industrial Case Study: Catalyst Regeneration

3.6.1 Description of the Problem

3.6.2 Application of the Proposed Planning Methodology

3.7 Industrial Case Study: E.D.T.A. Plant Recovery Procedures

3.7.1 Description of the Problem

3.7.2 Application of the Methodology

3.8 Summary

4 PLANNING WITH QUANTITATIVE CONSTRAINTS

4.1 Introduction

4.1.1 The Need for Posting Quantitative Constraints

4.1.2 Previous Work

4.1.3 Outline of the Chapter

4.2 Development of the Methodology

4.2.1 Description of EDTA Plant

4.2.2 Truth Criterion for Quantitative Constraints

4.2.3 Abstraction of Operators

4.2.4 Detection of Clobberers

4.2.5 Selection of Plan Modification Operations

4.3 Case Study: Gasoline Polymerization Plant

4.4 Summary

5 DESIGN ISSUES IN THE PLANNING OF PROCESS OPERATIONS

5.1 Introduction

5.2 Causes for Non-existence of a Feasible Plan

5.3 Algorithms for Generating Design Modifications
5.3.1 Flowsheet Modifications for Generating an ASVS 134
5.3.2 Flowsheet Modifications for Purge Sources and Routes 136

5.4 Industrial Case Study 138
5.4.1 Generation of Flowsheet Modifications for Creation of an ASVS 138
5.4.2 Flowsheet Modifications for Purge Source and Purge Route 138

5.5 Summary 139

6 STARTUP OF CONTINUOUS PLANTS 141

6.1 Introduction 142

6.2 Characteristic Feature of the Startup Problem 142
6.2.1 Two Phases of Startup 142
6.2.2 Startup Goals can Usually be Pre-specified 143
6.2.4 The Need for Structural Decomposition 144
6.2.5 Safety 146
6.2.6 Minimizing Startup Time 147

6.3 Previous Work 148
6.3.1 Startup of Conventional Processes 148
6.3.2 Startup of Complex Chemical Processes 148

6.4 Methodology for Planning Startup Operations 149
6.4.1 Industrial Case Study: Gasoline Synthesis Plant 149

6.5 Description of the Problem Formulation Phase 149
6.5.1 Initial State Specification 150
6.5.2 Goal State Specification 150
6.5.3 Knowledge Representation of Partial Plans 152
6.5.4 Specification of Operating Constraints 159

6.6 The Plan Generation Phase 163
6.6.1 Consequences of Structural Decomposition 163
6.6.2 Unit Constraint Transformation 164
6.6.3 Estimating the Size of the Solution Space 170
6.6.4 Synthesis of Complete Plans for Processing Sections 173
6.6.5 Coordinating the Startup of Sections 179

6.7 Industrial Case Study: A Section of a Heavy Water Plant 182
6.7.1 Problem Formulation Phase 182
6.7.2 Decomposition of the Process 183
6.7.3 Unit Constraint Transformation for Augmented Units 183
6.7.4 Heuristics Used in the Pruning 184
6.7.5 Synthesis of Complete Plans 184

6.8 Summary 186

7 BIBLIOGRAPHY 191
ABSTRACT

A computer-based methodology for the automatic synthesis of operating procedures for complete chemical plant is described. In particular, the planning of operating procedures for \textit{a priori}, or off-line, problems is addressed.

Modelling issues associated with the synthesis of process operations on a plant-wide scale are discussed, and a hierarchical, distributed, objected-oriented modelling framework described. It is shown that domain-independent planning methodologies using the functional operators required for the synthesis of operating procedures is computationally intractable. Consequently, domain-specific knowledge is exploited within the modelling structure presented here in order to define a tractable methodology for nonlinear planning of process operations.

An efficient planning methodology for synthesizing operating procedures for continuous chemical processes is developed. The important algorithms used in the various stages of planning are presented, and the properties of these algorithms, in terms of their correctness and completeness, are stated and proved.

A nonlinear planning methodology for handling qualitative mixing constraints is presented. The approach is based on the concept of the planning island and several algorithms derived from graph theory. The methodology is provably correct and all stated constraints are satisfied, thus ensuring that only feasible operating procedures are automatically synthesized.

Design issues related to the automatic modification of flowsheet structure to admit feasible plans are addressed. Algorithms are presented and analysed for the generation of piping and valving networks for the construction of feasible plans.

Finally, a modification of the general methodology is described which is designed to develop startup procedures in an efficient manner. The methodology exploits some of the specific characteristics of the startup problem to allow for more efficient planning.
INTRODUCTION

Planning process operations, especially on a plant-wide scale, is not a topic dealt with in the typical chemical engineering curriculum (O'Shima, 1982). Due to the large number of possible operations in even a medium-sized facility (Rivas and Rudd, 1974), and the subjective nature of the constraints and preferences placed on the performance of an operations plan (Lakshmanan and Stephanopoulos, 1987), as recently as 1987, no satisfactory formal theory for synthesis of such procedures had been developed (Fusillo and Powers, 1987a).

Computer-based process control systems (Pavlik, 1983) "do not know" how to plan, schedule and implement process operations beyond the local confines of single processing units. This is considered to be largely a supervisory task, left to the human operator. As a consequence, current industrial practice relies on (a) "recommended operating procedures" compiled in operating manuals, and (b) the experience accumulated by operating personnel (Stephanopoulos, 1987).

There is a natural tendency, however, towards increased automation in the control room (Garrison, et al., 1986) as well as at the design stage. In the problem of planning and scheduling process operations, this automation can take two forms: (a) a priori synthesis of process operations for subsequent implementation, and (b) automatic, on-line planning and scheduling in real time, while responding to feedback of information about changing conditions and process trends.

The first of these options is appropriate for planning operations such as startup, routine shutdown, or equipment and process change-over, and can be extended to include safety fall-back operations for hazardous situations which are known at the design stage. The second option accounts for a wide variety of operating conditions and involves on-line decision-making. Typical examples are automatic response to process faults and planned evolution to a new operating state for the purpose of optimizing operational performance. In this thesis, I address the problem of synthesizing operating procedures for complete chemical plants at the design stage, namely the a-priori planning problems listed above.
THESIS OBJECTIVES

(I) TO DEVELOP A MODELLING FRAMEWORK FOR PLANNING OPERATING PROCEDURES WITH THE FOLLOWING CHARACTERISTICS:
   (a) The functional behavior of the chemical plant is adequately represented
   (b) The interactions between units in a chemical plant are clearly identified
   (c) The models for the process should not be situation specific and the user of the software program should not have to do extensive modifications from application to application
   (d) Input of and access to values of state variables should be intelligently carried out

(II) TO DEVELOP EFFICIENT ALGORITHMS TO ADDRESS THE FOLLOWING NEEDS:
   (a) Efficient manipulation of the temporal constraints placed on the plant operation
   (b) Efficient handling of any mixing constraints that may be stated
   (c) Efficient handling of general quantitative inequality constraints

(III) TO DEVELOP ALGORITHMS FOR THE MODIFICATION OF FLOWSHEET STRUCTURE WHEN NO FEASIBLE PLAN IS FOUND FOR A GIVEN PROBLEM

(IV) TO INCORPORATE THE FEATURES MENTIONED ABOVE IN THE DEVELOPMENT OF AN INTELLIGENT GRAPHICS-BASED SOFTWARE PACKAGE
CHAPTER 1

HIERARCHICAL, STRUCTURED MODELLING FOR NONLINEAR PLANNING
1.1 THE GENERAL PLANNING PROBLEM

The planning of process operations involves specifying an ordered sequence of operations, or a partially ordered set of operations which, when carried out (applied), will perturb the state of the chemical plant from some (given) initial state, and cause it to eventually attain some pre-specified final, or "goal", state. Typically, this transformation cannot be achieved in a single step, and the plant must be taken through a series of intermediate states (Figure 1.1). While attempting to generate a plan, one can generate an intermediate state from which no operator is available to take the plan further (the so-called dead-end). Or one may find an intermediate state from which the goal state can be achieved, but no operator exists to take the plant from the initial state to this "unreachable state".

The operation of the plant at the intermediate states must be consistent with physical constraints such as conservation of mass, energy and momentum, and equilibrium and rate phenomena. An operations plan which satisfies this condition is termed "physically feasible". In addition to the physical constraints, intermediate states are required to satisfy certain "operating" constraints imposed on the plan (practical feasibility), such as upper bounds on reactor temperatures, prohibition of explosive mixtures, etc. Finally, for underconstrained planning problems, more than one feasible plan may exist. In such cases more stringent performance criteria are required to select a small set of "efficient" plans from among the many alternatives.

Research in the area of planning falls into two philosophies. One school of work focuses on those theoretical aspects of the problem which do not depend on the particular area of application, the so-called "domain-independent" theory of planning. Other research has concentrated on exploiting characteristics specific to a given problem domain in order to construct special-purpose planning programs. Clearly, if domain-independent planning theory could handle all of the complexities of practical planning problems, this approach would be preferable to one which was restricted to a particular domain of application. However, there is theoretical evidence to suggest
Figure 1.1: Schematic of a typical planning problem.
that building a provably correct, complete, domain-independent planner that is versatile enough to solve real-world planning problems, is impossible (Chapman, 1985).

1.1.1 Review of Domain Independent Planning Techniques

The bulk of domain-independent planning research has been carried out in the field of artificial intelligence. Major contributions by researchers in this area are summarized below. The work generally can be split into two categories, the so-called "linear" and "non-linear" planning approaches.

**Linear vs. Non-Linear Planning.** Early planning programs worked linearly. This means that they started at one end of a plan - either at the goal or at the initial state, and constructed a plan step by step, moving "closer" all the time to the finish (if planning started at the initial state) or to the start (if the planner moved backwards from the goal state). At every point in the planning process, all variables were assigned values corresponding to objects in the world, and the ordering of the operations performed up to the current state, was completely specified. Non-linear planners, on the other hand, work by constructing and refining "partial plans". A partial plan is a set of plan steps that leaves information unspecified. Perhaps the exact order in which two steps occur is not specified, or the planner may not specify a single operator for a task but rather choose a candidate set of operators (which is a subset of the large number of possible operators). A single partial plan actually describes a large number of "completions", or total plans. Thus, by conducting planning in terms of partial plans (non-linear planning), a single planning decision in actuality represents a decision about a large number of plans, resulting in a possibly exponential savings in efficiency (Chapman, 1985). The distinction between linear and nonlinear planning is clarified in the next chapter using a simple chemical engineering example.
**GPS: The General Problem Solver.** The pioneering work in domain-independent planning, due to Newell, Shaw and Simon (Newell, et al., 1960) was the computer program dubbed "The General Problem Solver" (GPS). GPS introduced the "Means-Ends Analysis" paradigm which has been extensively used in subsequent planners. This approach involves identifying "differences" between the goal and initial states and then selecting operators which "reduce" this difference. While a simple means-ends analysis was adequate for solving several of the problems on which it was first used - developing plans for a robot arm to stack toy building-blocks, for example - it suffers from several limitations. For instance, when the operational plan involves taking the plant through intermediate states whose "distance" from the goal state is not easily defined, means-ends analysis breaks down.

**STRIPS: An Attempt at Handling the "Frame Problem".** Application of an operator perturbs the state of a system, causing it to attain some new state. The problem of how to efficiently represent the effects of applying an operator is known as the "frame problem". Attempting to provide a practical solution to this problem, Fikes and Nilsson (Fikes and Nilsson, 1971) proposed the "STRIPS Operator Model" (Figure 1.2 (a)). This model has been used by all subsequent domain-independent planners, and several domain-dependent planners as well, and employs operators with pre-specified lists of pre- and post-conditions. While this scheme has been successful in dealing with a small range of problems, it is inadequate for modelling distributed, network-like structures, such as chemical plants.

**NOAH: The Advent of Non-Linear Planning.** In 1975, Sacerdoti (Sacerdoti, 1975) introduced the notion of a "non-linear planner". Previous planners were all "linear", and ran the risk of being exponentially less efficient than an equivalent non-linear algorithm.
Figure 1.2(a) Traditional STRIPS Operator
Planning with Constraints. In many domains, the statement of a planning problem includes the stipulation of certain constraints under which the plan is to be implemented. These constraints are not specific to a particular operator (in which case they could be stated in the form of pre-conditions) but, rather, are statements about situations which are either unacceptable, or are detrimental to the performance of the operations plan. This was first recognized by Stefik (1981) and implemented in his program MOLGEN which planned experiments in molecular genetics. By identifying and dynamically propagating constraints through a distributed system, the search through the space of possible solutions to a planning problem can be dramatically reduced.

Cleaning Up Three Decades of "Scruffy" Research: TWEAK. In 1985, much of the previously ad-hoc nature of planning was "cleaned up" by Chapman with his planning program TWEAK (Chapman, 1985). Chapman retained the standard STRIPS operator representation and developed a non-linear planning algorithm which was proven both correct and complete. If TWEAK produces a plan for a given problem, then that plan is guaranteed to solve the problem (correctness), and if the program fails to produce a plan (possibly by never terminating), then no plan which solves the problem exists.

1.2 Review of Planning Research in Chemical Engineering

Domain-dependent planning research has been carried out in a variety of domains. Applications range from planning operations for space satellites (Vere, 1983) to planning daily errands (Hayes-Roth, et al., 1979). In this thesis, I limit my review to those special purpose planners which have been built by researchers in the field of chemical engineering.

Rivas and Rudd (1974): Synthesis of Failure-Safe Operating Procedures. In this pioneering work, the plan-synthesis algorithm was built on top of an earlier constraint analysis methodology
presented as a "Computer-Aided Safety Interlock System" (Rivas, et al., 1974). The synthesis procedure was aimed at generating operating sequences that achieved certain operating goals while ensuring that given safety constraints were never violated. The constraints were restricted to statements about species which should never be present simultaneously in any location of the plant. Synthesis was carried out by formulating a hierarchy of goals (input by the user), which had to be specified down to the level of goals stated in terms of the presence or absence of individual chemical species, and whether these species were flowing, blocked or trapped within the segments of the plant network. Although this work included modelling techniques which were considered innovative at the time, their approach had several drawbacks.

(i) Quantification of driving forces which propel the species through the network was not possible, leading to ambiguities in cases where numerical (or even qualitative) values of pressures were needed to determine the direction of flow of a species.

(ii) A large portion of the tree of subgoals has to be specified by the user, including the order in which the goals at any particular level are to be achieved.

(iii) The sequential logic algorithm, which is equivalent to a restricted form of constraint propagation, was unable to provide any kind of explanation of its results. Keeping track of dependencies for the express purpose of generating an explainable plan, an attractive feature of standard constraint-propagation techniques, was impossible.

**Kinoshita, Umeda and O'Shima (1982).** The methodology consisted of dividing the plant (not automated) into sections around "key" equipment and their peripherals. Individual operations for these sections were then identified by the plant operator, and the following information was required as input: (i) state transition relationship between states of a unit (a tree of all possible sequences of operator application), (ii) minimum required time for each operation, (iii) maximum allowable time for each operation, (iv) information about the cost of each operation and (v) information about the constraints on the states of adjacent units. Although this approach yielded
efficient schedules, the overhead involved in setting up the input to the planning methodology was too great for the approach to be of practical utility.

**Optimal Startup Using Weighted Arcs.** Ivanov and co-workers (Ivanov, et al., 1980) developed algorithms for generating optimal startup sequences starting from a "transition network" representation of the chemical plant (input by the user). The nodes on the transition graph represented the possible combinations of system state parameters, and the arcs represented control operations which would carry the system between two states. Associated with each arc was a weighting factor which was based on some pre-determined optimality criterion. For practical industrial problems, however, a priori assignment of these weighting factors is difficult, and often subjective.

**Fusillo and Powers (1987a).** This research effort represents the first attempt made to define a formal theory for planning process operations. Their methodology employs the means-ends analysis which was modified to allow the computer to generate purge goals automatically when needed. They allowed for the statement of global and local design constraints which were checked at each new state to ensure feasibility of the plans. One of the most important contributions of this work was the concept of a "Stationary State". The authors define a stationary state as a condition at which the operating goals are partly met and the system does not change over time. Such states serve as a sound basis for guiding automatic decomposition of the flowsheet, and provide convenient stopping-off points for recovery from errors. In addition to defining the concept of the stationary state, this work represents the first attempt at using qualitative state descriptions more expressive than the primitive Boolean formulations favored by the pioneers in the field. The methodology does suffer some drawbacks, however:

(i) The oversimplified modelling approach: the state of a group of units was described in terms of variables lumped over the whole section of the plant.
(ii) The search algorithm used, although guaranteed to find a feasible solution, if one exists, places no emphasis on "efficiency" of the plans generated.

(iii) Plans are considered to be completely sequential. In many problems, significant savings in down-time can be achieved by carrying out operations in parallel.

(iv) It does not have the ability to generate plans in a non-linear manner.

1.1.3 Outline of this Chapter

This first chapter of this thesis is arranged as follows:

(a) Section 1.2 outlines how the modelling of primitive operations affects the properties of linear and non-linear domain-independent planning methodologies.

(b) Section 3 describes how the specific needs for planning chemical process operations dictate a departure from the well established models and details the formulation of functional operators. In addition, it describes a two-part model for representing the chemical plant for the purpose of planning process operations.

(c) Section 1.4 establishes the hierarchical structure used to model the operational state of a process at various levels of detail.

(d) Finally, Section 1.5 provides a series of examples to illustrate the modelling philosophy.

1.2 IMPACT OF MODELLING ON PLANNING METHODOLOGIES

It should be clear from the discussion on previous work on planning, both domain-specific and domain-independent, that the primary issue of concern is modelling. Modelling of the operational state of the system, modelling of the planning operators and their effects and, finally, modelling of the planning methodology itself. The manner in which this modelling is carried out has a significant impact on the practical applicability, tractability, versatility and completeness of the planning program.
1.2.1 Basic Modelling Needs of a Planning Program

A computer program which is designed to automatically generate operating procedures must have the capacity to do the following:

**Model the Operational State of the System.** A computer-based planning program must reason about the initial, final and intermediate states of a plan, hence a first basic modelling requirement is the representation of the operating state of the system for which planning is being carried out.

**Model the Primitive Operators.** An operations plan is implemented by carrying out a series of actions on special devices in the system (such as the valves, controllers and switches in a chemical plant) known as primitive operation devices. A planning program must contain models for these devices.

**Model the Effects of Operator Application.** Actions which are carried out by the primitive operation devices result in a perturbation of the state of a system, necessitating special forms of models to represent these effects.

**Model the Planning Methodology.** Finally, all of the information contained in the facilities above must be manipulated by some inferencing machinery which is responsible for constructing the sequences of operating procedures which define a plan.

The design of the modelling capabilities described above must be carried out with several criteria in mind: (i) ease of input/output of information by/to the user, (ii) ease of model setup for new problems, (iii) correctness/completeness of models and (iv) tractability/efficiency of planning methodology. The last two issues are especially important and have been the focus of significant research effort. The evolution of the original "linear" planning methodologies into the more recent "non-linear" techniques was prompted by the need to overcome the combinatorial explosion of possible plans, which was a major deficiency of the linear approach. In the next section I illustrate how the nature of the operator model affects the complexity of the planning methodology.
1.2.2 Effect of Operator Model on Complexity of Planning Methodology

The most commonly used model for representing the primitive operators and the effects of their application, is the STRIPS operator model. In this scheme of knowledge representation an operator, or step, is any action which can be performed on the system. A step has a finite number of pre-conditions, which are statements that must be true of the world before the action can be performed. A step also has a finite number of post-conditions, which are statements that are guaranteed to be true of the world after the action corresponding to that step has been carried out. Certain restrictions are placed on the form of the pre-conditions and post-conditions. The allowable form of these statements is as follows: (i) pre- and post-conditions must be expressed as propositions that have a content, which is a tuple of elements, and can be negated; (ii) the elements can be variables or constants, and there can be infinitely many of these; (iii) functions, propositional operators and quantifiers are not allowed. Linear planning with the STRIPS operator is relatively well understood. The problem of determining the truth of any given statement is facilitated by the following theorem:

**Theorem 1**: (Truth Criterion for Complete Plans) In a complete plan, a proposition \( p \) is necessarily true in a situation \( s \) iff there exists a situation \( t \) previous or equal to \( s \) in which \( p \) is asserted and there is no step between \( t \) and \( s \) which denies \( p \).

At each stage in the linear planning process, all information (sequence of operations carried out so far, variables which must be bound, etc.) is known so verification of the necessary truth of any statement \( p \) is straightforward.

Nonlinear planners, on the other hand, deal with plans which are only partially specified, and a mechanism is needed to guarantee that, when the partial plan is completed, a given proposition is still true. The following theorem due to Chapman (Chapman, 1985) gives the necessary and sufficient conditions for guaranteeing the truth of any given statement in a partial plan.
Theorem 2: (Modal Truth Criterion) A proposition $p$ is necessarily true in a situation $s$ iff two conditions hold: there is a situation $t$ equal or necessarily previous to $s$ in which $p$ is necessarily asserted; and for every step $C$ possibly before $s$ and every proposition $q$ possibly co-designating with $q$ which $C$ denies, there is a step $W$ necessarily between $C$ and $s$ which asserts $r$, a proposition such that $r$ and $p$ co-designate whenever $p$ and $q$ co-designate.

A polynomial-time algorithm based on the modal truth criterion exists, and it can be shown that all previous domain independent planners used one or more of the five techniques which this algorithm, TWEAK (Chapman 1985), employs to make a plan achieve a set of goals. These five actions are necessary and sufficient for constructing a correct and complete planner, subject to the assumption that all modelling is done in the form of the STRIPS operator model described earlier.

Advantages of the STRIPS operator model. The STRIPS operator model has been popular for several reasons. Firstly, operators are intuitively appealing. A plan consists of a discrete set of actions which effect changes in the world and can be conveniently expressed as an ordered set of operators. Thus, if all of the domain knowledge is expressed in the form of operators, the planning problem is reduced to that of determining a consistent ordering of the operators. Secondly, planning using the STRIPS operator, although undecidable has the special features described above, namely the existence of the modal truth criterion, and the existence of a polynomial time algorithm for its verification. Finally, for many domains (such as the robotics blocks world) where planning research has been carried out, objects do not interact with each other to any significant extent. In such cases, the behavior of the entire system is easily represented within the scope of the STRIPS operator model itself.

Disadvantages of the STRIPS model. In attempting to use the STRIPS operator model for the synthesis of operating procedures for chemical plants, I found that it suffers from some severe
limitations. Many physical processes exhibit behavior that is functionally dependent on the exact situation in which an action is carried out (see Section 1.3.2). As mentioned in the description of the STRIPS operator, functional dependency between the input and output states of an operator is not allowed.

There are two ways in which this situation can be handled: (i) represent the functional behavior of the system using a (possibly infinite) set of simple STRIPS operators, or (ii) approximate the behavior of the system with a simple, but inexact, STRIPS operator. Neither of these alternatives is practically acceptable, however, since the first option is detrimental to the ease of setup of new problems (the user is required to tailor-make a large number of operators for each new application) while the second impacts the correctness/completeness of the models.

In addition to the difficulty in dealing with functional operators, the STRIPS operator model is inappropriate for modelling systems, such as chemical plants, where there is a large amount of interaction between the sub-systems. These interactions are not likely to be known at the time when the operators are being constructed, since they stem from specific system configurations (such as the structure of a particular process flowsheet) which vary from problem to problem. Hence they cannot be included in the operator models.

Planning with Conditional Operators. Realizing the awkwardness which resulted from the restricted form of the propositions allowed in TWEAK, Chapman investigated the possibility of using conditional steps in a planner. A conditional step, or operator, is always applicable, but has two sets of post-conditions, the if-true post-conditions and the if-false post-conditions. The if-true post-conditions are true of the world after the operator is applied provided that all of the pre-conditions are satisfied. If any of the preconditions are not satisfied, the if-false post-conditions will be true of the world. A schematic of this type of operator is shown in Figure 1.2 (b). This representation scheme, however, yields the following theorem (Chapman, 1985):
Figure 1.2 (b) A Simple Conditional Operator

IF-TRUE POST-CONDITIONS

OPERATOR

PRE-CONDITIONS

POST-CONDITIONS

IF-FALSE POST-CONDITIONS

PRE-CONDITIONS

POST-CONDITIONS
Theorem 3: (First Intractability Theorem) The problem of determining whether a proposition is necessarily true in a nonlinear plan whose action representation is sufficiently strong to represent conditional actions is NP-hard.

Thus, no polynomial time algorithm is known for checking whether a proposition is necessarily true in a given situation. From this analysis it is clear that the tractability of a planning program is heavily dependent on the form of the models used to describe a system.

1.3 MODELLING ISSUES IN THE PLANNING OF PROCESS OPERATIONS

In the last section I discussed modelling issues that arise in the design of an automatic planning program. In this section I present some of these issues within the specific context of planning process operations.

1.3.1 Planning Using Operator Models

For certain types of process operations, it is possible to express the behavior of a chemical processing system in terms of operator models. A case in point is the routine startup of continuous chemical plants. For such problems, an operator-based planning methodology has been developed (See Chapter 4) which has been demonstrated to be suitable for planning operating procedures on a plant-wide scale. A STRIPS-like operator model is employed, and a chemical processing unit is represented by a data-structure known as a Unit Transition Network (UTN). A UTN describing a processing unit is a network that contains all feasible paths by which the unit could be brought to the final operating state, if it was operating in isolation (unconnected to the rest of the plant). Template models are supplied in compiled form, so that the user can use all of the graphic capabilities of DESIGN-KIT (Stephanopoulos, et al., 1987) to easily put together a flowsheet in a menu-driven manner. This methodology is described in detail in Chapter 4 of this thesis.
The operator-based planning methodology works well for planning routine startup of medium-sized chemical plants, and can be extended (by creating the appropriate UTNs for standard chemical equipment) to other process operations such as maintenance shut-downs and safety-fallback procedures from well-known process faults.

1.3.2 Inadequacies of the Operator-Based Approach

It is important to realize that the success of the operator-based approach depends on the ability of the user to define, a priori, operator models for individual units. This limitation renders the program unsuitable for planning operating procedures for such tasks as equipment/product change-over, optimizing control and certain types of safety fall-back. In addition there are specific modelling needs of chemical plants that are inadequately addressed by simple STRIPS-like operators.

**Functional Operators.** Consider the following process operation: "the inlet valve to a storage tank is opened". An operator must specify what changes are induced by this action. However, the behavior of the system depends on the exact conditions that prevail just before the action was performed. Whether or not fluid flows into the tank depends on the state of the system before the valve was opened. Was there liquid under pressure on the upstream side of the valve? Or was the pipe empty? The post-conditions of the operator are functionally dependent on the pre-conditions. However, according to the restrictions on the form of the propositions allowed in the STRIPS operator model, operators that include this functional dependence are not permitted.

Since the planning of process operations requires the use of operators whose post-conditions depend on the state of the system before the operator was applied, I investigate, here, the possibility of using such an operator representation. An example of an operator whose post-conditions depend on the input-situation, is shown in Figure 1.3. Instead of a conjunction of pre-conditions, this type of operator has a *set* of conjunctions of pre-conditions. Each element in
Figure 1.3: Schematic of the Functional Operator

OPERATOR-9

PRE-CONDITION-9.1.a
  .
  .
POST-CONDITION-9.1.a

PRE-CONDITION-9.1.b
  .
  .
POST-CONDITION-9.1.b

PRE-CONDITION-9.1.c
  .
  .
POST-CONDITION-9.1.c

PRE-CONDITION-9.2.a
  .
  .
POST-CONDITION-9.2.a

PRE-CONDITION-9.2.b
  .
  .
POST-CONDITION-9.2.b

PRE-CONDITION-9.2.c
  .
  .
POST-CONDITION-9.2.c

PRE-CONDITION-9.3.a
  .
  .
POST-CONDITION-9.3.a

PRE-CONDITION-9.3.b
  .
  .
POST-CONDITION-9.3.b

PRE-CONDITION-9.3.c
  .
  .
POST-CONDITION-9.3.c

PRE-CONDITION-9.4.a
  .
  .
POST-CONDITION-9.4.a

PRE-CONDITION-9.4.b
  .
  .
POST-CONDITION-9.4.b

PRE-CONDITION-9.4.c
  .
  .
POST-CONDITION-9.4.c

PRE-CONDITION-9.5.a
  .
  .
POST-CONDITION-9.5.a

PRE-CONDITION-9.5.b
  .
  .
POST-CONDITION-9.5.b

PRE-CONDITION-9.5.c
  .
  .
POST-CONDITION-9.5.c

PRE-CONDITION-9.6.a
  .
  .
POST-CONDITION-9.6.a

PRE-CONDITION-9.6.b
  .
  .
POST-CONDITION-9.6.b

PRE-CONDITION-9.6.c
  .
  .
POST-CONDITION-9.6.c

PRE-CONDITION-9.7.a
  .
  .
POST-CONDITION-9.7.a

PRE-CONDITION-9.7.b
  .
  .
POST-CONDITION-9.7.b

PRE-CONDITION-9.7.c
  .
  .
POST-CONDITION-9.7.c

PRE-CONDITION-9.8.a
  .
  .
POST-CONDITION-9.8.a

PRE-CONDITION-9.8.b
  .
  .
POST-CONDITION-9.8.b

PRE-CONDITION-9.8.c
  .
  .
POST-CONDITION-9.8.c

PRE-CONDITION-9.9.a
  .
  .
POST-CONDITION-9.9.a

PRE-CONDITION-9.9.b
  .
  .
POST-CONDITION-9.9.b

PRE-CONDITION-9.9.c
  .
  .
POST-CONDITION-9.9.c

PRE-CONDITION-9.9.d
  .
  .
POST-CONDITION-9.9.d
the set describes some possible situation which might exist before the operator is applied. For each element of the set of pre-conditions, there is a corresponding element in a set of post-conditions which specifies what facts are to be asserted in the world if the action is carried out with the corresponding pre-conditions being true before operator application. I shall refer to operators of this type as "functional operators".

**Theorem 4:** (Second Intractability Theorem) The problem of determining whether a proposition is necessarily true in a nonlinear plan whose action representation employs functional operators is NP-hard.

**Proof:** Theorem 4 is proved by demonstrating that the functional operators I have just described can be used to simulate conditional actions (see Theorem 3). For each conditional operator in a given planning problem, construct a functional operator as follows (see Figure 1.4):

1. Form a list of pre-conditions that is identical to the list of pre-conditions of the conditional operator. The corresponding post-conditions for the functional operator are the if-true post-conditions of the conditional operator.

2. For each of the propositions \( p_i \) in the conjunction of pre-conditions in the conditional operator, do the following:

   2.1 Construct a list of pre-conditions consisting of just one proposition, the negation of \( p_i \).

   2.2 Construct a list of post-conditions identical to the if-false post-conditions of the conditional operator. Assign this list to be the post-conditional set corresponding to each of the pre-conditions constructed in (2.1). The resulting set of operators will be able to express all of the actions described by the original set of conditional operators.

Thus, if we could construct a polynomial time algorithm for choosing and ordering a functional operator to achieve a given goal, then this algorithm would also solve the problem of planning with conditional operators. Theorem 3 asserts that this problem belongs to a class of problems which are
Figure 1.4  Simulation of conditional actions using the functional operator
commonly considered to be intractable, thus domain-independent planning with functional operators is also computationally intractable.

Distributed Nature of Chemical Processes. A chemical processing facility consists of a number of pieces of equipment which interact with one another. The interaction may be in the form of material being transported from one unit to another, or through energy transfer, due to heat integration between process streams. Due to the extensive interconnection between units, the effect of applying an operator is not confined to the immediate vicinity of the processing unit in which the operation was carried out. On the contrary, changes in the operating conditions of specific processing units lead to significant operational changes throughout the network of processing units.

Consider the effects of opening the inlet valve to a storage tank, discussed above. It was stated that whether or not fluid flowed into the tank was subject to the conditions which prevailed immediately upstream of the valve just before it was opened. Assuming that fluid flows into the tank, and the outlet valve of the tank is open, the fluid will flow out of the tank and proceed to equipment downstream. Subsequently, the fluid will flow into all equipment downstream unless it encounters a pressure head greater than the level of fluid in the tank, or some valve between the tank and a given processing unit is closed. On the other hand, if the outlet valve from the storage tank is closed, the effect of opening the inlet valve to the tank will be contained within the tank itself. We see that, in addition to the complexity introduced by the functional dependence of process behavior on initial conditions, the constraints which describe the behavior of the system stem from the network-like structure of the chemical process. Since the actual structure of the plant is not known at the time we formulate the operator model, we must rely on some other means for modelling these inter-unit constraints. The operator-based approach described above used very specific, simplified models which could be connected to form a network representing a chemical plant.
1.3.3 A Two-Part Model for Planning Process Operations

I now demonstrate how inter-unit constraints, and the functional nature of the primitive operators can automatically be derived from first principles. A chemical plant is modelled as a distributed, hierarchically decomposed network of devices, using a knowledge representation scheme that comprises two parts: (i) models of the primitive operators and, (ii) models for the state and behavior of the chemical plant. The final requirement of a planning program (see Section 1.2) namely, a model for the methodology, is the subject of the next two chapters of this thesis.

Modelling of Primitive Operations. In a chemical plant, devices such as pumps, valves and controllers, which are available for carrying out primitive actions are readily identified. All transformations in the operating state of the chemical plant must be effected using these devices, so it is important that the primitive operations be explicitly modelled. Following the object-oriented programming philosophy, each primitive action is represented as an individual object with slots for the pre- and post-conditions. Each operator object is "attached" to the model of the operator device which is responsible for executing that particular primitive operation. These primitive operator devices have various regions of operation, and a specific behavior model (a set of equations) is associated with each region. For example, a flow valve can be operating in the "open" or "closed" region. The pre-conditions are stated in terms of these operating regions. As illustrated by the data-structure representing the "OPEN VALVE" action (Figure 1.5), before we can apply this operator, the associated valve must be in the "closed" region. In addition, the outcome of applying the operator (post-condition) is that the valve is "open".

