NONMONOTONIC REASONING: THE SYNTHESIS OF OPERATING PROCEDURES IN CHEMICAL PLANTS

Chonghun Han, Ramachandran Lakshmanan,[1] Bhavik Bakshi,[2] and George Stephanopoulos

Laboratory for Intelligent Systems in Process Engineering
Department of Chemical Engineering
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

[1] Present address: Department of Chemical Engineering, University of Edinburgh, Scotland, UK.
[2] Present address: Department of Chemical Engineering, Ohio State University, Columbus, OH 43210, USA.

Planning a sequence of actions to take you from one point to another is usually a proposition whose inherent difficulty increases as more obstacles are placed in your way. The number of these obstacles (constraints), which you must skirt around, determines the complexity of the task, because any time you run into one of them you must *backtrack* and try an alternative step or path of steps. Such *serial* (or *linear* or *monotonic*) construction of a plan is fraught with pitfalls and repeated backtracking. The more the constraints, the more inefficient the monotonic planning. If, on the other hand, an action step (a Clobberer) leads to the violation of a constraint, then *do not backtrack*. Take another action step (a White Knight), which, when it precedes a Clobberer, negates the impact of the Clobberer, and you never need to backtrack. So, the more constraints, the more efficient your planning process. Such *nonserial* (or *nonlinear* or *nonmonotonic*) reasoning has become the essence of all modern and efficient planners, whether they are *logic-based* and *explicit*, or *implicit* enumerators of alternative plans. The purpose of this choice is twofold: (1) to introduce the ideas of *nonmonotonic reasoning* in the planning of process operations and (2) to demonstrate how nonmonotonic planning can be used to synthesize operating procedures for chemical processes, either off-line for standard tasks (e.g., routine startup or shutdown), or on-line for real-time response to large departures from desired conditions. It is shown that hierarchical modeling of process operations and operators is essential for the efficient deployment of nonmonotonic planning, and that the tractability of the resulting algorithms is strictly dependent on the form of the operators. In this regard, the ideas on modeling in this chapter draw heavily from the material of the first chapter in Volume 21 of this series. Nonmonotonic planners handle with superb efficiency constraints on (1) the temporal ordering of operations, (2) avoidable mixtures of chemical species, and (3) bounding quantitative conditions on the state of a process. Consequently, they could be used to generate explicitly all feasible operating procedures, leaving a far smaller search space for the selection of the optimum procedure by a numerical optimizer.

I. Introduction

The planning and scheduling of process operations, beyond the confines of single processing units, are fairly complex tasks. Thus, the synthesis of operating procedures for the routine startup or shutdown of a plant, equipment changeover, safe fallback from hazardous situations, changeover of the process to alternate products, and others, involves (1) a long list of noncommensurable objectives (operating economics, safety, environmental

impact), (2) models describing the behavior of all units in a plant, (3) human supervision and intervention, and (4) the degree of the required automation. While mathematical programming techniques, such as branch and bound algorithms for mixed-integer linear or nonlinear (MILP or MINLP) formulations, have been used (Crooks and Macchietto, 1992) to locate the "optimal" plan, computer-based process control systems (Pavlik, 1984) "do not know" how to plan, schedule, and implement complex, nonserial schedules of process operations. The latter is considered to be largely a supervisory task left to the human operator. Nevertheless, there exists a natural tendency for increased automation in the control room (Garrison *et al.*, 1986), which takes one or both of the following forms: (1) a priori synthesis of operating procedures and subsequent implementation through programmed-logic control systems or (2) online, automatic planning and scheduling of operating procedures with real-time implementation through distributed control systems. The former approach is favored for the routine tasks such as startup, shutdown, and changeover, whereas the latter is essential for the optimization of operating performance, the intelligent response to large departures from operating norms, and the planning of emerging fallback operations in the presence of unsafe conditions.

The synthesis of operating procedures involves the specification of an ordered sequence of primitive operations, such as, opening or closing manual valves, turning on or off motors, and changing the values of setpoints, which when applied to a process will change the state of a plant from some given initial state and eventually bring it to a desired *goal* state (Lakshmanan and Stephanopoulos, 1988a). Typically, the transformation from the initial to the goal state is attained through a series of intermediate states (Fig. 1), each of which must satisfy a set of constraints, such as

1. *Physical constraints*, imposed by mass, energy, and momentum balances, chemical and phase equilibrium conditions, and rate phenomena.
2. *Temporal constraints*, imposed by various considerations; e.g., you cannot reach the state of a full tank if the state of the feeding pump has not been previously turned on.
3. *Logical, mixing constraints*, to avoid explosive mixtures, poisoning of catalysts, generation of toxic materials, deterioration of product quality, and other consequences resulting from the unintentional mixing of various chemicals.
4. *Quantitative inequality constraints*, requiring that the values of temperatures, pressures, flowrates, concentrations, and material or energy accumulations remain within a range defined by lower and upper bounds, in order to maintain the safety of personnel and

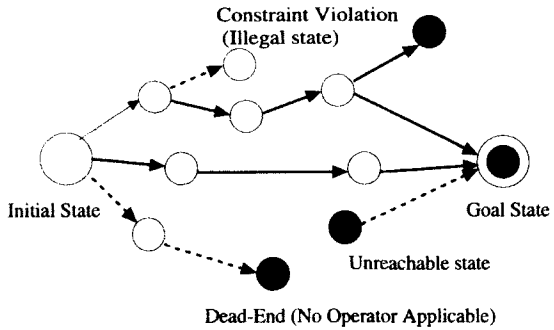


FIG. 1. Schematic of a typical planning problem (solid line signifies a feasible path and dotted line an infeasible path).

equipment, environmental regulations, and production specifications.

5. *Performance constraints*, in terms of operating cost, product turnaround time, and others.

A. PREVIOUS APPROACHES TO THE SYNTHESIS OF OPERATING PROCEDURES

Domain-independent theories of planning have attempted to generate formulations and solution methodologies that do not depend on the characteristics of the particular area of application, whereas *domain-dependent* approaches have attempted to capitalize on the specific characteristics of a given problem domain, in order to construct special-purpose planning programs. Although a domain-independent planning theory would be preferable, there exists sufficient theoretical evidence to suggest that building a probably correct, complete, domain-independent planner that is versatile enough to solve real-world planning problems, is impossible (Chapman, 1985).

Rivas and Rudd (1974) should be credited with the pioneering effort of synthesizing operating sequences that achieve certain operational goals, while ensuring that safety constraints were not violated. Chemical processes were modeled as networks of *valves* and *connectors* (e.g., pipes, vessels, unit operations), and the safety constraints were expressed as statements about the species that should never be present simultaneously in any location of the plant. The sequential logic algorithm that they proposed is equivalent to a restricted form of constraint propagation. Thus, high-level, abstractly expressed goals (input by the user) such as

"start-a-heater" were propagated to lower-level goals of higher detail and were converted into a series of valve operations. Unfortunately, keeping track of the dependencies in order to generate *explainable* plans (a highly desirable feature) was not possible, as it was impossible to account for additional classes of constraints.

Kinoshita *et al.* (1982) divided a plant into sections around "key" processing equipment (including peripherals and attached pipes and valves) and identified a list of individual operations for each section. During the first phase of their methodology, they generate a tree of all possible sequences of operations for each section. Each tree described the *transition relationships among the states* of each section, and was endowed with the following information: (1) minimum and maximum time required for each operation, (2) the cost associated with each operation, and (3) constraints on the states of adjacent units. During the second phase of their work, Kinoshita *et al.* attempted to coordinate the sequences of the independent trees in an effort to achieve consistency among the operations of interacting units, by adjusting the timing and thus the temporal order of the various primitive operations.

In a similar spirit, Ivanov *et al.* (1980) had previously constructed *state transition networks* for the representation of operating states taken on by a plant as it goes from the initial state to the goal state. While the nodes of the network represent the states of a plant, the edges (arcs) signify the primitive operations that carry the processing units from one state to the next. To account for quantitative performance measures, Ivanov *et al.* assigned to each arc a weighting factor, which depended on the type of the performance measure. Within such framework, the synthesis of the "optimal" operating procedure becomes a search problem through a set of discrete alternatives. Ivanov *et al.* applied this formulation to the synthesis of optimal startup sequences of operations.

Tomita, Nagata, and O'Shima (1986), on the other hand, represented the topology of a plant through a directed graph with the nodes signifying the constituent equipment and the directed edges representing the pipelines. The valves were assigned to the appropriate edges of the digraph. Propositional logic was used to express the topology of the digraph and the temporal ordering of the various operations, thus leading to the generation of logically feasible operating plans. Subsequent work (Tomita *et al.*, 1989a, b) focused on the use of logic-based techniques and other ideas from artificial intelligence toward the development of an automatic synthesizer of operating procedures.

The work of Fusillo and Powers (1987) has attempted to define a formal theory for the synthesis of operating procedures. First, it introduced more expressive descriptions of the plants and their behavior, going beyond the

Boolean formulations favored by the pioneers in the field. Second, the planning methodology allowed the incorporation of global and local constraints, all of which were checked at each intermediate state to ensure that the evolving operating procedure was feasible. Third, they introduced the concept of a "stationary state," i.e., a state at which the plant itself does not change significantly with time (e.g., can wait until next action is taken), and where the operating goals are partially met. The "stationary state" was a breakthrough concept. It allows a natural breakoff point from a large schedule of operations, thus leading to an automatic generation of *intermediate operating goals*, which simplify the search for feasible operating procedures by decomposing the overall planning problem to smaller subproblems. In addition, a "stationary state" represents a convenient stopping-off point for recovery from operational errors, thus allowing the process to retreat back to an intermediate state thereby avoiding a complete shutdown. In subsequent publications (Fusillo and Powers, 1988a, b) the same authors introduced local quantitative models, and they also carried out detailed studies on the synthesis of purge operations.

Foulkes *et al.* (1988) have approached the synthesis of operating procedures from a more empirical angle. They have extended the work of Rivas and Rudd (1974) for the synthesis of complex pump and valve sequencing operations, relying on the use of logical propositions (implemented as rule-based expert systems), which capture the various types of constraints imposed on the states of a processing system.

B. The Components of a Planning Methodology

From Fig. 1 it is clear that the synthesis of an operating procedure could start from (1) the initial state and proceed forward to the goal state, (2) the goal state and proceed backward to the initial state, or (3) any known intermediate state(s) and proceed in both directions to bridge the gap with the specified initial and goal states. Furthermore, it should also be clear that there exist many paths connecting the initial to the goal state, which can be evaluated in terms of (1) their *feasibility*, i.e., whether their corresponding constituent states violate the specified constraints or not, and (2) a *performance index*, i.e., associated cost, total operational time, or other. To check the feasibility of a path we need an explicit *set of constraints* determining the allowable states of the process, and to evaluate performance we need a quantitative description of the desired performance index (or a vector of performance indices). Finally, in order to be able to evaluate the state of the process, we need *models* that describe the

underlying physicochemical phenomena and the variety of primitive operations.

No planning methodology can be efficient or credible for the synthesis of process operating procedures, if it focuses only on the search for the "best" plan, delegating the modeling needs and the articulation of the requisite constraints to inferior tasks. The reverse is also true. Consequently, a concise methodology for the synthesis of operating procedures should provide a unified treatment of all three pivotal components, namely

1. Modeling of processing systems and primitive operations.
2. Explicit articulation of all constraints that an operating procedure should satisfy.
3. Search over the set of discrete plans and the identification of the feasible, and possibly of the "optimal" among them.

1. Formal Statement of the Operations Planning Problem

Consider a processing system composed of N subsystems, whose topological interconnections are determined by a set of streams, $S = \{s_{i,j}^{(p,q)}\}$; $i, j = 1, 2, \ldots, N$, with $s_{i,j}^{(p,q)} = 1$ if the pth output of the ith subsystem is the qth input to the jth subsystem, and zero otherwise. What is denoted as a subsystem could vary with the level of detail considered in the description of the processing system. At a high level of detail, a subsystem could be a processing equipment, an actuator, a safety device, or a controller. At a low level of detail, a subsystem could be a processing section, composed of several interconnected processing equipment with their actuators and controllers.

Let $\mathbf{x}_i$ denote the vector of state variables describing the behavior of the ith subsystem, and $\mathbf{u}_i^{(q)}, \mathbf{y}_i^{(p)}$ the vectors describing its qth input and pth output streams. The modeling relationships around each subsystem yield

$$\frac{d\mathbf{x}_i(t)}{dt} = \mathbf{f}_i\left[\mathbf{x}_i(t), \mathbf{u}_i^{(q)}(t); \quad q = 1, 2, \ldots, k_i\right]; \quad i = 1, 2, \ldots, N, \tag{1}$$

$$\mathbf{y}_i^{(p)}(t) = \mathbf{g}_i^{(p)}\left[\mathbf{x}_i(t), \mathbf{u}_i^{(q)}; \quad q = 1, 2, \ldots, k_i\right];$$

$$i = 1, 2, \ldots, N, \quad p = 1, 2, \ldots, m_i. \tag{2}$$

If the surrounding world is denoted as the zeroth subsystem, then the variables, $\mathbf{y}_0^{(p)}$; $p = 1, 2, \ldots, m_0$ represent the external inputs to the processing system, and in general they can be divided into two classes; the *real-valued*, $\mathbf{y}_{0,\mathrm{RV}}^{(p)}$ and the *integer-valued*, $\mathbf{y}_{0,\mathrm{IV}}^{(p)}$. The $\mathbf{y}_0^{(p)}$ variables can be

used to represent the actions of human operators or supervisory control systems.