The second part of the modelling structure is responsible for describing the changes in the operating state of the chemical plant induced by manipulation of the operator devices. It is this section of the modelling structure that accounts for the interactions among units in the plant network. This equation-based model is a complex, hierarchically structured collection of objects, and is described in detail in the next section.
<table>
<thead>
<tr>
<th>PRIMITIVE OPERATOR: OPEN-VALVE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTACHED TO: FLOW-VALVE</td>
</tr>
<tr>
<td>PRE-CONDITIONS: CLOSED (REGION (FLOW-VALVE))</td>
</tr>
<tr>
<td>OUTCOME: OPEN (REGION (FLOW-VALVE))</td>
</tr>
<tr>
<td>PRIMARY FUNCTION: FLOW REGULATION</td>
</tr>
</tbody>
</table>

Figure 1.5  Model of a Primitive Operator
1.4 A HIERARCHICAL STRUCTURE OF MODELS

In the preceding sections, previous attempts at modelling chemical processes for the purpose of planning process operations were reviewed. Having highlighted the strengths and weaknesses of these approaches, I now present a modelling structure based on fundamental chemical engineering principles. The description of this modelling structure is complemented by examples of the data-structures used to represent the models on the computer. The justification for the structure of the hierarchy stems from the various modelling requirements apparent at each level of the hierarchy. The attributes needed to describe the operational state of a segment of a chemical plant (e.g., a feed pre-treatment section, or a product recovery section) are quite different from those needed to describe the operational state of a single processing unit within a segment of the plant. These attributes, in turn, are different from those modelling a "variable". The distinct levels in the structure are linked to each other and methods are used to allow transfer of information both "up" and "down" the hierarchy: a user-induced command, such as an operating constraint between two sections of a chemical plant, can be refined to a set of detailed actions at the lowest level, while data on the state of variables and parameters at the lowest level are aggregated to describe the operational state at successively higher levels.

The structured modelling of chemical plants and their operating states consists of two parts: (i) hierarchical description of the operational state, and (ii) hierarchical, equation-based description of the structure of the process flowsheet. The first of the two hierarchies referred to above, illustrated in Figure 1.6 (a), is used to describe the operational state of any "system". The second structure of models provides the means by which the operational state of individual systems may be combined to form "constructive abstractions" of their subsystems (Figure 7 (b)).
MODEL FOR FLOW-VALVE: (SET OF EQUATIONS)

EQUATIONS

\[ H_1 - H_2 = 0 \]

\[ F - \text{Rho} \times C_v \times \sqrt{\Delta P} = 0 \]

\[ N(i,1) - N(i,2) = 0 \]

TERMS

F \times C_p \times T_1

F \times C_p \times T_2

V \times \text{Rho}

Pr_1 - Pr_2

F \times X(i,1)

F \times X(i,2)

VARIABLES

T_1 \quad C_p \quad V \quad \text{Rho} \quad T_2 \quad C_v \quad Pr_1 \quad Pr_2 \quad X(i,1) \quad X(i,2)

figure 1.6(a) Hierarchical description of operating states
Figure 1.6(b) Hierarchical description of process flowsheets
1.4.1 Hierarchical Description of the Operating State of a System

Under the modelling approach proposed in this chapter, the operational state of each system in a chemical plant is described in a hierarchical manner. This convention holds whether the system under consideration is a thermodynamic "simple" system (Modell and Reid, 1983), a single processing unit, an augmented unit or a whole section of a chemical plant.

First Level: Operational State of Variables and Parameters. At the most basic level the operational state of a system is captured by a set of variables and parameters. Typical examples are: "temperature", "pressure", "concentration" and "total-mass", physical properties and controller parameters. Unlike conventional modelling approaches, variables and parameters are modelled here as distinct objects with specific structure. This is done for several reasons:

(a) Variables and parameters represent the state of sensors and parametric data bases; basic sources of knowledge about the operational state of a plant. It is important to model these basic sources of information as explicitly as possible.

(b) Each variable or parameter represents a chunk of structured knowledge, e.g., the variable "temperature" could have the following attributes: "current-value", "current-trend", "range-of-values", "record-over-x-minutes", "average-over-x-minutes", etc. and it may be necessary, during planning, to reason explicitly about these attributes. Modelling variables and parameters as independent objects is an excellent means for achieving this goal.

It should be noted that, for simple systems, the state-variables (temperature, pressure, etc.) all have physical meaning. For composite systems, such as a section of a chemical plant, the internal state is usually some "aggregate" value reflecting the state of the sub-systems.

Second Level: The State of "Streams" and Other Compound Terms. Groups of variables and parameters form compound terms which are used to describe physically meaningful quantities such as: "flow-of-component-x", "enthalpy-flow", "work-input", "heat-flow", "diffusive-mass-flow", 

20
"reaction-rate", etc. Compound quantities are aggregates of the low-level variables and parameters, and are used to model the operational state of "streams". These "streams" refer to the flows into and out of a system, and can take several forms: (a) convective flow of material, (b) diffusive flow of mass, (c) flow of energy (heat, work), (d) flow of momentum and (e) generation or consumption of chemical species through reactions. "Streams" appear in the statement of the physical laws and engineering considerations that govern the operation of the system. Figure 1.6 (a) shows the structure of the object models used to represent the state of compound quantities. The construction of compound quantities from the basic variables is guided by definitions that stem from physical principles (e.g. the definition of enthalpy and heat flow) or engineering knowledge (e.g., statistical correlation of experimental data describing reaction rates, correlations describing work-input or pressure-drop).

Modelling "streams" and other compound quantities as individual objects is especially useful for input/output of information, where describing the system in a manner that is intuitively appealing to the operator is of importance. The equivalence relation below is an example of one such commonly used "stream".

\[
\text{ENTHALPY-IN} \Leftrightarrow \text{FLOW-IN} \times \text{SPECIFIC-HEAT} \times (\text{TEMP} - \text{REF-TEMP})
\]

This relationship enables the user to specify that the enthalpy flow (a physically meaningful quantity) into a particular device is (say) 10,000 kcal/hr, and have the computer deduce a relationship between the remaining variables in the equation. If, for example, the flow-rate and the specific heat were known, the temperature could be directly deduced. Alternatively, if the values of all of the terms on the right-hand side of the equivalence relationship are known, the user (or the planning methodology) can query the knowledge base for the value of the enthalpy flow into the system, and have the machine provide the appropriate response. Another commonly used group of variables is the pressure drop term often found in momentum equations:

\[
\text{DELTA-PRESSURE} \Leftrightarrow \text{UPSTREAM-PRESSURE} - \text{DOWNSTREAM-PRESSURE}
\]

While actual pressures upstream and downstream of a particular unit may not be available, it could
be the case that the pressure head across the unit is known (using a manometer, for example). By grouping the pressure variables into the more meaningful pressure-difference term, the computer program is able to make use of partial information when it is available.

Third Level: Operational State of Constraining Relationships. The operational state of "streams" is constrained by laws of physics and relationships which arise from engineering considerations. Typical examples of the former are (a) the conservation principle (for mass, energy and momentum) and (b) the second law of thermodynamics which imposes a directionality on such compound quantities as "heat-flow" and "diffusive-mass-flow". Typical examples of constraints imposed by practical or engineering considerations are: (a) limits on pressure drops, (b) desired directions of mass flows, (c) bounds on heat flows or work inputs dictated by the size of processing units, (d) experimental correlations between input and output "streams" and (e) controller designs. Explicit modelling of constraints necessitates objects with distinct structures. Figure 1.6 (a) shows the structure of the constraint data-models. The type of constraints exploited at this level can be boolean, qualitative, order-of-magnitude or quantitative, depending on the type of available knowledge. The models of Figure 1.6 (a) are generic in the sense that they can be used to represent any type of constraining relationships, thus allowing the execution of quantitative algorithms or the implementation of qualitative and semi-quantitative reasoning procedures (boolean, qualitative simulation, order-of-magnitude reasoning).

Fourth Level: Sets of Constraints - Operational State of a Process System. Processing units operate over extended operating regions which are often conceptualized as being distinct and discontinuous. The behavior in each of the distinct regions is described by a distinct set of constraining relations. For instance, we often distinguish the flow of fluids into laminar and turbulent regimes, and use different relations to represent system behavior in each region. Similarly, the behavior of a CLOSED control valve is represented by different relations than that of
an OPEN valve. Thus, in modelling the operational state of a system, separate physical relations are needed to represent the behavior of the system in each of the operating regions and the combined state of these constraining relations constitutes the state of the system.

**State Descriptions are Both Qualitative and Quantitative.** There is another important distinction between traditional chemical engineering modelling techniques and the modelling approach proposed here. Standard chemical engineering models are purely quantitative in nature. This means that the operational state at any level must invariably be associated with numerical values before the state of a system is considered to be specified. Purely quantitative modelling approaches suffer from the following disadvantages:

(i) Due to the overwhelming detail of the description of behavior, it is hard for humans to comprehend what is "really" going on.

(ii) Often hard, quantitative information is not available although some qualitative information, or order of magnitude estimates may be at hand. By forcing models to utilize only quantitative information, we unnecessarily restrict the applicability of the models.

(iii) Carrying out simulations with completely quantitative values is computationally intensive, and is often considered to be "infeasible". Simulations using qualitative data sometimes offer a tractable or cheaper solution to this problem.

In the modelling approach presented here, no restrictions are placed on whether values of state variables are qualitative or quantitative. Both types of information can be used. However, the rules for constraint propagation are specific to the type of representation (qualitative or quantitative) adopted. Thus, for a particular chemical operations planning application, the allowable qualitative and quantitative values must be clearly specified at each level of the modelling hierarchy, along with the appropriate rules for propagating these values through the relations describing the physical and operational constraints placed on the system.
1.4.2 Hierarchical Model of the Structure of a Chemical Process.

I have just described the modelling hierarchy used to represent the state of a system. It is now necessary to explain how systems can be combined to form new systems, and how the relationships between composite systems are to be represented. Topologically, a system consists of a boundary and one or more openings to permit material to flow in and out. These openings are referred to as "ports" of the system. For example, a continuous stirred tank reactor in which two chemical species A and B react to form a product C, would have three ports. Two of these ports are designed to permit input of the reactants, while the third is designed to remove the product. In addition to physical ports which transfer material in and out of a system and correspond to the convective mass flow "streams", a system may also have "conceptual" ports to allow for the presence of other, non-convective, "streams" such as diffusive heat flow, diffusive mass flow, consumption of a species, etc. In certain systems, such as pipelines, ports may not have a pre-specified input or output function. Whether a given port is an inlet or an outlet will depend on the particular flow situation that prevails during plant operation.

Modelling Composite Systems. Composite systems are clusters of systems which, as a group, perform some common function: e.g. the units of a feed-pretreatment section; the network of chemical reactors with associated heaters, coolers, pumps; the sequence of distillation columns in a recovery section. Such abstractions allow us to model the operational state of an entire processing segment as if it were a single system. For example, the state of a Reaction Section is described by the following set of constraining relations:

\[
\begin{align*}
\text{Mass Balance on A} & \quad (1a) \\
\text{Mass Balance on B} & \quad (1b) \\
\text{Mass Balance on C} & \quad (1c) \\
\text{Mass Balance on Make-Up Coolant} & \quad (1d)
\end{align*}
\]
Energy Balance on Cooling Water
Energy Balance on Steam
Total Energy Balance

Unlike the modelling of the operational state of a simple system, however, the operational state of a processing segment possesses *spatially distributed* capacities, i.e. properties distributed over the sub-systems of which the segment is composed. In order to maintain the analogy between the operational state of process segments and the operational state of simple systems, "pseudo-holdups" are introduced, indicating the accumulation of energy or mass within the boundaries of the processing segment. Such a model allows us to express operational states of the Reaction Section in the following forms: "increasing accumulation of energy", "imbalance in the utilization of make-up cooling water", "reduction in the consumption of steam", "operation approaching steady-state", etc. During the planning of operating procedures, it is useful to express operational goals in terms of entire processing segments, e.g. "feed-pretreatment section has been started up", "safety fall-back procedure for the network of reactors has been activated", "catalytic reactors 1 and 2 have been regenerated". A computer-based planning methodology should be aware of the abstraction involved in the definition of a composite system.

The generic structure used to represent the composition of systems is the Network Data-Structure. This model consists of the following generic template:

(1) **DEVICES**: A set (unordered list) of sub-system objects which comprise the network.

(2) **NODES**: A subset of the DEVICES which have only outputs or only inputs.

(3) **CONNECTIONS**: A set of pairs of system ports, each pair denoting that there exists a connection between two systems.

**Modelling of Processing Units.** The declarative modelling of a single processing unit is based on the network data structure described earlier, composed of the following parts:
(a) All of the simple sub-systems which comprise the unit (such as gas phases and liquid phases) are stored in the DEVICES slot of the network model. The operational state of each simple system is modelled by a data-model such as the structure illustrated in Figure 1.6 (a).

(b) The contact between phases is represented in the form of connection objects, and these are kept in the slot marked CONNECTIONS. The "connection" objects contain the relations describing the equilibrium and rates of transport between phases. Figure 1.7 shows the data-structures used to model a typical processing unit.

Modelling Processing Segments. Groups of chemical equipment can be abstracted into a single object known as a processing segment. A processing segment can be at the level of an augmented unit, such as a distillation column with its accompanying pumps, condensers and reboilers, or can be an entire section of a flowsheet. Representation of a processing segment is based on the network data structure described earlier, composed of the following parts:

(a) All of the processing units in the processing segment, along with the pipelines connecting them, are stored in the DEVICES slot of the network model. Some of the DEVICES may themselves be processing segments.

(b) A processing segment may contain objects which are "sources" and "sinks" of material and energy. These objects have very large capacities and can be assumed to have constant operational state. For example, a "source" could be a storage tank containing raw material, while a vent to the atmosphere is a "sink". Sources and sinks are stored both in the DEVICES slot as well as the special slot marked NODES.

(c) The interconnections among the devices and nodes are kept in the slot marked CONNECTIONS. The "connection" objects are produced automatically during the graphical construction of process flowsheets by the user, using any versatile design-oriented system such as the DESIGN KIT (Stephanopoulos, et al., 1987).
Figure 1.7 Data-Structure Modelling a Flow-Valve
Modelling of Complete Plants. The complete plant is a further abstraction based on the processing segments. Its topology is modelled by a similar network data-structure as that used for the process segments, while its operation is modelled by a data-model similar to that used for representing the operational state of a composite system.

1.5 CONSTRUCTION AND USE OF HIERARCHICAL MODELS

This section contains a series of examples demonstrating the construction of the modelling hierarchy for a plant, and the use of the structure for various tasks which are part of a comprehensive planning methodology. As mentioned earlier, a description of the complete methodology is the subject of the next chapter of this thesis.

1.5.1 Construction of an Example Hierarchical Model.

Broadly speaking, there are two ways in which a hierarchical model is assembled: (i) user-guided assembly of successive levels of abstraction, and (ii) automatic grouping of equipment based on knowledge/procedures encoded within the computer.

Input of the Two-Dimensional Flowsheet. The user begins by defining the "flat" structure of the plant (or section thereof). This simply implies that the user must put together a conventional two-dimensional flowsheet representing the system. Graphic input of a standard flowsheet is achieved using the interface provided by DESIGN-KIT. While the icons are selected from a menu using the mouse models (internal to the computer) representing the equipment are automatically created. Each individual type of equipment has a "template" model which consists of a structure for representing its operational state, and as the connections between equipment are made graphically, the corresponding network data-structure representing the two-dimensional flowsheet is created. Along the way (or at the end of the input) the user defines which of the equipment in the plant (steam sources, cooling water, storage tanks, etc.) are to be considered as "nodes". The example
considered here is the reaction section of a powerforming plant, pictured in Figure 1.8 (Rivas, 1973). The associated network constructed by the computer is shown in Figure 1.9 (a).

**User-Guided Generation of Hierarchical Structure.** The user of the planning program is offered the facility to create tailor-made hierarchies of models in a mouse- and menu-driven manner. This is done by selecting a group of equipment - a set of ports, valves, pipes and other equipment - and then specifying that they represent an "augmented object". The user is then prompted for a name for the new abstract object. Note that sets of augmented objects may also be grouped together to form successively more abstract descriptions of the chemical plant. Internally, the grouping of equipment into abstractions has the following effect: (i) the computer creates a new hierarchy (or accesses an existing hierarchy), (ii) collects together all of the units which the user has selected as comprising the group, (iii) creates a new "device" and names it with the label supplied by the user and, (iv) replaces the selected units in the hierarchy with the composite, or augmented device. The internal structure of the augmented device is itself a network, namely the sub-network of the "flat", two-dimensional, flowsheet which comprises the selected equipment. To illustrate this abstraction process on the flowsheet in Figure 1.8, consider the case where the user selects the auxiliary units surrounding REACTOR-1, provides the name "AUGMENTED REACTOR 1" and stipulates that the level of the hierarchy on which the abstracted unit is to be placed is the "AUGMENTED-UNIT-LEVEL". The purpose of this exercise is to indicate to the computer that the reactor, along with its accompanying peripherals, is to be lumped as a single "subsystem" within the system representing the entire reaction section. A new device is created which has a network structure shown in Figure 1.9 (b). Assuming that the user follows a similar procedure for creating an abstraction of the second reactor with its peripherals, and places it, too, on the augmented unit level, the resulting network representing the chemical plant is shown in Figure 1.9 (c). Comparison of Figures 1.9 (a) through 1.9(c) reveals the mutations in the data-structures caused by the abstraction process.
Figure 1.8 Reaction Section of a powerforming plant
<table>
<thead>
<tr>
<th>NETWORK: POWER-FORMING REACTION SECTION</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>DEVICES: (PIPE-1, JUNCTION-1, PIPE-2, PIPE-3, VALVE-1, VALVE-2, PIPE-5, JUNCTION-4, PIPE-17, PIPE-19, REACTOR-1, ......)</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>NODES: (HYDROCARBON-SOURCE, SINK-1 SINK-2)</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>CONNECTIONS:</th>
</tr>
</thead>
</table>

- ((PORT-1(PIPE-1) . OUTLET(HYDROCARBON-SOURCE))
- (PORT-2(PIPE-1) . PORT-1(JUNCTION-1))
- (PORT-2(JUNCTION-1) . PORT-1(PIPE-3))
- (PORT-3(JUNCTION-1) . PORT-1(PIPE-2))
- ...
- (PORT-2(PIPE-29) . INLET(SINK-2))

Figure 1.9 (a) Network data-structure before definition of augmented reactors
<table>
<thead>
<tr>
<th>NETWORK:: AUGMENTED REACTOR 1</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DEVICES:</strong></td>
</tr>
<tr>
<td>(PIPE-2, VALVE-2, PIPE-5,</td>
</tr>
<tr>
<td>JUNCTION-4, PIPE-17, PIPE-19,</td>
</tr>
<tr>
<td>REACTOR-1, PIPE-21, JUNCTION-6,</td>
</tr>
<tr>
<td>PIPE-28, VALVE-9, PIPE-27)</td>
</tr>
<tr>
<td><strong>NODES:</strong></td>
</tr>
<tr>
<td>()</td>
</tr>
<tr>
<td><strong>CONNECTIONS:</strong></td>
</tr>
<tr>
<td>((PORT-2(PIPE-2) . PORT-1(VALVE-2))</td>
</tr>
<tr>
<td>(PORT-2(VALVE-2) . PORT-1(PIPE-5))</td>
</tr>
<tr>
<td>(PORT-2(PIPE-5) . PORT-3(JUNCTION-4))</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>(PORT-2(VALVE-9) . PORT-1(PIPE-27))</td>
</tr>
</tbody>
</table>

Figure 1.9 (b) The Network representing Augmented-Reactor-1
**NETWORK**: POWER-FORMING REACTION SECTION

**DEVICES**: (PIPE-1, JUNCTION-1, VALVE-3
   VALVE-4, AUGMENTED-REACTOR-1
   AUGMENTED-REACTOR-2, VALVE-11
   VALVE-12, SINK-1, SINK-2,
   HYDROCARBON-SOURCE)

**NODES**: (HYDROCARBON-SOURCE, SINK-1
   SINK-2)

**CONNECTIONS**:

((PORT-1(PIPE-1) . OUTLET(HYDROCARBON-SOURCE))
(PORT-2(PIPE-1) . PORT-1(JUNCTION-1))
(PORT-2(JUNCTION-1) . PORT-1(AUG.-REACT.-2))
(PORT-3(JUNCTION-1) . PORT-1(AUG.-REACT.-1))
...
(PORT-2(AUG.REACT.-2) . INLET(SINK-2)))

Figure 1.9(c) Network data-structure after definition of augmented reactors
Automatic Generation of Abstractions by the Computer. Computer-generated abstractions of the two-dimensional plant model centers around two concepts: abstraction from templates and abstraction based on heuristic knowledge.

For many standard groups of equipment such as a distillation column with the accompanying pumps, condenser and reboiler, or a reactor with feed and effluent heat exchangers, it is possible to store in the computer, predefined templates of "augmented" equipment, and whenever these patterns are found within the flowsheet, the appropriate regrouping of equipment and network modification is performed (following the steps outlined above).

In some cases, it is possible for the computer to anticipate an effective decomposition of the flowsheet. These cases are usually not defined on the basis of theoretical grounds and stem from experience in planning operating procedures for chemical plants. An excellent example provided by Fusillo and Powers (Fusillo and Powers, 1987b) is the case where "stationary states" can be identified within the chemical plant. A stationary state is an operating state at which the overall operating goals are partially met and the state of the system does not change significantly over time: once a stationary state has been achieved in a section of a plant, operation can usually be maintained indefinitely within that section. It should be noted that achievement of a stationary state within a section of a chemical plant does not imply that the goal operating states of all equipment in that section have been achieved. Typically, many of the operating goals will have been achieved, but further planning may still be necessary to attain the goal state in some parts of the section. Stationary states can be associated with recognizable criteria:

(i) The presence of "simultaneous inverse operations". Simultaneous inverse operations take the following form: a transformation is performed in one part of the process and then the inverse transformation is performed in a connected part of the process. For example, in a closed flow loop with a pump in line, momentum is imparted to the fluid by the pump and is subsequently lost to friction in the piping. Another example is the operation at total reflux of a distillation column.
(ii) Systems with Large Capacitances. A system whose capacity for some physical quantity (such as thermal energy, material or pressure) is so large that its state varies slowly with respect to the rest of the process can serve as a stationary state during the operating procedure.

Planning to achieve stationary states has the following advantages.

(i) Stationary states serve as convenient stopping points during operations and can function as recovery states in case of emergencies.

(ii) They provide a useful means of focussing the plant-wide operation on a segment of the plant at a time, such that the attention of the monitoring personnel is not distributed.

By searching for the previously mentioned criteria within the structure of the chemical plant, the computer can generate abstractions of processing units which, as a group, possess a stationary state.

1.5.2 Examples Utilizing the Hierarchical Structure.

The remainder of this section is devoted to examples of the utility of the hierarchical, structured models for various planning tasks.

**Specification of Initial Operating State.** The input of initial state may be initiated at any level in the abstraction hierarchy that is convenient to the user (e.g., at the processing segment level, at the unit level, etc.) The user begins by specifying the values of known variables, and these may be either qualitative (+, -, 0) or quantitative. Referring to the abstracted flowsheet represented by Figure 1.9 (c), an example of a qualitative initial state specification could be a statement such as:

FLOW out of AUGMENTED-REACTOR-1 is POSITIVE.

This statement is made at the augmented unit level of the structural hierarchy, and the computer now traces down the hierarchy to the level of the individual processing units (Figure 1.8). It discovers that the exit port of AUGMENTED-REACTOR-1 is actually PIPE-27. This causes the value POSITIVE to be assigned to the variable representing the flow out of PIPE-27. In a similar
manner, other values of the initial state can be input into the hierarchy and inherited down to the lowest level.

**Propagating a Partial State Description.** Once the user has input a partial state description, the values of the variables can be propagated through the device models, and this results in specification of other variables in the system. Continuing from the initial state specification above, propagation of the constraint "FLOW through PIPE-27 is POSITIVE" results in the following conclusions: "The operating REGION of VALVE-9 is OPEN", "FLOW through PIPE-28 is POSITIVE", and, "FLOW in PORT-3 of JUNCTION-6 is NEGATIVE." After the constraint propagation algorithm has quiesced, if a portion of the operating state is left unspecified, the user is prompted to enter more information, and propagation is re-initiated until a complete description of the initial operating state of the system is obtained.

**Checking for Conflicts in Operating State.** It is possible that the user could accidentally overspecify the system, or could input a false initial state description which is inconsistent with the behavioral model of the plant. An example of such an error could be the specification that VALVE-9 is closed, while at the same time specifying that the FLOW through PIPE-27 is positive. The constraint propagation system will immediately spot such a contradiction (since it will conclude a dual value for a single variable) and, using the dependency network set up during the course of constraint propagation, the computer can backtrack to the source of the two conflicting statements. The user can then be asked to choose one of the two statements (fluid flowing, or valve closed, but not both) and to retract the other.
1.6 SUMMARY

In this chapter, I have discussed some of the issues involved in automating the synthesis of plant-wide operating procedures for continuous chemical processes. In particular, I have focussed on the question of how to effectively model, on the computer, knowledge about the operation of chemical plants. Previous attempts at modelling have either relied on the restrictive operator-based modelling paradigm, or have employed extremely simplified or situation-specific models. A modelling technique based on a functional operator structure was also considered. This representation accounted for the fact that the outcome of applying an operator is dependent upon the state of the system before execution, and was shown to reduce to the conditional action scheme investigated by researchers in artificial intelligence. Planning with conditional operators involves a plan generation step which is NP-Hard, and is therefore considered to be computationally intractable. This justifies the adoption of a domain-specific approach to the planning of process operations, and explains why no algorithmic solution has yet been developed.

Realizing the limitations of the operator model, a hierarchical, distributed, object-oriented modelling structure was described. The models are expressive enough to allow for automatic generation of inter-unit constraints, which is accomplished through propagation of known values through a network of relations representing the physical laws governing the behavior of a chemical plant and constraints induced by practical engineering considerations. The modelling framework is sufficiently general so as to allow the use of both quantitative information as well as qualitative knowledge, such as order-of-magnitude estimates.

Examples of the construction and use of the hierarchical modelling structure were presented. The non-linear planning methodology that makes use of the modelling structures described here is presented in subsequent chapters of this thesis.
CHAPTER 2

NONLINEAR PLANNING FOR TEMPORAL CONSTRAINTS
2.1 INTRODUCTION

In Chapter 1 I reviewed previous work in the area of operations planning and focussed on modelling issues involved in the synthesis of operating procedures. The two basic approaches to planning, namely, the linear and the non-linear, were described. It was pointed out that non-linear planning can be exponentially more efficient than linear planning, which often results in a combinatorially explosive number of plans. Previous methodologies developed for planning operating procedures were inherently linear in nature, and there is a need for a non-linear operations planning methodology.

Unfortunately, there are theoretical limitations on the type of models that are amenable to non-linear, general-purpose (domain-independent) planning. Those models which are suitable, namely, the STRIPS operator models, were shown to be inadequate for modelling a continuous chemical plant. The chief limitation of the domain-independent approaches is that they do not utilize knowledge specific to a particular domain, and hence lack the power to solve practical problems. In Chapter 1 I described a hierarchical, distributed, object-oriented modelling structure specially suited to the planning of chemical process operations. In this chapter I present a non-linear planning methodology based on the modelling structure developed in Chapter 1.

A major objective of this research undertaking was to establish a formal theory for planning process operations. Previous research efforts in the area have been ad-hoc in nature, and no attempt has been made to formally analyse the properties of the planning methodologies proposed. Here I describe the salient algorithms which comprise my planning methodology, clearly state their properties, and provide proofs for the same. This type of in-depth analysis is essential for any computer program that attempts to solve a potentially combinatorial problem. In the absence of such analysis, the validation of the methodology and the substantiation of claims about its performance can only be empirical.
2.1.1 Comparison of Linear and Non-linear Planning Methodologies.

In the initial phase of planning research, planners operated in what is referred to as a linear manner. This means that they started at one end of a plan - either at the goal or at the initial state, and constructed a plan step by step, moving "closer" all the time to the finish (if planning started at the initial state) or to the start (if the planner moved backwards from the goal state). At every point in the planning process, all variables were assigned values corresponding to objects in the world, and the ordering of all operations performed upto the current state, was completely specified. Non-linear planners, on the other hand, work by constructing and refining "partial plans". A partial plan is a set of plan steps that leaves certain information unspecified. In some cases the exact order in which two steps occur is not specified, in others the planner may not specify a unique operator for a given task, but rather specify a set of candidate operators (which is a subset of the large number of possible operators).

A single partial plan actually describes a large number of "completions", or total plans. Thus, by conducting planning in terms of partial plans (non-linear planning), pruning of a single alternative in a partial plan results in the elimination of a large number of complete plans. This results in a possibly exponential savings in efficiency (Chapman, 1985).

The differences between the two approaches are best illustrated within the context of an example. Consider the simple operations planning problem illustrated in Figure 2.1. Initially, oxygen is flowing into the system through inlet 1, and out of the outlet. The problem faced by the operations planner is to route the flow of methane gas from the inlet 2 to the outlet without running the risk of an explosion (by mixing oxygen and hydrocarbon). The actions available to the planner, and their pre- and post-conditions are shown in the figure. In addition, the following frame axioms hold:

\[(\text{PRESENT METHANE}) \& (\text{PRESENT OXYGEN}) \Rightarrow (\text{EXPLOSION}); (\text{FLOWING } *X*) \Rightarrow (\text{PRESENT } *X*)\]

Linear Planning. A linear planner begins at the goal state, and chooses one of the goals, e.g. the goal (FLOWING METHANE). It applies the operator (ESTABLISH-FLOW *X*) to METHANE, achieving
INITIAL-STATE      GOAL-STATE
(flowing oxygen)   (flowing methane)
not(explosion)      not(explosion)

OPERATORS
(stop-flow x)      (establish-flow x)
pre-conditions: (flowing x)      pre-conditions: ()
post-conditions: not(flowing x)  post-conditions: (flowing x)

(purge x)  pre-conditions:  
         not(flowing x); not(equal inert-gas x)
         post-conditions:  not(present x); (flowing inert-gas)

FRAME AXIOMS
(present methane) & (present oxygen) => (explosion!)

(flowing x) => (present x)

Figure 2.1 Example Operations Planning Problem
this goal. However, this causes first frame axiom to deduce (EXPLOSION), so the planner must backtrack. It then tries to achieve the other goal, namely NOT(EXPLOSION). The planner now has to make a choice. It must apply the (PURGE 'X') operator, and must decide whether to bind 'X' to OXYGEN or to METHANE. Since a linear planner does not "remember" earlier search paths, the planner may try to purge METHANE and then, later on fail, being unable to achieve (FLOWING METHANE). Clearly this is a very inefficient and simple-minded approach.

**Non-Linear Planning.** A non-linear planner faced with the above problem will proceed as follows. It chooses one of the goals, as before, and tries to find an operator which achieves it. Assuming that (ESTABLISH-FLOW METHANE) is chosen, the planner will again run into an explosive situation. However, when a non-linear planner runs into such a problem, it does not back-track. Instead, it tries to alleviate the constraint violation using one or more of the allowable operators, constraining the application of these operators to occur before the constraint violation. This is referred to as "promotion" (Chapman, 1985). In order to avoid an explosion, the planner constrains the action (PURGE 'X') to occur before (ESTABLISH-FLOW HYDROCARBON). Recall that in the case of the linear planner, all variables get bound immediately. This is not the case in a partial plan developed during non-linear planning. Thus the planner leaves the variable 'X' temporarily unbound. It now examines the sub-goals which have been generated by the two primitive actions chosen. The sub-goals to achieve are:

- NOT(FLOWING 'X') and NOT(EQUAL INERT-GAS 'X')

Since (FLOWING METHANE) is required in the goal state, the only suitable value for 'X' is OXYGEN. Thus, the computer attempts to achieve NOT(FLOWING OXYGEN) and, using the STOP-FLOW operator, succeeds.

The solution to the planning problem is now complete, and involves the following sequence of operating procedures:

(STOP-FLOW OXYGEN) > (PURGE OXYGEN) > (ESTABLISH-FLOW METHANE)
It is important to note that, while the linear planning methodology had to do a considerable amount of search, the non-linear methodology generated a feasible plan without backtracking.

2.1.2 Outline of this chapter.

This chapter presents a non-linear planning methodology which utilizes the modelling structure developed in Chapter 1. The description of the planning methodology in this chapter is structured as follows.

(a) Section 2.2 contains an outline of the two-phase, non-linear planning methodology, with a brief description of the function and purpose of each stage in the methodology. It also contains the description of an industrial case-study which will be used in Sections 2.3, 2.4 and 2.5.

(b) Section 2.3 describes, in detail, the first phase of the planning methodology, namely the Problem Formulation Phase.

(c) Section 2.4 presents the detailed methodology for synthesizing operating procedures. Details of the algorithms used during this phase, along with the proofs of their correctness/completeness are located at the end of the chapter.

(d) In Section 2.5 I explain how the planning methodology is applied to the industrial case study described in Section 2.2.
2.2 OUTLINE OF THE PLANNING METHODOLOGY

The methodology for non-linear planning of operating procedures consists of two distinct phases: (a) the Problem Formulation Phase and (b) the Plan-Synthesis Phase. Each of these phases can be broken down into a set of tasks, as described below.

2.2.1 Phase I: Problem Formulation Phase

The first phase of the planning methodology consists of four stages which must be carried out to in order to completely define the planning problem in a form that can be understood by the computer.

A. Description of the Initial State.

The first part of the problem formulation involves a description of the initial state of the plan. In many traditional planners, such as those developed by researchers in the field of artificial intelligence, the description of the initial state simply consists of a set of facts that are true about the system at the start of the plan. In typical chemical engineering processes, however, the network-like structure of a continuously operating plant makes it harder to ensure that the operational state of the system satisfies the physical laws of conservation (mass, energy and momentum) and the second law of thermodynamics. The process for describing the initial operating state of a chemical plant should consist of the following steps.

(i) **Input of the Initial Operational State of the Plant.** Specification, by the user, of portions of the operating state at various locations in the plant.

(ii) **Ensuring Completeness of the Initial State Description.** The partial initial state description input by the user has an impact on those portions of the plant whose state is still unspecified. The program contains equation-based models of these laws, and can propagate the user-supplied values through the network of equations to specify the initial state at other points in the system.
(iii) *Checking for Consistency of the Initial State Description.* During input of the initial operational state, it is conceivable that the user might enter initial values which are not consistent with the constraining equations. The planning program should have procedures for detecting these conflicts, efficiently identifying the sources of the conflicts, and providing transparent mechanisms for resolving them.

**B. Description of the Final, or Goal, Operational State**

The next step in the problem formulation phase is the definition of the goal state.

(i) *Input of the Planning Goals.* The input of planning goals is carried out in a manner analogous to the input of the initial state description, described earlier.

(ii) *Checking for Conflicts in the Goal State Specification.* The specification of inconsistent operational states is an important concern during the input of the goal state description, since the goal state is often only a "conceptual" state, which does not physically exist. Thus, while defining the goal state on the computer, it is especially important to carry out the conflict checking, identification and resolution referred to earlier.

(iii) *Generation of Concurrent Goals.* The user of a planning program typically specifies only a small number of goals at selected locations in the chemical plant. In chemical processing systems, and other systems with a large amount of interaction between sub-systems, achieving these goals usually implies achieving a number of other "concurrent" goals. The mechanism used by the planning methodology presented in this paper to identify concurrent goals is a constraint propagation technique analogous to the one used for ensuring completeness of the initial state description.

**C. Specification/Identification of Operational Constraints on the Plan**

In addition to the physical constraints governing the operation of the chemical process during
planning, there are usually several "operational" constraints that the user may require to be satisfied. After definition of the initial and goal states, these operational constraints are either specified explicitly by the user, or generated automatically by the computer. Alternatively, the computer could propose constraints to the user who could then select the ones he/she prefers.

D. Specification/Identification of Planning Islands

In planning problems where a long sequence of valve operations is expected, and where the process units will pass through a large number of intermediate states, it may be advantageous to identify what are known as planning islands (Chakrabarti, et al., 1986). These planning islands are partial descriptions of intermediate states which are known to lie on the paths of the most efficient, or feasible, plans. By specifying these intermediate states, the search through the many alternative operating routes can be efficiently controlled. The concept of the planning island, along with methods for its identification will be described in greater detail in Section 2.3.4.

2.2.2 Phase II. Plan Generation

Having defined the operations planning problem in a form which is understood by the computer, the next phase of the planning methodology consists of synthesizing plans to solve the stated problem. This generation of plans is carried out in three stages.

A. Identification of Primitive Actions

First, the primitive operations required to carry out the transformation from initial to final states must be identified. In this methodology, the means-ends analysis paradigm is employed to identify these primitive "operators".

B. Construction of a Partial Plan

The second stage involves the construction of a "partial plan". This consists of deriving a
partial ordering on the primitive operations. This partial ordering stems from the operational constraints placed on the operating plan together with the physical laws which govern plant operation, and is analogous to the constraint-posting philosophy of nonlinear planning described by Chapman (1985).

C. Synthesis of Complete, Feasible Plans

At present, the non-linear planning methodology only manipulates operational constraints which are temporal in nature. Some operational constraints stated by the user may not be amenable to direct transformation into temporal constraints between goals. In such cases, a systematic, linear, generate and test strategy based on the partial plan generated in the previous step is necessary to develop a feasible plan (or a set of feasible plans) which solve the problem.

2.2.3 Case Study: Regeneration of a Catalytic Reactor

The methodology for planning process operations outlined above is best described within the context of an industrial example. We describe, here, a system first analyzed by Rivas (Rivas, 1973). The chemical process consists of the powerforming reactors of a petroleum refinery, along with the accompanying catalyst regeneration system (see Figure 2.2). During operation of the plant, the activity of the catalyst in the reactor (chloroplatinic acid on an alumina base) gradually decreases due to the following processes: (i) coke formation on the catalyst, (ii) reduction in the chloride content, and (iii) size growth of the platinum crystals. Regeneration of the catalyst involves the following operations:

(i) **Coke removal.** Removal of accumulated coke is accomplished by injecting inert gas containing low concentrations of oxygen into the reactor at temperatures sufficiently high to initiate and sustain combustion. The inert gas is generated by burning natural gas and air to inertness.

(ii) **Addition of chloride.** Chloride is added to the catalyst by injecting carbon tetrachloride into
Figure 2.2 Flowsheet of the Catalyst Regeneration System
the heated inert gas. The CCl₄ breaks down to form free chlorine before it reaches the catalyst.

(iii) Inhibition of platinum crystal growth. Excessive growth of the platinum crystals is prevented by redispersing the crystals. This is done by proper chlorination and by oxygen soaking or "rejuvenation".

A computer-based planning program is required to synthesize the sequence of valve operations which will accomplish the above regeneration steps. In order to do this, however, a model of the process must be created on the computer. The procedure for creating the modelling structure which represents a process flowsheet was explained in Part I of this series. The creation of the models for the reaction section of the Powerforming Plant was described in detail, and is summarized below:

(i) The "flat" two dimensional flowsheet depicted in Figure 3 is constructed graphically in a menu-driven manner, using a mouse.

(ii) Groups of equipment are selected with the mouse and named as "augmented" equipment. This causes the computer to set up a hierarchical data-structure representing the flowsheet at different levels of abstraction.

(iii) For this example, the two meaningful abstractions created by the user are the augmented reactors 1 and 2, shown in Figure 4.
Figure 2.3 Flowsheet at the Augmented Unit Level
2.3 PROBLEM FORMULATION PHASE

At the end of the structure-definition process above, the computer has created a hierarchy of network models representing the reaction section of the Powerforming Plant. Initially, REACTOR-1 is operating at steady state, and its catalyst has become deactivated over time as a result of the phenomena described earlier. REACTOR-2, on the other hand, has fresh catalyst, and is waiting to be brought on-line. The first step in the catalyst regeneration process is to take REACTOR-1 off-line and to bring REACTOR-2 on-line. The automatic planning of the valve operations which will accomplish this change-over will be described in this section and in Section 4, and will serve to illustrate the various stages of the planning methodology.

2.3.1 Initial State Description

In addition to the modelling structure representing the chemical plant, the computer must be informed of the initial operating state of the plant. As mentioned earlier, the initial description is extracted from the user in an interactive session which involves the following: (i) input of certain portions of the operating state (by the user), (ii) automatic generation of the operating state of some of the associated equipment through constraint propagation, and (iii) checking for consistency of the initial state specification. This three part process is repeated until the complete initial operating state has been specified. Each of the steps in the cyclic process is described in more detail below.

Input of Initial State: The input of initial state may be initiated at any level in the abstraction hierarchy that is convenient to the user (e.g., at the processing segment level, at the unit level, etc.) The user begins by specifying the values of known variables, and these may be either qualitative (+, -, 0) or quantitative. In chapter 1 the relative merits of qualitative and quantitative state descriptions were discussed, and it was pointed out that limiting the state description to one or the other form presented an unnecessary restriction on the flexibility of the program. Thus the
methodology described here is general enough to handle both types of information. It should be noted, however, that if quantitative planning goals are stated, or if some of the operating constraints are quantitative in nature, a qualitative description of the initial state may be insufficient.

An illustration of the input of qualitative initial state specification at the augmented unit level appears in Figure 2.5. In this figure, the user has specified that the flow out of AUGMENTED-REACTOR-1 is positive, and that the reaction product is present in this stream. In addition, valves 3, 4, 11 and 12 are closed. The statement regarding the presence of hydrocarbon at the inlet of PIPE-1 is generated automatically by the computer.

**Ensuring Completeness of the Initial State Description.** Once the user has input a partial state description, such as the one in Figure 2.5, the values of the input variables are stored within the objects representing the individual sub-systems in the hierarchically structured system. At the most detailed level in the hierarchical structure, the computer has a constraint network which represents a complete set of equations for the physical laws governing the operational state of the system as a whole. The values of initial state variables input by the user can be propagated through the system models, resulting in the specification of other variables in the system. The constraint propagation algorithm utilizes the available partial state description and aggressively tries to deduce the state of the rest of the plant. After the algorithm has quiesced, a portion of the operating state is likely to be left unspecified. This is because we have not paid any special attention to ensuring that all the degrees of freedom have been specified by the user. If this is the case, the user is informed graphically of the portions of the system whose state is unspecified, and is prompted to enter more information about the initial operating state. The constraint propagation algorithm is then re-run and the cycle of input/propagation is continued until a complete description of the initial operating state of the system is specified. In the event that the constraint propagation algorithm, in utilizing qualitative values, yields multiple solutions, the user is asked explicitly to resolve such ambiguities. This is acceptable since the initial state of the plant is unique and is known.
Figure 2.3 The menu-driven input of an Operational State
Figure 2.5 Initial State Input at the Augmented Unit Level
The process of constraint propagation is illustrated beginning with the partial state description of Figure 2.5. Utilizing the equation-based models at the individual unit level, the computer begins by propagating the effects of the statements about the closed valves 3, 4, 11, 12. The shaded arrows in Figure 2.6 denote the "flow of information" as it is propagated through the plant network. From the equations governing a flow-valve operating in the CLOSED region, flows on either side of the valve are 0. Using this information and the models for other equipment such as the pipe, the flows in the following systems are set to 0: PIPE-17, PIPE-18, PIPE-23 and PIPE-24.

Next, the statement that flow through PIPE-27 is positive may be propagated backwards through REACTOR-1 and all the way to JUNCTION-1. (Since the machine has already concluded that the flow through PIPE-23 is 0, and that flow through PIPE-28 is positive, it can infer that out of JUNCTION-6, for continuity to hold, we must have positive flow through PIPE-21.) During this process, the computer has deduced that valves 2 and 9 are open. Having established that the flow through REACTOR-1 is positive, we can propagate the statement regarding the presence of product in PIPE-27 back up to the inlet of REACTOR-1. At this point, the state of the system is described by Figure 2.6.

An important point to be made here is that the order in which the constraints are propagated is an important factor determining the speed at which the constraint propagation algorithm converges. Knowledge about chemical processes tells us that materials are transported by the flows within and between the equipment in the plant. The constraint propagation algorithm used in the planning methodology here, being tailored for applications in the chemical engineering domain, attempts to first propagate quantities relating to material flow. Specifically, since we know that closed valves are good starting points (flows around them must be zero) we begin the propagation at those valves which the user states are closed. Once as many conclusions as possible have been made from these input statements, we proceed to other flow specifications and propagate them through the network. Having established the flows (to the extent possible) the algorithm turns its attention to other
Figure 2.6 Ensuring Completeness of the Initial State
quantities such as composition and temperature whose propagation depends on the flows existing in the plant. General purpose constraint propagation systems which do not incorporate domain knowledge are likely to be much less efficient.

Clearly, the state description in Figure 2.6 leaves several quantities unspecified. The constraint propagation network has concluded as much as is possible from the data supplied, and further information is required from the user in order to completely specify the initial operating state of the system. In an interactive session, the user states that valves 1 and 10 are CLOSED, and that hydrogen is present in PIPE-20, REACTOR-2, PIPE-22 and PIPE-30. Pipes 3, 4, 18 and 24 contain hydrocarbon. This information is sufficient to complete the specification of the initial state of the reactor system.

Checking Consistency of the Initial State Description. During the input of the initial state, it is possible that the user could accidentally overspecify the system, or could input a false initial state description which is inconsistent with the physical model of the plant. An example of such an error is the specification that a particular valve is closed, while at the same time specifying that there is a positive flow downstream of the valve. Such a contradiction will show up in the form of multiple values assigned to a single variable. For example, a valve could be specified as being both OPEN as well as CLOSED. The constraint propagation system will immediately spot such contradictions.

To keep track of the flow of computation during constraint propagation, a dependency network is maintained (Steele, 1980). While constraints are being propagated, every time a value is propagated through a "connection" between two sub-systems, the direction in which information flows through the "connection" is recorded in a slot on the connection object. Thus, at any point during the constraint propagation, the subsystems and their labelled connections will form a directed graph which describes the direction of computation. This graph indicates which values depend on which other values, and is therefore said to encode dependency information. Using this dependency network, the computer can backtrack to the source of the conflicting statements, and
ask the user to "retract" one of the two primary statements (fluid flowing, or valve closed) since both cannot co-exist.

2.3.2 Goal State Description

The methodology for description of the goal operating state shares many common features with the procedure for specifying the initial operating state. The main difference lies in the fact that the user need not specify a complete description of the goal operating state, while a complete description of the initial state is mandatory. An implicit assumption is made that the user will specify all "important" goals. Thus, any final operating state which achieves these "important" goals will be acceptable.

Input of Goal State. The input of the goal state can, like the input of the initial state, be made at any level of abstraction which the user may choose, and can be in qualitative or in quantitative form. The process is interactive as before, and allows the user to input a partial state description, observe what implications these goals have on the operating state of the rest of the plant, and then input other goals (or retract some of the previously stated goals) until he or she is satisfied with the statement of the goal operating state. Continuing with the formulation of the reactor change-over problem, the input of the goal state involves specifying that (i) AUGMENTED-REACTOR-1 is off-line, that is, there is no flow out of PIPE-27, and that (ii) AUGMENTED-REACTOR-2 is in operation (flow through PIPE-29 is positive). The partial description of the goal state is illustrated in Figure 2.7.

Checking for Goal Conflict/Generation of Concurrent Goals While specifying the goal operational state of the system, it is particularly easy for the user to specify goals which are in conflict with each other. This is because, unlike the initial state, the goal state is not always tangible, and may be incompletely defined in the user's mind. The constraint propagation
algorithm for generating concurrent goals operates in a manner similar to the algorithm used for ensuring completeness of the initial state. It aggressively propagates the user's stated goals and identifies the obvious conflicts when they occur. The dependency network facilitates easy identification of the source of the conflicting statements so that the user can conveniently retract one or more of the offending goals. The goal state description for the problem discussed in Section 2.3.1 is depicted in Figure 2.7. The shaded arrows in the figure denote the flow of information and indicate the dependence of derived quantities (such as the flow out of the hydrocarbon source and the flow into REACTOR-2) on primary goal specifications (such as the statement that the flow through PIPE-29 is positive).

2.3.3 Specification / Identification of Constraints

The operation of chemical processes is usually required to be consistent with certain "operational" constraints. During planning, these constraints must be satisfied in addition to the physical constraints (conservation laws) on the system.

Sources of Operational Constraints. The sources of these constraints have been discussed in the literature (Fusillo and Powers, 1987), and are as follows:

(i) Pre-conditions for Certain Unit Operations. To avoid damage to equipment, or to ensure normal operation, some condition must be achieved before a task can be performed. Examples of such pre-conditions include priming of pumps and cleaning of heat exchanger tubes.

(ii) Production Requirements. Various production requirements dictate maximum allowable impurity levels, as well as specifications on plant throughput and product concentrations.

(iii) Safety Hazards. Safety hazards, such as explosions or damage to the environment must be avoided. This can cause certain species to be prohibited in selected locations in the plant, or can prohibit the contacting of species which form explosive mixtures.
Figure 2.7  Input of Goal State/Generation of Concurrent Goals
(iv) **Materials of Construction.** The materials of construction of a processing vessel may be sensitive to certain operating conditions. Thus, in order to avoid undue stress on the equipment, corrosive chemicals or mixtures may be prohibited from entering the system, or temperatures and pressures may have to be maintained within allowable ranges.

Operational constraints pertinent to a given planning problem must be either specified by the user, or automatically identified by the program, before synthesis of the operating procedures is attempted.

**Specification of Operational Constraints by the User.** The planning program, currently assumes that all operational constraints will be supplied by the user. Operational constraints specified by the user take two general forms: (i) temporal (precedence) constraints and (ii) non-temporal constraints.

Temporal constraints are the most powerful for constraining the space of feasible plans, and it is especially important for a non-linear planning methodology to use these constraints in an efficient manner. Non-temporal constraints are harder to handle in a nonlinear manner, and are the subject of the next chapter.

Revisiting the catalyst regeneration problem, we identify a temporal constraint which stems from a production requirement. During the change-over from REACTOR-1 to REACTOR-2, production demands dictate that, at all times, product must be flowing into either SINK-1 or SINK-2. These two sinks lead to another process down stream in the refinery (not pictured) which sets the continuous demand for the product. Thus a temporal constraint is identified which states that before achieving the goal "flow out of AUGMENTED-REACTOR-1 = 0", we must first achieve the goals "flow out of AUGMENTED-REACTOR-2 is positive" and "product species is present in the effluent of AUGMENTED-REACTOR-2". Symbolically, this is written at the augmented unit level as:
(FLOW(P2(AUG.-REACTOR-2)) > 0) \land (X-PRODUCT(P2(AUG.-REACTOR-2)) > 0) \\
\implies (FLOW(P2(AUGMENTED-REACTOR-1)) = 0)

Thus, operational constraints may be stated at any level within the modelling hierarchy. Note that we have not yet specified how these constraints will be used, as this is part of the Plan Generation Phase.

**Automatic Generation of Operational Constraints.** The motivation for developing a computer-based planning methodology is the automation of an engineering activity which is currently considered to be a task for the human operator. It would be useful, therefore, to have a facility for automatically generating the operational constraints pertinent to a given operations planning problem. This automation is possible for the following classes of constraints: (i) safety constraints derived from the explosive or corrosive nature of certain chemical species, (ii) safety constraints, such as temperature and pressure tolerances, which stem from materials of construction, and (iii) standard pre-conditions for well-known unit operations (priming of pumps, etc.) which can be derived from experience with operation of chemical equipment. For each of these classes of constraints, the appropriate knowledge can be structured in the form of a standard rule-based expert system. By running the rules in a forward chaining manner on basic information supplied by the user (species present in the plant, models for the individual processing units, materials of construction, etc.) the appropriate operational constraints can be generated. The development of such a rule based system was considered to be outside the scope of this thesis.

2.3.4 Specification / Identification of Planning Islands

In cases where a long sequence of valve operations is expected, and where the chemical system will pass through a large number of intermediate states, it may be necessary to identify what are known as planning islands. These planning islands are intermediate states known to lie on the path of the most efficient, or feasible, plans. The specification of these intermediate states is
necessary in order to control the search through the many alternative operating routes. From a structural and implementational stand-point, the computer treats these islands as goal states, and breaks the problem up into a sequence of smaller planning problems. It then tries to synthesise a plan for achieving the first planning island, starting from the initial state, and then from the first planning island to the second. Planning proceeds, in this manner, from "island" to "island" as illustrated in Figure 2.8.

Typical examples of planning islands in chemical process planning include the formulation of purge goals and description of the so-called "stationary states". Purging problems can occur when dangerous mixtures are to be avoided and one of the species which constitutes the mixture must be purged (using an inert gas, for example). The statement of purge goals is difficult to automate within the framework of the means-ends analysis paradigm. This is because the presence of the inert purgative is often not specified either in the goal state or in the initial state description. To handle such situations, a planning island is generated by the computer (as described in the next chapter) which states, as an intermediate goal, the purging of one or more of the dangerous species.

Another example of the use of planning islands, first identified by Fusillo and Powers (Fusillo and Powers, 1987) is the case where "stationary states" can be identified within the chemical plant. A stationary state is an operating state at which the overall operating goals are partially met and the state of the system does not change significantly over time. If these stationary states are used as intermediate goals states during the planning process, several advantages are achieved.

(i) Stationary states serve as convenient stopping points during operations and can function as recovery states in case of emergencies.

(ii) They provide a useful means of focussing the plant-wide operation on a segment of the plant at a time, such that the attention of the monitoring personnel is not distributed.
Figure 2.8 Schematic Depicting Planning Islands
The identification of planning islands is not based on theoretical grounds, but rather on experiential knowledge about the planning of process operations in chemical plants. There are generally two modes for specifying these islands, and they are described below.

**User Specification of Planning Islands.** Description of a planning island by the user proceeds in exactly the same manner as the description of the goal state. A partial state description of a planning island is input by the user, reflecting the key intermediate goals in the system. For the case of the purge goal mentioned earlier, the partial state description must include statements regarding the flow of inert gas through appropriate pipelines, and absence of the dangerous species (for example "X-HYDROGEN in PIPE-27 = 0"). Proceeding from the partial state description, the computer propagates the user-specified values generating concurrent goals, and checking for conflicts, as in the case of the goal state specification process.

**Automatic Generation of Planning Islands.** Where natural stationary states exist, or where purge goals are to be formulated, the experiential knowledge pertaining to the generation of planning islands can be encapsulated in special purpose rule-bases which are the ideal vehicle for representing and utilizing such heuristic knowledge.

Fusillo and Powers (Fusillo and Powers, 1987) pointed out that stationary states can be associated with recognizable criteria:

(i) The presence of "simultaneous inverse operations". For example, in a closed flow loop with a pump in line, momentum is imparted to the fluid by the pump and is subsequently lost to friction in the piping. Another example is the operation at total reflux of a distillation column.

(ii) Systems with Large Capacitances. A system whose capacity for some physical quantity (such as thermal energy, material or pressure) is so large that its state varies slowly with respect to
the rest of the process can serve as a stationary state during the operating procedure.

In the chapter discussing the modelling issues I described the stationary state as being a sound basis for forming abstractions about the processing units. The justification for this stems from the fact that these stationary states are used to define planning islands, and by forming abstractions around the processing units involved, the specification of the operational state of these islands is facilitated.
2.4 PLAN GENERATION PHASE

The synthesis of operating procedures is carried out in three stages. First, the primitive operations required to carry out the transformation from initial to final states must be identified. Next, the planner must construct a "partial plan" which consists of specifying a partial ordering on the primitive operations. Finally, since some of the operational constraints (such as cost constraints) stated by the user may not be amenable to direct transformation into temporal constraints between goals, a systematic generate and test strategy is needed to ensure the feasibility of a complete operations plan.

2.4.1 Identification of Primitive Operations

The identification of primitive operations is achieved using the means-ends analysis paradigm. This paradigm was described in Chapter 1 of this thesis, and involves a comparison of the initial operating state of the plant with the goal state (or the first planning island, if such an island has been identified). Differences between the two states account for the driving force behind the plan generation process. Thus, for the methodology under consideration here, planning is viewed as a goal-driven process.

An important distinction exists between traditional AI planning domains and the process engineering domain. In traditional means-ends systems, it is necessary to compare all of the goal state variables with the initial state variables, and if a required goal is not already achieved in the initial state, the planner attempts to find a plan operator which will achieve that goal. In a chemical plant, however, the primitive operators, themselves, have an associated operational state. Valves can be open or closed, switches can be on or off and controllers can be active or inactive.

Thus, in order to determine which primitive actions are necessary to carry out a process operations plan, the means-ends analysis algorithm need only compare the initial and goal states of those equipment which are considered to be primitive operational devices with attached primitive operators (e.g. valves, switches and controllers). By taking this approach, the planning program
avoids specifying unnecessary, redundant, goals. A primitive operation "node" is then generated for each of the differences observed. The set of nodes generated by this process is the set of primitive operations which will effect the transformation from the initial state to the required goal state.

We now illustrate the application of this stage of the methodology to the initial and goal states of the reactor change-over problem. The initial and goal states of the primitive operators are shown in Table 2.1. An asterisk against a goal indicates that there is a mismatch between that goal and the initial state. For this example, as is often the case, there is more than one physically attainable state consistent with the stated goals. In all of the candidate goal states, valves 3, 4, 11 and 12 are closed, while valves 1 and 10 are open. There are three possibilities with regard to valves 2 and 9, however: (i) VALVE-2 closed, but VALVE-9 open; (ii) VALVE-2 open, but VALVE-9 closed; and (iii) both VALVE-2 and VALVE-9 closed. For later stages in the catalyst reactivation process, the last of these goal states is the preferred one. However, there is no way for the planner to know this a priori, so all three alternatives must be considered (unless the user states a preference for one of them). Means-ends analysis applied to Table 2.1 yields the following candidate operator sets:

\[
\{(\text{OPEN VALVE-1}), (\text{OPEN VALVE-10}), (\text{CLOSE VALVE-2})\}
\]

\[
\{(\text{OPEN VALVE-1}), (\text{OPEN VALVE-10}), (\text{CLOSE VALVE-9})\}
\]

\[
\{(\text{OPEN VALVE-1}), (\text{OPEN VALVE-10}), (\text{CLOSE VALVE-2}), (\text{CLOSE VALVE-9})\}
\]

Note that the above sets are unordered, and the sequence in which the valve operations are to be carried out is yet to be specified. This is accomplished in the final two stages of the Plan Generation Phase.

2.4.2 Construction of a Partial Plan

I illustrated earlier, the advantages of formulating partial plans of process operations before trying to develop complete operating procedures. In this section the generation of these partially
<table>
<thead>
<tr>
<th>Device</th>
<th>Goal State 1</th>
<th>Goal State 2</th>
<th>Goal State 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>VALVE-1</td>
<td>CLOSED</td>
<td>OPEN*</td>
<td>CLOSED</td>
</tr>
<tr>
<td>VALVE-2</td>
<td>OPEN</td>
<td>OPEN*</td>
<td>CLOSED</td>
</tr>
<tr>
<td>VALVE-3</td>
<td>CLOSED</td>
<td>CLOSED</td>
<td>CLOSED</td>
</tr>
<tr>
<td>VALVE-4</td>
<td>CLOSED</td>
<td>CLOSED</td>
<td>CLOSED</td>
</tr>
<tr>
<td>VALVE-9</td>
<td>OPEN</td>
<td>OPEN*</td>
<td>CLOSED</td>
</tr>
<tr>
<td>VALVE-10</td>
<td>CLOSED</td>
<td>CLOSED</td>
<td>CLOSED</td>
</tr>
<tr>
<td>VALVE-11</td>
<td>CLOSED</td>
<td>CLOSED</td>
<td>CLOSED</td>
</tr>
<tr>
<td>VALVE-12</td>
<td>CLOSED</td>
<td>CLOSED</td>
<td>CLOSED</td>
</tr>
</tbody>
</table>

Table 2.1 Initial and Goal States of Primitive Operators
specified plans is described. The technique used here derives from the constraint-posting paradigm first articulated by Stefik (1981).

As mentioned earlier, operational constraints may be stated at any level of the hierarchy which facilitates their expression. Thus, it must be clearly stated how a constraint stated at a given level of the modelling hierarchy is interpreted in "lower" levels of the hierarchy (where there is a higher degree of detail).

In addition, recall that these operational constraints are stated in terms of the user's goals, and the intermediate states in the plan. A partial plan, however, consists of a partial temporal ordering on the primitive operators. It is necessary, therefore, to describe how the operational constraints are transformed into constraints posted on the planning operators.

**Manipulation of Temporal Constraints** We first consider the temporal operational constraints of the form "GOAL-A must be achieved before achieving GOAL-B". One such constraint was identified in Section 2.3.3, namely:

\[ \{ \text{FLOW(P2(AUG.-REACTOR-2)) > 0} \land \{ \text{X-PRODUCT(P2(AUG.-REACTOR-2)) > 0} \right\} \]

\[ \Rightarrow \{ \text{FLOW(P2(AUGMENTED-REACTOR-1)) = 0} \right\} \]

A temporal constraint at a particular level in the abstraction hierarchy specifies a partial ordering on the goals in the level just below it. This causes the statement of new constraints on this level. Each of these new constraints specifies a partial ordering on the goals still one level lower in the hierarchy. In this manner, a constraint between goals which is stated at any given level is successively transmitted level by level down to the lowest level (level 1) which is made up only of individual processing units. In Figure 2.9, this process is illustrated schematically, where the greyed arrow at level "i", representing the stated constraint "GOAL A before GOAL B", is inherited down to the "(i-1)th" level, and generates new constraints at that level. A directed path from any one goal to any other implies that the first goal is constrained to be achieved before the second. Note that the hierarchy can extend above and below the two levels pictured.

56
Figure 2.9 Downward Transformation of Temporal Constraints
The complete algorithm for downward "inheritance" of temporal constraints is presented in the appendix to this chapter. This algorithm is provably correct, having the following property (proof in appendix):

**Property 1:** (Correctness of the Constraint Inheritance Algorithm)

Let \( C_i \) be a temporal constraint stated between two goals \( A_i \) and \( B_i \), on level "i" of the modelling hierarchy. Under this constraint, \( A_i \) must be achieved before \( B_i \).

Let \( A_{(i-k)} \) and \( B_{(i-k)} \) (\( k > 0 \)) be subgoals, on the \((i-k)^{th}\) level, of \( A_i \) and \( B_i \) respectively.

Then the Algorithm for Downward Inheritance of Temporal Constraints guarantees, upon termination that, for all \((A_{(i-k)}, B_{(i-k)})\), a directed path from \( A_{(i-k)} \) to \( B_{(i-k)} \) exists in the constraint network \( N_{(i-k)} \), and no corresponding path exists from \( B_{(i-k)} \) to \( A_{(i-k)} \).

In addition to systematic inheritance of constraints down the hierarchy of goals, the computer program allows easy detection of direct conflicts between temporal constraints. Conflicts manifest themselves as directed circuits at one or more levels in the goal hierarchy. Since constraints are introduced incrementally, and since we begin the constraint inheritance process with a cycle-free digraph, as each new constraint is added, a fairly straightforward polynomial-time algorithm will detect cycles (see appendix for algorithm). As soon as a cycle is found, the user is warned of its existence and asked to retract the offending constraint.

**Posting of Non-Temporal Constraints:** Certain constraints may not be stated in temporal form. For example, the user may define minimum or maximum allowable temperatures, or forbid dangerous mixtures at certain locations in the plant. In general, it is not obvious how to automatically transform these non-temporal constraints into temporal constraints on operations, which can then be efficiently manipulated in the manner described above. In fact, Chapman's
intractability theorem (described in Chapter 1) seems to imply that this is not possible in a completely domain-independent planner. It is, however, possible to accomplish this transformation for certain types of operational constraints, such as the mixing constraints, by exploiting domain-specific features of the problem.

Transformation of Mixing Constraints into Temporal Constraints: Constraints of the form "Species A and Species B should not come into contact with each other (for safety reasons)" are amenable to direct transformation into temporal constraints. The methodology is described in detail in the next chapter.

Transferring Inter-Goal Constraints onto Primitive Operators. It now remains to be explained how the temporal constraints on goals at the lowest level of the modelling hierarchy are transformed into constraints on the primitive operators. The algorithm for achieving this transformation is outlined in the appendix.

Property 2: (Correctness of Constraint Transformation Algorithm)

The constraint transformation algorithm accepts a network of goals partially ordered by constraints, and generates a constraint network on primitive actions, such that if there exists a path between goal A and goal B (i.e., A must be achieved before B) in the first network, and if OP-A is the primitive action which achieves A, and OP-B the action which accomplishes B, then OP-A and OP-B are labels on nodes in the generated network, and there exists a direct path from the node labelled with OP-A to the node labelled with OP-B.

The property stated above guarantees that constraints stated on goals will be correctly inherited down to the primitive operators which achieve those goals.

Returning to our illustration, for each candidate goal state, the production constraint
identified earlier is propagated downwards using the Downward Inheritance Algorithm, and then transformed into constraints on the primitive operators. The resulting constraint network with the primitive operators as nodes is shown in Figure 2.10 (a). This network represents a partial plan for the valve manipulations which take the first reactor off-line and bring the second into operation.

2.4.3 Synthesis of Complete Plans

If all of the operational constraints in a given problem are represented as strictly temporal relations between goals, then, by Property 2, any complete plan consistent with the generated partial plan is feasible. In cases where at least one of the operational constraints is non-temporal and not a mixing constraint, it is necessary to linearly generate and simulate candidate operating procedures and check for potential violations of these constraints.

The linear plan synthesis is accomplished by first constructing a transition network for the partial plan. A transition network is a directed graph consisting of a single start-node, with in-degree 0, a single end-node, with zero out-degree, and a set of intermediate nodes with in-degree and out-degree > 0. Each edge in the directed graph represents the application of some operator. Nodes represent the state of the system at various stages in the plan, and an operator (edge) connects one state (node) to another iff applying the operator in the first state transforms the operational state of the system to the second state. Once the transition network for a partial plan has been constructed, any path leading from the start-node to the end-node is a candidate plan. The network is searched until a path which does not violate any of the non-temporal constraints is found.

The algorithm for construction of the transition network is given in the appendix, and is shown to be complete, with the following important property.

Property 3: (Completeness of the Transition Network Generation Procedure)

All feasible, complete plans consistent with a given partial plan are contained in the transition network.
Figure 2.10 (a) a Partial Plan, and (b) its Completions

OPEN VALVE-1
CLOSE VALVE-2
CLOSE VALVE-9
OPEN VALVE-10

OP-1 => (OPEN VALVE-1)
OP-2 => (CLOSE VALVE-2)
OP-9 => (CLOSE VALVE-9)
OP-10 => (OPEN VALVE-10)

OP-2
OP-9

OP-1
OP-10

COMPLETION 1
COMPLETION 2
COMPLETION 3
COMPLETION 4

PARTIAL PLAN
The algorithm used here for the generation of transition networks has one more attractive feature. While the network is being constructed, the size of the solution space is simultaneously being computed (at no extra computational cost). Upon termination of the algorithm, the label of the end-node represents the total number of sequences which can be generated from the network. This metric can be used as a basis for deciding what fraction of the remaining sequences should be generated for screening or, equivalently, what type of search (breadth-first, depth-first, etc.) is appropriate.

Applying the generation procedure to the partial plan of Figure 2.10 (a), the four feasible plans shown in Figure 2.10 (b) are generated. For this simple example, since only temporal constraints were involved, the generate and test procedure is actually not necessary. Any plan consistent with the temporal ordering in the constraint network of Figure 2.10 (a) is guaranteed feasible, and we can simply pick the first operating sequence generated. The exhaustive enumeration in this case, was performed purely for illustrative purposes.

2.5 APPLICATION OF THE METHODOLOGY TO THE CASE STUDY

In the previous sections, the various stages of the planning methodology were described and illustrated using a small sub-problem within the overall Catalyst Regeneration Problem presented in Section 2.2.3. In this section we describe the difficulties involved with solving the entire problem, and show how it can be reformulated in a form amenable to the non-linear planning methodology described in this paper.

To begin with, let us examine the catalyst regeneration problem more closely. We notice that the plant actually undergoes a cyclic process, and that the initial and goal states appear to be the same. Clearly, any planner using the means-ends analysis paradigm will not be able to generate a plan for a problem which has identical initial and goal states. (There is no "driving force" to get the planner started.)
In reality, however, the initial and goal states in this problem are not identical. In the initial state, the catalyst in the first reactor has been depleted in chlorine, the crystals have grown to a sub-optimal size, and coke has been deposited on it. The modelling framework used to represent the operational state of the plant, is not able to capture these differences, thus leading to the apparently equivalent initial and goal states. Clearly this represents a limitation on the part of the modelling structure used, and it could be suggested that the framework should be extended to include such plant-specific state specifications as "coke deposit on catalyst", and descriptions of size distribution of the crystals. While this approach would enable the planner to identify the differences between the goal and initial states, it suffers from the following weakness:

(i) The modelling framework will not be generally applicable, and a single structure of models will no longer be suitable for several planning applications.

(ii) The overhead involved in identifying the modelling needs for a specific application is great. For a complicated problem such as the catalyst regeneration problem, it would be necessary for the human user to, in some sense, "solve the problem" him/herself before he/she would be able to define a suitable modelling structure.

There is, however, another viable option. Retaining the general purpose modelling framework discussed in the first chapter, the cyclic planning problem is stated as several individual planning problems. Each of these planning problems is characterized by its own initial and goal states, operational constraints and planning islands. This reformulation offers a practical solution to such difficult planning problems, given the inherent trade-offs involved between setup-overhead and automation.

The cyclic catalyst regeneration problem is broken up into the following sequence of planning problems:
(i) Take reactor 1 off-line and bring reactor-2 on-line (reactor change-over)

(ii) Pressurize reactor-1 with hydrogen

(iii) Circulate inert gas through reactor-1

(iv) Chlorinate catalyst in reactor-1

(v) Remove coke deposits by circulating natural gas in reactor-1

(vi) Pressurize reactor-1 with hydrogen

(vii) Change-over from reactor-2 to reactor-1.