The values of the inputs, $\mathbf{u}_i^{(q)}$, $q = 1,2,\ldots,k_i$; $i = 1,2,\ldots,N$, are determined by a set of operations, such as opening or closing a manual valve, turning on or off a motor, or changing the value of a controller's setpoint. Each of these operations can be associated with a particular input and describes how the value of input, $\mathbf{u}_j^{(q)}$, is determined from the value of outputs, $\mathbf{y}_i^{(p)}$;

$$\mathbf{u}_j^{(q)}(t) = Op_j^{(q)}\left[\mathbf{y}_i^{(p)}(t); \quad p = 0,1,2,\ldots,m_i \quad \text{and} \right.$$
$$\left. \forall i\colon S_{ij}^{(p,q)} = 1; \quad \alpha_j^{(q)}(t)\right] \tag{3}$$

where $j = 1,2,\ldots,N$.

Operations $Op_j^{(q)}$, can be seen as *operators* that can take on a *logical integer*, *analytic* (static or dynamic), or *hybrid* form. With each operation we have associated an integer-valued variable, $\alpha_j^{(q)}(t)$, which determines the discrete state that the operator $Op_j^{(q)}$, takes on with time.

Given an initial state of a processing system and the outputs of the surrounding world, one may solve the set of Eqs. (1), (2), and (3) and thus find the temporal evolution of the state describing the process behavior. As the various operators $Op_j^{(q)}$ take on different discrete states over time, the trajectory of the process' state goes through changes at discrete time points. Clearly, the order in which the various operations change state and the time points at which they change value, affect the temporal trajectory of the process evolution and determine the value of the performance index. Therefore, we are led to a *mixed-integer*, *nonlinear programming* (MINLP) problem with the following decision variables:

1. *Integer decisions*
   (a) $\alpha_j^{(q)}(t)$, i.e., the discrete state of each operator, $Op_j^{(q)}$.
   (b) The temporal ordering of the discrete-state changes of the various operators, e.g., $t_1 < t_2 < t_3$, where $\alpha_2^{(1)}(t_1) = 1$ (from 0), $\alpha_4^{(1)}(t_2) = 0$ (from 1), $\alpha_3^{(1)}(t_3) = 2$ (from 1).
   (c) $y_{0,\mathrm{IV}}^{(p)}(t)$, i.e., integer-valued variables, which are inputs from the surrounding world.
   (d) The temporal ordering of the discrete-value changes of the integer-valued inputs $y_{0,\mathrm{IV}}^{(p)}(t)$, from the surrounding world.
2. *Continuous decisions*. The time-dependent trajectories of the real-valued variables that are inputs from the surrounding world.

In addition to these above, the values that the states, $\mathbf{x}_i$, $i = 1,2,\ldots,N$; the outputs, $y_i^{(p)}$; $p = 1,2,\ldots,m_i$; $i = 1,2,\ldots,N$; and the inputs, $\mathbf{u}_i^{(q)}$; $q = 1,2,\ldots,k_i$; $i = 1,2,\ldots,N$ can take are limited by a set of engineering

constraints, which are dictated by safety and operability considerations, physical limitations, or/ and production specifications. Finally, the selection of the "best" plan of operations requires the formulation of an objective function, or a vector of objective performance indices. Thus, for the general multiobjective synthesis of operating procedures we can state the following mixed-integer, dynamic optimization problem (termed *Problem* 1):

$$\underset{\mathbf{A};\, \mathbf{y}_0^{(p)}(t)}{\text{Minimize}}\ P = [P_1, P_2, \ldots, P_l]$$

subject to

$$\frac{d\mathbf{x}_i(t)}{dt} = \mathbf{f}_i[\mathbf{x}_i(t), \mathbf{u}_i^{(q)}(t); \quad q = 1,2,\ldots,k_i]; \quad i = 1,2,\ldots,N, \tag{1}$$

$$\mathbf{y}_i^{(p)}(t) = \mathbf{g}_i^{(p)}[\mathbf{x}_i(t), \mathbf{u}_i^{(q)}; \quad q = 1,2,\ldots,k_i];$$
$$p = 1,2,\ldots,m_i, \quad i = 1,2,\ldots,N. \tag{2}$$

$$\mathbf{u}_j^{(q)}(t) = Op_j^{(q)}[\mathbf{y}_i^{(p)}(t), \quad p = 1,2,\ldots,m_i, \forall i\colon s_{ij}^{(p,q)} = 1; \quad \alpha_j^{(q)}(t)];$$
$$q = 1,2,\ldots,k_j \quad \text{and} \quad j = 1,2,\ldots,N, \tag{3}$$

$$\mathbf{h}_i(\mathbf{x}_i; \mathbf{u}_i^{(q)}; \quad q = 1,2,\ldots,k_i; \quad \mathbf{y}_i^{(p)};$$
$$p = 1,2,\ldots,m_i) \leq 0; \quad i = 1,2,\ldots,N, \tag{4}$$

where $\mathbf{A}$ is the vector of all $\alpha_j^{(q)}(t)$, for $q = 1,2,\ldots,k_j$ and $j = 1,2,\ldots,N$.

Problem-1 is a formidable challenge for mathematical programming. It is an NP-hard problem, and consequently all computational attempts to solve it cannot be guaranteed to provide a solution in polynomial time. It is not surprising then that all previous efforts have dealt with simplified versions of Problem-1. These simplifications have led to a variety of

(a) Representation models for the behavior of the subsystems, i.e., the form of the f_i and $\mathbf{g}_i^{(p)}$ functions.
(b) Descriptions for the operations, i.e., the form of the $Op_j^{(q)}$ operators.
(c) Search techniques for feasible or optimal plans.

2. *The Character of Constraints and the Modeling Requirements*

The character of the constraining functions, $\mathbf{h}_i$, given by Eq. (4), determines to a large extent the form of the required modeling functions, $\mathbf{f}_i$, $\mathbf{g}_i^{(p)}$, and $Op_i^{(p)}$. For example, Rivas and Rudd (1974) stipulated that the only constraints they were concerned were related to the avoidance of certain mixtures of chemicals, anywhere in the process. This being the

case, all states, input and output variables can be characterized by Boolean values, and the required models are simple expressions of propositional calculus. Subsequently, the work of Fusillo and Powers (1987, 1988a, b) introduced qualitative and static quantitative constraints, which required more detailed $\mathbf{f}_i$ and $\mathbf{g}_i^{(p)}$ descriptions.

Another important consideration regarding the character of constraints, $\mathbf{h}_i$, is their *hierarchical articulation at multiple levels of detail*. For example, suppose that one is planning the routine startup procedure for a chemical plant. Safety considerations impose the following constraint on the temporal ordering of operations:

Start the reaction section last,

which is expressed at the abstract level of processing section. On the other hand, the constraint

in starting a heat exchanger, start the cold side first

is expressed at the more detailed level of processing equipment. Since both types of constraints must be accounted for during the planning of an operating procedure, it is clear that the computer-aided modeling of a processing system should allow for hierarchical representations at various levels of detail.

The final comment on the character of constraints is related to the available technology for equation solving. The general form of constraints (4) involves restrictions on the time-dependent trajectories of states, inputs, and outputs. Since the set of decisions of Problem-1 involves both integer and continuous variables, we need numerical methodologies that can solve large sets of differential algebraic equations [i.e., Eqs. (1), (2), and (3)] with continuous and discrete variables. The recent works in dynamic simulation (Pantelides and Barton, 1993; Barton, 1992; Marquardt, 1991; Holl *et al.*, 1988) are spearheading the developments in this area, but until these efforts produce efficient and robust solution algorithms, the formulations of Problem-1 will be forced to accept simple statements of constraints [Eq. (4)].

3. Search Procedures

The solution of Problem-1 requires extensive search over the set of potential sequences of operations. Prior work has tried either to identify all feasible operating sequences through *explicit* search techniques, or locate the "optimum" sequence (for single-objective problems) through the *implicit* enumeration of plans. The former have been used primarily to solve planning problems with Boolean or integer variables, whereas the latter have applied to problems with integer and continuous decisions.

Explicit search techniques for the identification of feasible operating procedures originated in the area of artificial intelligence. Typical examples involve the pioneering "means–end analysis" paradigm of Newell *et al.* (1960), and the various flavors of *monotonic planning* (e.g., Fikes and Nilsson, 1971), or *nonmonotonic planning* (e.g., Sacerdoti, 1975; Chapman, 1985). With the exception of the work of Lakshmanan and Stephanopoulos (1988a, b, 1990), who have used nonmonotonic planning, all other works on the synthesis of operating procedures for chemical processes have employed ideas of monotonic planning.

Starting either from the desired goal or the given initial state of a processing system, monotonic planning constructs an operating procedure step by step, moving "closer" all the time to the other end (i.e., initial or goal states, respectively). After each operating step the values for all state, input, and output variables are known by solving Eqs. (1), (2), and (3), respectively, and the satisfaction of constraints [Eq. (4)] is tested. The last operating step is revoked (or modified) if any of the constraints (4) is violated. Nonmonotonic planning, on the other hand, works by constructing and refining partial plans. A *partial plan* is a set of operating steps that leaves a certain amount of information unspecified. Perhaps the temporal order in which two operating steps are executed is not specified, or a particular step of the operating procedure may be assigned a *set* of potential feasible operations, rather than a *single* operation. Thus, at the end, *a single partial plan actually describes a large number of "completions" or total operating procedures*. This feature is extremely important for two reasons:

1. *Efficiency*. By conducting planning in terms of partial plans, nonmonotonic reasoning allows a single planning decision to represent a large number of plans, resulting in possibly exponential savings in efficiency (Chapman, 1985; Lakshmanan and Stephanopoulos, 1989).
2. *Integration*. The fact that nonmonotonic planning can capture large sets of feasible plans through single decisions implies that it can be integrated very naturally with the mixed-integer mathematical programming techniques if the identification of the "optimum" plan is required.

Implicit enumeration techniques for the identification of the optimum operating procedure originated in the area of operations research and are based largely on variants of the *branch-and-bound* paradigm. Through recent developments, they have become fairly efficient in locating the "optimum" solution, but they are still awkward in defining the set of feasible plans. Continued progress in the development of more efficient formulations of the planning problem and more efficient branch-and-bound strategies, to handle decision variables with many integer values, will

provide a *practical tractability* for the solution of larger and richer versions of Problem-1.

C. Overview of the Chapter's Structure

Section II addresses the most important facet of an efficient planning methodology, namely, "how to represent the behavior of processes and process operating steps." Three basic operators—simple propositional, conditional, and functional—are explored for their representational power to model process operations. In addition, the representation of process operational constraints dictates the need for a hierarchical modeling of processes and their subsystems, thus leading a modeling framework very similar to that of MODEL.LA. (see first chapter in Volume 21). In fact, we have used the formalism of MODEL.LA. to articulate all modeling needs, related to nonmonotonic planning of process operations.

Section III introduces the concept of nonmonotonic planning and outlines its basic features. It is shown that the tractability of nonmonotonic planning is directly related to the form of the operators employed; simple propositional operators lead to polynomial-time algorithms, whereas conditional and functional operators lead to NP-hard formulations. In addition, three specific subsections establish the theoretical foundation for the conversion of operational constraints on the plans into temporal orderings of primitive operations. The three classes of constraints considered are (1) temporal ordering of abstract operations, (2) avoidable mixtures of chemical species, and (3) quantitative bounding constraints on the state of processing systems.

In Section IV we provide illustrations of the modeling concepts presented in Section II and how the strategy of nonmonotonic planning has been used to synthesize the switchover operating strategy for a chemical process.

Section V extends the above ideas into a different problem, i.e., how to revamp the design of a process so that it can accommodate feasible operating procedures.

II. Hierarchical Modeling of Processes and Operations

In this section we will discuss a systematic approach for the representation of processes and operating steps. These representations, or *models*, conform to the general requirements presented in Section I,B; they can

(1) capture Boolean, integer (i.e., categorical, or qualitative), or real-valued variables of a process; and (2) emulate descriptions of the process or of the operations at multiple levels of detail with internal consistency.

A. Modeling of Operations

The characterization of operating steps, i.e., the characterization of the operators, $Op_j^{(q)}$, depends on (1) the type of operation (i.e., Boolean, integer, analytic real-valued) and (2) the level of the desired detail (abstract operation over a section of a plant, primitive operations on valves, pumps, etc.). Let us see the classes of possible models, all of which could be available and used even within the same planning problem.

The STRIPS-Operator

Figure 2a shows the simplest possible model of an operating step, introduced by Fikes and Nilsson (1971) and known as the STRIPS operator. The *preconditions* are statements that must be true of the system and its surrounding world before the action can be taken. Similarly, the *postconditions* are statements that are guaranteed to be true after the actions corresponding to the specific operation have been carried out. The allowable forms of pre-, and postconditions should satisfy the following requirements:

(a) Pre- and postconditions must be expressed as propositions that have content, which is a tuple of elements, and can be negated.
(b) The elements of the preceding tuples can be variables or constraints and could be infinitely many of them.
(c) Functions, propositional operators, and quantifiers are not allowed.

The STRIPS operator is very simple and intuitively appealing. In attempting to use it for the synthesis of operating procedures for chemical processes, one finds immediately that it suffers from severe limitations, such as the following:

1. The temporal preconditions and postconditions in the STRIPS model vary from process to process, thus making impossible the creation of a generic set of operators a priori.
2. Given a set of preconditions [i.e., the values of $\mathbf{y}_i^{(p)}$ and $\alpha_j^{(q)}$; see Eq. (4)], the actions of process operation lead to new values for $\mathbf{u}_j^{(q)}$, which when propagated through Eq. (1) generate the new state, i.e., the action's postconditions. But Eq. (1) does include functional relationships, $\mathbf{f}_i$, thus leading to an overall functional operator, which is disallowed in a strict STRIPS model.

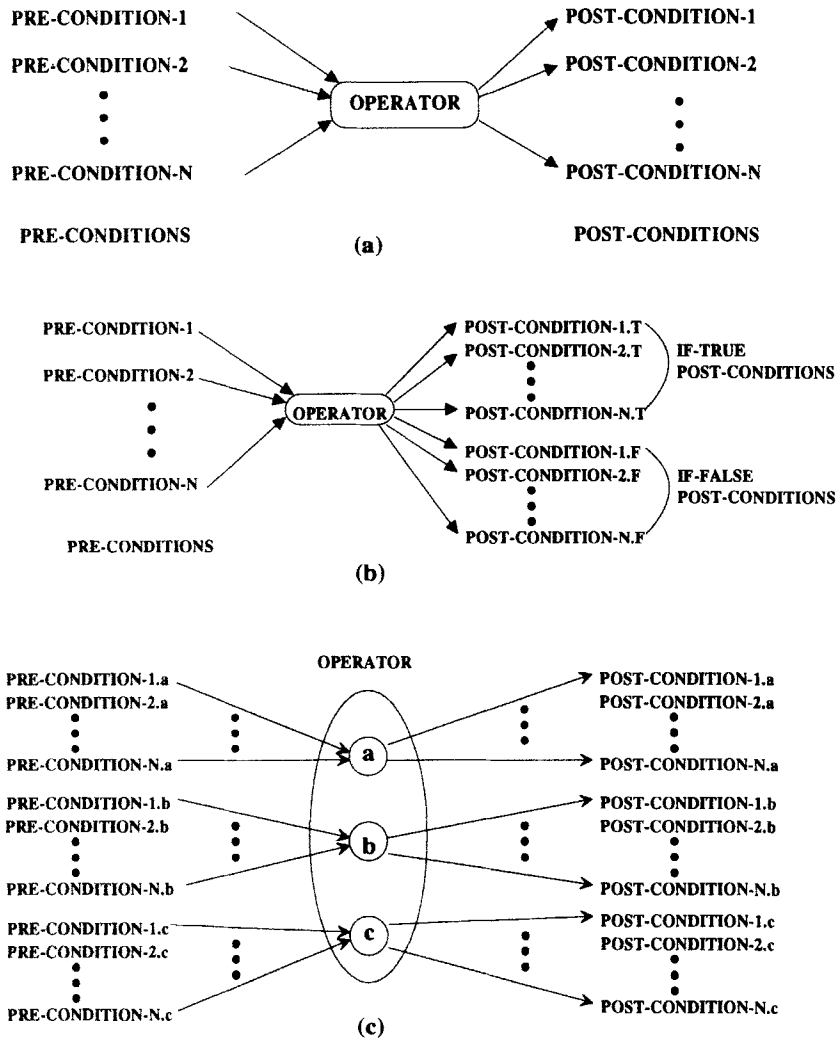


FIG. 2. The three planning operators: (a) STRIPS, (b) conditional, (c) functional.

Nevertheless, early work in the synthesis of operating procedures (Rivas and Rudd, 1974; Ivanov *et al.*, 1980; Kinoshita *et al.*, 1982; Fusillo and Powers, 1987) did employ various variants of the STRIPS operator on limited-scope problems with very useful results. Lakshmanan and Stephanopoulos (1987) also demonstrated the value of the STRIPS operators in ordering sequences of feasible operations during routine startup.

2. Conditional Operators

To overcome the limitations of the STRIPS operator, Chapman (1985) suggested the use of *conditional operators*, which produce two sets of postconditions (Fig. 2b), depending on whether all preconditions are true or any one of them is false. Conditional operators can describe a broader class of operations, but they do possess similar drawbacks when they are considered for the synthesis of operating procedures. In this regard, it is important to realize that the success of the operator-based approaches depend on the ability of the user to define, a priori, operator models for individual processing units, since such operators represent a fusion (i.e., simultaneous solution) of Eqs. (1)–(3) so that they can relate preconditions to the resulting postconditions. It is clear that their use is severely limited and cannot cover planning of operating procedures for (1) equipment/ product changeover, (2) optimizing control, and (3) certain types of safety fallback strategies.

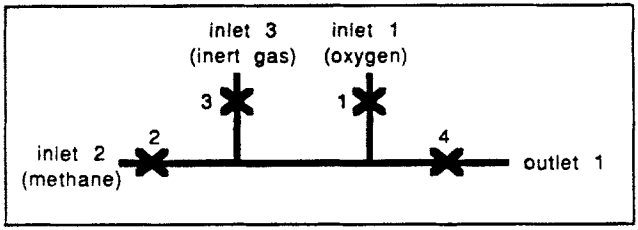
3. Functional Operators

Instead of a conjunction of preconditions, as used by the STRIPS and conditional operators, the *functional* operator has a *set* of conjunctions of preconditions (Fig. 2c). Each element in the set describes some possible situation that might exist before the operator is applied. For each element of the set of preconditions, there is a corresponding element in the set of postconditions. The functional operator is a more flexible model than the STRIPS or conditional operators. It comes closer to the modeling needs for the synthesis of operating procedures for chemical processes, but as we will see in the next section, we need to introduce additional aspects in order to capture the network-like structure of chemical processes.

4. An Illustration

Consider the simple pipes-and-valves system shown in Fig. 3. Initially, oxygen is flowing into the system through inlet 1, and out through the outlet. We would like to develop an operating procedure to solve the following operational problem: "Route the flow of methane gas from the inlet 2 to the outlet without running the risk of explosion (i.e., avoid mixing oxygen and the hydrocarbon)."

Since the only conditions of interest relate to the presence or absence of materials in the various piping segments, the required models for the operations are very simple and can be given by the following three



INITIAL-STATE	GOAL-STATE
(flowing oxygen)	(flowing methane)
not(explosion)	not(explosion)

OPERATORS

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

(establish-flow x)
pre-conditions: ()
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)

FIG. 3. Example operations planning problem. (Reprinted from *Comp. Chem. Eng.*, **12**, Lakshmanan, R. and Stephanopoulos, G., Synthesis of operating procedures for complete chemical plants, Parts I, II, p. 985, 1003, Copyright 1988, with kind permission from Elsevier Science Ltd., The Boulevard, Langford Lane, Kidlington 0X5 1GB, UK.)

STRIPS-like operators:

1. Operator-1: (STOP_FLOW x)
   Preconditions: (flowing x)
   Postconditions: not(flowing x)
2. Operator-2: (ESTABLISH_FLOW x)
   Preconditions: ( )
   Postconditions: (flowing x)
3. Operator-3: (PURGE x)
   Preconditions: not(flowing x); not(equal inert-gas x)
   Postconditions: not(present x); (flowing inert-gas)

Using the initial and goal states as indicated in Fig. 3, it is easy to construct the following sequence of operations that satisfy the desired objectives:

(STOP_FLOW Oxygen),
(PURGE Oxygen), (ESTABLISH_FLOW Methane)

B. Modeling of Process Behavior

A processing facility is a network of unit operations, connected through material and energy flows. Through these connections, the effect of an operation is not confined to the operational behavior of the processing unit it directly affects and its immediate vicinity, but it may propagate and effect the state of all units in the plant. Thus, in addition to the complexity introduced by the functional dependence of process behavior on initial preconditions (see Section II,A), the conditions describing the state of a process resulting after the application of an operation must satisfy the constraints imposed by the network-like structure of the plant. Since the actual structure of the plant is not known at the time when we formulate the operator-models, we must rely on some other means for modeling the constraints characterizing the interactions among different units.

In the first chapter of Volume 2 (hereinafter referred to as **21**:1) we presented the general framework of MODEL.LA., a modeling language that can capture the hierarchical and distributed character of processing systems. We will employ all aspects of MODEL.LA. in order to develop a complete and consistent description of plants that will satisfy the modeling needs for the synthesis of operating procedures.

1. The Hierarchical Description of Process Topology

MODEL.LA.'s primitive modeling elements (see **21**:1), *Generic-Unit*, *Port*, and *Stream*, are capable of depicting any topological abstraction of a chemical process. In particular, the specific subclasses emanating from the *Generic-Unit* allow us to describe a process as (see **21**:1): (1) an overall *Plant-Sections*, where each section represents a grouping of units with a common operating framework, e.g., train of distillation columns; (2) a network of *Augmented-Units*, which encapsulate the structure of processing units along with their ancillary equipment, e.g., a distillation column with its feed preheater, condenser, and reboiler; and (3) a network of *Units*. Figure 4 shows a schematic of this topological hierarchy.

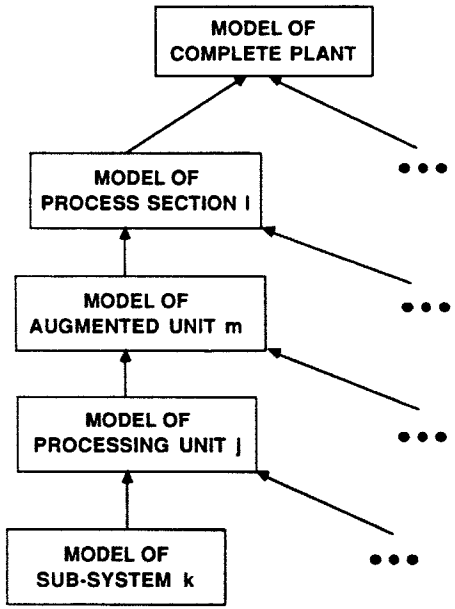


FIG. 4. Hierarchical description of process flowsheets.

Figure 5 shows the sections of a plant producing gasoline from the dimerization of olefins. Using the explicit hierarchical descriptions given above, we can formulate the following series of abstract operators, which (as we will see in Section III) can provide significant help in reducing the number of alternative plans:

Plant level. (START_UP_PLANT);

Section level. (START_UP_FEED_PREPARATION), (START_UP_REACTION), (START_UP_RECOVERY), (START_UP_REFINING);

Augmented-Unit level. (START_UP_DEPROPANIZER), (START_UP_DEBUTANIZER), etc.

Any of these operators can be represented as a STRIPS, conditional, or functional operator, depending on the character of the constraints imposed on the particular operating procedure to be synthesized.

2. *The Hierarchical Description of the Operating State*

Whether we are dealing with an overall plant and its sections, augmented units, or individual units, we must view each system as a thermody-

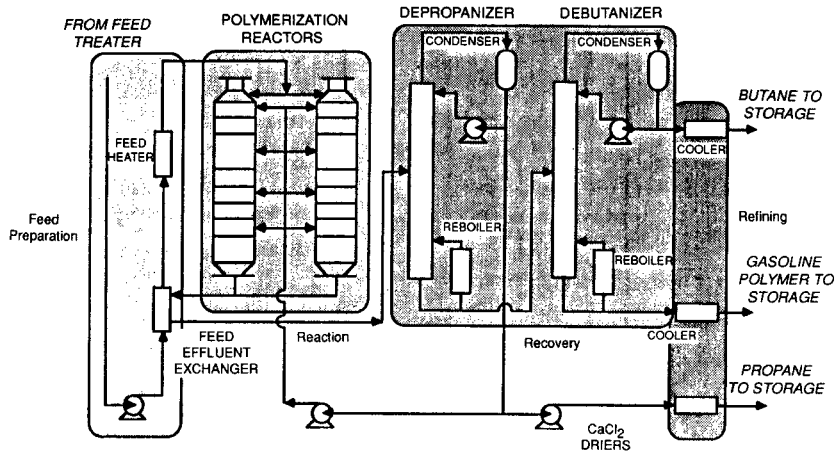


FIG. 5. Decomposition of gasoline polymerization plant.

namically "simple" system (Modell and Reid, 1983), which is characterized by a "state." From an operations planning point of view, this "state" can be viewed at different levels of detail, thus giving rise to a hierarchy of descriptions. MODEL.LA. provides the requisite modeling elements, which can be used to provide the necessary hierarchical descriptions of operating states:

a. Operational State of Variables. The modeling element, Generic-Variable (see **21**:1), allows the representation of the structured information that describes the behavior of "temperature," "pressure," "concentration," "flow," etc. Each of these instances of the *Generic-Variable* is an object with a series of attributes such as "current-value," "current-trend," "range-of-values," "record-over-x-minutes," and "average-over-x-minutes." Through these attributes we can capture all possible preconditions and postconditions that may be needed by the individual operators.

b. Operational State of "Terms". MODEL.LA.'s *Generic-Variable* possesses a series of subclasses (see **21**:1), whose sole purpose is to provide declarative description of certain compound variables, called *Terms*, such as "flow-of-component-x," "enthalpy-flow," "heat-flow," "diffusive-mass-flow," and "reaction rate." The declarative, rather than procedural, representation of compound variables enables the computer-aided planner to have an explicit description of the quantities involved in the preconditions and postconditions of an operation, thus replacing a numerical

procedure by a series of explicit numerical or logical inferences, as the planning needs may be.