Each of these individual sub-plans can be easily described using the modelling structure developed in Chapter 1 of this thesis. Planning islands are currently identified by hand, but following the approach outlined in Section 2.3.4 can, in the future, be automatically generated. In this example, the planning islands involve only purge goals, which are easily identified by the computer.

APPENDIX: Algorithms, Properties and Proofs

Piggin Algol:

Algorithms in this appendix are expressed in a language known as Piggin Algol. This language is an accepted standard for describing combinatorial algorithms, and is described elsewhere (Papadimitriou and Steiglitz, 1982). For readers familiar with PASCAL, ALGOL or PL/I, algorithms in this language should be trivial to understand. Piggin Algol should be viewed as an informal notation, rather than a high-level programing language. Miscellaneous statements which do not conform to the stated syntax of the language are allowed, provided that they are "readable and reasonably unambiguous".

Definition 1 (Maximal Nodes): In a digraph D(V,E) which is free of directed cycles, the following statements are equivalent:

(i) The node v ∈ V has the property that ∀u ∈ V, u ≠ v, such that u and v are connected, adding the edge v→u to the digraph introduces a directed cycle.

(ii) The node v ∈ V is MAXIMAL.
**Definition 2 (Minimal Nodes):** In a digraph D(V,E) which is free of directed cycles, the following statements are equivalent:

(i) The node v ∈ V has the property that ∀u ∈ V, u ≠ v, such that u and v are connected, adding the edge u→v to the digraph introduces a directed cycle.

(ii) The node v ∈ V is MINIMAL.

**Algorithm for Downward Inheritance of Constraints:**

**Input:** (i) A hierarchical structure of goals H(L). Each level "1" in the set L of the hierarchy is a network N₁(V₁,E₁). The goals on a given hierarchical level appear as nodes in the set V₁ corresponding to that level, while the precedence constraints between goals are represented as edges in E₁. The lowest level in the hierarchy (most detailed information) is labelled level 1.

(ii) A Temporal Precedence Constraint at Level "i" in the Hierarchy. This precedence constraint consists of an ordered pair of operational goals, a pre-condition-node and a post-condition-node.

**Output:** A hierarchical structure of goals H(L) which contains all previously stated constraints between goals plus the additional constraints (on levels below level "i") induced by the new temporal constraint in level "i".

**Note:** The "subgoals" method accepts an abstract goal as argument and returns the set of goals which comprise the abstract goal in the level just below that of the abstract goal.

```
begin
  level:= i; constraint-list:= {constraint};
  while level>1 do
  begin
    new-constraints:= {};
    for every constraint c in constraint-list do
    begin
      pre-goal:= pre-condition-node[c];
      post-goal:= post-condition-node[c];
    end
  end
end
```
pre-subgoals := subgoals[pre-goal];
post-subgoals := subgoals[post-goal];
for every node p in maximal-nodes[pre-subgoals] do
    begin for every node q in minimal-nodes[post-subgoals] do
        begin e := edge(p,q), E_1 := E_1 || (edge(e));
            new-constraints := new-constraints || (edge(e));
        (comment: the symbol "||" stands for concatenation. concatenation of two ordered sets results in their union, with the elements of the first set ordered before those of the second)
        end
    end
end;
constraint-list := new-constraints;
level := level-1
end
end

procedure maximal-nodes(list-of-nodes)
(comment: finds all maximal nodes [according to the definition above] in the list of nodes supplied)
begin maximal-node-list := {};
    for every node n in list-of-nodes do
        begin flag := true;
            for every node m in maximal-node-list do
                begin if there exists an edge from m to n then delete[n, maximal-node-list]
                (comment: delete[v,V] deletes the node v from the ordered set V)
                else if flag = t and there exists an edge from n to m then flag := false;
            end;
            if flag = t then maximal-node-list := maximal-node-list || {n}
        end;
end

Note: The algorithm for finding minimal nodes is symmetric to the one for
maximal nodes. The only difference is that the words "from" and "to" (underlined) are interchanged.

Property 1: (Correctness of the Constraint Inheritance Algorithm)

Let $C_i$ be a temporal constraint stated between two goals $A_i$ and $B_i$, on level "i" of the modelling hierarchy. Under this constraint, $A_i$ must be achieved before $B_i$.

Let $A_{(i-k)}$ and $B_{(i-k)}$ ($k > 0$) be subgoals, on the $(i-k)^{th}$ level, of $A_i$ and $B_i$ respectively. Then the Algorithm for Downward Inheritance of Temporal Constraints guarantees, upon termination that, for all $A_{(i-k)}$, $B_{(i-k)}$, a directed path from $A_{(i-k)}$ to $B_{(i-k)}$ exists in the network $N_{(i-k)}$, and no corresponding path from $B_{(i-k)}$ to $A_{(i-k)}$ exists.

Proof: (By Contradiction) Consider a constraint introduced in the $i^{th}$ level of the hierarchy, between $A_i$ and $B_i$. The constraint inheritance algorithm creates an edge from each of the maximal sub-goals of $A_i$ in the $(i-1)^{th}$ level, to each of the minimal sub-goals of $B_i$. Assume that there exists a pair of nodes $A_{(i-1)}$ and $B_{(i-1)}$ that are not connected. From the definition of a maximal node, it is clear that in a given set of nodes, any individual node is either maximal, or a path exists from that node to some maximal node in the set. Thus $A_{(i-1)}$ is either maximal, or a path exists from $A_{(i-1)}$ to some $A_{(i-1)}'$, which is maximal. Similarly, $B_{(i-1)}$ is either minimal, or a path exists from some minimal node $B_{(i-1)}'$ to $B_{(i-1)}$. But all of the $A_{(i-1)}'$ are connected to all of the $B_{(i-1)}'$. (Contradiction)

Since there are no directed cycles in the networks, no path can exist from any $B_{(i-1)}$ to any $A_{(i-1)}$. This proves the result between the $i^{th}$ and the $(i-1)^{th}$ levels. The reasoning can be extended inductively down to the $(i-k)^{th}$ level. Q.E.D.

Note: A polynomial-time algorithm for finding the strongly-connected components of a directed graph is available (Even, 1979). A slightly modified version of this algorithm is used to detect the existence of a directed cycle.

Algorithm for Transforming Constraints from the Lowest Level of the Goal Hierarchy into Constraints on Primitive Operator Nodes:

Input: A constraint-network $C(V,E)$ which is a copy of the goal network at the bottom-most level of the goal hierarchy.
Output: A network whose nodes are labelled with primitive operators and whose edges represent precedence constraints on the operators. (An edge $u \rightarrow v$ implies that the operation in label[$u$] must be carried out before the operation in label[$v$].)

begin
for each node $v \in V$ do
  if the goal associated with node $v$ can be directly achieved by a primitive operator op, then label[$v$] := op
  else
    begin
      in-edges[$v$] := the set of in-edges of node $v$;
      out-edges[$v$] := the set of out-edges of node $v$;
      for every $e_1 \in \text{in-edges}[v]$ and every $e_2 \in \text{out-edges}[v]$ do
        begin
          pre-nodes[$v$] := pre-nodes[$v$] || from-node[$e_1$];
          post-nodes[$v$] := post-nodes[$v$] || to-node[$e_2$];
          delete($e_1$, E); delete($e_2$, E);
        end;
      end;

      (Comment. At this point pre-nodes[$v$] and post-nodes[$v$] contain the list of goal nodes just "previous to" and just "after" node $v$, and all edges incident to $v$ have been deleted from the network $C(V,E)$)

      for every pair $(m, n)$ st. $m \in \text{pre-nodes}[v]$ & $n \in \text{post-nodes}[v]$ do
        e := (m, n), if edge(e) is not $\in E$ then $E := E || \text{edge}(e)$
      end
    end
end

Property 2: (Correctness of Constraint Transformation Algorithm)

The constraint transformation algorithm accepts a network of goals partially ordered by constraints, and generates a constraint network on primitive actions, such that if there exists a path between goal A and goal B (i.e., A must be achieved before B) in the first network, and if OP-A is the primitive action which achieves A (and OP-B, the action which accomplishes B), then OP-A and OP-B a.e labels on nodes in the generated network, and there exists a directed path from the node labelled OP-A to the node labelled OP-B.
Proof: The constraint transformation algorithm copies over the initial goal network, and deletes nodes iff they are not directly achieved by a primitive operator. Also, if a goal node is not deleted (because it is directly achieved by some operator OP), then the label of this node is set to OP. This proves the statement above: "if there exists a path between goal A and goal B (i.e., A must be achieved before B) in the first network, and if OP-A is the primitive action which achieves A (and OP-B, the action which accomplishes B), then OP-A and OP-B are labels on nodes in the generated network".

It now remains to be shown that there exists a directed path from the node labelled OP-A to the node labelled OP-B. In the original goal constraint network G(V,E), such a path existed. During execution of the algorithm, whenever a node was removed, the edges in the node were also removed, and a new set of edges was added to the network. This new set of edges consisted of all possible matchings between the pre-nodes and the post-nodes of the node which was deleted. Thus, if two nodes u and v in the original G(V,E) were connected by a path which passed through the node just deleted, then an edge which "bridges" the gap caused by deleting the node will be added to the network. Q.E.D.

Algorithm for Construction of Transition Networks.

**Input:** A partial plan. (A digraph D(V,E) with operators for node. An edge leading from one node to another indicates that the first operation must be performed before the second.)

**Output:** A transition network T(V',E',s,t) (where s ∈ V' is the start-node, and t ∈ V' is the end node).

begin
operators:= V (comment: the set of nodes in the partial plan);
s:= create-node[], label[s]:= 1, candidate-ops[s]:= operators;
V':= {s}, current-level:= {s};
while candidate-ops[first[current-level]] ≠ {} do
begin
new-nodes:= {};
for every node c in current-level do
  for every node op in V do

67
if applicable?[op,c] = "yes" then
begin n:= create-node[];
    e:= create-edge[c,n], label[e]:= op, E':= E'\{e\};
candidate-ops[n]:= delete[op, \{candidate-ops[c]\}];
    new-nodes:= new-nodes\{n\};
end;
new-nodes:= merge-redundant-nodes[new-nodes];
for every node n in new-nodes do
    begin V':= V'\{n\};
      label[n]:= sum over all labels of previous-nodes[n]
      (comment: previous-nodes[n] are located by back-tracking along the edges leading into n)
    end;
current-level:= new-nodes
end;
t:= first[current-level]
end

procedure applicable?(op,st)
(comment: checks to see if operator "op" is applicable in state "st")
if op \in\ minimal-nodes[candidate-ops[st]] then "yes" else "no"

procedure merge-redundant-nodes(nodes)
begin
    non-redundant-nodes:= \{
for every node n in nodes do
    if \exists v \in non-redundant-nodes s.t. candidate-ops[v]=candidate-ops[n]
        then in-edges[v]:= in-edges[v]\{in-edges[n]\}
        else non-redundant-nodes:= non-redundant-nodes\{n\};
return non-redundant-nodes
end
Property 3: (Completeness of the Transition Network Generation Procedure)
All feasible, complete plans consistent with a given partial plan are contained in the transition network.

Proof: (By Contradiction) In order to prove that a given transition network, \( T(V,E) \) is complete, we must show that no feasible plan is omitted. Let us assume that some plan \( p \) which is consistent with the constraints of the partial plan is not contained in \( T(V,E) \). The first state on \( p \) is the start-state (no operations achieved) thus the first state on \( p \) is definitely contained in \( T(V,E) \) namely, the start-node. Searching the paths in \( T(V,E) \) in a breadth-first manner, let \( s_i \) be the first state of \( p \) which is not on at least one of the paths in the network up to the current level. (Such a state must exist, since \( p \) is not contained in \( T(V,E) \). The state \( s_{(i-1)} \) just previous to \( s_i \) in \( p \) is a feasible state contained in \( T(V,E) \), since \( s_i \) is the first "unreachable" state discovered. However, if \( s_{(i-1)} \) is a feasible state in \( T(V,E) \) then, during construction of the transition network, an edge corresponding to each operator that is applicable in \( s_{(i-1)} \) is created and added to the list of out-edges of the node \( s_{(i-1)} \). The operator in \( p \) which achieves the transformation from \( s_{(i-1)} \) to \( s_i \) must be one of these operators (since \( p \) is consistent with the partial plan), thus there must exist some edge in the transition network leading from \( s_{(i-1)} \) to \( s_i \). (Contradiction)
CHAPTER 3

PLANNING IN THE PRESENCE OF QUALITATIVE, MIXING CONSTRAINTS
3.1 INTRODUCTION

In the first two chapters of this thesis a foundation was laid for a theory and methodology for the a priori planning of process operations. An extensive review of previous work both in the area of chemical engineering and in the area of artificial intelligence, was provided, and the importance of developing provably correct, complete and efficient planners was emphasized. Towards this end, a general methodology and framework for the so-called non-linear planning of process operations was developed although the methodology, as presented, was only partially non-linear and required a linear search procedure to complete the planning process. At present, only certain types of operating constraints can be efficiently handled in a non-linear manner, while the other constraints must be handled by a generate-and-test strategy. In the second chapter, a methodology for handling hierarchical, temporal constraints in a non-linear manner was presented.

3.1.1 Binary, Qualitative Mixing Constraints

A very important class of constraints that commonly arise in typical chemical engineering planning problems is the so-called class of mixing constraints. Mixing constraints occur whenever there are two or more chemical species in a plant which form a potentially dangerous or undesirable mixture. This type of constraint is of particular interest in situations where safety considerations are of primary importance. Recognizing this fact, several researchers have developed a variety of methodologies for handling mixing constraints (Rivas and Rudd, 1974; O'Shima, 1982; Fusillo and Powers, 1988). However, all of the methodologies proposed in the past have been linear in nature in that they make limited, if any, use of existing constraints during the plan generation process. These methodologies do not make effective use of knowledge about the structure of a chemical plant, and engineering knowledge about its operation which may be used to constrain the search for feasible solutions. Thus, there is a need for an efficient non-linear planning methodology for handling situations where mixing constraints occur.

In its most general form, the mixing constraint involves the specification of a set of chemical species, and composition ranges for these species within which a dangerous situation
might occur. Very often, however, operations planners encounter a specific type of mixing constraint, namely, the binary, qualitative mixing constraint. This class of constraint is stated in the form of an unordered pair of chemical species. A constraint such as \((x_i,x_j)\) implies that the components \(x_i\) and \(x_j\) should never be present at the same time in the same location within the chemical plant. Unlike the more general form of the mixing constraint, no allowable range of concentrations is specified - the species are simply forbidden to come into contact with one another. In this chapter we will show that a non-linear planning methodology exists for handling binary, qualitative mixing constraints.

3.1.2 Outline of the Chapter

This chapter, the third in a series on the planning of operating procedures, extends the non-linear planning methodology, which originally only handled purely temporal constraints, to cases in which non-temporal, qualitative mixing constraints are involved. The chapter is organized as follows.

(a) Previous work on planning to avoid mixing constraint violations is critiqued in Section 3.2 and the requirements of a practical methodology are outlined.

(b) Section 3.3 contains the outline of a methodology for automated synthesis of operating plans in the face of binary, qualitative mixing constraints.

(c) Section 3.4 describes in more detail the portion of the methodology which deals with the identification of possible constraint violations and the transformation of the mixing constraints into the temporal constraints which were dealt with in chapter 2.

(d) Subsequently we describe methods for automatic generation of planning islands which avoid these potential constraint violations, and a method for non-linear plan generation is proposed.

(e) Finally, Sections 3.6 and 3.7 focus on the application of the methodology to two industrial case-studies.
3.2 PREVIOUS WORK

An extensive review of previous work in the area of operations planning appears in the first chapter in this thesis. Here we briefly mention only those planners which focus on the handling of binary, qualitative mixing constraints.

3.2.1 Synthesis of Failure-Safe Operating Procedures

The first methodology for the synthesis of operating plans proposed by researchers in the chemical engineering domain (Rivas and Rudd, 1974) was developed primarily to handle mixing constraints. While the method successfully generated operating procedures for selected industrial examples, a large portion of the planning decision-making often had to be carried out by the human. In addition, the plan generation procedure was linear in nature and hence ran the risk of being inefficient. In several instances, the user had to manually supply intermediate states to guide the planner. The work of Rivas and Rudd was extended by O'Shima (1982), but the limitations described above still remained.

3.2.2 Planning of Purge Operations

Fusillo and Powers (1988) proposed a different approach to the problem of handling mixing constraints. Instead of concentrating on the detailed valve manipulations, they focused on the "top-level" goals which a human operator formulates in his or her mind in order to solve a given planning problem. An example of such a goal might be the statement "purge hydrogen from reactor 3 to vent 27 using nitrogen". Using this approach, the planner generated plans which were more easily understood (and therefore more easily debugged) by the human operator. However, the methodology still suffered from some limitations:

(i) the process by which the planner would translate the top-level goals to the more detailed valve manipulations was not obvious,

(ii) the method of plan generation used a brute-force, depth-first linear search which is inefficient and
(iii) the modelling approach used so-called global variables, so differences in temperatures, pressures, concentrations, etc. between pieces of equipment could not be represented.

As will be demonstrated in the following sections, methodologies exist which effectively plan process operations in a non-linear manner (i.e. without having to resort to search) and no simplification in the modelling equations need be made. In addition, the planning can be automated right down to the detailed level of valve manipulations, and does not need the human operator to achieve the transformation from a high-level operating procedure to a low-level one.

3.3 OUTLINE OF THE METHODOLOGY

The planning methodology for handling mixing constraints is an augmentation of the non-linear planning methodology, described in the first two chapters, and is composed of two distinct phases: (a) the Problem Formulation Phase and (b) the Plan Generation Phase. The extension to include Mixing Constraint problems involves the addition of two stages in the Plan Generation Phase and these modifications will be described in detail. The Plan Generation Phase can be broken down into a set of tasks as described below:

A. Identification of Primitive Actions

First, the primitive operations required to carry out the transformation from initial to final states must be identified. In our approach we have adopted a traditional means-ends analysis for this task.

B. Transformation of Mixing Constraints into Temporal Constraints

The next stage, which is specifically geared towards the situations where qualitative mixing constraints arise, involves the transformation of these constraints into temporal constraints on the primitive operators. This transformation is accomplished as outlined below:

(i) Construction of Influence Graphs for Dangerous Species. A graphical construction known as the "influence graph" is generated for each potentially dangerous species.
(ii) Identification of Potential Constraint Violations: Using the influence graphs for each pair of dangerous species, the program identifies potential sources of constraint violation.

(iii) Generation of Temporal Constraints between Primitive Operators: If a potential constraint violation is detected, the computer uses a series of algorithms to state precedence constraints between primitive valve manipulations which will disallow the mixing of the dangerous species in the steady state.

C. Generation of Evacuation/Purge Planning Islands

After the temporal constraints have been generated, the methodology guarantees that, at any given intermediate state, no mixing constraint violation will occur. However, we must also cover the situation where a constraint violation occurs during the transition between two intermediate states. This is achieved by the formulation of evacuation or purge planning islands, and is based on the philosophy of generating purge goals outlined by Fusillo and Powers (Fusillo and Powers, 1988) coupled with efficient graph algorithms which take into account the distributed character of the chemical plant.

D. Construction of a Partial Plan

The fourth stage involves the construction of a "partial plan". This is a partial ordering of the operators which stems from the operational constraints placed on the operating plan. It is analogous to the constraint-posting philosophy of nonlinear planning described by Chapman (1985).

E. Synthesis of Complete, Feasible Plans

At present, the non-linear planning methodology only manipulates operational constraints which are temporal in nature (aside from the mixing constraints which are transformed into temporal constraints). Some operational constraints stated by the user may not be amenable to direct transformation into temporal constraints and a linear, generate and test strategy based on the partial plan generated in the previous step must be used.
It should be noted here that the proposed methodology for handling mixing constraints is *conservative* in nature. In other words, the program will never generate a plan which could *possibly* lead to the violation of any of the mixing constraints, but some feasible sequences may be missed. However, this is not a serious practical limitation, as mixing constraints generally stem from safety operating constraints which are of the "hard" variety, and in most cases, it is preferable to provide a safe operating procedure within a reasonable amount of time, even at the expense of achieving sub-optimal operation. The alternative to this approach would be to use some form of linear search procedure. Such a procedure could guarantee that all possible sequences would be investigated but, as was pointed out earlier, this is generally infeasible for reasonably sized problems. We have therefore chosen to sacrifice some degree of *completeness* in order to achieve practical efficiency.
3.4 TRANSFORMATION OF MIXING CONSTRAINTS

The general philosophy for handling mixing constraints involves the transformation from their typical form, namely, "the pair of species \(x_i, x_j\) should not come into contact with each other" into temporal constraints between low-level operations (such as valve manipulations or controller set-point changes). As described in the earlier papers, these temporal constraints can be handled efficiently and, if the transformation of the mixing constraints into temporal constraints can be accomplished correctly and without unreasonable computational expense, then a useful automation capability has been gained.

In this section we describe in detail the process by which the binary, qualitative mixing constraints are transformed into temporal constraints between primitive operators. The detailed description of the stages of the methodology which precede this phase appears in Chapter 2 and will not be discussed further.

3.4.1 Construction of the Influence Graphs

The first stage of the constraint transformation methodology involves the construction of the so-called Influence Graphs. These graphs are a representation of the possible extent to which a given chemical species is present within the flowsheet in a given operational state. They are a graphical description of the "influence" of a particular species, in a given operating state. Each Influence Graph is associated with a particular chemical species as well as a particular operating state. The Influence Graph forms the basis of the next stage in the methodology where the computer attempts to check for possible constraint violations.

The procedure for constructing an Influence Graph begins with a directed-graph representation of the topology of the chemical process, at a given state. All state information regarding temperatures and concentrations (beyond qualitative "+" or "0" values) is temporarily suppressed, and only pressure information for the sources and sinks of material is retained. In any given piece of equipment, if the direction of the bulk flow is known at the time when the Influence Graph is being constructed, then a directed edge in the Topological Graph depicts this direction. If
the direction of the flow has not as yet been computed, directed edges are introduced in both
directions.

**Procedure for Constructing an Influence Graph**

With the topological graph as input, the construction of the Influence Graph for component
\( x_i \) in the state \( L_p \) proceeds as follows:

**STAGE 1:**

1. Locate all sources \( s_{ik} \) of component \( x_i \).
2. Let \( N_i \) be \( |S_i| \), where \( S_i \) is the set of all \( s_{ik} \). Create a new node \( s'_i \) to function as a
   "super source" of component \( x_i \).
3. For each pair of super-source/source vertices \( (s'_i, s_{ik}) \), \( k = 1, 2 \ldots N_i \), add a directed
   edge from \( s'_i \) to \( s_{ik} \).
4. Remove all edges that are immediately adjacent to closed valves.

**STAGE 2:**

1. Perform the labelling algorithm, DEFINE-INFLUENCE (see Appendix) which is of
time complexity \( O(|E|+N_i) \) (proof of complexity in Appendix), \( E \) being the set of edges in
the topological graph. This algorithm identifies all of the process flows which may contain
the chemical species \( x_i \)

One can then prove the following theorem:

**Theorem 1:** If there exists a directed path from the super source \( s'_i \) to any given node, \( v \), then
the label-set of that node contains the label \( x_i \).

Proof of Theorem 1 is given in the appendix.
3.4.2 Identification of Potential Constraint Violations.

In his definitive work on planning, Chapman introduces the concept of the Clobberer and the White Knight. The Clobberer is a plan operator which negates, prevents or undoes one or more of the planning goals. A White Knight is a plan operator which rectifies, or prevents, any "damage" the Clobberer might do. The success of any non-linear planning methodology lies in the ability to identify these Clobberers and White Knights without having to resort to a generate-and-test approach. Once these operators have been identified, the planner must ensure that whenever a Clobberer is used in the plan, an accompanying White Knight is also present.

In the case of the operations planning problem, the problem of detecting Clobberers reduces to the detection of potential constraint violations. In this section, we describe how this may be done in an efficient (polynomial time) manner for problems involving mixing constraints. Later we shall show how automatically generated planning islands can function as White Knights that prevent the planner from generating plans which run the risk of violating one or more of the mixing constraints.

Outline of the methodology for identification of potential constraint violations: Potential violations of mixing constraints are determined in the following manner

STAGE 1: (Detecting constraint violations in Initial and Goal States)

(1) For each dangerous component (i.e., any component which appears in one or more of the mixing constraints) construct the Influence Graph for both the Initial and Goal States.

(2) Determine whether there are any potential mixing constraint violations in these two states. It should be clear from Theorem 1 that if, in a particular Influence Graph, any given node is marked with a pair of species which form a forbidden mixture, then a constraint violation could occur, since there is a possibility that both species may be present at the same time in the same place. The algorithm CHECK-LABELS described in the appendix achieves this in $O(|C|^2|V|)$ time, where $C$ is the set of mixing constraints for the particular planning problem.
Since the Initial State is presumed to be a known feasible operating state, it is generally not the case that it will contain potential constraint violations, and the algorithm is run purely for the purposes of "debugging" any errors the user may have made in the Initial State input. It is very possible, however, that the user of the program may specify an infeasible goal state. In such cases, the specification of the goal state will have to be altered by the user in order to eliminate the conflict.

By looking for a "Minimal Separation Valve Set", (the smallest number of valves which, when closed would prevent the mixing of the two dangerous species) and allowing the user to modify this set interactively, the program can assist the user in defining a consistent goal state. An algorithm for efficiently determining a Minimal Separation Valve Set is described later on in this section.

Having taken care of the possibility of constraint violations occurring in the initial or goal states (for which the Influence Graphs have been constructed), we must also anticipate that some constraint violation might occur in one of the intermediate states. The second stage of the constraint detection process addresses this problem.

STAGE 2: (Detecting potential constraint violations in the intermediate states)

(1) Construct the "worst-case" Topological Graph. This "fictitious" Topological Graph, as the name implies, is intended to simulate the worst possible scenario in terms of potential mixing constraint violations. This is achieved as follows:

(1.1) Identify all valves which must be kept open in the final state, but which are closed in the initial state.

(1.2) A copy of the Initial State Topological Graph is made, and all the valves which must be opened in the goal state are marked as being opened. This modified Initial State Topological Graph represents the worst case situation, as it has the maximum potential for dangerous species to come in contact with one another.
(2) Construct the Influence Graph for each dangerous component on the Worst Case Topological Graph.

(3) If there are no constraint violations in this Influence Graph, then there is no danger of a potential constraint violation. If, on the other hand, any of the nodes is labelled with a pair of dangerous species, then there is a danger that, for this particular node, in some intermediate state, these two species might mix causing, for example, an explosion.

3.4.3 Generation of Temporal Constraints on Primitive Operations

If the program has determined that there is a possibility that there will be a constraint violation at a given location, steps must be taken to prevent this situation from occurring. As alluded to earlier, this is analogous to searching for one of Chapman's "White Knights" to alleviate the difficulties caused by the "Clobberers", namely the valves which were opened in the "worst-case" intermediate state. In order to do this, the program tries to determine a set of valves which, when closed, will prevent the species from mixing. After this has been done, temporal constraints are generated which specify that these "separating" valves should be closed before the valves which lead to a constraint violation are opened. Note that such a Separation Valve Set (SVS) must exist, since we have already (presumably) determined that the goal state is feasible. The complete methodology for determining a suitable SVS is outlined below:

STAGE 1: (Determine a Minimal Separation Valve Set).

(1) Construct the Reduced Valve Network corresponding to the Topological Graph. For every node in the Topological Graph do the following:

(1.1) If the current node represents a VALVE, a sink or a "super source", do nothing.

(1.2) If the current node represents any other type of processing equipment, do the following:

(1.2.1) Delete the current node from the Topological Graph.
(1.2.2) Let U be the set of nodes incident to the in-edges of the current node.

(1.2.3) Let V be the set of nodes incident to the out-edges of the current node.

(1.2.4) For every pair \((u, v)\) where \(u\) is in U and \(v\) is in V, create a directed edge \((u, v)\) and add it to the Topological Graph.

In the transformed Topological Graph, the only nodes are valves, sinks and "supersources" and an edge leads from one node to another iff there is a path connecting the corresponding nodes in the topological graph, and no other valve, sink or source lies on this path. This transformation is demonstrated graphically in Figure 3.1.

(2) We now replace all directed edges with undirected edges. The problem of finding a Minimal Separation Valve Set (MSVS) now reduces to the problem of determining a vertex separator (see appendix for definition) in the Reduced Valve Network for the pair of supersources for the dangerous species under consideration. A polynomial-time \((O(|V|^2 \cdot |E|))\) algorithm exists for determining an MSVS (Even, 1979) and is described in the appendix.

The Minimal Separation Valve set may not be an Acceptable Separation Valve Set for the given planning problem. This is because it is possible that some valve in the MSVS is required to be open in the goal state (to achieve a particular operating goal), or the user may simply have some preference regarding the state of some of the valves. However, from a heuristic standpoint, the Minimal Set is a good starting point for evolving into an ASVS, since closing the MSVS upsets the flow of species in the intermediate topological graph only "minimally". The procedure for evolving from an MSVS to an ASVS is described in Stage 2.

STAGE 2: (Determine an Acceptable Separation Valve Set)

(1) Let the current Separation Valve Set (SVS) of valves be the MSVS already determined.
Figure 3.1: Transformation of a Topological Graph into a Reduced Valve Network
(a) The original Topological Graph
(b) The intermediate stage: all nodes which are not valves, sources or sinks are removed; greyed edges are removed and replaced with the black edges shown
(c) The final Reduced Valve Network
(2) Identify the valves (if any) in the current SVS which must be open in the goal state. If no such valve exists, then the current SVS is feasible. Go to (3). If, on the other hand, there are such valves, mark them as "unsatisfactory" and proceed to step (4).

(3) Present the MSVS graphically to the user (by highlighting the valves on the flowsheet, for example). Ask user to specify any valves which he or she prefers to keep open. Mark all such valves as "unsatisfactory". If there are no unsatisfactory valves, return the current SVS as the ASVS.

(4) "Close" all "satisfactory" valves in the Reduced Valve Network. For each "unsatisfactory" valve, do the following:

   (4.1) Find an MSVS which separates the valve from each of the two supersources under consideration. Replace the "unsatisfactory" valve in the current SVS with one of the two new MSVS's which have just been determined. The choice between which of the two sets to use can be made in the following manner: ask the user for any preference he or she might have, or investigate both alternatives. In investigating the alternatives, the following heuristic is used: choose the new MSVS that has the minimum number of nodes which will be marked as "unsatisfactory" in step (3).

(5) Go to step (3).

Once an Acceptable Separation Valve Set has been found, the final stage of the constraint transformation can be accomplished as follows:

STAGE 3: (Posting of temporal constraints between valve manipulations)

(1) In the Goal State, constrain all valves in the ASVS to be closed.

(2) Temporal constraints of the form described in Part II of this series are generated as follows:

   (2.1) Let P be the set of valves in the Initial State which must be opened to achieve the Goal State. This set of valves can be easily identified by simply comparing valve states in the Initial
and Goal States. (The comparison is especially easy in the new implementation of the software, as the different states of each equipment are saved in the same data-structure.)

(2.2) For every pair \((p, q)\) such that \(p\) is an element of \(P\) and \(q\) is an element of \(ASVS\), the following temporal constraint is generated:

\[
\text{CLOSE}(q) \text{ before OPEN}(p)
\]

At the end of this stage, the planning algorithm is ready to move into the next section of the planning methodology, namely the generation of planning islands.

3.5 GENERATION OF EVACUATION/PURGE PLANNING ISLANDS

Once a separation valve set for each pair of dangerous species has been identified, the temporal constraints will guarantee that, in the steady-state, no constraint violation can occur. However, it is often the case that in the transition from one state to another, dangerous species may come in contact with each other. For example, if oxygen must flow through a pipe that once contained hydrogen, there is a possibility that the two will come in contact even if the hydrogen inlet valve upstream is closed before the oxygen is allowed to enter. To avoid this, the hydrogen must either be purged or evacuated before allowing the oxygen to flow.

In general, if we intersect the Influence Graph of one component in the initial state with that of its dangerous counterpart in the goal state, wherever there is an intersection, we should worry about purging (or evacuating) one of the components before admitting the other into that section of the flowsheet.

Fusillo and Powers have pointed out (Fusillo and Powers, 1988) that when purging is to be done, the following must be selected:

(i) A purgative -- a fluid that pushes out and/or dilutes the purged component.

(ii) A method -- the manner in which the purge operation is accomplished (e.g., a system can be swept through with a gas, evacuated using a pump, etc.)

(iii) The destination -- the location to which the purged material is to be removed.
In their paper they list several heuristics based on such factors as the phase of the material to be purged as well as its chemistry, which aid in the selection of the first two items. However, due to the lumped nature of their model of the chemical process, they did not provide satisfactory methods for deciding the route by which the component would be purged through the complex network which comprises a typical chemical plant.

In this methodology, the emphasis has been on providing support for the last of these three tasks. We assume that the purge will be done using the first available inert gas and no attempt is made here to optimize this selection. It is a relatively straightforward task to include the heuristics presented by Fusillo and Powers in the choice of the purgative. In addition, we have not tried to include the heuristics for choosing the type of operation to be used and we focus solely on the sweep method.

The destination and route for the purge are decided upon using graph algorithms which seek out paths within the topological graph. The paths provide a means by which the purgative may be introduced into the area where the species to be purged is present, and provide a means for pushing the species out of a vent. The destination and route finding is accomplished in the following manner:

STAGE 1: (Path and destination finding).

(1) Identify all locations at which the species to be purged is located. (This is done by scanning the labels on the Influence Graphs.)

(2) For each node where the dangerous species is present in the Initial State but not in the Goal State, find a path leading from the supersource of the purgative to the node, and another leading from the node to the sink. Ensure that all valves along this path are open. This is accomplished using the Find-Path algorithm described in Papadimitriou and Steiglitz (1982). The time complexity of this algorithm is O(|E|+N_i) for each species x_i.
Once the routes for the purge have been found, a planning island is introduced to guide the planner through the purge operation before attempting the reach the Goal State. This intermediate planning island is automatically generated in the following manner:

STAGE 2: (Generation of the purge planning island)

(1) Create an intermediate state in which all of the valves along the purge paths are constrained to be OPEN.

(2) Place this Intermediate State in between the Initial State and the Goal State, and constrain the planner to achieve the goals in the Intermediate State before opening the valves in the Initial State which were earlier identified as being open in the Goal State.

Once the planning islands have been established as above, the problem is effectively broken down into two problems: the first in which the planner treats the planning island as a pseudo-Goal State, and the second in which the planner treats the planning island as the Initial State, and plans to achieve the Goal State once this "Initial State" has been achieved.
3.6 INDUSTRIAL CASE STUDY: CATALYST REGENERATION

The methodology for planning process operations described above is best illustrated within the context of an industrial example. We describe, here, a system first analyzed by Rivas (Rivas, 1973) and a simplified formulation of which was used in the first two papers of this series. In this paper, we attack the problem as initially formulated by Rivas and Rudd (1974) using the techniques described in this series of papers.