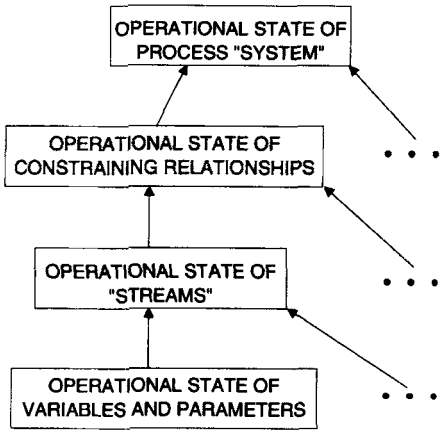
c. Operational State of Constraining Relationships. The operational state of "terms" is constrained by the laws of physics and chemistry and relationships that arise from engineering considerations. Typical examples are (1) the conservation principle (applied on mass, energy, and momentum); (2) phase or reaction equilibria; (3) reaction rates or transport rates for mass, energy, and momentum; (4) limits on pressure drops, heat flows, or work input, dictated by the capacity of processing units; and (5) experimental correlations between input and output "terms," etc. MODEL.LA.'s modeling element, *Constraint*, and its subclasses, *Equation*, *Inequality*, *Order-of-Magnitude*, *Qualitative*, and *Boolean* (see **21**:1) offer the necessary data structures to capture the information associated with any type of constraining relationships.

d. Operational State of a Process System. MODEL.LA's modeling element, *Modeling-Scope*, captures the consistent set of modeling constraints (described in the previous paragraph), which apply on a given process system. As the operational state of a system changes, the content of the *Modeling-Scope* may change; e.g., transition from a single to a two-phase content, transition from laminar to turbulent flow.

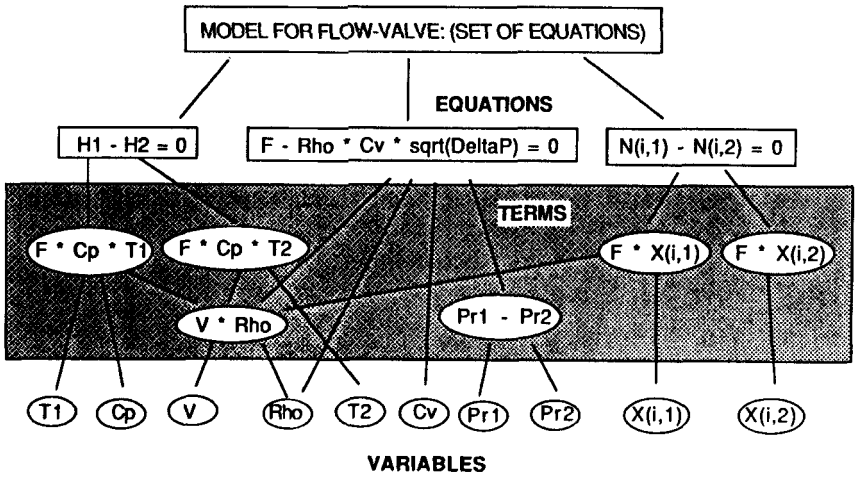
e. Example. The four-level hierarchy for the description of process operational states as shown in Fig. 6a offers a specific example. It offers a very rich and flexible modeling framework to capture any conditions (pre- or post-) associated with the definition of operators.

3. Maintaining Consistency among Hierarchical Descriptions of Process Topology or State

Operators, describing process operations, can be declared at any level of abstraction, but they should maintain consistent relations with each other, since they refer to the same process. For example, the top-level operator, (START_UP_PLANT), could be refined to the following sequence of operators (see also Fig. 5): (START_UP_RECOVERY), (START_UP_REFINING), (START_UP_FEED_PREPARATION), and (START_UP_REACTION). Clearly, the preconditions of (START_UP_PLANT) are distributed and represent a subset of the preconditions for all four more detailed operations. Similarly, the postconditions, derived from the startup operation of the four sections, should be



(a)



(b)

FIG. 6. Hierarchical description of (a) operational states and (b) operational relationships. (Reprinted from *Comp. Chem. Eng.*, **12**, Lakshmanan, R. and Stephanopoulos, G., Synthesis of operating procedures for complete chemical plants. Parts I, II, p. 985, 1003, Copyright 1988, with kind permission from Elsevier Science Ltd., The Boulevard, Langford Lane, Kidlington 0X5 1GB, UK.)

consistent with the postulated postconditions of the overall (START_UP_PLANT).

To achieve these consistencies, MODEL.LA. provides a series of semantic relationships among its modeling elements, which are defined at different levels of abstraction. For example, the semantic relationship (see **21**:1), **is-disaggregated-in**, triggers the generation of a series of relationships between the abstract entity (e.g., overall plant) and the entities (e.g., process sections) that it was decomposed to. The relationships establish the requisite consistency in the (1) topological structure and (2) the state (variables, terms, constraints) of the systems. For more detailed discussion on how MODEL.LA. maintains consistency among the various hierarchical descriptions of a plant, the reader should consult **21**:1.

III. Nonmonotonic Planning

Let us return to the simple operations planning problem of Fig. 3 (see also Section II,A,4). A non-monotonic planner faced with this problem proceeds as follows:

Step 1. Choose one of the desired goals (e.g., flowing methane).

Step 2. Select an operator that achieves the selected goal, e.g., (ESTABLISH_FLOW METHANE), which produces the desired goal as its postcondition.

Step 3. Propagate derived postconditions through the frame axioms (see Fig. 3) and take:

$$\text{(flowing methane)} \Rightarrow \text{(present methane)}$$
$$\text{(flowing oxygen)} \Rightarrow \text{(present oxygen)}$$

Step 4. Check for violation of constraints. In this case the constraint is also a frame axiom:

$$\text{(present methane) AND (present oxygen)} \Rightarrow \text{(explosion)}$$

and is confirmed.

Step 5. Since the constraint was violated, the non-monotonic planner attempts to identify the proper operator among the set of available operators, which can alleviate the constraint violation, if this operator were "forced" to be applied before the constraint violation. In this example such an operator is (PURGE X). Thus, (PURGE X) must be applied before (ESTABLISH-FLOW METHANE). The variable X remains temporarily unbound to any particular value.

Step 6. Operation (PURGE X) stipulates the a priori satisfaction of the following two preconditions:

not(flowing X) AND not(equal inert-gas X)

Since the final goal state requires that (flowing methane) is true, then the value of X can be bound to oxygen. Then the following postconditions of (PURGE OXYGEN) are true:

not(present oxygen) (flowing inert-gas)

Step 7. The first precondition of (PURGE OXYGEN) becomes the new goal and requires that the following operation must take place before (PURGE OXYGEN):

(STOP_FLOW OXYGEN) producing not(flowing oxygen)

So the solution to the planning problem is now complete, and is given by the following sequence of elementary operations:

(STOP_FLOW OXYGEN) > (PURGE OXYGEN)
> (ESTABLISH_FLOW METHANE)

where the symbol > signifies the relative temporal ordering of two operations.

In summary, nonmonotonic planning of process operations has the following distinguishing features:

1. *Property 1: goal-driven approach.* This always starts from a desired goal state and tries to identify the requisite operations that will achieve it.

2. *Property 2: constraint-posting without backtracking.* Instead of backtracking, when a constraint violation is detected, nonmonotonic planning attempts to find an operator that would negate the preconditions leading to the constraint violation and forces this operator to be applied *before* the operation that causes the constraint violation. It leads to the posting of a new constraint determining the temporal ordering of operations. The operator that causes the violation of a constraint, negates, prevents, or undoes one or more of the planning goals, is called *Clobberer*. On the other hand, an operator that rectifies, or prevents any of the damage caused by a Clobberer will be called *nonmonotonic White Knight* (Chapman, 1985). Clearly, the success of any nonmonotonic planning methodology lies in its ability to identify these Clobberers and White Knights without having to resort to an exhaustive *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 present.

3. *Property 3: generator of partial plans.* Nonmonotonic planning generates partially specified plans where a certain amount of information is not bound to specific values. A typical example was the value of X in the operation (PURGE X), that we saw in the illustration at the beginning of this section. Partial plans are abstractions, which can represent large sets of plans, and can lead to every efficient search strategies.

A. Operator Models and Complexity of Nonmonotonic Planning

The correctness of complete plans is encapsulated by the following theorem (Chapman, 1985).

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

In this theorem, any proposition **p** can represent an operator (i.e., an operation step), whereas the situations **t** and **s** represent any intermediate state of the process. Although the validity of the theorem is general, its practical utility is confined to monotonic planning with STRIPS-like operators. For example, in nonmonotonic planning the plans are at any point when partially specified and a new mechanism is needed to guarantee that when the partial plan is completed, a given proposition (i.e., a given operation) is still true (i.e., consistent).

Chapman's work produced the following theorem, which provides the necessary and sufficient conditions for guaranteeing the truth of any given statement in a partial plan, if all operations are modeled by STRIPS-like operators.

Theorem 2 (Modal Truth Criterion). *A proposition* **p** *is necessarily true in a situation* **s** *if and only if the following two conditions hold:*

(a) *There is a situation* **t** *equal or necessarily previous to* **s** *in which* **p** *is necessarily asserted.*

(b) *For every step* **C**, *possibly before* **s**, *and every proposition,* **q**, *possibly codesignating with* **p**, *which* **C** *denies, there is a step* **W** *necessarily between* **C** *and* **s** *that asserts* **r**, *a proposition such that* **r** *and* **p** *codesignate whenever* **p** *and* **q** *codesignate.*

So, if **C** is the step in an operating procedure that introduces the

Clobberer, **q**, denying the validity of the postconditions of operation **p**, then **W** is the operating step (necessarily after **C**) at which the White Knight, **r**, comes to rescue and negates the effects of **q**. Theorem 2 also indicates that for each operation **p**, any preceding Clobberer must have the corresponding White Knight, if the nonmonotonic plan is to be correct.

For STRIPS-like operators, Chapman (1985) developed a polynomial-time algorithm, called TWEAK, around five actions that are necessary and sufficient for constructing a correct and complete plan. As soon as we try to extend these ideas to nonmonotonic planning with conditional operators, we realize that no polynomial-time algorithm can be constructed, as the following theorem explicitly prohibits (Chapman, 1985):

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

It is not surprising, then that, as we require functional operators to approximate more closely the majority of operations during the synthesis of process operating procedures, we must give up any expectation for a tractable algorithm (Lakshmanan, 1989):

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

B. Handling Constraints on the Temporal Ordering of Operational Goals

This section and the following two will cover the treatment of three distinct classes of constraints within the general framework of nonmonotonic planning. Let us first consider constraints on the temporal ordering of operational goals. They are of the form "GOAL-A must be achieved before achieving GOAL-B" and appear at a certain level of process abstraction. Examples are

(a) (START_UP_FEED_TREATMENT_SECTION) before (START_UP_REACTOR).
(b) (START_FLOW_IN_COLD_SIDE) before (START_FLOW_IN_HOT_SIDE) of a heat exchanger.
(c) (OPEN_VALVE_1) before (OPEN_VALVE_2).

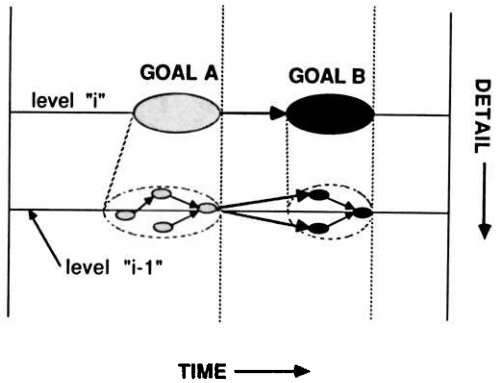


FIG. 7. Downward transformation of temporal constraints. (Reprinted from *Comp. Chem. Eng.*, **12**, Lakshmanan, R. and Stephanopoulos, G., Synthesis of operating procedures for complete chemical plants. Parts I, II, p. 985, 1003, Copyright 1988, with kind permission from Elsevier Science Ltd., The Boulevard, Langford Lane, Kidlington 0X5 1GB, UK.)

A temporal constraint at a particular level of the abstraction hierarchy specifies a partial ordering on the goals in the level just below (more detailed), *causing the posting of new temporal constraints* on the goals of this level. Each of these new constraints specifies in turn a set of new temporal constraints at the level below it. In this manner, constraints at a high-level of abstraction are successively transmitted from level to level until they reach the most detailed level of individual processing units. All the newly generated constraints must be taken into account and satisfied by the non-monotonic planner. Figure 7 shows a schematic of the propagation of temporal constraints through two levels of abstraction. The directed path indicates the temporal ordering of goals.

Lakshmanan and Stephanopoulos (1988b) developed an algorithm, which automates the downward propagation of temporal constraints all the way to the level of processing units, where these constraints can be incorporated in planning of primitive operators (e.g., open or close valve, turnon or shutdown motor). This algorithm is provably correct having the following property.

1. *Property 1: Correctness of the Algorithm for the Downward Propagation of Temporal Constraints*

Let C_i be a temporal constraint, stated between two goals, A_i and B_i, at the ith level of the modeling hierarchy (see Section II,B), which indicates that A_i must be achieved before B_i. Let $A_{(i-k)}$ and $B_{(i-k)}$, with $k > 0$, be subgoals at the $(i-k)$th level of A_i and B_i, respectively. Then, the algorithm for the downward propagation of temporal constraints guaran-

tees, on 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 $A_{(i-k)}$.

The successive introduction of new constraints in the temporal ordering of operational steps at lower levels of the goal hierarchy, may create internal conflicts. For example, the propagation of constraint C_i may result in ordering A_i before B_i, while the propagation of constraint C_j may lead to ordering B_j before A_j. These conflicts manifest themselves in the form of directed circuits (cycles) at one or more levels in the goal hierarchy. A polynomial-time algorithm has been developed to detect the generation of directed cycles as new constraints are introduced (Lakshmanan and Stephanopoulos, 1988b). The user is informed and the offending constraint may be detracted.

a. Transformation of Integer–Goal Ordering Constraints onto Orderings of Primitive Operators. The temporal ordering of goals at higher levels of the goal hierarchy is successively transformed to temporal orderings of goals at lower levels. It now remains to be seen how the temporal ordering of goals at the lowest level of the hierarchy can be transformed into constraints on the ordering of primitive operators. Lakshmanan and Stephanopoulos (1988b) have developed such an algorithm which possesses the following property.

2. Property 2: Correctness of the Constraint Transformation Algorithm

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

This property guarantees that constraints on the temporal ordering of goals are correctly transformed into constraints on the temporal ordering of primitive operations that achieve these goals.

C. Handling Constraints on the Mixing of Chemicals

Safety, environmental, health, and performance considerations dictate that certain chemicals should not coexist anywhere in the plant. Rivas and Rudd (1974), O'Shima (1982), and Fusillo and Powers (1988a) have dealt

with this important class of constraints, and have proposed approaches all of which are monotonic in character. In its most general form, a constraint on the mixing of chemicals involves the specification of a set of chemical species, and composition ranges for those species that ought to be avoided anywhere in the plant. Very often, though, the high cost of the consequences of an unsafe operation (e.g., an explosion) forces the operations planners to adopt a more conservative posture and require that the coexistence of a set of chemicals be avoided altogether. This class of constraints is expressed as an unordered set of chemical species $(x_1, x_2, \ldots, x_n)$ that should never be present at the same time anywhere in the plant. In this section we will examine how nonmonotonic planning handles such constraints and converts them into constraints on the temporal ordering of primitive operations through the following sequence of steps:

Step 1. *Construction of the influence graphs*, which represent the extent to which a given chemical species is present within the process in a given operational state.
Step 2. *Identification of potential constraint violations*, resulting from selected operations during the construction of partial plans.
Step 3. *Generation of temporal constraints on primitive operations*, to negate the violation of mixing constraints.

In the following paragraphs we will discuss the technical details involved in each of the preceding steps.

1. *Construction of the Influence Graphs*

Consider the directed graph representing the topology of a chemical process at a given operating state, i.e., the nodes of the graph represent process equipment and the directed edges of the graph the material flows. If the direction of the flow has not been computed or is uncertain, the corresponding edges are bidirectional. Each edge conveys information related to the type of chemical species flowing and the directionality of flow. All other state information (e.g., temperature, concentration) has been suppressed. Then, using the directed graph that represents the topology of the chemical process, the construction of the influence graphs (IGs) proceeds as follows (Lakshmanan and Stephanopoulos, 1990):

1. Locate all sources, s_i, of the chemical species, x_i.
2. Remove all edges that are immediately adjacent to closed valves.
3. Any node, v, which is on a directed path from any source, s_i, is labeled with x_i. Let N_i be the set of nodes labeled with x_i and let E be the set of associated edges. Nodes labeled with x_i and the attached edges contain the chemical species.

At the end of this three-step procedure we have generated a graph (i.e., a subgraph of the process graph) whose edges indicate the flow of species x_i and whose nodes contain the chemical x_i, at the given operating state.

2. *Identification of Constraint Violations*

Violations of the mixing constraints may occur at the *initial*, *goal*, or any *intermediate* state in the course of an operating procedure. Since the initial state is presumed to be a known, feasible state, it is generally not the case that it will contain potential mixing constraint violations. It is possible though that the human user or the planning program may specify a goal operating state that violates the mixing constraints.

The detection of potential mixing constraints violations at an intermediate operating state is based on a "worst-case scenario" with the maximum potential for the species to coexist, and which corresponds to the following situation:

(a) Keep open all the valves which are OPEN at the initial state.
(b) Keep open all the valves which must be OPEN at the goal state.

Construct the IGs for all species present in the various mixing constraints. Then:

If a node, v, carries both labels, x_i and x_j, where (x_i, x_j) is an unacceptable pair of coexisting species, then the potential for a mixing constraint violation has been detected.

Clearly, if under the worst-case scenario no node can be found in the IGs that is labeled with all species present in a mixing constraint, then no potential violation of a mixing constraint exists. If, on the other hand, a potential violation has been detected, then we need to *generate additional constraints on the temporal ordering of primitive operations so that we can prevent or negate the preconditions of a mixing constraint*.

3. *Generation of Constraints on the Temporal Ordering of Primitive Operations*

The potential violation of a mixing constraint is equivalent to the presence of a potential *Clobberer*. Therefore, according to Theorem 2, we need to identify a *White Knight* and force him to proceed the action of the Clobberer. In terms of the synthesis of operating procedures, the "Clobberers" are the valves that are open under the worst-case scenarios, described in the previous paragraph. Obviously, the "White Knights" represent the set of valves that, when closed, will prevent the chemicals from mixing. We will call this set the *minimal separation valve set* (MSVS). Once the valves that should be closed have been identified, temporal

ordering constraints are imposed, which specify that these valves be closed, *before* the valves that lead to a mixing constraint violation were opened. Lakshmanan and Stephanopoulos (1990) have developed a graph-based algorithm with polynomial-time complexity, which identifies the MSVS. However, the MSVS may not be an *acceptable separation valve set* (ASVS) for the given planning problem, since (1) some valve in the MSVS is required to be open in the goal state (in order to achieve a certain operating goal), or (2) the user may have some subjective preferences regarding the state of some valves. A concrete algorithm exists (Lakshmanan and Stephanopoulos, 1990) to convert the initial MSVS to an ASVS.

Once an ASVS has been found, the constraints on the temporal ordering of the valve operations in the ASVS must be established. This is systematically accomplished through an algorithm that undergoes through the following steps.

1. Let P be the set of valves that must be open under the worst-case scenario.
2. If p is an element of P and q is an element of ASVS, then, to negate the violation of a mixing constraint, the following temporal constraint must be generated:

$$\text{CLOSE(q) before OPEN(p)}$$

4. *Purging a Plant from Offending Chemicals*

Once the ASVS has been identified and the temporal ordering of valve operations has been determined, no mixing constraints will be violated in the steady state. However, it is possible to violate the mixing constraints during the transient from one steady state to another. For example, during the changeover from oxygen to methane in the network of Fig. 3, it is possible that oxygen and methane will come into contact with each other even if the upstream oxygen inlet valve is closed, before the methane is allowed to enter. To avoid the contact between the offending chemicals, the first (e.g., oxygen) must be purged, before the second (e.g., methane) is allowed to flow in.

Lakshmanan (1989) has indicated that "*purging or evacuation operations are necessary, whenever the intersection of the IG of one component in the initial state with the IG of another (offending) component in the goal state is non-zero.*" In such case, nonmonotonic planning generates a new intermediate goal, i.e., purge the plant of chemical x_i, and tries to achieve this goal first before admitting the offending chemical x_j in the process.

Consequently, the overall planning is broken into two phases and proceeds as follows (Lakshmanan and Stephanopoulos, 1990):

a. Phase 1. Find the path and destination of purge operations.

1. Identify all locations of a plant where a certain species must be purged. To accomplish this it is sufficient to scan the labels of the nodes in the IGs and identify those nodes that are labeled, e.g. x_i and x_j in two successive operating states, where (x_i, x_j) is a prohibited mixture of chemicals.
2. For each node v, where the offending chemicals (x_i, x_j) are present in the initial and goal states, respectively, find a directed path that leads the purgative material from its source to the sink through node v.

b. Phase 2. Generation of the purge planning island, i.e., intermediate planning goals.

1. Create an Intermediate operating State, i.e., introduce the intermediate goals requiring that *all valves along the purge path be open.*
2. Place the intermediate operating state between the initial and goal states 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 (viz., worst-case scenario).

D. Handling Quantitative Constraints

During the synthesis of operating procedures, the operating state may be required to satisfy certain quantitative constraints. For example, during the operation of a reactor, the temperature may not be allowed to exceed a certain maximum, or 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:

$$\{\text{HCN should not mix with HCHO at } T \geq T_{\max}\}$$
$$\{[\text{HCN}] \leq 0.1 \text{ mol/L}\}, \text{ in the reactor at all times.}$$

Such constraints commonly arise from safety requirements, product specifications, environmental regulations, etc.

In this section we will present a formalized methodology that allows the transformation of quantitative bounding constraints into constraints on the temporal ordering of operators within the spirit of nonmonotonic planning.

1. Truth Criterion for Quantitative Constraints

The efficiency of nonmonotonic planning depends on its ability to identify Clobberers and White Knights efficiently. Theorems 3 and 4 (see Theorems 1–4 in Section III,A) have established the intractability of nonmonotonic planning for conditional and functional operators, which are rich enough to ensure adequate representation of process operations. Consequently, practical solutions can be derived and intractability can be avoided only through the use of domain-specific knowledge. 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 $x.*.k$[2] the truth criterion is stated as follows:

Theorem 5: (Modal Truth Criterion for Quantitative Constraints). *A constraint $x.*.k$ is necessarily satisfied in a situation s iff there is a situation t equal to or necessarily previous to s in which $x.*.k$ is satisfied and for every step C possibly before s that possibly violates $x.*.k$, there is a step W necessarily before C that ensures $x.*.k$ in s.*

The statement and proof of the truth criterion (see Lakshmanan, 1989) for quantitative constraints is along the lines of Chapman's truth criterion for domain independent nonmonotonic 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

[2]The notation '.*.' indicates relationships of the type $\langle, \rangle, =$, etc. between X and k.

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.

2. Identification of Clobberers of Quantitative Constraints

A goal is often achieved by multiple primitive (e.g., valve) operations. All the primitive operations that will together result in violation of a constraint will be Clobberers of that constraint. Consequently, the first step in the quantitative constraint-posting methodology is to identify abstracted operators as the sets of primitive operations. This is accomplished through the following procedure:

```
    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
```

This algorithm provides a complete identification of all requisite operators, simple or abstract (i.e., sets of operators), as the following theorem guarantees (Lakshmanan, 1989).

Theorem 6: *Given the list of primitive operations to achieve a goal state, the algorithm* ABSTRACT_OPERATOR *will detect every simple or abstracted operator corresponding to the set of operations required for starting or stopping flow of materials from each source, s_j, to the point of constraint specification, e_i, necessary to achieve the goal state.*

The algorithm ABSTRACT_OPERATOR 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 operators (e.g., valves) and E is the number of directed edges (i.e., connections) in the graph representing the topology of a process flowsheet. 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 (Theorem 5), 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.

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
```

The following theorem guarantees the completeness of the above algorithm:

Theorem 7. *Every abstract operator, that will result in violation of a quantitative constraint, if implemented first and alone, will be detected as a Clobberer by the procedure* CLOBBERER_DETECTION.

3. *Selection of Plan Modification Operations: The White Knight for Quantitative Constraints*

Once the Clobberers that lead to violations of quantitative constraints have been identified, we must search for modifications of the operating procedure, which will negate the effects of the Clobberers on the procedure and thus restore its feasibility. There exist two general mechanisms for achieving this objective, and in the following paragraphs we will discuss each one of them.

a. Mechanism 1: 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 provide greater temporal order to the operators in the plan.

b. Mechanism 2: Identification of White Knights. If all operators for obtaining a feasible plan (i.e., satisfies both goals and constraints) have been correctly identified, then the quantitative constraints can be transformed into constraints on the temporal ordering of operations and demotion will always work. Nevertheless, if every operator identified for a single equipment is a potential Clobberer, then quantitative constraint violation cannot be avoided by the current set of operators and a White Knight has to be identified. The procedure for identifying White Knights is composed of two steps:

Step 1. 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.

Step 2. 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 complete algorithm for the identification of White Knights is as follows:

```
procedure WHITE_KNIGHT
Input: Plant flowsheet in final state, F; constraint being clobbered, q.
Output: White Knight, W for the constraint q.
begin
for q do
    begin
    if q ≡ 'X(<, ≤)k' then L := 'decrease X' {comment: L – load}
    if q ≡ 'X(>, ≥)k' then L := 'increase X'
    if q ≡ 'X = k' and X < k then L := 'increase X'
    if q ≡ 'X = k' then X > k then L := 'decrease X'
    { comment: Such rules may be written for each type
      of constraint encountered }
    end
while v ≠ manipulatable valve do
    QUALITATIVE – PROPAGATE(L, F)
return v and required qualitative change
end
```

The following theorem (Lakshmanan, 1989) guarantees the completeness of the preceding algorithm.

Theorem 8. *The algorithm* WHITE-KNIGHT *will detect every manipulation, that will possibly avoid a quantitative constraint violation, as a White Knight.*

E. Summary of Approach for Synthesis of Operating Procedures

The synthesis of operating procedures for a chemical plant, using nonmonotonic planning ideas, consists of two distinct phases: (1) formula-

tion of the planning problem and (2) synthesis of operating plans. In the following two subsections we will examine the features of each phase and we will discuss the technical details for their implementation.

1. Phase 1: Formulation of the Planning Problem

The complete formulation of the planning problem implies the complete specification of the following four items.

a. Description of the Initial State. We must make sure that the specified starting operational state of a process is complete and satisfies the balances (mass, energy, and momentum), equilibrium, and rate phenomena in the process. This task is composed of the following steps:

1. Input of the initial operational state of the plant, using the linguistic primitives of MODEL.LA. (see **21**:1).
2. The specification of the initial state may be partial and lead to incomplete descriptions of various segments of the plant. The modeling facilities of MODEL.LA. contain a complete set of the balance equations, phase and chemical equilibrium, and rate relationships. These relationships are used to propagate the user-supplied specifications and thus complete the description of the initial state throughout the plant.
3. During the specification of the initial state, it is conceivable that the user-supplied information is not consistent with the physical constraining relationships. The planning program should possess procedures for detecting such conflicts and provide mechanisms for their resolution.

b. Description of the Final, Goal State. This proceeds in a manner similar to that for the description of the initial state.

1. Input the specifications of the desired goal state.
2. Check for potential conflicts in the specification of the goal state, and provide resolution for its consistent definition.
3. Generate concurrent goals, which usually come from the interaction among the various subsystems of the plant. This is accomplished through constraint propagation mechanisms, using the set of modeling relationships describing the process, analogous to the mechanism that ensures the completeness in the description of the initial state.

c. Specification and Identification of Operational Constraints in the Plan. In addition to the physical modeling constraints that govern the operation of the chemical process, we require that several "operational" constraints be met by an operating procedure, such as (1) temporal ordering in the execution of process operations, (2) avoiding the creation of undesirable mixtures of chemical species, and (3) maintaining the state variables within specific ranges of numerical values, e.g., bounding quantitative constraints.

d. Specification, Identification of Planning Islands. When an operating procedure is expected to be long and to involve a large number of intermediate states, it may be advantageous to explicitly identify intermediate goals, known as *planning islands* (Chakrabarti *et al.*, 1986), through which the operating procedure must pass, as it goes from the initial to the goal state. These planning islands are partial descriptions of intermediate states, which are known to lie on the paths of the most efficient, or feasible, plans, thus decomposing the overall operating plan into a series of subplans, which can be synthesized separately in a sequence. Lakshmanan (1989) has established specific procedures for the identification of planning islands.

2. Phase II: Synthesis of Operating Plans

Having defined the planning problem in a form that is "understood" by the computer, we can proceed to the synthesis of operating procedures through the following sequence of steps.

a. Identification of the Primitive Operators. The "means–ends analysis" of Newell *et al.* (1960) is employed to (1) identify the differences between the goal and initial states and (2) select the operators (i.e., operating actions) that would eliminate these differences.

b. Construction of Partial Plans. This consists of deriving a partial temporal ordering in the application of primitive operators. The partial ordering is driven by the constraints placed on the states of the desired operating plan, and proceeds as follows:

1. User-specified, temporal ordering of operational goals at higher levels of abstraction is propagated downwards in the hierarchy goals and is ultimately expressed as temporal ordering of primitive operations (see Section III,B).
2. Constraints on the disallowed mixtures of chemical species (a) are transformed into temporal orderings of primitive operations, or/ and

(b) introduce additional intermediate goals, e.g., planning islands for purge or/ and evacuation operations (see Section III,C).

3. Quantitative constrains (a) are transformed into temporal orderings of primitive operations (e.g., demotion of Clobberers) or (b) dictate the introduction of new operators (e.g., White Knights) to recover the feasibility of plans.

c. Synthesis of Complete Plans. At the present, nonmonotonic planning covers the three classes of constraints discussed above. For other classes of constraints, not amenable to direct transformation into temporal constraints between goals, the generate-and-test strategy of monotonic planning is employed. Thus, the partial plans generated by the constraint propagation of nonmonotonic planning are put together to develop feasible plans to solve the problem.

IV. Illustrations of Modeling and Nonmonotonic Operations Planning

In this section we will offer several illustrations of the various aspects of nonmonotonic operations planning (discussed in earlier sections) including the following: (1) development of hierarchical models for the process and its operations, (2) conversion of constraints to temporal orderings of primitive operations, and (3) synthesis of complete plans.

A. Construction of Hierarchical Models and Definition of Operating States