3.6.1 Description of the Problem

The chemical process consists of the powerforming reactors of a petroleum refinery, along with the accompanying catalyst regeneration system. During operation of the plant, the activity of the catalyst in the reactor (chloroplatinic acid on an alumina base) gradually decreases due to the following processes: (i) coke formation on the catalyst, (ii) reduction in the chloride content, and (iii) size growth of the platinum crystals. Regeneration of the catalyst involves the following operations:

(i) Coke removal. Removal of accumulated coke is accomplished by injecting inert gas containing low concentrations of oxygen into the reactor at temperatures sufficiently high to initiate and sustain combustion. (The inert gas is generated by burning natural gas and air to inertness.)

(ii) Addition of chloride. Chloride is added to the catalyst by injecting carbon tetrachloride into the heated inert gas. The CCl₄ breaks down to form free chlorine before it reaches the catalyst.

(iii) Inhibition of platinum crystal growth. Excessive growth of the platinum crystals is prevented by redispersing the crystals. This is done by proper chlorination and by oxygen soaking or "rejuvenation".

A computer-based planning program is required to synthesize the sequence of valve operations which will accomplish the above regeneration steps. In order to do this, however, a model of the process must be created on the computer. The procedure for creating the modelling structure which represents a process flowsheet was explained in Chapter 1 of this series.
Figure 3.2 Structure of the Catalyst Regeneration System
3.6.2 Application of the Proposed Planning Methodology

The planning problem which will be solved here is the coke removal from the catalyst described in the section above. We shall step through the problem formulation phase of the planning methodology and state the planning goals as we see fit, and then require the computer to generate a feasible operating procedure using the methodology described in this paper.

**Description of the Initial State:** In describing the initial state to the computer, the user clicks on a particular piece of equipment, and a menu appears see Figure 3.3 which asks the user to choose the state which he or she would like to modify. In this case we choose INITIAL STATE, at which point the computer pops up another menu which contains a list of the state variables corresponding to that unit. The user can modify these values directly on the menu. Once this has been done, the values input by the user are propagated to neighboring units, and the process continues.

The values which were input for this example are:

```
valve-1-stem-position = closed
valve-3-stem-position = closed
valve-4-stem-position = closed
valve-5-stem-position = closed
valve-6-stem-position = closed
valve-7-stem-position = closed
valve-8-stem-position = closed
valve-10-stem-position = closed
valve-11-stem-position = closed
valve-12-stem-position = closed
valve-13-stem-position = closed
valve-14-stem-position = closed
valve-15-stem-position = closed
```
Figure 3.3: The menu-driven input of an Operational State
valve-16-stem-position = closed
valve-17-stem-position = closed
reactor-1-flow-rate = positive (flow from 1 to 2)
reactor-1-port-2-product-x = positive (reaction is occurring)

This is enough information for the computer to automatically set the values of the hundreds of state variable values using the qualitative constraint propagation techniques mentioned earlier.

**Description of the Goal State:** The mechanics of the Goal State specification follow closely those of the initial state specification. When the first menu pops up, the user selects GOAL STATE, and modifies (or inputs variables) as desired. For this example, the following goal state variables were specified.

reactor-1-flow-rate = positive
reactor-1-port-2-product-x = 0
reactor-1-port-2-oxygen-x = positive
reactor-2-flow-rate = positive
reactor-2-product-x = positive

The program propagates these values as they are being input, and this results in a partial description of the goal state.

**Specification of operational constraints on the plan:** The following operational constraints are input by the user.

Production constraint:

"Reactor 2 must be brought on line before Reactor-1 is taken off-line"

(reactor-2-flow-rate = positive) & (reactor-2-port-2-product-x = positive)

*before* (reactor-1-port-2-product-x = 0)

Mixing Constraints:

The following forbidden mixes are defined:
(OXYGEN . HYDROGEN)

(HYDROCARBON . OXYGEN)

(NATURAL-GAS . OXYGEN)

Note that there are other mixing constraints pertinent to the overall regeneration procedure, however, for simplicity we only consider here the pairs which are of importance for the removal of coke deposits stage.

**Specification/Identification of Planning Islands:** In this case, there are no known planning islands. Later on, during the planning process, the computer will generate a planning island for purposes of purging hydrocarbon from reactor-1. However, at this stage, this cannot be anticipated either by the user or by the program. In reality, the goal state for this sub-problem, namely the removal of coke build-up is the first planning island in the overall catalyst regeneration problem. For purposes of presentation, we have only considered this first phase, and treated the planning island as a goal state, without any loss of generality.

At this point, we are ready to move into the next phase of the planning process, namely, the Plan Generation Phase.

**Identification of primitive actions:** The means-ends analysis is used to identify the primitive actions which must be carried out in order to effect the transformation from initial to goal state. In this case, the goal state is simply compared with the initial state, and all of the valves whose states are different in the two states are marked as primitive operators. In cases where quantitative constraints occur, this becomes more complicated, as the quantitative changes in control action must be determined: it is not sufficient to simply state that a valve must be switched from open to closed or vice-versa. Future research effort is aimed at addressing the issue of how to efficiently handle quantitative constraints.
Transformation of mixing constraints into temporal constraints: The influence graphs for oxygen, hydrogen, hydrocarbon and natural gas are constructed. No conflicts in the initial state occur. In the goal state, we discover that valve-7 forms a separation set for natural-gas/oxygen, and does not interfere with the achievement of the other goals. So this valve is specified to be closed in the goal state. Similarly valve-6 is required to be closed to prevent the mixing of hydrogen and oxygen.

This concludes the specification of a consistent goal state, and the program is ready to search for potential constraint violations. The algorithm is run on the worst-case intermediate state, and a potential constraint violation (explosion caused by mixing oxygen and hydrocarbon) is detected in reactor 1 and its immediate surroundings. A minimal separation valve set is found using the transformed network flow algorithm mentioned earlier. The set of valves chosen is \{valve-17\}. This set, while minimal, is unacceptable, as it conflicts with the goal "reactor-1-port-2-oxygen-x = positive". Thus the computer evolves from this set, replacing its members with \{valve-14, valve-15\} and so on until the set \{valve-2, valve-4, valve-12\} is found which is a separation set and is also consistent with the goal state specification. Thus this set of valves is stipulated to be closed in the goal state, and for each of these valves, a temporal constraint is introduced which states that the valve must be closed before any valve (form the marked set of primitive operators) is opened.

Generation of Evacuation/Purge Planning Islands: Once the separation set has been found and temporal constraints have been stated, this guarantees that the program will be working with steady states which are consistent with the stated operating constraints. However, we still need to address the fact that the hydrocarbon which was flowing into reactor 1 may come into contact with the oxygen while the process is undergoing a transition from one state to another. The program handles this situation by attempting to purge the hydrocarbon before admitting the oxygen.

As discussed earlier, there may be several alternative purge methods and fluids but, for the current implementation, only sweep is considered, and the first available purgative is used. It is relatively straightforward to extend the methodology so that it considers other alternatives. In this
Figure 3.4: Influence Graphs of oxygen and hydrocarbon in Worst Case State
Figure 3.5: Influence Graph after the ASVS has been chosen
case, algorithm FINDPATH, described earlier, is run and it is discovered that the inert gas from
the inert gas source may be used as a purgative, and that the hydrocarbon may be vented to sink 1.
Thus a planning island is formulated by the program, which contains the following state
specifications (see Figure 3.6):

reactor-1-flowrate = positive
reactor-1-port-1-inert-gas-x = positive
valve-15-stem-position = closed
valve-13-stem-position = closed
valve-5-stem-position = closed
valve-4-stem-position = closed
valve-8-stem-position = closed
valve-11-stem-position = closed

This automatically specifies that valve-3 and valve-14 must be opened (using the constraint
propagation algorithm).

This planning island is inserted (by the computer) between the initial and goal states, and
can be viewed and modified by the user (if desired) using the same menu driven utilities which
were used to view and modify the initial and goal operating states.

**Synthesis of an operating procedure.** From this point onwards the methodology is identical to the
one described in the second part of this series. A partial plan is constructed, and then a complete
plan is synthesized. The complete plan is presented to the user in an EMACS-type editor, and the
user is allowed to cut and paste operations in the plan, if the initial plan is not completely
acceptable. The computer takes this modified plan and compares it with the partial plan, which is a
concise representation of the mixing and temporal constraints, and informs the user of any potential
constraint violations.

For our example, the plan generated was:

OPEN VALVE 1 > OPEN VALVE 10 > CLOSE VALVE 2 > OPEN VALVE 3 >
Figure 3.6: The Path Finding algorithm sets up a Purge Planning Island
This plan was deemed acceptable as the only constraints stated in the problem were either temporal or mixing, and the plan is guaranteed to satisfy these constraints, thus no final modifications were made.

3.7 INDUSTRIAL CASE STUDY: E.D.T.A. PLANT RECOVERY PROCEDURES

Another area in which the application of the planning methodology has been successful is the planning of recovery procedures for faults which the designer anticipates in the chemical plant. We shall illustrate this aspect of the methodology within the context of another industrial case study. The system we will be considering is a slightly modified version of the ethylenediaminetetraaceticacid plant which was analysed elsewhere in the literature (Foulkes et al., 1988)

3.7.1 Description of the Problem

Ethylenediaminetetraaceticacid (E.D.T.A.) is produced using the cyanomethylation process. The process flowsheet is shown in Figure 3.7. The main pieces of equipment which come into play in this particular planning problem are the pumps, valves, mixing tanks, storage tanks and the reactor. Other details of the plant, such as the P&I Diagram have been abstracted away for simplicity.

The manufacture of E.D.T.A. is accomplished in the following manner. Sulphuric acid, formaldehyde and hydrogen cyanide are pumped into a glass-lined mixer (mixer 1), with care being taken to achieve these operations in that order. This must be done to ensure that the mixture in the mixer 1 is stable at all times. In a separate flow structure of the plant, ethylenediamine (E.D.A.) and dilute sodium hydroxide are mixed (mixer 3). The solutions from mixers 1 and 2 are now pumped to the reactor.
Figure 3.7 Flowsheet of the Ethylenediamineacetic Acid Plant
When the reaction is complete, the product is tested for traces of hydrogen cyanide. If there is any HCN present, a dilute solution of formaldehyde is prepared in mixer 2 and is added to the reactor until there are no more traces of HCN.

In this example we shall study the operating procedures which will keep the plant operating even in the face of such events as the failure of a pump, or a mixing tank. In planning the response to such a failure, the operator must bear in mind the following safety constraint: hydrogen cyanide and formaldehyde should never come into direct contact with each other. There are also other safety constraints, but they are not discussed here as they are generally inactive constraints under the circumstances of this example.

The specific situation which we will be considering in this example is as follows. Hydrogen cyanide is being handled using the topmost line in the flowsheet (see Figure 3.7). Sulphuric acid and formaldehyde are being handled using the next two lines while the fourth line, which passes through pump P04 has been shut down for routine maintenance. At some point in the operation, the cooling jacket on mixer 1 springs a leak, and the mixer has to be shut down. Mixer 2 must now be used as a standby, at least until mixer 1 has undergone maintenance. During the recovery, the operator must be careful that hydrogen cyanide and formaldehyde do not come in contact with each other.

3.7.2 Application of the Methodology

We shall now illustrate the performance of the planning algorithm on the industrial case study described above.

**Description of the Initial State:** The Initial State description is achieved in a similar manner to that described in the first case study. For this particular example, the state specification involved the following statements:

\[
\text{valve-7-stem-position} = \text{closed}
\]

\[
\text{valve-9-stem-position} = \text{closed}
\]
Figure 3.8 Influence Graphs in the Initial State
valve-14-stem-position = closed
valve-3-stem-position = closed
valve-17-stem-position = closed
valve-24-stem-position = closed
valve-25-stem-position = closed
valve-26-stem-position = closed
valve-30-stem-position = closed
valve-32-stem-position = closed
valve-27-port-1-flowrate = positive
valve-28-port-1-flowrate = positive
valve-29-port-1-flowrate = positive
valve-22-port-1-flowrate = positive
valve-23-port-1-flowrate = positive

With these values input, the computer is able to propagate the information to complete the state specification. (Note, for the sake of brevity in the presentation of the example, the state of the system downstream of the mixers has been left unspecified, as it does not affect the planning in this particular instance.)

**Description of the Goal State:** The following state variables describe the primary Goals of the planning problem.

mixer-1-port-1-flowrate = 0
mixer-2-port-1-flowrate = positive
mixer-2-port-1-HCN-x = positive
mixer-2-port-1-acid-x = positive
mixer-2-port-1-HCHO-x = positive

From this partial state description, the program tries to complete a Goal State description by propagating these values through the model equations.
Specification of Operational Constraints on the Plan and Planning Islands: The only operating constraint which is important for this problem is the mixing constraint which disallows the mixing of HCN and formaldehyde. This stated in the usual manner as

\[(HCN \cdot FORMALDEHYDE)\]

As in the case of the catalyst reactivation system, no planning islands can be anticipated a priori. As will be seen later, however, it is necessary for one to be generated during the actual planning process.

Transformation of mixing constraints: The generation of the primitive actions parallels closely that of the previous example, and so this is not described in detail in this section. However, the transformation of the mixing constraints into temporal constraints is more interesting and occurs as follows:

1. No conflicts in the initial state occur.
2. In the Goal State, valve-10 serves as a vertex separator for hydrogen cyanide and formaldehyde. Closing this valve does not adversely affect the achievement of any of the stated goals, and so this valve is "closed" in the Goal State.
3. Intersection of the Influence Graphs of HCN and formaldehyde in the worst case intermediate state lead to the discovery of potential constraint violations. The MSVS detected in this case, is again valve-10. Thus constraints are generated which state that this valve must be closed before any valve in the Initial State is opened.

Generation of evacuation/purge planning islands: Having found an Acceptable Valve Separation Set, and having stated the appropriate temporal constraints, it is necessary to formulate a purge planning island. The path finding algorithms locate a path from the sulphuric acid storage through pump P03 and valves 20, 29 and 32. This path is stated as a planning island,

\[\text{valve-20-port-1-flowrate = positive}\]
valve-20-port-1-acid-x = positive

and is constrained to lie before the operations which allow HCN into the pipe lines, and after the ASVS is closed.

**Synthesis of an operating procedure:** The final stage of the methodology, namely the synthesis of a complete operating procedure is carried out, and the following plan is recommended.

\{CLOSE VALVE 27, CLOSE VALVE 28, CLOSE VALVE 31\} >
\CLOSE VALVE 10 > OPEN VALVE 9 > CLOSE VALVE 12 >
\{OPEN VALVE 24, OPEN VALVE 25, OPEN VALVE 26\}

The operations which appear in curly brackets have no particular precedence ordering. They may be achieved in any order relative to one another, or even simultaneously. In actuality, this representation captures all of the detail of a set of complete plans. Each one of these plans, of which there are 36, is feasible, and are equally appropriate. An exhaustive generate and test procedure would be unable to present this kind of concise description of a set of plans, and would present all 36 to the user.

This completes the planning for this example, and the computer has generated a feasible operating strategy for recovery from the fault which was postulated, namely the leak in the cooling jacket of the first mixer.

**3.8 SUMMARY**

The synthesis of operating plans for complete chemical plants is an interesting problem and one that has many practical applications. The thrust of a computer-based planning methodology should be toward increased automation, improvement in the efficiency of planning (planning should be achieved in polynomial time, wherever possible), and guarantees that the planning methodology produces correct (feasible) plans. It is also desirable that the plans generated are optimal, or at least cost efficient in their implementation.
In developing this methodology, I have addressed the first three of these considerations within a modelling framework that is easy to use, menu-driven and which utilises models with which process engineers are generally familiar. There has been an implicit tradeoff between safety (correctness) and optimality. Future work should address the issue of the completeness of the methodology (i.e., if a feasible plan exists, it will be found) which can not be currently guaranteed without the exhaustive (and practically infeasible) generate and test approach.
APPENDIX

Algorithm: DEFINE-INFLUENCE \((x_i, L_p)\)

Input: Graph of the flowsheet created in STEP 1 for the given chemical species \(x_i\), and the operational state \(L_p\).

Output: The same graph with all nodes marked that could possibly be visited by species \(x_i\) in the operating state \(L_p\).

\[
\text{begin} \\
\quad Q := \{s_i\}; \\
\quad \text{while } Q \neq \emptyset \text{ do} \\
\quad \quad \text{begin} \\
\quad \quad \quad \text{let } v \text{ be any element of } Q; \\
\quad \quad \quad \text{remove } v \text{ from } Q; \\
\quad \quad \quad \text{label}(v) := \text{label}(v)||x_i; \\
\quad \quad \quad \text{for all } v' \text{ in } A_o(v) \text{ do} \\
\quad \quad \quad \quad \text{(comment: } A_o(v) \text{ is the set of adjacent out-edges of } v) \\
\quad \quad \quad \quad \quad \text{if } x_i \text{ is not a member of } \text{label}(v') \text{ then add } v' \text{ to } Q; \\
\quad \quad \quad \text{end;} \\
\quad \quad \text{end} \\
\text{end}
\]
Proof of Complexity Estimate for Algorithm Define-Influence.

In section 4.1 it was claimed that the time complexity of the algorithm used to construct the Influence Graph of a given species is $O(|E|+N_i)$. Clearly, the total number of edges, $|E'| = |E|+N_i$. Now the complexity of the algorithm has three components:

(i) Initialization, which takes constant time,

(ii) Maintaining the set $Q$; there are at most $2(|V| + 1)$ additions and removals from $Q$ (worst case estimate, when no directional information is known). Each addition or deletion can be done in two or three operations (see Papadimitriou and Steiglitz, 1982), an $O(|V|)$ bound follows.

(iii) Searching the adjacency lists $A_0$; we do constant work for each element of each adjacency list. Since the sum of the lengths of the adjacency lists is $2|E'|$, the time required is $O(|E'|)$, or $O(|E|+N_i)$.

Assuming that there are no isolated nodes, the overall time complexity is $O(|E|+N_i)$.

(Since this is then always greater than or equal to $O(|V|)$

Proof of Theorem 1

Theorem 1: If there exists a directed path from the super source $S_p'$ to any given node, $v$, then that node is labelled with the species $x_p$.

Proof: (by induction on the length of the path from $S_p'$ to $v$)

Consider the first node, $v'$, on the shortest path (of length L) from $S_p'$ to $v$. Clearly, it will be labelled with $x_p$, since an edge will exist from $S_p'$ to $v'$ (by construction) and $v'$ will be in the set $A_0(S_p')$. Consider the node $v''$ which lies adjacent to $v$ on this path, and whose distance from $S_p'$ is $(L-1)$, and assume that this node is labelled with $x_p$. If this is the case then, since $v$ is in the set $A_0(v'')$ it will also be labelled. Q.E.D.
**Vertex Separators** (Even, 1979) and the **Minimal Separation Valve Set** (MSVS)

*Definition:* Let $G(V, E)$ be a finite undirected graph, with no self-loops and no parallel edges. A set of vertices, $S$, is called an $(a, b)$ vertex separator if $\{a, b\}$ is a subset of $V - S$ and every path connecting $a$ and $b$ passes through at least one vertex of $S$.

Clearly, if $a$ and $b$ are connected by an edge, no $(a, b)$ vertex separator exists. Let $a \neq b$ mean that there is no such edge. In this case, let $N(a, b)$ be the least cardinality of an $(a, b)$ vertex separator. Also, let $p(a, b)$ be the maximum number of pairwise vertex disjoint paths connecting $a$ and $b$ in $G$; clearly, all these paths share the two end-vertices, but no other vertex appears on more than one of them.

Perform the following construction on the graph $G(V, E)$. Construct a digraph $G^+(V^+, E^+)$ as follows. For every $v$ in $v$ put two vertices $v'$ and $v''$ in $V^+$ with an edge $v' \rightarrow v''$ of unit capacity. For every edge $u \rightarrow v$ in $G$, put two edges $u'' \rightarrow v'$ and $v'' \rightarrow u'$ in $G^+$ each of infinite capacity. Now define a network with source $a''$ and sink $b'$. Edges of the $v' \rightarrow v''$ variety are referred to as internal edges, while the others are called external edges. An example of this construction is shown in Figure 3.11.

The total maximum flow $F$ from $a''$ to $b'$ in this network can be shown to be equal to $p(a, b)$ (see Even, 1987). We can find a minimum $(a, b)$ vertex separator as follows. Once the flow has been maximised (in this case, using the standard Dinic algorithm), apply the construction of the layered network. The set of vertices which appear also in this layered network, $S$, defines a minimum cut which consists of internal edges only. Let $R$ be the vertices of $G$ which correspond to the internal edges in $(S; S^+)$. $R$ is a minimum $(a, b)$ vertex separator in $G$. The additional work of constructing the layered network is of time complexity $O(|E|)$. Note: proofs of these claims can be found in the book by Even.
Figure 3.11
Algorithm for Checking Labels on Influence Graphs

The following algorithm is used in the determination of potential violations of mixing constraints. As described in Section 4.2, the method for determining the locations in the chemical plant in which a pair of dangerous species might come in contact with each other involves checking the labels on the nodes of the Influence Graph. If any node in the graph is labelled with both species from any dangerous pair, then that node corresponds to a piece of equipment which may present a safety hazard. The following algorithm performs this checking and is of time complexity $O(|C|^2 + |V|)$.

Algorithm: Check-Labels(IG, C)

Input: The Influence Graph, IG, of any given Operational State.

Output: The same graph in which a node is labelled with a constraint $c$ if there is a potential for that constraint to be violated in the piece of equipment corresponding to that node, in the given operational state.

begin
    $Q := V$;
    while $Q \neq \emptyset$ do
        begin
            let $v$ be any element of $Q$;
            remove $v$ from $Q$;
            violation($v$) := $\emptyset$;
            for every pair $c$ in $C$ do
                (comment: $C$ is the set of mixing constraints which are to be checked for violation.)
                begin
                    flag := nil
                    for every species $s$ in label($v$) do
                        if $s$ is an element of $c$ and flag \neq s then
If flag = nil then flag := s;
else if flag is an element of c then
    violation(v) := c || violation(v);
end;
end;
end

Analysis of the Time Complexity of the CHECK-LABELS Algorithm

In the CHECK-LABELS algorithm, each node in the Influence Graph is operated upon in the following manner:

1. A constraint c is picked from the set C, of which there are |C| possibilities.

2. Each element in the label of the node is then scanned exactly once, and compared with the constraint c. Each of these comparisons takes constant time. The number of elements in the label of any given node is at most equal to the number of dangerous species, since the labelling of the Influence Graph is only performed for such species. An upper bound on the number of dangerous species is 2|C| which is the case when no dangerous species occurs in more than one constraint.

Thus step (2) when performed for a given constraint and a given node takes $O(|C|)$ time. Step (1) is performed $|C|$ times, resulting in a $O(|C|^2)$ bound for steps (1) and (2). Since steps (1) and (2) are performed once on each node, of which there are $|V|$, the overall time complexity is $O(|C|^2 \cdot |V|)$. 
CHAPTER 4

PLANNING WITH QUANTITATIVE CONSTRAINTS
4.1 INTRODUCTION

The first three chapters of this thesis have dealt with the issues of representation, a nonlinear planning methodology, and planning with qualitative mixing constraints. The methodology developed in those chapters can generate plans that are guaranteed to satisfy temporal constraints on the goals, and qualitative mixing constraints. Temporal constraints include explicit statements about the order in which some of the operations must be performed e.g.

\{Achieve (Start Coolant) before (Start Feed)\}

Qualitative mixing constraints disallow certain chemical species from coming in contact e.g.

\{HCN should not mix with HCHO\}

Besides temporal and qualitative mixing constraints, a plan may have to satisfy other types of constraints. It is important to develop methods for posting different types of constraints that are efficiently transformed into temporal constraints on the primitive operators.

The fundamentals of nonlinear planning and its advantages over linear planning are outlined in Chapter 2. The success of nonlinear planning relies heavily on the ability to transform various types of constraints in an efficient manner. The task of developing efficient constraint transforming methodologies is therefore an important one. The efficiency of constraint posting depends on the complexity of determining clobberers and white knights for the type of constraint being considered. Those constraints that cannot be transformed to temporal constraints nonlinearly, have to be posted in a linear manner, by resorting to a generate and test procedure. It has been shown in Chapter 3 that qualitative mixing constraints may be transformed to temporal constraints using polynomial time algorithms. The methodology is specific to the domain of chemical plants and uses structural knowledge of a chemical plant. It is desirable to develop techniques for transformation of other types of constraints that occur in a chemical plant.
4.1.1 The Need for Posting Quantitative Constraints

Several constraints in a chemical plant include numerical values of variables and may be classified as quantitative constraints. During the operation of a reactor, the temperature may not be allowed to exceed a certain maximum. The distillate from a distillation column may be required to have a minimum purity. Two chemicals may not be allowed to mix if the temperature is below a minimum. These constraints involve numerical values and may be stated as follows:

\{HCN should not mix with HCHO at \( T \geq T_{\text{max}} \} \)

\{[HCN] \leq 0.1 \text{ mol/lit}, in the reactor at all times. \}

Such constraints commonly arise from safety requirements, product specifications, environmental regulations etc. It is essential for any operating procedure to satisfy such constraints. The methodology developed so far cannot post quantitative constraints in an efficient manner and tests for their violation in a linear manner by resorting to generate and test. In this chapter I give a formalized methodology for transforming this very important class of constraints into temporal constraints on the operators.

4.1.2 Previous Work

A detailed review of previous work in planning of process operations is given in Chapter 1. All the approaches to synthesizing operating procedures have relied on a linear, generate and test type method. In most cases, quantitative constraints have also been dealt with in the same generate and test format.

Lau, (1982) in his methodology for planning of start-up procedures requires the user to transform quantitative constraints into temporal constraints to ensure their satisfaction. For example, a constraint of the type

\[ T_{\text{out},h} \leq T_{\text{max}} \]
cannot be used as such and is transformed by the user to a temporal constraint of the form

\{ (\text{Start cold side flow}) \text{ before } (\text{Start hot side flow}) \} \\

Such transformation avoids dealing with the constraint directly and requires understanding of
the process and the operations being performed.

The planning methodology of Fusillo and Powers (1987, 1988) transforms the
quantitative problem into a qualitative one by using qualitative measures of low medium and
high to express quantitative ranges. A temperature constraint of the type

\{ \text{Do not allow } O_2 \text{ if } T \geq 100 \, ^\circ \text{C} \} \\

is stated as

\{ \text{Do not allow } O_2 \text{ if } T \geq \text{medium} \} \\

Once again, quantitative constraints are not dealt with explicitly, and the procedure requires the
user to give some definition of low, medium and high for the relevant variables.

The constraint posting methodology proposed in this paper puts an end to the various ad hoc
and generate and test type approaches. Quantitative constraints are handled directly and their
transformation to temporal constraints is performed efficiently and automatically.

4.1.3 Outline of the Chapter

Systematic development of the methodology is given in Section 4.2. Algorithms for
abstraction of primitive operators, identification of clobberers, the process of demotion of
clobberers and identification of white knights are described with formal proofs where
applicable. Each step is illustrated with examples of a process for the manufacture of EDTA.
The complete methodology is illustrated on a case study of a gasoline polymerization plant in
Section 4.3. A summary of the paper is given in Section 4.4.
4.2. DEVELOPMENT OF THE METHODOLOGY

Each step of the methodology for posting of quantitative constraints is illustrated with examples of various situations in an EDTA manufacturing plant (Foulkes et al, 1988).

4.2.1 Description of EDTA Plant

Ethylene Diamine Tetraacetic Acid is manufactured by the cyanomethylation process. The flowsheet is shown in Figure 4.1. The reactions involved are given below:

\[ HCHO + HCN + H_2SO_4 \xrightarrow{} H \quad | \quad H - C - OH \quad | \quad CN \]

Cyanohydrin

\[ H_2N - C_2H_4 - NH_2 + 4 H - C - OH + NaOH \xrightarrow{} \]

EDA

\[ HOOCCH_2 \quad N - C_2H_4 - N \quad CH_2COOH \]

\[ HOOCCH_2 \quad + \quad 4 NH_3 \]

EDTA

Mixer M1 is used for cyanohydrin formation and is glass lined. Mixer M2 may be used in emergencies but is not glass lined. It may be damaged by the formation of paraformaldehyde. Ethylene Diamine and Sodium Hydroxide are mixed in mixer M3. The final reaction occurs in reactor R1. Mixer M2 is also used for making a dilute solution of Formaldehyde and feeding it to reactor R1 to get rid of excess HCN, if present. The cyanohydrin formation takes place at a pH of 1. Reactor R1 should have no traces of HCN. HCN and HCHO should not be allowed to mix, except in a mixer to avoid a safety hazard.
4.2.2 Truth Criterion for Quantitative Constraints

The important contribution of Chapman's work is the use of a provably correct modal truth criterion for nonlinear planning and thus avoiding any scruffiness in the planning methodology. The efficiency of nonlinear planning depends on the ability to evaluate this truth criterion i.e. on the ability to identify establishers, clobberers and white knights, efficiently. Unfortunately, the truth criterion can be evaluated efficiently only for a simple action representation. No side effects of actions are allowed. This is clearly stated by the intractability theorem.

Intractability Theorem: The problem of determining whether a proposition is necessarily true in a nonlinear plan whose action representation is sufficiently strong to represent conditional actions, dependency of effects on input situations, or derived side-effects is NP-hard.

Most realistic domains, including the process operations planner's require the use conditional operators (Chapter 1). Intractability may be avoided by using domain-specific knowledge in the planning process. A truth criterion, valid for the specific domain should be identified and proved. Also, efficient ways of evaluating the domain-specific truth criterion should be given. Such a truth criterion has been identified and proved for synthesizing plans that are guaranteed to satisfy quantitative constraints in chemical plants. For posting quantitative constraints of the form $x \cdot \dot{} k$ the truth criterion is stated as follows:

Modal Truth Criterion for Quantitative Constraints: A constraint $x \cdot \dot{} k$ is necessarily satisfied in a situation $s$ iff there is a situation $t$ equal to or necessarily previous to $s$ in which $x \cdot \dot{} k$ is satisfied and for every step $C$ possibly before $s$ that possibly violates $x \cdot \dot{} k$, there is a step $W$ necessarily before $C$ that ensures $x \cdot \dot{} k$ in $s$.

---

1 '.\dot{}' indicates relationships of the type $<, >, =$ etc. between $X$ and $k$. 

109
Proof: See Appendix.

The statement and proof of the truth criterion for quantitative constraints is along the lines of Chapman's truth criterion for domain independent nonlinear planning. From this criterion, the plan modification operations that would ensure satisfaction of constraints are:

1. If C is constrained to lie after s, its clobbering effect will not be felt at situation s. This process of constraining C to lie after s is called demotion.

2. A white knight, W may be selected and constrained to lie before C and therefore s, so as to ensure that the clobbering effect of C will not be felt.

The time required to evaluate the truth criterion depends on the complexity of the procedures for determining clobberers and white knights. The algorithms for these operations are presented below. Clobbering of a constraint may often be caused by a set of valve operations that achieve an effect together. Such a set of valves is identified as an abstracted operator.

4.2.3 Abstraction of Operators:

In planning of process operations we are primarily interested in the effects of achieving the flow of various species from each source. A valve is the primitive operator. Means-ends analysis identifies the primitive valve operations required to achieve the goal state. A goal is often achieved by multiple valve operations, and a single valve operation may or may not have any effect on the operation of the process. All the valve operations that will together result in violation of a constraint will be clobberers of that constraint. The first step in the quantitative constraint posting methodology is to identify abstracted operators as the sets of primitive operations that together, start or stop flow of a single species from one source. The algorithm for generation of abstract operators is as follows:

```
procedure ABSTRACT-OPERATOR

Input: Initial state, I; Final state, F; List of goals to be achieved, G and list of primitive valve operations required to achieve the goal state, V.
```
Output: Abstract operators, A consisting of the set of valve operations necessary to start or stop flow from each source to point where goal is specified

begin

for each g ∈ G do

if RHS (g) = 0 then STATE:= I  \{comment: if goal is to start/stop flow, consider\}  
else STATE:= F  \{final/initial state as 'state'\}  

remove closed valves from STATE

j = 0

for each point of goal specification, e_i do

begin

j = j + 1

for each source, s_j do

FIND-PATH (e_i, s_j)

(comment: Search for paths from point where the goal is specified to sources of species and mark valves along it)

AO_j:= Intersection (open/closed valves in V, marked valves), for start/stop flow

end

end

Theorem: Given the list of primitive valve operations to achieve a goal state, the algorithm will detect every abstracted operator corresponding to the set of valve operations required for starting or stopping flow of material from each source, s_j to the point of constraint specification, e_i, necessary to achieve the goal state.

Proof: See appendix.

This algorithm runs in time O(g (V +s (V+E+V \ln V)) where g is the number of goals, s is the number of sources, V is the number of valves and E is the number of edges (connections) in
the process flowsheet. From the complexity analysis presented in the appendix, this reduces to O(g (E^2 + VE lnV + V^2 lnV)).

Illustration 1: Consider the EDTA plant case study given in Part III. Due to a leak in mixer M1, it is required to change over to mixer M2. The initial state of the plant is shown in Figure 2. The three reactants are being fed to M1. The goal state is to have the reactants flowing to M2 and no flow in M1. This is stated as follows:

\[
G = \{\text{mixer-1-port-1-flowrate} = 0 \\
\text{mixer-2-port-1-flowrate} = \text{positive} \\
\text{mixer-2-port-1-HCN-x} = \text{positive} \\
\text{mixer-2-port-1-H_2SO_4-x} = \text{positive} \\
\text{mixer-2-port-1-HCHO-x} = \text{positive}\}
\]

Note that the goals are specified only at two points in the process: e_1 = mixer-1-port-1 and e_2 = mixer-2-port-1.