Consider the flowsheet shown in Fig. 8. It represents the power forming section of a refinery, along the ancillary storage tanks and piping network required for the catalyst regeneration. During the operation of the plant, the activity of the catalyst (chloroplatinic acid on alumina base) in the operating reactor, e.g., REACTOR-1, decays as a result of (1) coke formation, (2) reduction in the chloride content, and (3) size growth of the platinum crystals. We want to synthesize an operating procedure that will discontinue the operation of REACTOR-1, start the regeneration of its catalyst (i.e., remove coke, add chloride, redisperse the platinum crystals), and initiate the operation of the standby REACTOR-2. For a fairly detailed description of the plant, its modeling, and the synthesis of

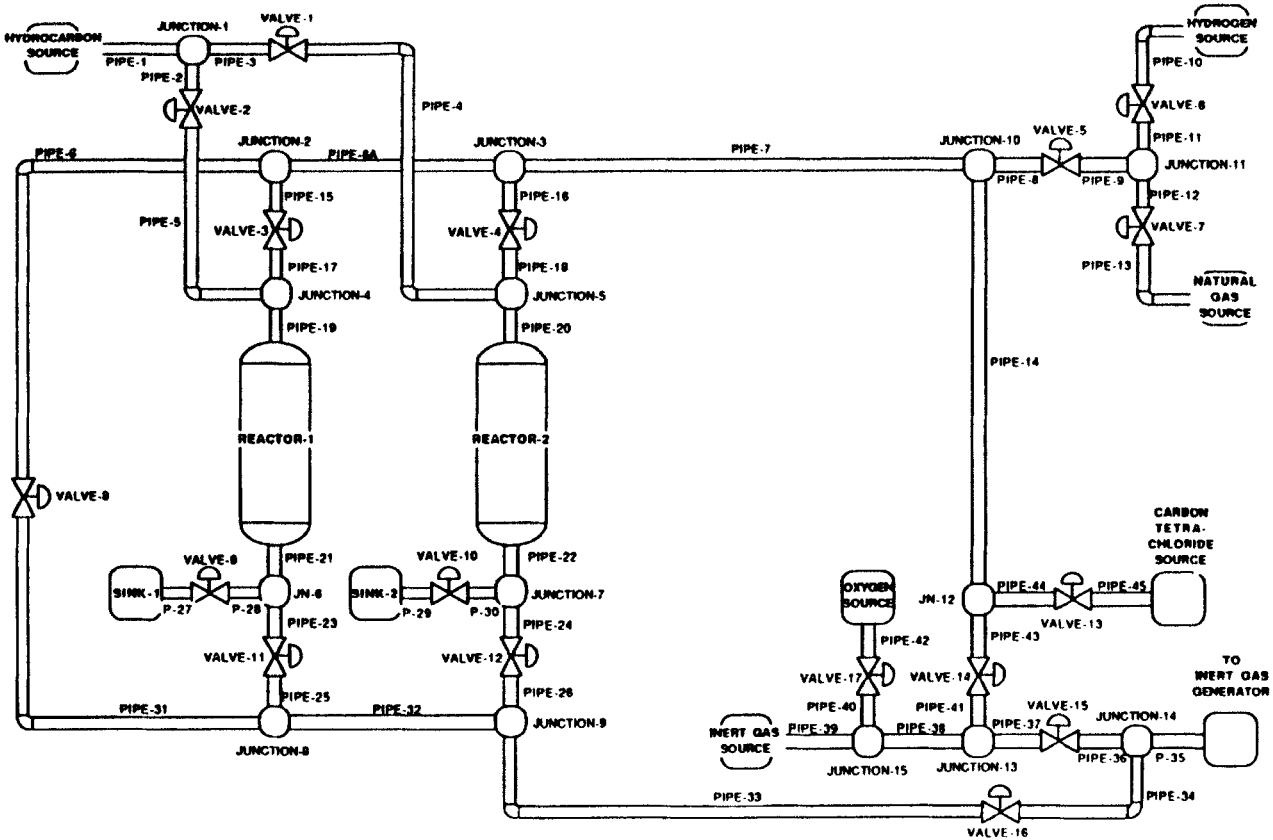


FIG. 8. Structure of the catalyst regeneration system. (Reprinted from *Comp. Chem. Eng.*, **12**, Lakshmanan, R. and Stephanopoulos, G., Synthesis of operating procedures for complete chemical plants, Parts I, II, p. 985, 1003, Copyright 1988, with kind permission from Elsevier Science Ltd., The Boulevard, Langford Lane, Kidlington 0X5 1GB, UK.)

changeover operating procedures, the reader is referred to Lakshmanan (1989).

The flowsheet editing facilities of *Design-Kit* (Stephanopoulos *et al.*, 1987) allow the user to construct an iconic representation of the flowsheet shown in Fig. 8, while the system at the same time creates the requisite data structures to represent the topology of the process (i.e., units and their interconnections). These data structures represent instances of the basic modeling elements of MODEL.LA. (see **21**:1). For example, Fig. 9a shows the relevant attributes in the declarative description of "Power-Forming-Plant," which is an instance of the modeling element (see Section II,B), *Plant*. The latter is, in turn, a subclass of the generic modeling element, *Generic-Unit*. Every device in the plant of Fig. 9a is an instance of the modeling element, *Unit*. Figure 9b shows the relevant declarative description of the unit REACTOR-1. Since abstraction in the representation of a plant is essential for the synthesis of operating procedures, the user can represent the "Power-Forming-Plant" as composed of two instances of *Plant-Section* called "Power-Forming-Reaction-Section" and "Power-Forming-Catalyst-Regeneration-Section" (see Fig. 9c). Figure 9d shows the relevant declarative elements of the "Power-Forming-Reaction-Section," whose boundaries (i.e., component units and streams) have been defined by the user. MODEL.LA. allows the creation of mathematical models around any abstraction of the processing units, while maintaining internal consistency between the model of an abstract entity (e.g., Plant-Section) and those of its constituents. Figure 10 shows the classes modeling a flow-valve and their interrelationships. For more details on the generation of the mathematical models, see **21**:1.

1. *Defining an Operating State and Ensuring Its Completeness and Consistency*

Any operations planning problem requires the *complete* and *consistent* definition of the initial and the goal states. In addition, during the evolution of an operating procedure any intermediate state must be fully specified by the operating actions and the modeling relations. Let us focus our attention on one segment of the desired operating procedure, e.g., "take REACTOR-1 off-line and bring REACTOR-2 on-line."

In the *initial state*, REACTOR-1 is operating while REACTOR-2 is on standby. Consequently,

Condition-1: Hydrocarbon-Flow-in-REACTOR-1 > 0
Condition-2: Product-Flow-from-REACTOR-1 > 0
Condition-3: Hydrocarbon-Flow-in-REACTOR-2 = 0
Condition-4: Product-Flow-from-REACTOR-2 = 0

Instance: POWER-FORMING-PLANT
Is-a-member-of: PLANT
Is-composed-of: [REACTOR-1, REACTOR-2, VALVE-1, VALVE-2, ..., VALVE-17,
JUNCTION-1, ..., JUNCTION-15, PIPE-1, ..., PIPE-42,
HYDROCARBON-SOURCE, SINK-1, SINK-2, HYDROGEN-SOURCE,
NATURAL-GAS-SOURCE, CARBON-TETRACHLORIDE-SOURCE,
⋮ OXYGEN-SOURCE, INERT-GAS-SOURCE, TO-REGENERATOR]
⋮
Note: "X" is-a-member-of "Unit"
where "X" is any device in the above list.

(a)

Instance: REACTOR-1
Is-a-member-of: UNIT
Is-composed-of: []
Ports: [PORT-1, PORT-2]
⋮
Note: "PORT-1" is-a-member-of "PORT"
"PORT-2" is-a-member-of "PORT"

(b)

Instance: POWER-FORMING-PLANT
Is-a-member-of: PLANT
Is-composed-of: [POWER-FORMING-REACTION-SECTION,
⋮ POWER-FORMING-CATALYST-]-REGENERATION-SECTION]
⋮

(c)

Instance: POWER-FORMING-REACTION-SECTION
Is-a-member-of: PLANT-SECTION
Is-composed-of: [REACTOR-1, REACTOR-2, VALVE-1, ..., VALVE-4, VALVE-9, VALVE-12,
JUNCTION-1, ..., JUNCTION-9, PIPE-1, ..., PIPE-7, PIPE-15, ..., PIPE-26,
PIPE-31, ..., PIPE-33, HYDROCARBON-SOURCE, SINK-1, SINK-2]
⋮
⋮

(d)

FIG. 9. Object-oriented descriptions of the Power-Forming-Plant at two levels of abstraction: (a) in terms of processing units, and (c) in terms of processing sections. The descriptions of a processing unit (b) and a processing section (c).

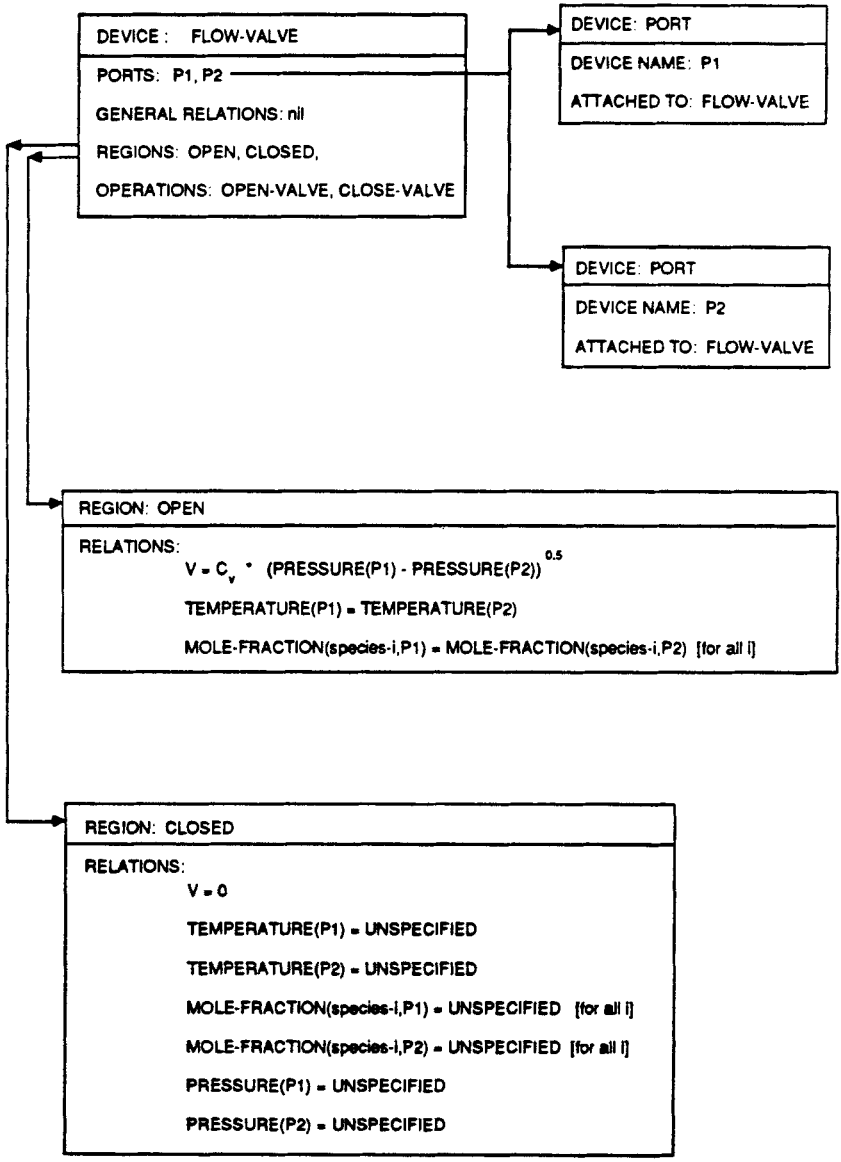


FIG. 10. Data structure modeling a flow-valve. (Reprinted from *Comp. Chem. Eng.*, **12**, Lakshmanan, R. and Stephanopoulos, G., Synthesis of operating procedures for complete chemical plants, Parts I, II, p. 985, 1003, Copyright 1988, with kind permission from Elsevier Science Ltd., The Boulevard, Langford Lane, Kidlington 0X5 1GB, UK.)

Condition-1, when propagated through the modeling relationships, yields the following; VALVE-2 is OPEN and VALVE-3 is CLOSED. These values are then stored in the instances of the corresponding valves. Similarly, we take the following associations:

Condition-2 ⇒ [VALVE-9 is OPEN; VALVE-11 is CLOSED];
Condition-3 ⇒ [VALVE-1 is CLOSED; VALVE-4 is CLOSED];
Condition-4 ⇒ [VALVE-10 is CLOSED; VALVE-12 is CLOSED]

The propagation of these conditions through the modeling relationships is fairly straightforward, given the declarative richness of the data structures (e.g., see Fig. 10) representing the units of the plant, their interconnections and their physical behavior. Figure 11 shows the information flow through the modeling dependencies of variables, as Conditions 1–4 are propagated and produce the states of the associated operators (valves).

2. *Ensuring the Completeness of the Initial State*

The propagation of initial conditions through the modeling relationships is also used to ensure completeness in the definition of an initial state. For example, Condition-1 implies that the state of PIPE-1, PIPE-2, and PIPE-3 is characterized by the state of the flowing (or, stagnant, in PIPE-3) hydrocarbon. Similarly, Condition-2, when propagated backward, implies that the state of PIPE-27, PIPE-28, PIPE-21, and PIPE-23 is determined by the state of the flowing or stagnant product. The available initial conditions may not be able to define all facets of the initial state of a plant. For example, Conditions-1–3 cannot produce unambiguously the value of the state for VALVE-8, PIPE-6, and PIPE-31 (see Fig. 8). This is due to the fact that there is no directed path in the graph representing all modeling relationships, between the variables of the four conditions and the variables describing the state of VALVE-8, PIPE-6, and PIPE-31. In such cases the user must intervene and complete the specification for the initial state of the whole plant.

3. *Checking the Consistency of the Initial State*