The final goal state as identified by the program is shown in Figure 3. The operations identified by means-ends analysis to achieve this goal state are:

\[
V = \{\text{CLOSE VALVE 27, CLOSE VALVE 28, CLOSE VALVE 31, CLOSE VALVE 10, OPEN VALVE 9, CLOSE VALVE 12, CLOSE VALVE 19, OPEN VALVE 24, OPEN VALVE 25, OPEN VALVE 26, OPEN VALVE 32, OPEN VALVE 14, OPEN VALVE 21, OPEN VALVE 33}\}
\]

Consider the first goal in the list, g_1. Following the algorithm, STATE = initial state (Figure 2). The valves marked by the find-path algorithm lie along the paths shown in Figure 4. Taking the intersection set of V and the marked valves for each source, the abstract operators corresponding to each source are given below, with the physical significance of each abstracted operator. Of course, the abstraction algorithm does not automatically assign any physical significance to the operators.
Figure 2. EDTA Plant Initial State
AO-1:  \{CLOSE VALVE 27\} ⇒ Stop-Flow-HCN-M1
AO-2:  \{CLOSE VALUE 12, CLOSE VALUE 19, CLOSE VALUE 28\}
        ⇒ Stop-Flow-H\textsubscript{2}SO\textsubscript{4}-M1
AO-3:  \{CLOSE VALUE 10, CLOSE VALUE 31\} ⇒ Stop-Flow-HCHO-M1

Similarly, the paths and the marked valves found for the point of goal specification, $e_2$ are shown in Figure 5 and the abstract operators are as follows:

AO-4:  \{OPEN VALVE 24, OPEN VALVE 25, OPEN VALVE 26, OPEN VALVE 32\}
        ⇒ Start-Flow-HCN-M2
AO-5:  \{OPEN VALVE 9, OPEN VALVE 32\} ⇒ Start-Flow-H\textsubscript{2}SO\textsubscript{4}-M2
AO-6:  \{OPEN VALVE 14, OPEN VALVE 21, OPEN VALVE 33\}
        ⇒ Start-Flow-HCHO-M2

Abstraction of operators has been used extensively in increasing the efficiency of problem solving. ABSTRIPS (Sacerdoti, 1974), a modification of STRIPS (Fikes et al, 1972) used operator abstraction to increase planning efficiency. Abstraction of operators in a chemical plant may enable such application. More research is required in this direction.

4.2.4 Detection of Clobberers:

Any set of primitive operators that results in possible violation of a quantitative constraint will be a clobberer of that constraint. For satisfaction of the truth criterion, we need to identify those operators that could result in constraint violation. Clobberers of quantitative constraints may be identified efficiently only when the effect of just one operator is considered at a time. When multiple operators are considered simultaneously, the quantitative effect of each operator will depend on the magnitude of the others. Numerical simulation based on assumptions of the values of the variables may then be required to detect clobberers. Since each variables could take any value within a range, such simulation would be intractable. The following illustration clarifies this point.
Illustration 2: Continuing with the situation in illustration 1, let us consider the process of loading mixer M2. The three abstracted operators that need to be applied are

\{
(START-FLOW-HCN-M2),
(START-FLOW-H2SO4-M2),
(START-FLOW-HCHO-M2)
\}.

Consider the initial process conditions to be:

\[ [\text{HCN}]_0 = 1 \text{ mol/l} \]
\[ [\text{H}_2\text{SO}_4]_0 = 1.5 \text{ mol/l} \]
\[ [\text{HCHO}]_0 = 2 \text{ mol/l} \]

Mixer M2 is empty.

and quantitative constraints on the mixer to be:

\[ [\text{HCN}] \leq 0.1 \text{ mol/l}, \text{ for all } t \]
\[ [\text{H}_2\text{SO}_4] \geq 1 \text{ mol/l}, \text{ for all } t \]

Starting the flow of HCN could result in possible constraint violation. If the flow of HCN is started first and alone, it will result in \([\text{HCN}] = 1 \text{ mol/l}\) in M2 and therefore immediate constraint violation. If all the flows are started simultaneously, \([\text{HCN}]\) and \([\text{H}_2\text{SO}_4]\) are given by:

\[ [\text{HCN}] = [\text{HCN}]_0 \frac{V_{\text{HCN}}}{(V_{\text{HCN}} + V_{\text{H}_2\text{SO}_4} + V_{\text{HCHO}})} \]  \hspace{1cm} (1)
\[ [\text{H}_2\text{SO}_4] = [\text{H}_2\text{SO}_4]_0 \frac{V_{\text{H}_2\text{SO}_4}}{(V_{\text{HCN}} + V_{\text{H}_2\text{SO}_4} + V_{\text{HCHO}})} \]  \hspace{1cm} (2)

Thus, \([\text{HCN}]\) and \([\text{H}_2\text{SO}_4]\) will depend on the relative magnitudes of \(V_{\text{HCN}}, V_{\text{H}_2\text{SO}_4}, V_{\text{HCHO}}\). Constraint violation will have to be checked by solving equations 1 and 2 for different values of \(V_{\text{HCN}}, V_{\text{H}_2\text{SO}_4},\) and \(V_{\text{HCHO}}\) within a given range of values. Potentially infinite combinations of values are possible.

For the detection of clobberers, it is necessary to propagate the quantitative effects of each abstracted operator through the plant to the point where the constraint is applicable. The algorithm for performing this task is given below.
procedure Clobberer-Detection

Input: Abstracted operators, A; final state, F; quantitative constraints, Q

Output: Abstracted operators that are clobberers, C

begin

for each q ∈ Q do

for each a ∈ A do

begin

QUANTITATIVE-PROPAGATE (q, e_i, F)

(comment: propagate the quantitative values of the variables through the model equations)

if constraint, q is violated then a ∈ C

end

end

Theorem: Every abstract operator that will result in violation of a quantitative constraint if implemented first and alone will be detected as a clobberer.

Proof: See Appendix.

This algorithm runs in O(qV(V+E)) time and is illustrated in the example below.

Illustration 3: The abstracted operators for stopping flow to M1 and starting flow to M2 are

{(STOP FLOW HCN-M1), (STOP FLOW H_2SO_4-M1), (STOP FLOW HCHO-M1), (START FLOW HCN-M2), (START FLOW H_2SO_4-M2), (START FLOW HCHO-M2)}

The process conditions in mixer M2 are given in illustration 2. The conditions in M1 are considered to be unknown. Considering the effect of each abstracted operator individually, it is clear that (START FLOW HCN-M2) will clobber the constraint [HCN] ≤ 0.1 mol/l, for all t, and (START FLOW HCN-M2) and (START FLOW HCHO-M2) will clobber the constraint [H_2SO_4] ≥ 1 mol/l, for all t.
4.2.5 Selection of Plan Modification Operations:

The modal truth criterion for quantitative constraints gives two possible plan modification operations. The methodology for application of each of these is given below.

Demotion of Clobberers: Demotion of the clobberer involves constraining the clobberer to lie after the situation at which constraint satisfaction is being considered. Thus, if \{AO-1, AO-2, AO-3\} are three unordered operators, and AO-2 is a clobberer of a quantitative constraint, the plan may be modified by demotion of AO-2 to give \{AO-1, AO-3\} > \{AO-2\} i.e. AO-1 and AO-3 may be applied at situation \(s\), but AO-2 is constrained to lie after \(s\). Thus, posting quantitative constraints provides greater temporal order to the operators in the plan.

Illustration 4: The plan in terms of abstracted operators, after means-ends analysis and before posting qualitative mixing constraint is as follows:

\{(STOP FLOW HCN-1), (STOP FLOW H_{2}SO_{4}-1), (STOP FLOW HCHO-1), (START FLOW HCN-2), (START FLOW H_{2}SO_{4}-2), (START FLOW HCHO-2)\}

For the conditions and constraints given in illustration 2 above, the following clobberers are detected:

\[ [HCN] \leq 0.1 \text{ mol/l} \Rightarrow (START \text{ FLOW HCN-M2}) \]
\[ [H_{2}SO_{4}] \geq 1 \text{ mol/l} \Rightarrow (START \text{ FLOW HCN-M2}) \ (START \text{ FLOW HCHO-M2}) \]

Therefore, the plan after demotion of the clobberers will be:

\{ (STOP FLOW HCN-M1), (STOP FLOW H_{2}SO_{4}-M1), (STOP FLOW HCHO-M1), (START FLOW H_{2}SO_{4}-M2) \} > \{ (START FLOW HCN-M2), (START FLOW HCHO-M2) \}

Qualitative mixing constraints may be posted before or after posting quantitative constraints. In any case, if new operators (white knights) are detected, they will be considered to be new abstracted operators and it will be necessary to check these operators for constraint violation. The plan after both quantitative constraint posting and qualitative mixing constraint posting is:
{(STOP FLOW HCN-M1), (STOP FLOW HCHO-M1), (STOP FLOW H2SO4-M1),
(CLOSE MSVS)} > {(START FLOW H2SO4-M2)} > {(START FLOW HCHO-M2),
(START FLOW HCN-M2)}

If demotion of clobberers does not ensure constraint satisfaction, the next plan modification operation: identification of white knight will have to be applied.

Identification of White Knights: The operators identified by means-ends analysis ensure achievement of the goals, but not satisfaction of constraints. If all the operators for obtaining a feasible plan (one that satisfies both goals and constraints) have been correctly identified, then the constraints provide only sequencing information and demotion will always work. Otherwise, it may be necessary to identify new operators or planning islands as white knights to ensure constraint satisfaction. Thus, if every operator identified by means-ends analysis for a single equipment happens to be a clobberer, then constraint violation cannot be avoided by the current set of operators and a white knight has to be identified. For qualitative mixing constraints, closing the minimum separation valve set (MSVS) and/or the purge goal is used as a white knight. The procedure for identifying white knights for quantitative constraints is developed below.

This procedure involves two steps. First, it is necessary to identify the constraint being possibly violated and the qualitative change required in the variable to avoid the violation. For example, if a "T < T_{max}" constraint is violated, then the qualitative change required in T to avoid the violation is "Decrease T". Having identified the property to be affected and the direction of desired change, we now need to identify manipulations that may help us meet our objective. The qualitative value of the desired change is propagated through the steady-state model equations of the plant equipment, following the constraint propagation procedure of Steele (1980). Manipulations that cause the desired change and that are feasible are identified as white knights and are constrained to lie before the situation of interest, 's', in accordance with the truth criterion.
The algorithm for identification of white knights is as follows:

\textbf{procedure} WHITE-KNIGHT

\textbf{Input:} Plant flowsheet in final state, $F$; constraint being clobbered, $q$.

\textbf{Output:} White knight, $W$ for the constraint $q$.

\textbf{begin}

\textbf{for} $q$ \textbf{do}

\textbf{begin}

\textbf{if} $q \equiv 'X (<, \leq) k'$ \textbf{then} $L := 'decrease X'$ (comment: $L$ - load)

\textbf{if} $q \equiv 'X (> , \geq) k'$ \textbf{then} $L := 'increase X'$

\textbf{if} $q \equiv 'X = k'$ \textbf{and} $X < k$, \textbf{then} $L := 'increase X'$

\textbf{if} $q \equiv 'X = k'$ \textbf{and} $X > k$, \textbf{then} $L := 'decrease X'$

(comment: Such rules may be written for each type of constraint encountered)

\textbf{end}

\textbf{while} $v \neq$ manipulatable valve \textbf{do}

\textbf{QUALITATIVE-PROPAGATE} ($L$, $F$)

\textbf{return} $v$ and required qualitative change

\textbf{end}

\textbf{Theorem:} The algorithm will detect every manipulation that will possibly avoid a constraint violation as a white knight.

\textbf{Proof:} See Appendix.

The complexity of this algorithm for determining a white knight for a constraint being clobbered is $O(V+E)$.

\textit{Illustration 5:} Consider the situation of Illustration 4 with an additional constraint on the temperature of mixer M2. Let the temperature in the storage tanks, and the constraint on the temperature of the mixer be as follows:
\[ T_{\text{HCN}} = T_{\text{HCHO}} = T_{\text{H}_2\text{SO}_4} = 100^\circ C \]
\[ T_{\text{M}_1} < 90^\circ C, \text{ for all } t \]

Clobberers for constraint \( T_{\text{M}_1} < 90^\circ C \) are:

\text{START FLOW HCN-M2, START FLOW HCHO-M2, START FLOW H}_2\text{SO}_4\text{-M2}

Thus, all the abstracted operators for mixer M2 are clobberers of this constraint. None of these operators may be applied without immediate violation of the temperature constraint. It is therefore essential to search for a white knight that will nullify the effect of the clobberers. The white knight will have to cool down the reactor since the feed is too hot. Searching the plant for the relevant manipulations, we identify the following white knight

\text{(OPEN VALVE CW) } \Rightarrow \text{ Start jacket cooling water}

Constraining the white knight to lie before the clobberers the new plan obtained is:

\text{(OPEN VALVE CW) } \Rightarrow \text{ (START HCN-2, START HCHO-2, START H}_2\text{SO}_4\text{-2)}

The complete methodology for posting quantitative constraints is now illustrated for the EDTA plant.

\text{Illustration 6: In this illustration we will synthesize a plan for loading the reactor R1. The initial state of the plant is shown in Figure 6 and is stated below.}

\text{Initial state:}

\begin{align*}
\text{HCN-Mixer-M1} &= \text{ present} \\
\text{H}_2\text{COHCN-Mixer-M1} &= \text{ present} \\
\text{H}_2\text{SO}_4\text{-Mixer-M1} &= \text{ present} \\
\text{EDA-Mixer-M3} &= \text{ present} \\
\text{NaOH-Mixer-M3} &= \text{ present} \\
\text{Mixer-M1-port-1-flowrate} &= 0 \\
\text{Mixer-M1-port-2-flowrate} &= 0 \\
\text{Mixer-M2-port-1-flowrate} &= 0
\end{align*}
Figure 6. EDTA Plant Initial State for Illustration 6
Mixer-M2-port-2-flowrate = 0
Mixer-M3-port-1-flowrate = 0
Mixer-M3-port-2-flowrate = 0
Reactor-R1-port-1-flowrate = 0
Reactor-R1-port-2-flowrate = 0
[HCN]_{M1} = 0.05 \text{ mol/l}

Goal State:

Reactor-R1-port-1-flowrate = positive
Mixer-M1-port-2-flowrate = positive
Mixer-M3-port-2-flowrate = positive

Constraints:

Reactor, R1: Reactor-R1-HCN = absent ([HCN] = 0 \text{ mol/l})
Mixer, M2: [HCHO] < 0.1 \text{ mol/l}

The final state satisfying all the goals is as shown in Figure 7. The primitive operators identified by means-ends analysis are:

\{OPEN VALVE 34, OPEN VALVE 38, OPEN VALVE 37, OPEN VALVE 41\}

Abstract operators are found to be

AO-1 = \{OPEN VALVE 34, OPEN VALVE 38\}
AO-2 = \{OPEN VALVE 37, OPEN VALVE 41\}

Consider the effect of each abstracted operator on the reactor. AO-1 clearly violates the constraint and is a clobberer. Therefore, the plan after demotion of the clobberer is

AO-1 > AO-2

\{OPEN VALVE 34, OPEN VALVE 38\} > \{OPEN VALVE 37, OPEN VALVE 41\}

Given these two operators, the only way [HCN] may be maintained to 0 \text{ mol/l} in R1 is by not feeding any material from M1 to the tank, since HCN reacts only with HCHO which is absent
in both the streams being fed to R1. This fact may be recognized by solving the mathematical equations describing [HCN] in terms of the other flows and concentrations. If this information is supplied to the system, AO-1 will also be considered to be a clobberer and it will be necessary to search for a white knight.

The desired change to ensure constraint satisfaction is to "decrease HCN". Propagating this change through the model equations, "start HCHO" is detected as the white knight, since HCHO reacts with HCN. Therefore, the new plan would be

\[
\text{Start-HCHO-M2} > \{\text{AO-1, AO-2}\}
\]

The operator Start-HCHO-M2 violates the constraint [HCHO] < 0.1 mol/l in M2. Searching for a white knight to avoid this constraint violation results in detection of Start-H2O-M2. The final plan is

\[
\text{Start-H2O-M2} > \text{Start-HCHO-M2} > \{\text{AO-1, AO-2}\}
\]

The final goal state satisfying all the constraints is shown in Figure 8.

**Theorem:** The final plan is guaranteed to satisfy quantitative constraints in the situation being considered.

**Proof:** See Appendix.

Correctness of the constraint posting methodology is claimed only for the situation of the plant at which values of all the variables are available. For mixer M2, in illustrations 1 to 5, the situation considered during constraint posting is the initial state of the mixer. Once the flow of chemicals is started, constraint satisfaction will depend on the relative magnitudes of the various flowrates. Apriori determination of the values of the various flowrates and their rate of change such that constraint satisfaction is guaranteed at all times requires detailed simulation and an expensive generate and test procedure.
4.3 CASE STUDY - GASOLINE POLYMERIZATION PLANT:

The utility and application of the procedure of posting quantitative constraints in synthesis of an operating procedure for a large chemical plant is shown in this case study. The plant produces C₆, C₇ and C₈ from olefines and paraffins with a UOP solid phosphoric acid catalytic process, and is shown in Figure 9. The polymerization reactors are fed with fresh, preheated feed and are cooled by recycled propane from the product recovery section. The reaction products partially preheat the feed in a feed effluent exchanger. Butane, gasoline polymer, and part of the propane, are separated in distillation columns, dried, and are sent to storage. This plant has been studied in the past for design of optimizing control structures (Arkun and Stephanopoulos, 1981), and for the synthesis of operations and control structures during start-up (Lau, 1982). The planning methodology developed in this series of papers is applied to develop a start-up procedure, the main focus being on posting of quantitative constraints.

Following Lau's approach, the plant may be decomposed into four distinct sections: feed preparation, reaction, product refining and recovery as shown in Figure 10. The operating procedure for each section may be developed separately, and then the four sections be sequenced. The industrially recommended start-up sequence is recovery > product refining > feed preparation > reaction. The initial state of the plant has all valves and equipment shut down. The final state requires everything to be operating. The abstracted operators required to

<table>
<thead>
<tr>
<th>Equipment</th>
<th>Operators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pump, P-1</td>
<td>Start feed flow (1)</td>
</tr>
<tr>
<td>Feed Effluent Exchanger, HEx-1</td>
<td>Start cold feed flow (2)</td>
</tr>
<tr>
<td></td>
<td>Start reactor effluent flow (9)</td>
</tr>
</tbody>
</table>

Table 1. Abstracted Operators for Gasoline Polymerization Plant.
<table>
<thead>
<tr>
<th>Component</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feed Heater, HEx-2</td>
<td>Start cold feed flow (3)</td>
</tr>
<tr>
<td></td>
<td>Start heating fluid flow (45)</td>
</tr>
<tr>
<td>Reactor, R1, R2</td>
<td>Start reactant flow (5, 6)</td>
</tr>
<tr>
<td></td>
<td>Start interstage cooling (35, 36, 37, 38, 39, 40, 41, 42)</td>
</tr>
<tr>
<td>Depropanizer, D1</td>
<td>Start feed flow (11), Start top product (12)</td>
</tr>
<tr>
<td></td>
<td>Start bottom product (17)</td>
</tr>
<tr>
<td>Condensor, Cd1</td>
<td>Start hot fluid (12), Start coolant (49)</td>
</tr>
<tr>
<td>Reboiler, Rb1</td>
<td>Start cold fluid (18), Start steam (50)</td>
</tr>
<tr>
<td>Debutanizer, D2</td>
<td>Start feed flow (16), Start top product (21)</td>
</tr>
<tr>
<td></td>
<td>Start bottom product (27)</td>
</tr>
<tr>
<td>Condensor, Cd2</td>
<td>Start hot fluid (21), Start coolant (51)</td>
</tr>
<tr>
<td>Reboiler, Rb2</td>
<td>Start cold fluid (28), Start steam (52)</td>
</tr>
<tr>
<td>Cooler, Cl1</td>
<td>Start feed flow (25), Start coolant (46)</td>
</tr>
<tr>
<td>Cooler, Cl2</td>
<td>Start feed flow (30), Start coolant (47)</td>
</tr>
<tr>
<td>Drier, Dr1</td>
<td>Start feed flow (43), Start drying (48)</td>
</tr>
</tbody>
</table>

...go from the initial to the final state are shown in Table 1. The initial values of the various variables (temperatures, concentrations etc.) are assumed to be known. The operating procedure based on these quantitative constraints for each sub-section of the plant is presented below:

**Feed Preparation**: "T >" constraint on stream 4:

START-45 > START-3
Reaction: "T <" constraint on reactor R1:

{START-35, START-37, START-39, START-41, START-8} > {START-5}

"T <" constraint on reactor R2:

{START-36, START-38, START-40, START-42, START 7,} > {START-6}

Product Recovery: "x >" constraint on stream 16:

{START-13, START-14} > START-16

"x >" constraint on stream 20:

{START-17, START-18} > START 20

"T <" constraint on stream 13:

START-49 > START-12

"T >" constraint on stream 19:

START-50 > START 18

"x >" constraint on stream 25:

{START-22, START-23} > START-25

"x >" constraint on stream 30:

{START-27, START-28} > START 20

"T <" constraint on stream 22:

START-51 > START-21

"T >" constraint on stream 29:

START-52 > START 28
Product Refinement: "T < “ constraint on stream 26:

START-46 > START-25

"T < “ constraint on stream 31:

START-47 > START-30

"x_{H2O} < “ constraint on stream 44:

START 48 > START 44

The order of the rest of the streams is given from the flow structure of the process flowsheet. Final sequence is given by

Recovery > Product Refining > Feed Preparation > Reaction

The overall final plan corresponds to that given by Lau. Thus, transformation of quantitative constraints to temporal constraints gives a partial plan for start-up of the gasoline polymerization plant. No white knights were required for this example since the goal state was completely specified and happened to be consistent with the quantitative constraints.

4.4 SUMMARY:

Following the methodology of nonlinear planning, we have developed a procedure for transforming quantitative constraints on equipment into temporal constraints. This avoids using an exhaustive generate and test procedure for developing plans that are guaranteed to satisfy quantitative constraints and results in considerable savings in terms of algorithmic complexity.

The constraint posting methodology for quantitative constraints is based on a formal truth criterion. Correctness of the various algorithms used in the constraint transformation has been formally proved. The final plan is guaranteed to satisfy the quantitative constraints in the situation being considered. Satisfaction of the constraint at any time after this situation depends
on the magnitudes of the various variables. These magnitudes may be decided by simulating
the behavior of the process for different values of variables.
APPENDIX

Modal Truth Criterion for Quantitative Constraints: A constraint $x \cdot \circ k$ is necessarily satisfied in a situation $s$ iff there is a situation $t$ equal to or necessarily previous to $s$ in which $x \cdot \circ k$ is satisfied and for every step $C$ possibly before $s$ that possibly violates $x \cdot \circ k$, there is a step $W$ necessarily before $C$ that ensures $x \cdot \circ k$ in $s$.

Proof: This proof is analogous to the proof for Chapman's truth criterion. The criterion is proved by construction of specific completions of plans that satisfy various conditions and is proved in three parts. The first two parts show that necessary satisfaction of a constraint $x \cdot \circ k$ (or $q$ for short) requires the presence of an establisher and the absence of a clobberer. This is done by proving the contrapositive, that the absence of an establisher or presence of a clobberer ensures that $q$ is not necessarily satisfied. The third part of the proof shows the implication in the other direction: if there is an establisher and no clobberer, $q$ is necessarily satisfied.

If there is no establisher, then the constraint is being violated, and no available operator can affect the variable so as to change its value to satisfy the constraint. Thus, putting any available operator before or at $s$ will not result in constraint satisfaction, and by the truth criterion for complete plans, the constraint will not necessarily be satisfied if an establisher is absent. (For example, if a temperature constraint is being violated and no means of affecting it are available, then it will not be possible to satisfy this constraint).

In the next case, we prove that if a clobberer $C$ is present, then again we can construct a plan where $q$ will not be satisfied. The clobberer is possibly before $s$ and possibly violates $q$. Constrain $C$ to lie before $s$. This is possible because $C$ is a clobberer. Put all steps that are possibly after $s$ to be after $s$ and all steps possibly before $s$ to be before $s$. Some step lying before $C$ may affect the constrained variable favorably and cause satisfaction of the constraint $q$, but this would mean that it is a white knight. This is disallowed because then the clobberer will not be a clobberer. Such a plan will not necessarily satisfy the constraint $q$.
In the last case, we show that if there is an establisher and no clobberer, then we can construct a plan in which the constraint is necessarily satisfied. Since there is an establisher, the constraint is necessarily satisfied in a situation t. Every clobberer possibly violates q. For every clobberer that actually does violate q, there must exist a white knight lying before each C to nullify its effect. (Since there is no clobberer in the final plan). Since there are only a finite number of steps in the plan, eventually there will be a white knight that ensures satisfaction of the constraint in s.

**algorithm ABSTRACT-OPERATOR**

**Theorem:** Given the list of primitive valve operations to achieve a goal state, the algorithm will detect every abstracted operator corresponding to the set of valve operations required for starting or stopping flow of material from each source, s_j to the point of constraint specification, e_i, necessary to achieve the goal state.

**Proof:** (By contradiction) Let us assume that the algorithm is unable to detect an abstracted operator corresponding to start of flow from a certain source of material, s_j to an equipment, e_i where the constraint is specified. This implies that the algorithm did not mark the valves that need to be operated to achieve flow from s_j to e_i. No paths from e_i to s_j could be found in the final state. Since the goal state is correctly specified, if flow from s_j to e_i is to occur, then a path should exist between these points in the final state (=STATE). This path will not be found only if FIND-PATH does not find the correct path from s_j to e_i. This contradicts the fact that FIND-PATH is a provably correct algorithm (Papadimitriou and Steiglitz, 1982). Therefore the assumption is wrong, and the algorithm will find the set of operators that achieve flow from s_j to e_i.

In the case of stopping flow from s_j to e_i, the path between these points exists in the initial state, and the valve that is closed to stop the flow is given by means-ends analysis. Arguments
similar to those given above may be used to prove the theorem for finding abstracted operators corresponding to stopping of flow. Q.E.D.

**Complexity Analysis:** Time taken by each operation in the algorithm:

- Removing closed valves from STATE: $O(V)$
- FIND-PATH ($e_i$, $s_j$): $O(V + E)$
- Intersection: $O(V \ln V)$ (Since each valve cannot occur more than once in each list of operators)

Therefore, complexity of the algorithm: $O(g (V + e (V + E + V \ln V)))$

Also, $e = O(V + E) = \text{maximum number of equipment in plant}$

Complexity $= O(g (V + (V + E)^2 + (V + E) V \ln V))$

$= O(g (E^2 + VE \ln V + V^2 \ln V))$

**Algorithm Clobberer-Detection:**

**Theorem:** Every abstract operator that will result in violation of a quantitative constraint if implemented first and alone will be detected as a clobberer. (Assuming that QUANTITATIVE-PROPAGATE is correct).

**Proof:** (By contradiction) Assume that an abstract operator that will result in violation of a quantitative constraint if operated first and alone is not detected as a clobberer. This implies that the value propagated to the point of constraint specification does not violate the constraint. Since QUANTITATIVE-PROPAGATE is correct, this is possible only if the value of the variable at the source is such that it will not violate the constraint when flow is started by executing the abstract operator first and alone. Therefore, the abstracted operator is not a clobberer, which contradicts the previous assumption. Therefore, the algorithm will detect all the abstract operators that will result in constraint violation if executed first and alone as clobberers. Q.E.D.
Complexity Analysis: The procedure QUANTITATIVE-PROPAGATE takes $O(e)$ time where $e$ is the number of equations through which the quantitative value of the constrained variables is propagated. Also, $e = O(V+E)$. This propagation is done for each abstracted operator (a) and for each quantitative constraint (q). The maximum number of abstracted operators is equal to the number of sources of material in the plant = $O(V)$. Therefore, total complexity of the algorithm is $O(qa(V+E)) = O(qV(V+E))$.

**algorithm** WHITE-KNIGHT

**Theorem:** The algorithm will detect every manipulation that will possibly avoid a constraint violation as a white knight. (Assuming that QUALITATIVE-PROPAGATE is correct).

**Proof:** If a constraint of the form '$X .\bowtie k$' is violated, then it means that $X$ has a value .\bowtie than $k$. To avoid constraint violation, the value of $X$ needs to be changed in (\bowtie .\bowtie ) direction.

Propagation of 'increase/decrease $X$' through the equations by QUALITATIVE-PROPAGATE will not result in ambiguity because, only one qualitative value is being propagated at a time through the equations. All the variables that could affect the value of $X$ are present in the model equations for $X$. Given that the model equations are correct, a search through the equations will lead to all the manipulated variables that could affect $X$, and the direction of change required.

Complexity Analysis: For the quantitative constraint being violated, the first step is to generate the proper load, $L$. This takes time equal to the types of constraints that may occur, which is constant. The procedure QUALITATIVE-PROPAGATE takes time equal to the number of constraints, i.e. $O(V+E)$ for this simple case, with only one variable and everything else constant. Therefore, total complexity of the algorithm is $O(V+E)$.

**Theorem:** The final plan is guaranteed to satisfy the constraints in the situation being considered. If no plan is found, or it does not halt, no solution exists.
Proof: This theorem and its proof are analogous to Chapman's Correctness/Completeness Theorem for TWEAK. The modal truth criterion for quantitative constraints gives the conditions that have to be satisfied to ensure satisfaction of the constraints. The nondeterministic goal achievement procedure gives only finitely many ways of producing a new plan from an old one, and eventually each of these ways is examined. The theorems given above prove that the algorithms for detecting clobberers and white knights are both correct and complete. Thus all clobberers and white knights will be found. The control structure goes through all the constraints one by one and tries to satisfy them. Any operator that is a clobberer will be detected. If there is any way of satisfying a constraint but is not given in the primitive operators, the white knight identification procedure will find it. Completeness of the white knight identification algorithm means that all possible white knights will be ultimately explored and if a solution exists, it will be found. If no solution is found, or the planner does not halt, no feasible plan exists.
CHAPTER 5

DESIGN ISSUES IN PLANNING OPERATING PROCEDURES
5.1 INTRODUCTION

In the first four chapters of this thesis I have described a nonlinear (constraint-posting) planning methodology for the automatic synthesis of operating procedures in complete chemical plants. In particular we saw efficient methodologies for handling situations where temporal and qualitative mixing constraints. In all of the examples shown, however, I have implicitly assumed that a feasible plan exists. No attempt has been made to determine, a-priori, whether or not a feasible plan exists. Furthermore I have not yet addressed the issue of what is to be done if no feasible plan exists. In this chapter I shall describe some of the reasons why feasible plans may not exist, and present algorithms for generating modifications to the flowsheet structure so that feasible plans may be found.

We shall see in subsequent sections that the cases in which no plan is found correspond to the cases where no white knights can be found to counteract the effects of clobberers detected by the planner. Thus the methodology for generating design modifications can center around flowsheet structure changes which introduce into the plant equipment that can serve as white knights for whatever clobberers have been detected. This fact provides a focus for the design of the algorithms which come up with the design modifications.

At this point it is important to bear one fact in mind. When generating a complete goal state with which to start the planning process, the planner will construct one of several goal states that are consistent with the partial state specification declared by the user. Thus, if no feasible plan is found at this stage, it does not necessarily follow that none exists. This is because there may be another feasible goal state (also consistent with the partial state specification provided by the user) for which a feasible plan can be found. Thus, before investigating the possibility of making potentially expensive design modifications to the flowsheet, the user should investigate all of the feasible goal states to see whether one of them allows the generation of a feasible plan.

132
5.2 CAUSES FOR NON-EXISTENCE OF A FEASIBLE PLAN

During the planning of operating procedures for complete chemical plants, a very important stage is the one in which the computer tries to identify whether there is a possibility that one or more of the operating constraints will be violated. This is analogous to the detection of Clobberers in Chapman's program TWEAK. As is the case in TWEAK, when Clobberers are detected, they must be countered by a suitable White Knight unless they are subdued by demotion or promotion. In the planning methodology presented in this thesis, the Worst Case State lies between the Initial and Goal States, and promotion or demotion after or before these two states is clearly not possible. Thus the only alternative is to for the program to locate a White Knight to counter the Clobberer.

It should be obvious that there is no guarantee that a suitable White Knight will exist to counter each Clobberer that is detected. In those cases where no suitable White Knight can be found to counter at least one of the Clobberers, no feasible plan will be found. In these instances a modification must be made to the process flowsheet so as to introduce structures which will serve as White Knights. In this chapter I describe algorithms which allow a computer based methodology to automatically generate the necessary process modifications.

Let us look more closely at the situation where a Clobberer has been detected. In the case of qualitative mixing constraints this involves an overlap of influence graphs in the Worst Case State. The White Knights that are needed to counter this Clobberer are:

(a) An Acceptable Valve Separation Set and

(b) A Purgative and a Purge Route.

In this analysis, I shall assume that the absence of White Knights is due to one of the above two situations. Thus the modifications to the flowsheet must provide either an Acceptable Valve Separation Set or a Purgative and Purge Route (or both) depending on the needs of the situation at hand. The remainder of this chapter provides a methodology for accomplishing both these tasks.
5.3 ALGORITHMS FOR GENERATING DESIGN MODIFICATIONS

In this section I present the algorithms which are to be used by a computer program that attempts to make design modifications to a flowsheet structure so as to allow the system to generate feasible operating plans where none existed before. The algorithms are classified into two types: those that are used to ensure the existence of an Acceptable Separation Valve Set, and those that are used for generating a Purge Source and a Purge Route.

5.3.1 Flowsheet Modifications for Generating an ASVS

The methodology for proposing flowsheet modifications to ensure the existence of an Acceptable Valve Separation Set is described in this section.

STAGE 1: (Determine a Minimal Pipe Separation Set)

(1) Construct the Reduced Pipe Network corresponding to the Topological Graph of the Worst Case State.

(1.1) If the current node represents a PIPE or a "super source", do nothing.

(1.2) If the current node represents any other type of processing equipment, do the following:

(1.2.1) Delete the current node from the Topological Graph.

(1.2.2) Let U be the set of nodes incident to the in-edges of the current node.

(1.2.3) Let V be the set of nodes incident to the out-edges of the current node.

(1.2.4) For every pair (u,v) where u is in U and v is in V, create a directed edge (u,v) and add it to the Topological Graph.

In the transformed graph, the only nodes are pipes and "supersources" and an edge leads from one node to another iff there is a path connecting the corresponding nodes in the Topological Graph, and no other pipe or source lies on this path.

134
(2) Now, replace all directed edges with undirected edges. The problem of finding a Minimal Separation Pipe Set (MSPS) now reduces to the problem of determining a vertex separator in the Reduced Pipe Network for the pair of supersources for the dangerous species under consideration. A polynomial time algorithm exists for determining a vertex separator (see Even, 1979).

The Minimal Separation Pipe Set may not be an Acceptable Separation Pipe Set for the given planning problem. This is because it is possible that some pipe in the MSPS is required to have flow through it in the Goal State, or the user may simply have some preference regarding the state of flow through some of the pipes. However, from a heuristic standpoint, the Minimal Set is a good starting point for evolving into an ASPS, since disturbing the flow in these pipes upsets the overall flow only minimally. The procedure for evolving from an MSPS to an ASPS is described in Stage 2.

STAGE 2: (Determine an Acceptable Separation Pipe Set)

(1) Let the current Separation Pipe Set of edges be the MSPS already determined.

(2) Identify the pipes (if any) in the current SPS which are constrained to have positive or negative flows in the Goal State. If there are no such pipes, then the current SPS is feasible. Go to (3). If, on the other hand, there are some such pipes, mark them as unsatisfactory and proceed to Step (4).

(3) Present the MSPS graphically to the user (by highlighting the pipes on the flowsheet for example). Ask the user to specify any pipes which he or she prefers to keep open. Mark all such pipes as "unsatisfactory". If there are no unsatisfactory pipes, return the current SPS as the Acceptable Separation Pipe Set.

(4) Remove all "unsatisfactory" pipes from the Reduced Pipe Network. Go to STAGE 1, Step 2.
Once an Acceptable Separation Pipe Set has been found, the final stage of the methodology can be started:

STAGE 3: (Generation of an Acceptable Valve Separation Set)

1. For each pipe in the Acceptable Pipe Separation Set locate a corresponding pipe in the Topological Graph. For each of the pipes located in the Topological Graph, place a Valve on the pipe. The set of valves which are added to the flowsheet in this step will form an Acceptable Separation Valve Set for the planning problem under consideration.

Thus if the planner fails to come up with a feasible plan, and the reason for failure is that no Acceptable Separation Valve Set was found, the above algorithm may be used to propose flowsheet modifications (in the form of extra valves) to create an appropriate Acceptable Separation Valve Set.

5.3.2 Flowsheet Modifications for Purge Sources and Routes

Another reason why the planner may fail to find a feasible plan is that no suitable purgative or purge route can be found. The modification of the flowsheet structure to allow for this purging is described in this section. The algorithms provided here will, however, only address the structural modifications which need to be performed, and will not address the chemistry-related issues of deciding on a suitable chemical species to introduce as a purgative. This aspect of the problem can be addressed by encoding chemistry knowledge in the form of a rule-based system or by asking the user to choose an inert component. We also make the assumption that all of the flows in the Worst Case State are defined. This may not necessarily be the case, but as yet no algorithm is known which will propose structural modifications to the flowsheet in the absence of such flow information.
ALGORITHM CREATE-PURGE-ROUTE

(1) Construct a copy of the Initial State Topological Graph. In this graph close all of the valves which are members of the Acceptable Separation Valve Set. This will partition the Topological Graph into two subgraphs $G_A$ and $G_B$ each of which will contain the source of only one of the two dangerous components A and B respectively.

(2) Locate all the nodes in $G_A$ where B was present in the Initial State. Let the subgraph which consists of these nodes be called $G'$. Similarly, locate all the nodes in $G_B$ where A was present in the Initial State and create a subgraph $G''$ which consists of these nodes and their incident edges. For each of the graphs $G'$ and $G''$ do the following:

\begin{enumerate}
  \item[(2.1)] Find all the Minimal and Maximal Nodes in $G'$ (or $G''$). (For a definition of Minimal and Maximal Nodes see Appendix to Chapter 2).
  \item[(2.2)] Create a Super Source of the chemical species that was identified (either by the user or by a rule-based system) as a suitable purgative. For each Minimal Node in $G'$ introduce a pipe node and appropriate connecting edges leading from the Super Source to the Minimal Node.
  \item[(2.3)] Create a Sink for the chemical species used as a purgative. For each Maximal Node in $G'$ introduce a pipe node and the appropriate connecting edges leading from the Maximal Nodes to the Sink.
\end{enumerate}

At the end of the construction process above, the flowsheet will have been suitably modified so as to allow the purging of one dangerous species before admitting the second.
5.4 INDUSTRIAL CASE STUDY

In this section we examine two modifications of the EDTA plant discussed earlier in this thesis. The purpose of the illustration is to demonstrate the working of the algorithms which generate flowsheet modifications when

(i) no Minimal or Acceptable Separation Valve Set can be found and

(ii) no Purge Source or Purge Route is available.

5.4.1 Generation of Flowsheet Modifications for Creation of an ASVS

In this example the flowsheet of the Ethylenediaminetetraacetic acid process has been modified by removing the following valves: v10, v11, v13 and v20. The result of the modifications is shown in Figure 5.1.

When the planning algorithms are run on this plant no Minimal Valve Separation Set is found since, in the worst case state, there are no valves which will prevent hydrogen cyanide and formaldehyde from mixing in the region of overlap. Thus the computer will try to use the methodology described in Section 5.3.1 to come up with flowsheet modifications that will allow the computer to find an MSVS.

The algorithms described in Section 5.3.1 are run on the modified plant, and the computer suggests that a valve be placed between the water inlet and the junction just above it. This modification is shown in Figure 5.2 and this valve serves as the MSVS in the planning problem.

5.4.2 Flowsheet Modifications for Purge Source and Purge Route

The second example which is used to illustrate the design methodologies is a modification to the EDTA problem as follows. The structure of the flowsheet is exactly the same as the one presented in Chapter 3. The only change to the problem is the statement that sulphuric acid and formaldehyde should not be allowed to come into contact with each other.

It may be recalled that in the initial analysis sulphuric acid was used as a purge species. It is
Figure 5.1 Flowsheet of the modified Ethylenediaminetetraacetic Acid Plant
Figure 5.2 Flowsheet of the Ethylenediaminetetraacetic Acid Plant with MSVS
obvious that this is not possible anymore since one of the dangerous components, namely formaldehyde, cannot be brought into contact with sulphuric acid. Thus the computer must generate new piping and source and sink structure to allow the computer to find a purge route.

The algorithm CREATE-PURGE-ROUTE described in Section 5.3.2 is run on the Worst Case State of the original EDTA plant with the added mixing constraint, and piping modifications are generated in the flowsheet. The final flowsheet with the purge route is shown in Figure 5.3.

Note that, as mentioned earlier, the program does not attempt to specify what component is actually used as the inert purgative. A rule-based system with extensive knowledge about explosive mixtures is needed to accomplish this task. The construction of such a knowledge-base, while useful, has little academic value and is assumed to be outside the scope of this research undertaking. The program is designed only to propose the piping and valving modifications to the flowsheet.

5.5 SUMMARY

In this chapter we present the first systematic effort towards automating the design tasks associated with the planning of operating procedures for chemical plants. The algorithms presented here build on the planning methodology presented in earlier chapters in this thesis and address the problem of dealing with situations where the existing flowsheet structure does not admit any feasible operating plan which satisfies all of the operating constraints.

The algorithms are easily implemented on top of the existing methodology as they use such constructs as the Topological Graph and the Reduced Valve Network which were introduced during the course of this thesis.

The working of the algorithms developed in this chapter is demonstrated using two examples. Both illustrations involve modifications of the Ethylenediaminetetraacetic acid (EDTA) plant. In the first case study, a situation in which there is no acceptable valve separation set is encountered and the algorithms are used to generate flowsheet modifications which allow for an
ASVS. In the second example, no purge route is found and the methodology presented in Section 5.3.2 is used to generate a purge source and a purge route.

At this point we should mention the following limitations:

(i) The design algorithms only address the case where the operating constraints are of the boolean qualitative mixing type. Future work will have to address the cases where the more general quantitative constraints are important.

(ii) The design modifications are state dependent in that for each consistent Goal State there is a potentially different Worst Case State and hence a different design modification. There is a need for a methodology to systematically compare all of these design modifications and select the most economic ones.
CHAPTER 6

STARTUP OF CONTINUOUS CHEMICAL PLANTS
6.1. INTRODUCTION

In previous chapters, we have discussed the problem of planning plant-wide process operations in general terms. However, it is often the case that a particular class of planning problems has specific features which may be exploited to improve on the performance of a general purpose operations planner. Or, it may be the case that the specific needs of a particular class of planning problems are not satisfactorily addressed by the general purpose planner. One such case is the planning of startup procedures in continuously operating chemical plants. In this chapter, we shall discuss some of the features of this problem which set it apart from other planning problems, and propose some modifications to the general planning methodology discussed in earlier chapters.

6.2 CHARACTERISTIC FEATURES OF THE STARTUP PROBLEM

The characteristic features of the problem of planning startup procedures has the following special characteristics which set it apart from the other classes of planning problems.

6.2.1 Two Phases of Startup

In industry, the startup of a plant begins at low throughput (Lau, 1982). That is, the initial phase involves establishing steady state with low feed rates into the system, and operating at low, or moderately low temperature and ambient pressure. During this initial period, the startup operations consist of valve manipulations which induce material, energy and momentum flows in the system. Control loops are activated when needed to regulate process variables at safe and economically favorable levels. This phase of the startup has also been referred to as the Flow Build-Up stage (Doldan, 1986).

After steady state at low throughput has been achieved, the plant is brought up to the designed operating conditions by one of two methods:

- moving the plant gradually through a series of feasible steady states, or
• using constraint control.

This second phase of the startup procedure has been analysed in the literature (Han, 1970; Brooks, 1979; Kao, 1980) and we shall not address this problem here. On the other hand, few attempts have been made to put forward an explicit strategy for synthesis of startup procedures during the first stage described above. In this chapter, the framework for such a methodology, utilizing techniques from graph theory, the theory of computation and artificial intelligence, is proposed.

6.2.2 Startup Goals can usually be Pre-specified

If we examine each individual piece of equipment in a chemical plant we can identify distinct functions that it is required to perform in order that the plant operate in a feasible manner and at the same time achieve its processing objectives (Lau, 1982; Nishitani and Westerberg, 1982). These functions form the Goals in the initial plan specification of startup for the given piece of equipment. In the general case of planning process operations, these goals must be specified by the user during the Problem Formulation Phase. In the case of startup, however, these goals can be compiled (or deduced) ahead of time, thus improving significantly the ease of using a computer-based planning methodology. Similarly, the initial state of the equipment is also generally known a-priori, since the equipment is usually devoid of any material and is open to the atmosphere in the initial state. It is thus possible to generate an initial state specification for the individual units before the actual structure of the chemical flowsheet is known.

Once the goals have been identified for each unit, it is necessary to define how they will be combined in order to provide consistent state specifications for the complete chemical plant. This is accomplished using the equation-based models of the equipment coupled with the topology of the flowsheet. Techniques for planning using these pre-compiled initial and goal states are the subject of this chapter.

Some common types of equipment and their Unit Goals are listed below:
Heat Exchanger

- cold side fluid is flowing
- hot side fluid is flowing
- heat exchange is achieved

Pump

- pump is primed with fluid
- fluid is pressurized

Tank

- liquid level is achieved
- outlet flow is started

In order to achieve these goals, certain operations must be performed. For instance, in order to achieve the Heat Exchanger Goal **cold-side-fluid-flowing**, we must open the inlet and outlet valves to the cold-side of the exchanger. Typically, process operating personnel need not be told these valve operations explicitly, and we can represent the two valve operations as one operation, **establish-cold-side-flow**, assuming that the human operator will be able to translate this statement into the appropriate set of valve operations. There are two reasons why we make this simplification, or *abstraction*, as it is commonly called. First, by grouping several valve operations under one common operation, the computer has a vastly reduced number of possibilities to consider. More importantly, we deliver information to the operator in a form which he or she can easily comprehend.

6.2.4 The Need for Structural Decomposition

When planning and executing the startup of a chemical plant, it is customary to decompose the flowsheet into sections of units. This decomposition differs somewhat to the structural hierarchical decomposition which was discussed in Chapter 2 of this thesis. In that chapter, sets of
equipment were grouped together for convenience in specifying constraints and goals associated with the planning problem. However, the individual sections of the plant interacted with each other at all times, and these interactions restricted the number of planning alternatives considerably. In the case of startup it is customary to augment the basic structure of the chemical plant with auxiliary equipment (such as piping for a pump around loop, or extra hold-up tanks) so that when a structural decomposition is made, these sections of the plant may be operated independently for an extended period of time. This increases significantly the number of feasible startup alternatives and consequently more efficient pruning techniques must be used.

A number of commonly accepted rules-of-thumb, or heuristics, have been developed in industry which assist the designer in achieving an effective decomposition. Some of these are:

(i) a decomposition in which the number of process streams leading from one sub-section to another is small, is probably better than one which has a very large number of such streams.

(ii) information about time constants of the interactions between sections can be useful in deciding where to partition the plant - interactions which are sluggish provide a convenient point for inserting a boundary between two sections.

(iii) by grouping units according to their common functionality*, we can reduce the startup problem to the problem of scheduling the startup of groups of equipment.

(*By functionality, we mean the key operation, such as separation, which a group of equipment - a separator and its auxiliary equipment - is required to achieve.

The rationale behind decomposition of a problem into sub-tasks has been studied extensively in the areas of artificial intelligence (Stefik, 1981; Hayes-Roth, 1985; Sacerdoti, 1977; Mostow, 1985) mathematics and philosophy.

Some of the specific benefits obtained from decomposing the startup problem into weakly interacting sub-problems are the following:

(i) It allows the designer to focus all his/her problem solving skills on each of the sub-sections of the plant in turn. The entire problem is usually too complex for him/her to tackle all
at once.

(ii) By starting up sections of the plant in parallel, it may be possible to reduce startup time and hence favorably influence the process economics.

(iii) In cases where the safe startup of critical equipment - a reactor in which a strongly exothermic reaction is taking place, for example - is important, it is usually necessary to have all other processing units fully operational before starting up the critical piece of equipment. In such cases, grouping units according to common functionality is a good approach.

(iv) The actual implementation of startup is made clearer to the operator if the startup is achieved in sections of equipment which share common functional purpose.

(v) Safe recovery from unexpected startup errors or accidents is facilitated by carrying out the startup section by section.

6.2.5 Safety

Apart from the coordination problem which comes about because of the decomposition approach, we have the added objective of achieving start up in a safe manner. This involves not only the so-called hard constraints such as: "Hydrogen and oxygen should not be present at the same time in any of the pipelines of the plant," but also involves soft constraints: "When a process vessel is being filled, avoid leaving it unattended, since there is a possibility that the fluid may overflow." While the first type of constraint can be handled by the Boolean-logic methodology of Rivas and Rudd (Rivas and Rudd, 1974), no mathematical formulation for the second type of constraint is currently known.

A computer program which automates the synthesis of startup sequences for chemical plants must have a mechanism for understanding these safety constraints and recognising situations during startup which would lead to actual or potential violations of these constraints. If this can be achieved, it will be possible for the machine to completely eliminate startup alternatives that violate hard constraints, and favor those alternatives which have few potential constraint violations.
6.2.6 Minimizing Startup Time

So far the discussion has focussed mainly on problems dealing with feasible startup of chemical plants, with some attention given to the economic impact of using a decomposition approach. There is, however, another cost associated with starting up a chemical plant. It was mentioned earlier that the startup of the plant was accomplished in two stages, and that the initial stage was carried out at low throughputs. A continuous chemical plant is designed to operate at a given cost-optimal steady-state, thus operating for any length of time at sub-optimal levels is uneconomic. Thus one of our objectives is to minimize the time taken to bring the plant upto optimal steady-state operation.

Achieving the startup of a chemical plant involves performing a set of specific operations on individual processing units in the plant. The duration of each of the individual operations is, in general, independent of the order in which the operations are performed. Thus, if we follow a simple sequential approach to startup, we cannot significantly affect startup time. On the other hand, if we somehow manage to startup individual units, or even groups of units, in parallel, clearly there will be a reduction in startup time. In many cases, however, this parallel startup is in conflict with other capital and operating costs. For instance, consider the case of two distillation columns in series with heat integration between them. In order to start these columns up in parallel, extra dollars must be invested in heaters to satisfy the additional heating load since we cannot take advantage of heat integration and start up the two columns in parallel.

In addition, starting up many sections of the plant in parallel requires a large staff of operators to oversee the process. The same task, done sequentially, could be implemented and supervised by fewer operators, since their attention is not divided among several tasks which are to be performed at the same time. This is yet another illustration of the multi-objective nature of the problem.
6.3 PREVIOUS WORK

There are two major works in the chemical engineering literature that are devoted to the synthesis of operating procedures for the startup of complete, continuously operating chemical plants.

6.3.1 Startup of Conventional Chemical Processes

In 1982 Lau developed a methodology for planning the startup procedures for conventional chemical plants. By 'conventional' we mean those plants which have the following easily identifiable sections: (a) pretreatment section, (b) reaction section, (c) separation section and (d) refining section. The methodology utilizes the idea of predefined unit goals but makes no attempt to use efficient methods of constraint transformation for the generation of the operating procedures.

6.3.2 Startup of Complex Chemical Processes

A systematic methodology for a human designer to develop the startup operating procedures for complex chemical plants was presented by Doldan, et al. (1985) The methodology required the human to define the basic starting block namely, the Elementary Technical Unit, and then to follow a systematic procedure of enumeration of the startup alternatives. Although this approach was the first to address the startup of complex chemical plants, it suffered the following limitations:

(i) The definition of the Elementary Technical Unit was not easily implemented in a computer program.

(ii) The enumeration and screening procedure used heuristics which were applied in an ad-hoc manner by the human, and no attempt was made to describe how to automate this process on a computer.

(iii) The methodology used a generate-and-test approach that was combinatorially explosive.
6.4 METHODOLOGY FOR PLANNING STARTUP OPERATIONS

The specific methodology for the planning of startup operations is described in this section. The methodology, like the general methodology, consists of two phases: (a) The Problem Formulation Phase and (b) The Plan Generation Phase. The complete methodology is described within the context of a simple industrial case study and is followed by the application to a more complex plant.

6.4.1 Industrial Case Study - Gasoline Synthesis Plant

In order to illustrate more clearly the problems involved in planning the startup of a chemical plant, we shall describe the procedure in the context of an actual industrial plant. The process we will be following (see Figure 6.1) is the Universal Oil Products Solid Phosphoric Acid Catalyst Process that produces gasoline from light hydrocarbons (Arkun, 1979; Lau, 1982).

The feed composed of olefins and paraffins enters the polymerization reactors after being preheated by heat exchange with the effluent from the reactors and by separate feed heaters. In the reactors, dimerization of the olefins takes places. Due to the highly exothermic nature of the reaction, a recycle propane stream is used as a quench to provide interstage cooling between the individual catalyst beds. The reactor effluent is processed through a sequence of distillation columns to separate propane, butane and the gasoline product streams.

6.5 DESCRIPTION OF THE PROBLEM FORMULATION PHASE

The Problem Formulation Phase is composed of three basic parts:

(i) Specification of the Initial State
(ii) Specification of the Goal State
(iii) Specification of the Constraints on the Operation

Each of these stages is described in detail below:
6.5.1 Initial State Specification

In the typical startup problem, the initial state of the plant is completely defined. This is because the equipment is usually empty to begin with, and therefore the initial state is just the state of the equipment when it is open to the atmosphere. Due to the fact that the initial state is standard for the startup problem, it is possible to capture the initial state in compiled form within the model of the equipment. This is indeed the approach which is taken here. This leaves the user with the sole task of graphically creating the flowsheet, as the initial state description is automatically then set up behind the scenes.

6.5.2 Goal State Specification

The specification of the Goal State description can also be automatically achieved. By comparing the models for the initial state and the steady-state continuous operation of the equipment it is possible to automatically deduce the Unit Goals ahead of time. This process of automatic generation of unit goals is described below. Clearly, once the computer has generated the Unit Goals for a particular type of equipment they can be compiled and used in a variety of flowsheets. Thus, by specifying the graphic flowsheet, the user has completely defined both the Initial and the Goal States.

The fact that startup Unit Goals may derived and compiled ahead of time is crucial to the extra efficiency which is achieved by this specialised methodology. The main reason for this is that the inefficiencies of qualitative and quantitative constraint propagation are avoided.

Automatic Generation of Unit Goals from Steady-State Models: The automatic generation of the Unit Goals centers around the achievement of the user defined "streams" or "terms" in the steady-state equation-based models. In Chapter 1, these streams or terms were described as the building blocks on which the hierarchical modelling structure is built. The streams and terms are built in turn on lower-level streams and terms as well as individual variable-objects. The general
object of achieving the first phase of startup (as described in Section 6.2.1) is to achieve or establish all of the top level terms in each of the individual units of the plant. Clearly, in the Initial State, none of these is established and the task of generating Unit Goals is now one of identifying the top-level independent terms. This can be done using the following simple algorithm which parses the hierarchical network of objects.

**ALGORITHM FIND-TOP-LEVEL-STREAMS**

**Input:** The Structured Modelling Equations describing the Steady-State Operation of the Unit  
**Output:** The corresponding Unit Goals

```plaintext
begin  
Q := the set of equations;  
terms := Ø  
for eq in Q do  
begin  
terms-internal := top-level-terms in eq;  
for term in terms-internal do  
    if term is a subterm in one of the members of terms then  
        do remove term from terms-internal;  
    else if one of the members trm of terms is a subterm in term then  
        do remove trm from terms;  
end;  
end;
```
6.5.3 Knowledge Representation of Partial Plans

In this section I discuss issues regarding the representation of partial startup plans for individual process units using the pre-compiled unit goals. The knowledge representation scheme discussed here should be visualized as a layer on top of the models which were described in Chapter 1 of this thesis. I then describe how these pre-compiled models can be automatically combined using structural information derived from the flowsheet to form partial plans for groups of equipment.

Partial Plans for Individual Units: Modelling of chemical processing equipment has traditionally been carried out in a variety of ways. The most prevalent of these is the mathematical approach. Here, algebraic and differential equations are formulated which represent the fundamental laws of conservation of mass and energy, and the second law of thermodynamics. One approach to modelling equipment units during startup analysis would be to construct detailed dynamic models of the processing equipment and numerically simulate the startup of the plant. This is an extremely expensive and time consuming task, and is usually considered impractical, if not impossible. If we can tackle the problem using simpler and intuitively appealing models, much computational efficiency can be gained, and the results can be delivered in a form that is easily interpreted. To this end, I propose that a processing unit be modelled in two parts:

- a Structural Flow Model
- a Transition Network

These two models are described in the following sections.

Structural Flow Model: Although a processing unit is usually visualised as a single entity, it is really composed of one or more sub-units or compartments which are physically separated from each other, such that material transfer between sub-units is not possible. For instance, a jacketed continuous-stirred-tank-reactor consists of a reaction vessel into which reactants are charged and a
jacket through which a heating medium is circulated. The two sub-units are physically separated to prevent intermixing of the fluids. We would like the structural model of the CSTR to contain the following information:

- that there are two sub-units
- that the only interaction between them is in the form of energy transfer.

The Structural Model of the jacketed CSTR depicted below by nodes and arrows (see figure 6.2) makes this information explicit. In this structural model, each node represents a self-contained compartment which is physically isolated from the others. Solid arrows indicate material flow through the sub-units, and greyed arrows indicate energy transport. These structural models are essentially the lowest level of the hierarchical model that represents the flowsheet structure and consists of the simple subsystem which make up the processing unit.

![Diagram of a jacketed CSTR and its structural model](image)

*figure 6.2*

As will be shown later, this form of representation enables the program to identify the need for additional startup equipment, as well as facilitating the combination of Unit Transition Networks,
described in the next section.

**Unit Transition Networks:** In addition to the structural models described above, it is also necessary to have a model which represents the partial plans of the processing unit as operations are performed. In the case of a steady state simulation, the process state is simply the vector of the process variables. The representation technique proposed here is based on the *Transition Networks* used in natural language processing, and *Deterministic Finite Automata* used in theoretical studies of computability.

Consider a process heat exchanger before any of its Unit Operations have been performed. None of its Unit Goals have been achieved. We can therefore write:

$$
\neg \text{HEX-GOAL-1} \& \neg \text{HEX-GOAL-2} \& \neg \text{HEX-GOAL-3}
$$

where: $$\neg X$$ is an abbreviation for "not (ACHIEVED X)"

This conjunction of negated goals describes the initial state of the heat exchanger. If we apply one of the heat exchanger operators, say **HEX-OP-1**, we achieve the first goal, but not the others. The state of the exchanger is now:

$$\text{HEX-GOAL-1} \& \neg \text{HEX-GOAL-2} \& \neg \text{HEX-GOAL-3}$$

The final state is the conjunction of all three goals. Including the start and end states, there are 8 permutations of the conjuncted goals and their negations. However, since there are physical constraints on the order in which the operators can be applied, not all these states are physically achievable. The precedence ordering for the heat exchanger is:

$$\text{HEX-GOAL-1} \Rightarrow \text{HEX-OP-3}$$

$$\text{HEX-GOAL-2} \Rightarrow \text{HEX-OP-3}$$

This precedence ordering is the analog of writing a partial plan for a single piece of equipment. A *Transition Network* is a network consisting of a starting node, an ending node, and *k* intermediate nodes. Nodes are connected by directed graph edges which are labelled with the Unit Operations of the processing equipment. A node labelled with a particular operation **OP** connects two nodes.
State\((i-1)\) and State\(^i\) iff carrying out OP when the processing unit is in State\((i-1)\) will take the unit to State\(^i\). From this definition, the Unit Transition Network (UTN) for the heat exchanger can be constructed (see figure 6.3). The greyed arrow indicates that the final operation is a process that occurs spontaneously once the first two goals have been achieved.

![Transiton Network for Heat Exchanger](image)

*Figure 6.3*

The Transition Networks for the equipment in the Gasoline Synthesis plant are shown in figures 6.4 and 6.5. These diagrams contain all possible paths by which individual units may be taken from no goal status (depicted by a circle with a tick mark) to full goal status (depicted by a "bulls-eye") and contain embedded information about the partial plan for each piece of equipment.
TRANSITION NETWORKS FOR EQUIPMENT IN GASOLINE SYNTHESIS PLANT

**PUMP**

```
 PUMP-OP-1 -> S1 -> PUMP-OP-2
```

**HEAT EXCHANGER**

```
 HEX-OP-1 -> S1 -> HEX-OP-2 -> S3 -> HEX-OP-3
 HEX-OP-2 -> S2 -> HEX-OP-1
```

**STEAM HEATER**

```
 HTR-OP-1 -> S1 -> HTR-OP-2 -> S3 -> HTR-OP-3
 HTR-OP-2 -> S2 -> HTR-OP-1
```

*Figure 6.4*
TRANSITION NETWORKS (continued)

PRESSURE CONTROL VALVE

SPLITTER

JUNCTION

POLYMERIZATION REACTOR

WATER COOLER

Figure 65
Properties of Unit Transition Network Models: Having described the Unit Transition Network, we explain some of the properties of the networks that will make the scheduling problem easier.

Transition Networks Constrain the Solution Space: It was pointed out earlier, that the set of physical constraints governing individual processing units are completely contained within the unit's Transition Network. Thus, if we were to generate the set of alternative startup sequences using the Transition Networks as a basis, we would only generate potential solutions which are consistent with these physical constraints. In this manner, I avoid being as simple-minded and inefficient as when we attempted a straightforward linear planning approach. I have made use of constraints that govern the physical behavior of our system to focus our attention on a sub-space of the combinatorially large solution-space. In fact, in our case, the exact size of the sub-space of solutions generated from UTNs can be compared with the size of the complete solution-space as described in the next section.

Metric for Estimating Size of the Solution Sub-Space: In the previous section, it was pointed out that using the Transition Network to model the state-transition behavior of processing equipment allows us to focus our attention on a particular sub-space of possible solutions. It is useful at this point to have some method of determining how effectively the physical restrictions contained in the Transition Networks cut down the number of alternatives which we have to consider. The two formulae presented below are metrics which directly allow us to evaluate the pruning.
Total Number of Sequences = \[
\left[ \sum_{i=1}^{N} G_i \right]!
\]

Number of Sequences Remaining = \[
\left[ \sum_{i=1}^{N} G_i \right]! \cdot \prod_{j=1}^{N} P_j \prod_{k=1}^{N} G_k!
\]

where:

\( G_i \) = Number of Unit Goals for Equipment "i"

\( P_i \) = Number of Paths from Zero Goal Status to Full Goal Status for Equipment "i"

The first formula gives the total number of alternatives which would be generated using the generate-and-test paradigm, while the second (see derivation in appendix) gives the number of sequences which would be produced by blind enumeration starting from a Transition Network representation.

If we look closely at the second formula, we see that the factorial still dominates, but is attenuated by a factor which approaches a constant less than 1 raised to the nth power. However, we have thus far only used physical constraints which apply within individual processing units, and have still to incorporate information about physical constraints between pieces of equipment. As long as we do not go ahead and actually generate all these potential solutions until we have sufficiently constrained the solution space, we are "still in business".
Transition Networks Prune the Solution Space Conservatively: Clearly by using the Transition Network representation, we reduce the number of alternatives that we have to consider. But how can we be sure that we have not left out some potentially viable solution?

Let us examine the case that some feasible solution was missed by the generation technique using UTNs. This feasible path will consist of a series of "states" with operators leading from one state to the next. Each state is "feasible" within the physical constraints which were placed on the individual pieces of equipment. And each of these states contains a set of goals which have been achieved - one set for each piece of equipment. According to our hypothesis, this feasible path was "overlooked" by the generation technique which used the UTN representation. This means that one or more of the states on this feasible path must not be "reachable" using the UTNs. The first state on the feasible path is the state where no goals have been achieved. This state is obviously reachable using the UTN representation, and is in fact the first state on any path generated from a UTN. So let us "walk" along our "overlooked" feasible path and find the first state that could not be reached by the UTN technique. Call this the "first unreachable state". The state directly preceding the "unreachable" state is "reachable" (the "unreachable" state was the first one of its kind on the feasible path). In this last state reachable from the UTN, certain Unit Goals have already been achieved, and after applying the next operator on the feasible path, one more Unit Goal is achieved, and the system moves to the "unreachable" state. However, in the definition of a Unit Transition Network, we stated that an operator connected two states if and only if applying the operator when the process was in the first state would take the plant up to the second state. Our earlier reasoning about an "unreachable" state contradicts the "if" clause of the UTN definition, leading us to conclude that our initial hypothesis about leaving out a feasible path was erroneous.

Unit Transition Networks do not know about the physical constraints between pieces of equipment which arise due to the network of interconnections in a chemical plant. However, since they constrain the solution space conservatively, addition of new constraints in the future will only
serve to further constrain the solution space.

6.5.4 Specification of Operating Constraints

In the previous section I described a knowledge representation scheme which embedded the operating constraints within a structure which represented the partial plan for the particular processing unit. In this section, I describe how the user may input information about the operating constraints and have the computer automatically generate the partial plan representation for the particular piece of equipment in question. Let us take the example where we create the Transition Network for a Heat Exchanger.

To begin with, we need to identify the functional goals which are to be achieved by the equipment, and the operations that will achieve these goals. In the case of the Heat Exchanger, the functional goals are:

\[ G1 : \text{'cold side fluid is flowing'} \]
\[ G2 : \text{'hot side fluid is flowing'} \]
\[ G3 : \text{'heat exchange is achieved'} \]

and the operations which achieve these goals are:

\[ OP1 : \text{'establish cold side fluid flow'} \]
\[ OP2 : \text{'establish hot side fluid flow'} \]
\[ OP3 : \text{'allow heat exchange to take place'} \]

Note that these operations and the goals they achieve can be easily specified. They are the standard operations which an operator is told to perform on heat exchangers. The user is also required to specify whether a particular operation occurs spontaneously, or not. To begin constructing the Transition Network, the computer creates the start node and asks the question:

Which of the following operators can I apply right away?

\[ OP1 : \text{'establish cold side fluid flow'} \]
\[ OP2 : \text{'establish hot side fluid flow'} \]
OP3 : "allow heat exchange to take place"

To which the user replies:

OP1 : "establish cold side fluid flow" and
OP2 : "establish hot side fluid flow"

The computer now creates the following Graph (internally) (figure 6.6):

![Graph](image)

*figure 6.6*

and then asks:

If the "**cold side fluid is flowing**", which of the following can I perform?

OP2 : "establish hot side fluid flow"

OP3 : "allow heat exchange to take place"

The user then replies:

OP2 : "establish hot side fluid flow"

Which leads the computer to expand upon the graph as shown below (figure 6.7):

160
Now the computer asks:

If the "hot side fluid is flowing", which of the following can I perform?

**OP1**: "establish cold side fluid flow"

**OP3**: "allow heat exchange to take place"

To which the answer is, of course:

**OP1**: "establish cold side fluid flow"

The computer now does two things. It first generates a new node as before, which represents the conjunction of \(G1\) and \(G2\). It then compares this new state with the state \(S3\). Recognising that they are one and the same state, it merges them as shown in *figure 6.8*. 
The computer now does not have to ask any more questions and can directly arrive at the final form of the Network (see figure 6.3). Since the third operation was specified as being spontaneous, the arc corresponding to this operation is greyed.
6.6 THE PLAN GENERATION PHASE

In the methodology for the planning of startup operating procedures, the second phase
namely, the plan generation phase consists of the following stages:

(i) Transformation of Unit Constraints and Interaction Constraints into Partial Plan
(ii) Synthesis of Complete Plans.
(iii) Coordination of Startup Sections

First look at what the consequences of our earlier decomposition are.

6.6.1 Consequences of Structural Decomposition

Decomposition of a chemical plant for the purpose of synthesizing operating procedures
reduces a large scale problem into a set of smaller sub-problems with one major expense: a
coordination analysis must be performed among this set of sub-problems. Consequently, the
criterion for using a decomposition approach should be that computational complexity is reduced,
plant-site implementational efficiency is improved (Lau, 1982), or both. Clearly the decomposition
must be achieved in such a way that the coordination effort is minimized.

To see what we mean by a coordination analysis, let us take a look at our industrial example.
We have a conventional chemical plant which we divide into four sections according to the
functionality of the equipment.

• a pre-treatment section, where a feed stream gets heated - partly by heat exchange with the
reactor effluent, and partly by external heating
• a reaction section
• a recovery section and
• a refining section

We notice that there are process streams which pass between sections of the plant. If we were to
tear these streams at the section boundaries (which is what is implied by the decomposition), we
have to find some way of simulating the function of these streams within each section of the plant.
For example, in order that the recovery section be considered as an individual entity, we must have storage tanks containing a mixture of propane, butane and gasoline polymer, ready to supply the feed stream to the first distillation column, and and pumps to circulate (recycle) the product and by-product streams from the column until the refining section is ready for them.

Suppose we decide to start up the plant following the most common sequencing order used in industry, namely, \textit{recovery $\rightarrow$ refining $\rightarrow$ pre-treatment $\rightarrow$ reaction}. We realise that this problem does not arise, since we can feed the product and by-product streams directly into the refining sections which are then started up in parallel. We do need, however, the other storage tanks and raw materials mentioned, since, when the recovery section is being started up, the reaction section is not yet operational.

Other problems occur when we try to startup the pre-treatment section independently of the reaction section. Due to the fact that there is heat integration between the two sections, we see that unless excess heat is supplied by the steam heater, we may not be able to heat up the feed stream enough so that when it enters the reactor, reaction occurs. This means increased capital costs, since the steam heater may have to be redesigned to accommodate the increased heating load; and increased operating costs because of the need for additional utilities. These various costs will have to be anticipated, and justified, when selecting an appropriate decomposition strategy.

\subsection*{6.6.2 Unit Constraint Transformation}

Having described the methodology for modelling individual partial plans, it is now time to address the issue of how these individual partial plans can be combined with the structural models to generate partial plans for groups of equipment. As before, the discussion will focus on two aspects of modelling:

- Structural Models for Groups of Processing Equipment, and
- Extension of Transition Networks to Handle Groups of Equipment.

In each case, methodologies are described which automatically synthesize the compound models
from the individual "cells".

The explanation of the approach is more clearly understood with a practical example, and here we consider the pre-treatment section of the Gasoline Synthesis Plant.

**Structural Model for a Group of Equipment** The Structural Model for a group of processing equipment is a simple extension of the concept applied to individual units. The model consists of a set of nodes which again represent sub-units which are physically isolated from each other, except that now solid arrows are drawn between sub-units of one piece of equipment and those of adjacent units whenever there is material flow through a pipe connecting the sub-units. *Figures 6.9 and 6.10* illustrate the pre-treatment section of the Gasoline Synthesis Plant and the Structural Model derived from it. The advantage of this representation scheme over previous Process Flow Digraph representations is that in the Structural Model energy interactions between sub-units are made explicit, and are clearly differentiated from material flow interactions.

**Partial Plans for Groups of Equipment** In a similar manner that we extended the Unit Structural Model to cover groups of equipment, it is possible to extend the UTN concept as well. The methodology is somewhat more involved, but still relatively straightforward. It was mentioned earlier that in addition to physical ordering constraints within an individual equipment, there are constraints *between* pieces of equipment which specify a partial ordering on Unit Operations. These inter-unit constraints can be inferred from the Structural Model of the group of units, making additional use of the fact that each arc in the Structural Model can be labelled with the type of flow achieved by the particular operation which the arc represents (e.g. mass, momentum, or energy). For our particular example, these inter-unit constraints are as follows:

- **PUMP-GOAL-1 > HEX-OP-1**
- **HEX-GOAL-1 > HTR-OP-1**

Let us now take the *union* of all of the individual Unit Goals and call this set the "Unit Goals" for
PRE-TREATMENT SECTION OF GASOLINE SYNTHESIS PLANT

Figure 6.9

STRUCTURAL MODEL FOR PRE-TREATMENT SECTION

Figure 6.10
the group of equipment (consisting of pump, heat-exchanger and steam-heater). We could now, theoretically, write down all of the constraints which were used to generate the individual Unit Goals, add to them the inter-unit constraints mentioned above, and we could go ahead and construct a Group Transition Network in much the same way that we did for individual units. This would be wasteful, however, since we have already done some of the construction in the first step when we generated the Unit Transition Networks. Instead we propose an algorithm to combine existing Transition Networks (whether they be for a single unit or for a group of units) along with external (structural) constraints, to generate a Composite Transition Network. This Composite Transition Network contains all of the constraint information which was contained in the individual Unit Transition Networks and incorporates, in addition, the inter-unit constraints.

**Creating Partial Plans for Groups of Equipment from Unit Partial Plans** In the next two sub-sections we shall describe techniques for automatically combining the unit models - both structural and UTN - to form the corresponding models for groups of equipment.

**Combining Structural Models for Groups of Equipment:** The Structural Model in figure 6.10 was constructed from individual Structural Models for the pump, heat exchanger and the steam heater along with the process flowsheet. The methodology for constructing the Structural Model is a trivial bookkeeping exercise. Every inlet and outlet stream in a given processing unit is pre-defined to correspond to a particular solid arrow in the Structural Model for the same processing unit. (For adding new types of equipment, this correspondence can be established graphically and in a menu-driven style). As the Process Flow Sheet is constructed, the connection between the nodes in the Structural Model is established by keeping track of the process streams which connect the individual processing units.
Combining Unit Transition Networks: The methodology for combining Transition Networks is described below.

(i) First, all the *inter-unit* constraints are identified (from the Structural Model). In our case, as pointed out in the previous section, the constraints are:

- **PUMP-GOAL-1** > **HEX-OP-1**
- **HEX-GOAL-1** > **HTR-OP-1**

The goals which must be achieved before a particular operator can be applied are called the *pre-conditions* for that operator.

(ii) Form a list of pairs, where each pair has as its first element, the operator which achieves one of the preconditions stated in the inter-unit constraints, and as its second element, that operator whose pre-condition is the first element. Call this list the *constraint list*. (The constraint list for our example is shown below)

\[ \text{Constraint List} \leftarrow (\text{PUMP-OP-1} \cdot \text{HEX-OP-1}) (\text{HEX-OP-1} \cdot \text{HTR-OP-1}) \]

(iii) Construct a *composite end-node* and label it with the *conjunction* of all of the goals for the individual processing units which form the group. For our example, the composite end-node would be:

\[ \text{Goals Achieved} \leftarrow \{\text{PUMP-GOAL-1} \& \text{PUMP-GOAL-2} \& \]

\[ \text{HEX-GOAL-1} \& \text{HEX-GOAL-2} \& \text{HEX-GOAL-3} \& \]

\[ \text{HTR-GOAL-1} \& \text{HTR-GOAL-2} \& \text{HTR-GOAL-3} \}

(iv) Create a list which contains this node and label this list the *current node-list*.
(v) Now form a list of edges which are incident to the *end-state* of each of the individual Transition Networks. Call this list the *current candidate set*.

(vi) Let the *current node* be the *composite end-node*.

(vii) If any element of the *current candidate set* is labelled with an operation which is the first element of a pair in the *constraint list*, and the operation which is the second element of the same pair does not appear on the path between the *current node* and the *composite end-node*, delete that arc from the *current candidate set*. (This may sound confusing, since to begin with, the current node is the *composite end-node*, but as the algorithm progresses, the *current node list* will contain nodes which are further and further away from the *composite end-node.*) When all such nodes have been deleted, create one arc for each of the arcs in the *current candidate set*, and label these arcs with the same operations as those of the arcs in the *current candidate set*.

(viii) Now make the arcs in the *current candidate set* incident to the *current node* (in this case, the *composite end-node*), and for each of the arcs, create a new state node. Each new state node is labelled with all of the goals of the current node (here, the *composite end-node*), except the goal corresponding to the operation on the arc leading backwards to that node. This particular goal is negated. This means, for example, the arc labelled PUMP-OP-2 will originate from a new node which has the following goal set:

\[
\{ \text{PUMP-\text{GOAL}-1} \land \neg \text{PUMP-\text{GOAL}-2} \land \text{HEX-\text{GOAL}-1} \land \text{HEX-\text{GOAL}-2} \land \text{HEX-\text{GOAL}-3} \land \text{HTR-\text{GOAL}-1} \land \text{HTR-\text{GOAL}-2} \land \text{HTR-\text{GOAL}-3} \}\]

At this point, the Composite Transition Network for our example looks like this:
(ix) Now remove the current node from the current node-list and push all of the new nodes on the end of the list. Pick the first node on the current-node-list and call this the new current-node.

(x) If the current-node has been labelled with the conjunction of the negation of all of the goals, then TERMINATE. Otherwise, go back to step (vii).

At the end of the procedure mentioned above, the Composite Transition Network should look something like figure 6.12. This Composite Transition Network shares one feature with the Unit Transition Networks, the pruning has been conservative. So far, only those constraints which we know to be hard physical constraints have been used, thus guaranteeing, as before, that no feasible alternatives are pruned from the solution space. However, this means that we still have to deal with a (possibly large) number of infeasible alternatives which have escaped the initial pruning. If the number of alternatives remaining is small (on the order of about 10) then this may be a sufficiently small set to present to the user. On the other hand, as is our current situation, we may still have a large number of alternatives, and further analysis is necessary. In any case, it
would be nice to have some way to estimate the size of the solution space, and as it turns out, for this particular graphical representation, namely the Transition Network, a simple, efficient method for doing this is available (see next sections).

6.6.3 Estimating the Size of the Solution Space

We have been doing as much as we can to constrain the space of solutions. It would be useful to have a means of estimating the efficiency of the pruning we are achieving at any given point. As it turns out, the Transition Network representation is a knowledge representation scheme which conveniently allows such estimates to be made. To explain how this is so, we have to open a parenthesis and mention some results from graph theory.

**Incidence Matrices for Group Transition Networks:** The Group Transition Networks which we have been discussing so far can be considered as a special class of a mathematical structure, namely the Directed Graph, or Digraph defined as follows (Even, 1979):

A Digraph $G(V, E)$ is a structure which consists of a set of vertices $V = \{v_1, v_2, \ldots\}$ and a set of edges $E = \{e_1, e_2, \ldots\}$; each edge $e$ is incident to the elements of an ordered pair of vertices ($u$, $v$) which are not necessarily distinct. The vertex $u$ is known as the start-vertex, and the vertex $v$, the end-vertex.

Actually, the Group Transition Networks have certain special properties of their own.

(i) Each Network has a pair of distinct nodes, $\{s, t\}$, where $s, t$ are elements of $V$, and every path from $s$ to $t$ traverses the same number of edges $L$, where $L$ is the total number of Unit Goals to be achieved in starting up the group of equipment.

(ii) No edge has the node $s$ as its end-vertex, and no edge has the node $t$ as its start-vertex.

A convenient way of representing Digraphs is through the use of an Incidence Matrices defined as follows:
Let $G(V, E)$ be a Digraph and let the nodes, $V$, be labelled with the Natural Numbers $1 \leq i \leq |V|$. The Incidence Matrix, $M$, is defined as the square matrix of dimension $|V|$, with element $m_{ij}=1$ iff the edge $(v_i, v_j)$ exists, and 0 otherwise.

As we will see in the next section, it is useful to have the Incidence Matrix for the Group Transition Network.

**Metric for Determining Current Size of Solution Space** In this section, a metric is presented, using the Incidence Matrix for the Group Transition Network described in the previous section, which allows us to determine the size of the solution sub-space which is consistent with the current set of constraints implied by the GTN. Given a particular GTN which represents a set of equipment, every distinct path in the network leading from the node $s$ to the node $t$ is a potential startup alternative which must be investigated. If we had a way to count all of these paths without actually tracing them, we could gauge the current size of our solution sub-space. Consider the following theorem (proof given in appendix)

**Theorem:** Let $G(V, E)$ be a directed graph with unweighted edges and incidence matrix $M$. Let the nodes of $G$ be numbered $v_i$, $1 \leq i \leq n$, such that the subscript $i$ denotes the ordinality of the row/column in $M$ which corresponds to the vertex $v_i$. The number of distinct paths of length $l$ leading from a vertex $v_j$ to a vertex $v_k$ is given by the $(j,k)^{th}$ element in the matrix $M^l$.

We pointed out earlier, that the length of each path leading from $s$ to $t$ was equal to the total number of Unit Goals which have to be achieved in order to start up the group of equipment. If this number is $L$, then the number of distinct paths leading from $s$ to $t$ is given by the appropriate entry in the matrix $M^L$.

Matrix multiplication is a somewhat expensive operation. Although we can make use of a compact representation (the matrix $M$ is upper triangular) to speed things up, a much faster
algorithm also exists. The actual computer implementation of the startup methodology utilized this algorithm which involves labelling the nodes in the transition network, *while it is being constructed*. The size of the space is determined by simply looking at the label on the start node, rather than by performing a series of matrix multiplications. The matrix theorem is presented here since it illustrates the concepts behind the metric more transparently than the description of the labelling algorithm.
6.6.4 Synthesis of Complete Plans for Processing Sections

Having described the modelling techniques used to represent a chemical process, we now present a methodology for using these models to synthesise startup schedules for complete chemical plants. The methodology is illustrated with the industrial case study described earlier.

**Heuristics: Preferences Developed out of Industrial Experience** In the preceding sections we have described several types of knowledge which help us to partially specify the order in which unit operations should be carried out in starting up a given chemical plant. We have mentioned physical constraints which are present within individual processing units, as well as those which arise between units as a result of the flow structure of the specific plant under consideration. We have also pointed out that there are processing constraints, both hard and soft, which eliminate certain startup alternatives, thus serving to further specify the partial ordering of the operations. In practice, however, there still remains an exceedingly large set of alternatives (as will be illustrated in the next section), and further screening must be done to choose from among the remaining possibilities. Several heuristics have been developed through practical experience and are used in industry to achieve the final pruning (Lau, 1982):

(i) establish material flow through each unit first; exceptions arise in those units which require energy and momentum flows to ensure safety.

(ii) schedule material and energy saving devices, e.g., recycles and heat integrations, to startup last.

(iii) minimize the number of situations where potential violations of soft constraints may occur.

(iv) whenever feasible, retain startup in the same section of the plant

(v) always maintain the possibility for safe fall-back.
Reducing the Number of Alternatives Using Heuristics

Constructing a Composite Transition Network prunes the space of alternatives from the sub-space that remains after the Unit Transition Networks have been constructed. We can use the metrics described in the previous section to check how efficient the pruning has been. For our example, the Unit Transition Networks were composed of the following:

**PUMP:**
- Number of Goals = 2
- Number of Paths = 1

**HEAT EXCHANGER:**
- Number of Goals = 3
- Number of Paths = 2

**STEAM HEATER**
- Number of Goals = 3
- Number of Paths = 2

Using the formula derived earlier, the number of possible solutions represented by the Unit Transition Networks is 2,240. (The total number of alternatives is 40,320). The number of alternatives in figure 3.11 is 430 (computed using the matrix size-of-space metric). This is clearly an impressive reduction from the total number of solutions, but is still too large a number to be presented to a human for further evaluation. It is at this point that we start looking at the knowledge which we earlier referred to as heuristic. These heuristics consist of information which can help us decide which of the sequences in a transition network are better than others.

It is important to notice the difference between a hard physical constraint such as:

"BEFORE HEAT EXCHANGE CAN BE ACHIEVED IN A HEAT EXCHANGER, THE HOT- AND COLD-SIDE FLUIDS MUST BE FLOWING"
and the heuristic statement:

"TO SAVE ON ENERGY, START UP THE COLD-SIDE OF A HEAT EXCHANGER BEFORE STARTING UP THE HOT-SIDE". 

We shall refer to a statement of the first type as a FACT, while a statement of the second type is considered to be an ASSUMPTION.

Two Basic Types of Heuristic are Used Heuristics are of two general types:

(i) General Heuristics

(ii) Situation Specific Rules.

Examples of the first type are:

(i) When achieving energy flows within a single unit, proceed from the coldest streams to the hottest. (So as not to waste energy by having a hot stream flowing without a cold stream to absorb the heat). Note that this refers to exchange carried out above ambient conditions, the opposite of this heuristic is used when carrying out refrigeration.

(ii) Operations whose sole purpose is to achieve heat integration should preferably be scheduled to occur last.

Situation specific rules include such statements as:

(i) Distillation columns should be operated at total reflux until tray hydraulics are achieved.

Heuristics Generate Assumptions By applying the heuristics to specific equipment units, we generate assumptions about the order in which operations should be performed. For example, applying the two general heuristics mentioned in the previous section, we generate a set of assumptions, as shown in figure 6.12.
HEURISTIC I: "DO COLD OPERATIONS AND THEN HOT"

HEAT EXCHANGER

COLD-SIDE → HOT-SIDE

STEAM HEATER

PROCESS FLUID → STEAM

HEURISTIC II: "ACHIEVE HEAT INTEGRATION LAST"

HEATING IN HTR → HOT-SIDE OF HEX

PUMP MOTOR → HOT-SIDE OF HEX

Figure 6.12
"FACTS" vs. "ASSUMPTIONS"

**FACT:** "BEFORE HEAT EXCHANGE CAN BE ACHIEVED IN A HEAT EXCHANGER, THE HOT AND COLD-SIDE FLUIDS MUST BE FLOWING"

**ASSUMPTION:** "TO SAVE ON ENERGY, START UP COLD-SIDE OF HEAT-EXCHANGER FIRST"
**Effect of Using Assumptions**  In the previous section, we stated that we were using heuristic knowledge, in the form of assumptions, in order to reduce the number of alternative startup sequences that the user has to consider. We saw that the assumptions have the same form as facts, except that they are not believed as strongly as the facts. Facts in our representation are statements of the physical constraints on the schedule of operations. Assumptions, while they are not hard constraints constrain the search space according to industrial preferences which have evolved into the rules of thumb, or heuristics. This is illustrated in *figure 6.13*. By making assumptions, we cut "holes" in the search space. (The grey region in the figure corresponds to the mass of arrows on the Group Transition Network.) The task of finding a startup schedule for the group of equipment involves finding a route from the start node (notated with a tick mark) to the end-node. With the holes cut in the search space, the number of alternatives we have to consider are greatly reduced.

This phenomenon is illustrated by the case study we have been considering. Recall that the transition network for the pretreatment section contained possible startup paths (*figure 6.14*). We now apply the first heuristic ("Do cold operations, and then hot") and we observe the "holes" which appear in the transition network. The number of sequences to be considered has now been reduced to 70 (*figure 6.15*). Applying another heuristic ("Achieve heat integration last") reduces the number of sequences to just 5 sequences (*figure 6.16*).

The only difference between these sequences is the point at which the pump motor is turned on, and this operation can be performed anytime before the process fluid is introduced into the steam heater.

The pruning results for the other sections of the plant are shown in *figure 6.17*.

**Heuristic Knowledge Deals With Conflicting Goals**  The heuristic knowledge we used in the previous section was not derived from any fundamental theory, but rather, from previous
PRUNING OF THE SEARCH-SPACE

ASSUMPTION-1

ASSUMPTION-2

figure 6.13
PRE-TREATMENT SECTION

NETWORK CONTAINS 430 PATHS

Figure 6.14
PRE-TREATMENT AFTER FIRST HEURISTIC

NUMBER OF SEQUENCES = 70

*Figure 6.15*
PRE-TREATMENT AFTER SECOND HEURISTIC

5 SEQUENCES REMAINING

Figure 6.16
## SUMMARY OF PRUNING RESULTS

<table>
<thead>
<tr>
<th>SECTION</th>
<th>TOTAL</th>
<th>HEURISTIC I</th>
<th>HEURISTIC II</th>
<th>OTHERS</th>
<th>FINAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRETREAT.</td>
<td>430</td>
<td>70</td>
<td>5</td>
<td>...</td>
<td>5</td>
</tr>
<tr>
<td>REACTION</td>
<td>20</td>
<td>...</td>
<td>...</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>RECOVERY</td>
<td>$1.4 \times 10^4$</td>
<td>$1.01 \times 10^{10}$</td>
<td>239200</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>REFINING I</td>
<td>2</td>
<td>1</td>
<td>...</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>REFINING II</td>
<td>2</td>
<td>1</td>
<td>...</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>REFINING III</td>
<td>2</td>
<td>...</td>
<td>...</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

*Figure 6.17*
experience with starting up chemical plants. When these heuristics are applied by the practitioners in the field, they are usually based on certain assumptions which are held implicit in the designer's mind, or in the mind of the expert who developed the piece of knowledge. By definition, an assumption is a statement which is not guaranteed, but is believed to be true. What happens if an assumption which a designer made turns out to be false? Well, if the designer is clever, he will be able to detect a contradiction when it arises, and use his knowledge, common sense and experience to figure out which of the assumptions he made was invalid. Our computer program, on the other hand has not (as yet) been given the ability to detect when a contradiction has occurred; nor does it have the natural reasoning ability to figure out how to back-track through its tree of decisions, and to try a new assumption.

**Conflicting Assumptions Form Directed Cycles** With our representation of facts, it turns out that there is a simple way of detecting conflicts which lie hidden in the assumptions made by the computer. Consider the set of facts (solid arrows) and assumptions shown in figure 6.14. Two of the facts state:

**OPERATION-1 > OPERATION-2**

**OPERATION-2 > OPERATION-3**

the two assumptions state:

**OPERATION-3 > OPERATION-4**

**OPERATION-4 > OPERATION-1**

Clearly, this is impossible! However, each of these assumptions, on its own, presents no problem: the combination of facts in figures 6.18 b-c are perfectly consistent. The difference lies in the fact that there is a directed cycle in the first situation while no directed cycle occurs in the other two cases. Thus, locating directed cycles in the set of assumptions and facts provides the computer...
CONTRADICTIONS

(a)

(b)

(c)

Conflicting assumptions set up directed cycles

Figure 6.18
with a means for detecting conflicts.

**Applying Assumption-Based Truth Maintenance Ideas** The *Assumption Based Truth Maintenance System* described by de Kleer, (de Kleer, 1986a, b, c) has some very nice features which enable program designers to think about how to represent assumptions and facts and, very importantly, to efficiently organizing the *contexts* in which assertions are made. This system of truth maintenance is used as the primary means of keeping track of the contextual relevance and validity of assumptions. Attractive features of the technique are simplicity, intuitive appeal and efficiency. One minor implementational extension of the mechanism is incorporated here. In the original formulation of the paradigm, it was not deemed necessary to keep track of all facts that remain valid in a given context. In our planning application, it is important to know the sequences of operations which are valid in any given context of assumptions. The data structures which are used to implement the methodology are suitably modified to allow this facility.

The general working of an ATMS is illustrated in *figure 6.19*. Given a set of 4 assumptions (A - D), any combination of these assumptions is treated as a *context*. When the ATMS is told, by the graph algorithm for detecting conflicts, that assumptions A and D together introduce a conflict, this context, and all contexts which are supersets of this context are also marked as "no good". This allows the user to make a set of assumptions and then have the ATMS *inform* him/her which pruned set of sequences is valid under which set of assumptions.

In this manner, the space of possible startup sequences can be efficiently screened for only those sequences which are (heuristically) better than the others.
6.6.5 Coordinating the Startup of Sections

Earlier I presented a methodology for generating and screening the possible ways of starting up a section of a chemical plant. However, as mentioned earlier, decomposing a problem into sub-problems implies a later coordination of the sub-problem solutions to produce a solution to the complete problem.

**Interactions Between Sections Set Precedence Constraints** As yet, no systematic methodology has been developed for examining all possible coordinations of sections in an efficient manner. However, by identifying the streams which connect sections, and by examining the cost of "simulating" these streams, one can specify a precedence ordering for the startup of the sections. This results in a PERT diagram (see next section for description) which describes the network of startup operations, taking into account the constraints imposed by the interaction of sections.

For the case of the Gasoline Synthesis plant, the only interesting interaction between the sections occurs between the pre-treatment and the reaction sections. *Figure 5.1* illustrates how the interactions between these sections leads to the construction of the operations network.

Having constructed the network, techniques from the field of operations research (described in the next sections) are useful for analyzing the performance of a given startup sequence. Various precedence orderings of the sections of the plant can be postulated, the networks generated, and the results of the analysis described below can be used to choose between the precedence orderings.

**Two Applications of PERT Digraphs** To address the problem of parallel startup in order to minimize downtime, the concept of the *PERT Digraph* is useful.

A *PERT Digraph* is a finite digraph $G(V, E)$ with the following properties:

(i) There is a vertex $s$, called the *start vertex*, and a vertex $t \neq s$, called the *termination*
vertex.

(ii) $G$ has no directed circuits.

(iii) Every vertex $v$ in $V\setminus\{s, t\}$ is on some directed path from $s$ to $t$.

The PERT digraph has the following interpretation (Even, 1979). Every edge represents a process. All processes which originate from the start node can be started right away. For every vertex $v$, the processes represented by edges leaving the node can be started when all the processes represented by the edges incident to $v$ are completed.

Two results from graph theory are interesting for analyzing such networks.

(i) Critical Path Method for determining Minimum Startup Time

(ii) Network Flow Technique for Allocation of Resources.

**Estimating Minimum Startup Time**  The Critical Path Method assumes that the edges of the graph are labelled with the time taken to complete the operations they represent. In the case of startup, shortcut mathematical models can be used to estimate these times. The method also assumes that there is no restriction on the number of resources. This means that any number of operations can be successfully performed simultaneously, without ever running out of operators to perform them. Thus, the CPM is a useful technique for estimating **lower bound** on the time taken to start up a section of the plant, and as long as our computer program is "aware" of the assumptions involved in using the technique, this is useful information. An algorithm of $O(E)$ time complexity exists for finding the longest path from start to finish in the digraph. Such a path is called a **critical path**, and the length of this path is a lower bound on the time taken to achieve all of the operations in the digraph.

In addition, determining the critical paths in the network helps operators to detect the
bottlenecks in the plant startup (which must lie on critical paths).

**Estimating Resource Requirements** The other interesting result from graph theory involves a network flow technique, and is aimed at obtaining an estimate on the "resource requirement" of the network. It assumes that, while an operation is being performed, one processor will be utilized - in our case, the attention of one operator will be occupied. The problem here is to find out what is the minimum number of processors (operators) required so that no process is delayed due to lack of a processor. A minimum flow problem is solved to find a set of concurrent edges. A set of edges is said to be concurrent if for no two edges in the set, there is a directed path which passes through both. The number of elements in the set of concurrent edges is a lower bound on the number of operators required. This is a very conservative estimate, and in actual fact, far fewer operators may be needed than is specified by the algorithm. However, it is still useful as a first approximation of the complexity of parallel start up of a chemical plant.
6.7 INDUSTRIAL CASE STUDY: A SECTION OF A HEAVY WATER PLANT

I now analyse the planning of startup procedures within the context of a more complex case study, namely, a section of an Isotope Exchange H₂S-H₂O Heavy Water Plant. The description of the plant follows. From a storage tank, treated water is fed into the absorption packed tower CR1004 where it is saturated absorbing most of the H₂S of the purge stream. Column CP1001 operates at low temperature (30°C) and a deuterium rich gas (DHS) exchanges the deuterium with the raw water enriching it. This takes place by means of an isotope exchange reaction. On the contrary column CP1002 operates at high temperature (130°C) enriching the gas in deuterium and depleting the water from the deuterium gained in column CP1001. A stream of deuterium enriched water is obtained between the columns. The water that leaves column CP1002 with less deuterium than the raw material and is fed into the stripper CP1007 being finally sent to waste after exchanging heat. A recirculating stream of cooled liquid is passed through the bottom section of the column CP1001. This stream dehumidifies and cools the hot gas. Similarly, the cold gas from CP1001 passes through a compressor and is humidified and heated by a recirculating stream in the bottom section of column CP1002. Appropriate heat exchange is done for energy saving purposes.

6.7.1 Problem Formulation Phase

The problem formulation phase in the case of planning of startup procedures is achieved simply by constructing a graphical depiction of the flowsheet on the computer. This is done using the graphic interface capabilities of the planning program described earlier. As described in earlier sections, the initial and goal states are already compiled for the individual equipment. By constructing the graphical flowsheet and establishing the connections between equipment, the initial and goal states for the entire plant are generated by the computer.
6.7.2 Decomposition of the Process

In the case of the heavy water plant, there are no easily identifiable sections of the conventional kind, such as pretreatment, recovery, etc. In addition, the equipment are interconnected very tightly. Thus the decision on how to decompose the process is made along the lines of key equipment and their auxiliary equipment.

The key equipment identified in the heavy water plant are as follows:

(i) Purge column CR 1004
(ii) Column CP 1001
(iii) Column CP 1002
(iv) Stripper CP 1007

The augmentation of these units with their auxiliary equipment are shown in Figure 6.21.

6.7.3 Unit Constraint Transformation for Augmented Units

Having divided the plant into augmented units, the next step is for the program to generate the partial plans for each of the sections. This is done following the procedure outlined in Section 6.6.4. Using the metric for determining the size of the solution space, we can estimate the number of complete plans for each of the augmented units. The results are tabulated below.

| AUGMENTED PURGE                      | 6 |
| AUGMENTED COLUMN CP1001              | 478 |
| AUGMENTED COLUMN CP1002              | $5.7 \times 10^{23}$ |
| AUGMENTED COLUMN CP1007              | 568 |

It appears from the numbers above that the startup of individual augmented units can be achieved fairly easily with the exception of, perhaps, the augmented column CP1002. However, once the combinations of augmented units with each other are computed, the results are combinatoric. It is
FIG. 6.20: FLOWSHEET OF THE ISOTOPIC EXCHANGE FIRST STAGE OF A HEAVY WATER PLANT
Figure 6.23: A portion of the transition network for the augmented unit cp1007

568 ALTERNATIVES
Figure 6.24: Transition network for the Augmented Purge Section
necessary to use heuristics to screen out some of the alternatives.

6.7.4 Heuristics Used in the Pruning

The following heuristics have been used in the pruning of the startup alternatives (aside from the ones already used for the Gasoline Polymerization Plant).

1. Start liquid flow through columns before gas
2. Start bottoms flow in columns before tops
3. For safety reasons do not vent H\textsubscript{2}S to atmosphere in large quantities
4. Do not operate the stripping section before augmented CP 1002

Applying the heuristics above to the coordination and startup of the equipment we can generate a transition network for the complete plant.

6.7.5 Synthesis of Concrete Plans

Even after the application of the several heuristics that are outlined above, the size of the transition network is extremely large and it is not illustrative to the designer to view all of these alternative startup routes. It is thus necessary for some sort of optimization to be carried out in order to identify the most promising of the feasible startup alternatives. In this research undertaking on the planning of operating procedures, I have concentrated mainly on the generation of feasible operating procedures and have not paid much attention to rigorous optimality.

The program is thus expected to provide a feasible and not necessarily optimal startup procedure. One such procedure which is also consistent with the heuristics which provide for better operation is presented below.

BOTTOMS-FLOW-CP1001 > COLD-SIDE-FLOW-IP1003 > COLD-SIDE-FLOW-IE1001
> TOP-FLOW-CP1001 > BOTTOMS-FLOW-CP1002 > HOT-SIDE-FLOW-IP1003 >
COLD-SIDE-FLOW-CPIH1001 > COLD-SIDE-FLOW-IP1002 > TOPS-FLOW-CP1002 >
COLD-SIDE-FLOW-IH1002 > LIQUID-FLOW-CP1007 > HOT-SIDE-FLOW-IP1001 >
HOT-SIDE-FLOW-IP1002 > COLD-SIDE-FLOW-IP1001 > FLOW-THROUGH-UI1001
> FLOW-THROUGH-UI1003 > WATER-FLOW-CR1004 > COLD-SIDE-FLOW-IE1003
STEAM-FLOW-IE1001 > STEAM-FLOW-IH1002 > STEAM-FLOW-IE1003 >
VAPOR-FLOW-CP1007 > VAPOR-FLOW-CP1002 > VAPOR-FLOW-CP1001 >
FLOW-THROUGH-KC1001 > VAPOR-FLOW-CR1004

The operating procedure above is presented to the user who can then ask the computer to generate
the next procedure and so on.
6.8 SUMMARY

In the first 5 sections of this thesis I presented a generalised methodology for the synthesis of operating procedures for complete chemical plants. In this chapter I have discussed some of the characteristics of the startup problem which set it aside from the other a-priori planning problems and which dictate that a special purpose methodology be developed for planning startup procedure. In particular the need for decomposition of the plant for startup purposes and the fact that startup goals can be precompiled by the computer allow for a much more efficient planning methodology.

The methodology described in this section had the same general components of the general purpose methodology, namely the Problem Formulation and the Plan Generation Phases, and the notion of constructing partial plans by transforming operating constraints. The major improvement in efficiency lies in the fact that the decomposition of the plant into sections allows us to develop partial plans within a section and also among sections.

The algorithms and the analysis thereof are presented in detail and are then applied to first a simple, conventional, continuously operating chemical plant and then to a more complex system.
APPENDIX

A.1 Number of Startup Alternatives Metric

*Formula:* If the State Diagram method is used to represent startup constraints on individual process units, and if the total number of process units is $N$, then the number of possible startup sequences is given by the following formula:

$$\text{Number of Sequences Remaining} = \left[ \sum_{i=1}^{N} G_i \right]! \cdot \prod_{j=1}^{N} P_j \prod_{k=1}^{N} G_k!$$

where:

$G_i = \text{Number of Unit Goals for Equipment "}i\text{"}$

$P_i = \text{Number of Paths from Zero Goal Status to Full Goal Status for Equipment "}i\text{"}$

*Derivation:* Consider a single process unit. Let us assume that there are $p_1$ distinct paths by which the equipment can be started up. By picking any one of the $p_1$ paths, we completely specify the startup procedure for the single process unit. Now consider a second and a third unit with $p_2$ and $p_3$ paths respectively. Even if we choose one path for each of the three units, the startup procedure has not been completely specified. We still have to decide how to *interleave* the
individual operations on each of the paths. In other words, if we apply the operators one by one, after every application we must decide whether the next operator is going to come from unit1, unit2, or unit3. To make this easier to see, let us rephrase the problem slightly. Let us represent the operators of unit1 by the letter A, those of unit2 by B and those of unit3 by C. If there were n1 operations on each path of unit1, n2 on each path of unit2, and n3 on each path of unit3, then the problem is reduced to finding the number of distinct words that can be formed by the following letters:

n1 A's, n2 B's and n3 C's.

Here, we have assumed that we have already picked a path for each unit.

The total number of distinct words is given by:

\[
\frac{\left( \binom{n_1^+\ n_2^+\ n_3^+}{n_1\ n_2\ n_3} \right)!}{n_1!\ n_2!\ n_3!}
\]

The number of ways of choosing three paths, one from each of the units is

\[ p_1 \times p_2 \times p_3 \]

Thus the total number of sequences is the product of these last two terms. Generalizing this formula to the case of N process units, we obtain the required formula.
A.2 Incidence Matrix Metric

Theorem: Let $G(V, E)$ be a directed graph with unweighted edges and incidence matrix $M$. Let the nodes of $G$ be numbered $v_i$, $1 \leq i \leq n$, such that the subscript $i$ denotes the ordinality of the row/column in $M$ which corresponds to the vertex $v_i$. The number of distinct paths of length $l$ leading from a vertex $v_j$ to a vertex $v_k$ is given by the $(j,k)^{th}$ element in the matrix $M^l$.

Proof: By induction on the length of the path, $l$.

Induction Base: Clearly, for paths of length 1, the matrix $M^1$ satisfies the claim.

Induction Hypothesis: Assume that the theorem is true for $1 < p < l$.

Induction Step: The $(j,k)^{th}$ element of $M^l$ is given by:

$$m_{j,k}^l = \sum_{i=1}^{n} m_{j,i}^{(l-1)} \cdot m_{i,k}$$

According to the induction hypothesis, the term:

$$m_{i,k}^{(l-1)}$$

represents the number of distinct paths of length $(l-1)$ leading from node $v_j$ to node $v_i$.

If the element $m_{i,k}$ is non-zero, it represents a path of length 1 leading from node $v_i$ to
node $v_k$, and the product of the two terms represents the number of distinct paths of length 1 leading from node $v_j$ to node $v_k$ and passing through node $v_i$. By summing over all $i$, we obtain the total number of paths. Q.E.D.
7 BIBLIOGRAPHY


Han, C. D.: "Determination of Startup Conditions for Chemical Reactor Stability";


Kao, Y. K.: "A Simple Startup Policy for a Catalytic Reactor with Catalyst Deactivation";


Tate, A.: "Project Planning Using a Hierarchic Nonlinear Planner"; Department of Artificial
Intelligence Report No. 25, University of Edinburgh, Edinburgh, August 1976


Williams, B. C.: "Doing Time: Putting Qualitative Reasoning of Firmer Ground",
Proceedings, AAAI, 1986