Whereas the propagation of conditions through the modeling relationships always produces consistent completions in the definition of the initial state, this may not be true with the user-driven specifications. It is possible that a valve specified by the user to be OPEN, is "found" by the propagation of other conditions to be CLOSED. To detect such potential inconsistencies, we have developed a *dependency network* (Steele, 1980) to keep track of the flow of computations during constraint propagation. The dependency network is built on the undirected graph that represents the

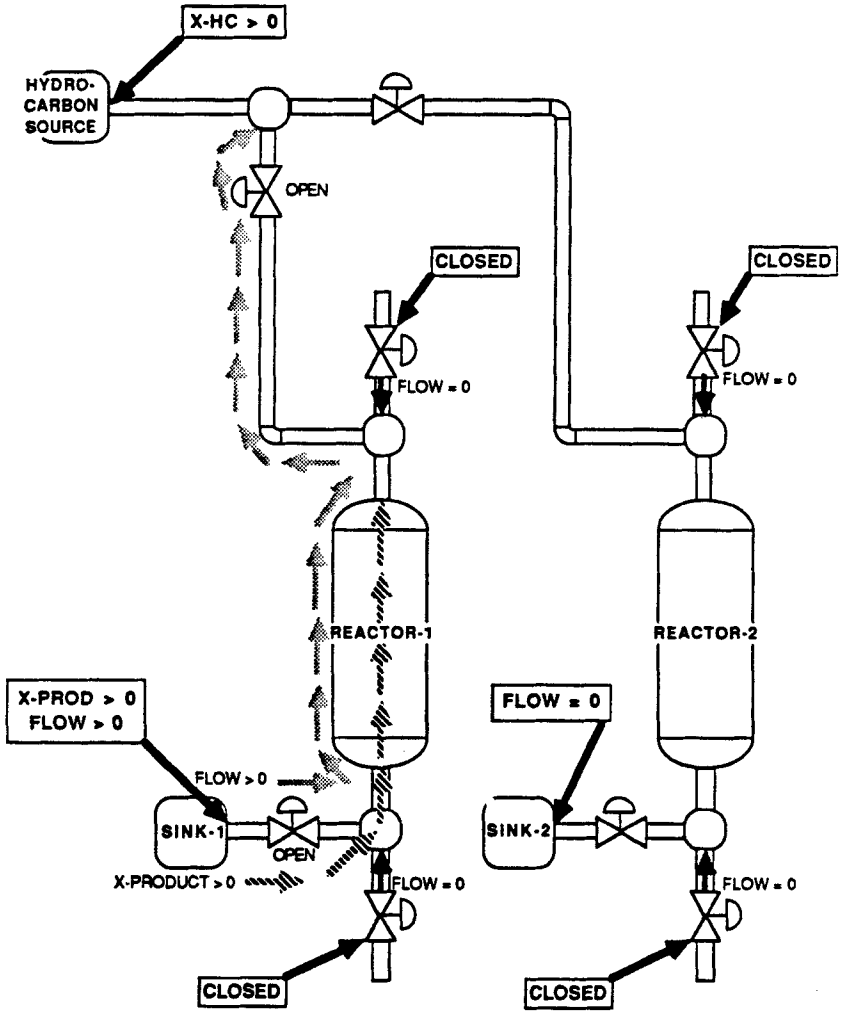


FIG. 11. Ensuring completeness of the initial state. (Reprinted from *Comp. Chem. Eng.*, 12, Lakshmanan, R. and Stephanopoulos, G., Synthesis of operating procedures for complete chemical plants, Parts I, II, p. 985, 1003, Copyright 1988, with kind permission from Elsevier Science Ltd., The Boulevard, Langford Lane, Kidlington 0X5 1GB, UK.)

set of modeling relationships. With the given initial states as input variables, the undirected graph of modeling relationships is converted to a directed graph (dependency network), indicating the directionality of dependencies during the computation of initial states. If a conflict is encountered, backtracking through the dependency network reveals the source of conflict.

4. Complete and Consistent Definition of the Goal State

The mechanics for the definition of the goal state are fairly similar to those used for the definition of the initial state. Since the goal state is far more vague in the planners' mind than the initial state, the potential for conflicts is higher. Figure 12 shows the partial specification of the goal state, as well as the propagation of conditions to ensure completeness in the definition of the goal state.

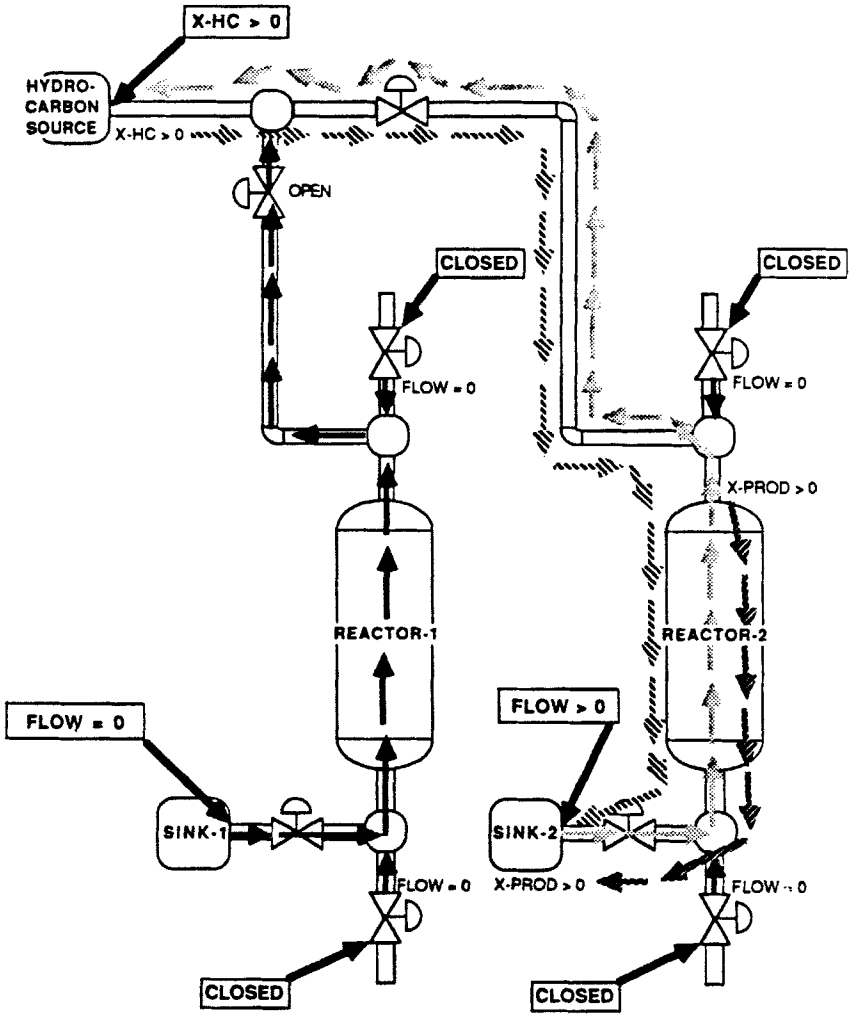
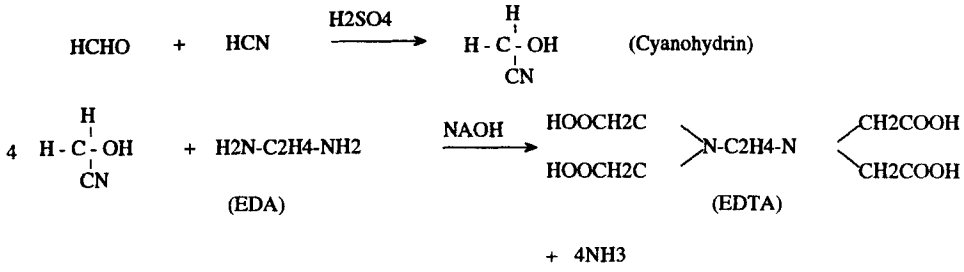


FIG. 12. Input of goal state and generation of concurrent goals. (Reprinted from *Comp. Chem. Eng.*, **12**, Lakshmanan, R. and Stephanopoulos, G., Synthesis of operating procedures for complete chemical plants, Parts I, II, p. 985, 1003, Copyright 1988, with kind permission from Elsevier Science Ltd., The Boulevard, Langford Lane, Kidlington 0X5 1GB, UK.)

B. Nonmonotonic Synthesis of a Switchover Procedure

Ethylenediaminetetraacetic acid (EDTA) is manufactured by the cyanomethylation process, whose main reactions are



Sulfuric acid, formaldehyde, and hydrogen cyanide are pumped into a glass-lined mixer (mixer 1, M1, of Fig. 13). Particular care is exercised so that the three charge operations are carried out in the order indicated above, to ensure the stability of the mixture at all times. In a separate segment of the plant, ethylenediamine (EDA) and dilute sodium hydroxide are charged and mixed in mixer 3 (M3 in Fig. 13). The solutions from mixer 1 and mixer 3 are pumped to the reactor (REACTOR, R1, in Fig. 13). When the reaction is complete, the reaction mixture is tested for traces of hydrogen cyanide. Dilute solution of formaldehyde is prepared in mixture 2 and is added to the reaction mixture, if there is any HCN present.

1. The Operating Situation Requiring a Switchover Procedure

The specific situation for which we want to synthesize an operating procedure is as follows. Hydrogen cyanide is handled by the top most line of piping, pumps, and valves (Fig. 13). Sulfuric acid and formaldehyde are handled by the next two flowlines in the flowsheet. The fourth flowline, which passes through pump PO4, has been shut down for routine maintenance. At some point during the operation of the plant, a leak is detected in the cooling jacket of mixer 1 and it has to be shut down in order to be repaired. Mixer 2 can be used in emergencies for limited periods of time only, because it is not a glass-lined vessel. As a result of this situation, we need to develop an operating procedure that will (1) shut mixer 1 down and disconnect it from the rest of the plant and (2) divert the flow of these materials, i.e., hydrogen cyanide, sulfuric acid, and formaldehyde, to mixer 2.

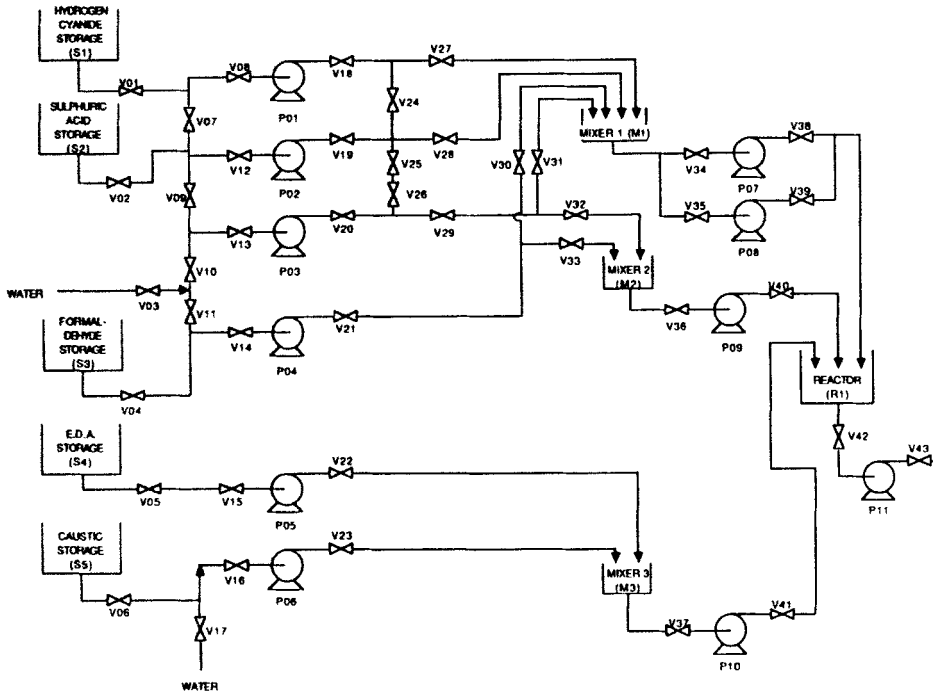


FIG. 13. Flowsheet of the ethylenediaminetetraacetic acid (EDTA) plant.

2. *Constraints on the Operating Procedure*

As we switch the operation from mixer 1 to mixer 2, certain constraints must be kept satisfied at all times. These constraints are as follows:

Constraint-1. Nowhere in the plant is HCN allowed to mix with formaldehyde.

Constraint-2, 3. The concentrations of HCN and H_2SO_4 in the mixer should satisfy the following constraints at all times:

$$[\mathrm{HCN}] \leq 0.1 \text{ mol/ L}$$

$$[\mathrm{H_2SO_4}] \geq 1 \text{ mol/ L}$$

Constraint-4. The temperature in the mixer should not exceed 30°C:

$$T_{M2} \leq 30°\mathrm{C}$$

3. Initial and Goal States

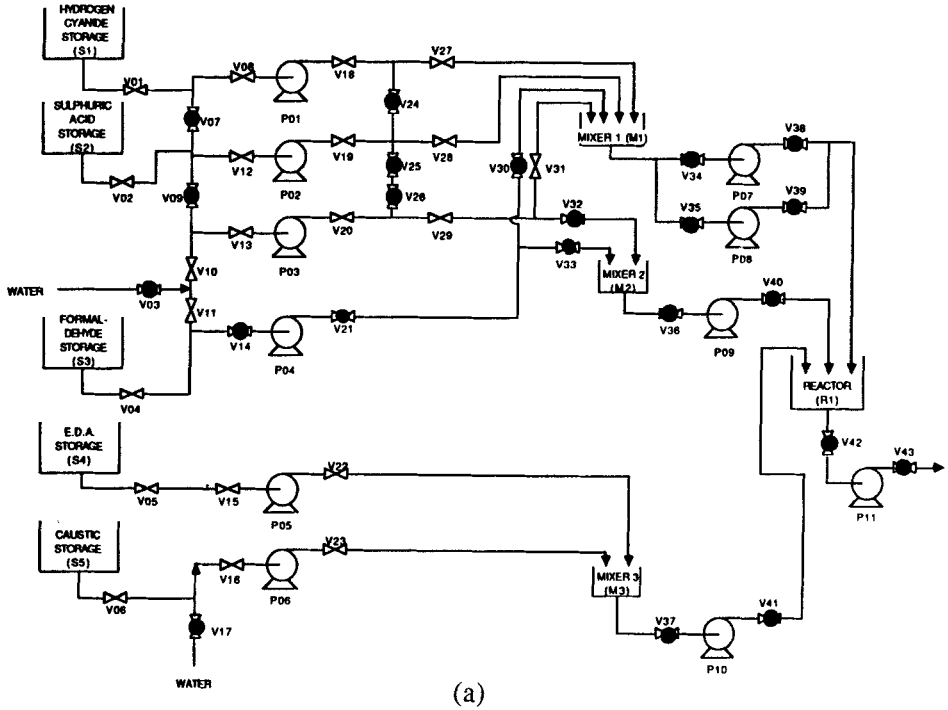
The starting situation is partially defined by the state of the following valves:

$$\{\text{CLOSED-VALVES}\} = \{\text{V03, V07, V09, V14, V17, V24, V25, V26, V30, V32}\}$$
$$\{\text{OPEN-VALVES}\} = \{\text{V22, V23, V27, V28, V29}\}$$

Propagation of the above states through the modeling relationships completes the definition of the initial state of the plant. The results are shown in Fig. 14a. (Note that valves V34 through V43, which are downstream of the mixers, are considered closed, and they do not affect the synthesis of the switchover procedure.) The desired goal state is operationally defined by the following conditions:

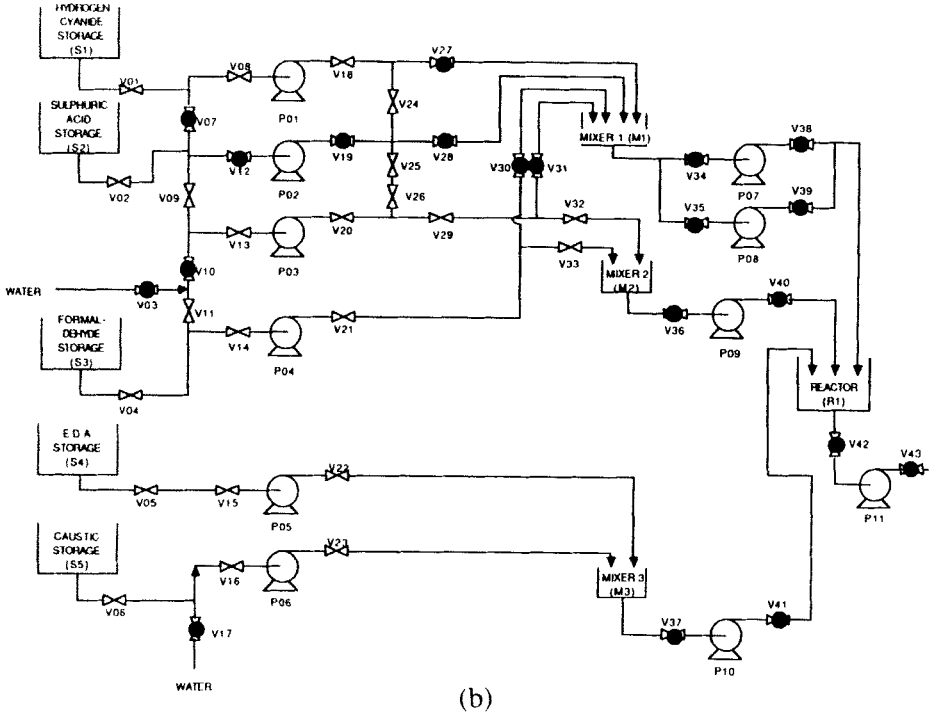
Mixer-1-input-flow-i = 0 i = HCN, H_2SO_4, HCHO
Mixer-1-output-flow = 0
Mixer-2-input-flow-i = positive i = HCN, H_2SO_4, HCHO

Propagating these values through the modeling equations, we can complete the definition of the goal state (see Fig. 14b). Comparing initial and



(a)

FIG. 14. The (a) initial and (b) goal states for the EDTA planning problem.



(b)

FIG. 14. *Continued*.

goal states, we conclude that the means–ends analysis requires the availability of the following operations to achieve the desired goal state:

V = {CLOSE V27, CLOSE V28, CLOSE V31, CLOSE V10, OPEN V9, CLOSE V12, CLOSE V19, OPEN V24, OPEN V25, OPEN V26, OPEN V32, OPEN V14, OPEN V21, OPEN V33}.

4. *Abstraction of Operators*

In Section III,D we presented an algorithm for the aggregation of primitive valve actions into abstract operators. Applying this algorithm over the set V of actions to achieve the goal state, we take:

AO-1: {CLOSE V27} ⇒ Stop-Flow-HCN-M1
AO-2: {CLOSE V12, CLOSE V19, CLOSE V28} ⇒ Stop-Flow-H_2SO_4-M1
AO-3: {CLOSE V10, CLOSE V31} ⇒ Stop-Flow-HCHO-M1
AO-4: {OPEN V24, OPEN V25, OPEN V26, OPEN V32}
⇒ Start-Flow-HCN-M2
AO-5: {OPEN V09, OPEN V32} ⇒ Start-Flow-H_2SO_4-M2
AO-6: {OPEN V14, OPEN V21, OPEN V33} ⇒ Start-Flow-HCHO-M2.

Then, the set V takes the following form:

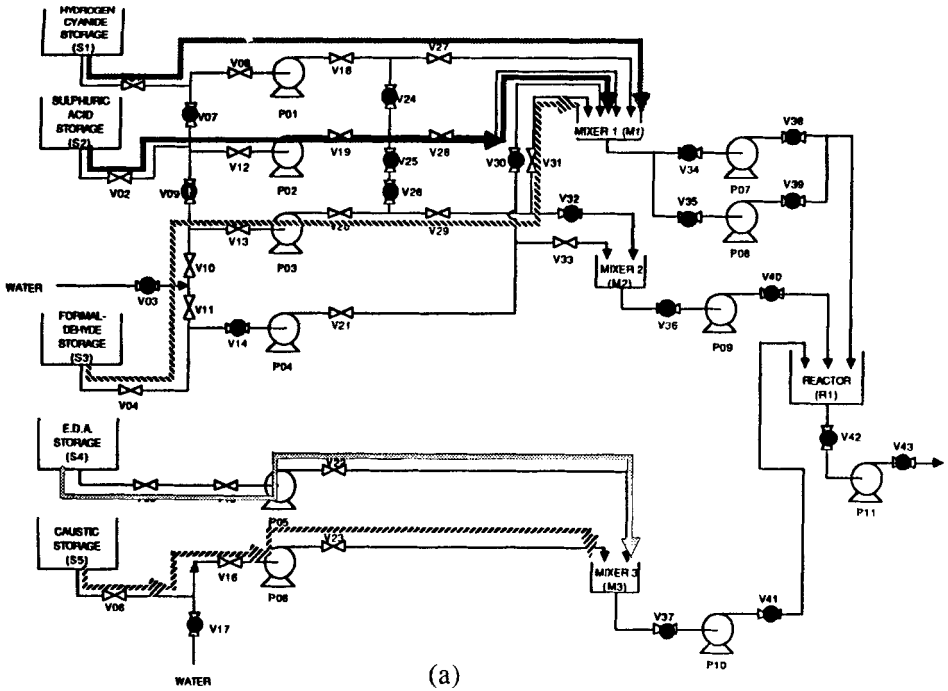
$$V = \{AO\text{-}1, AO\text{-}2, AO\text{-}3, AO\text{-}4, AO\text{-}5, AO\text{-}6\}.$$

5. Transformation of Constraint-1 (Mixing Constraint)

In Section III,C we argued that one should identify the worst-case scenario for the violation of Constraint-1 and take action to ensure that a White Knight operation is carried out before any offending operation (i.e., Clobberer), in order to negate its impact (i.e., violation of mixing constraint). The worst-case scenario occurs when the plant has the following two sets of valves OPEN at the same time:

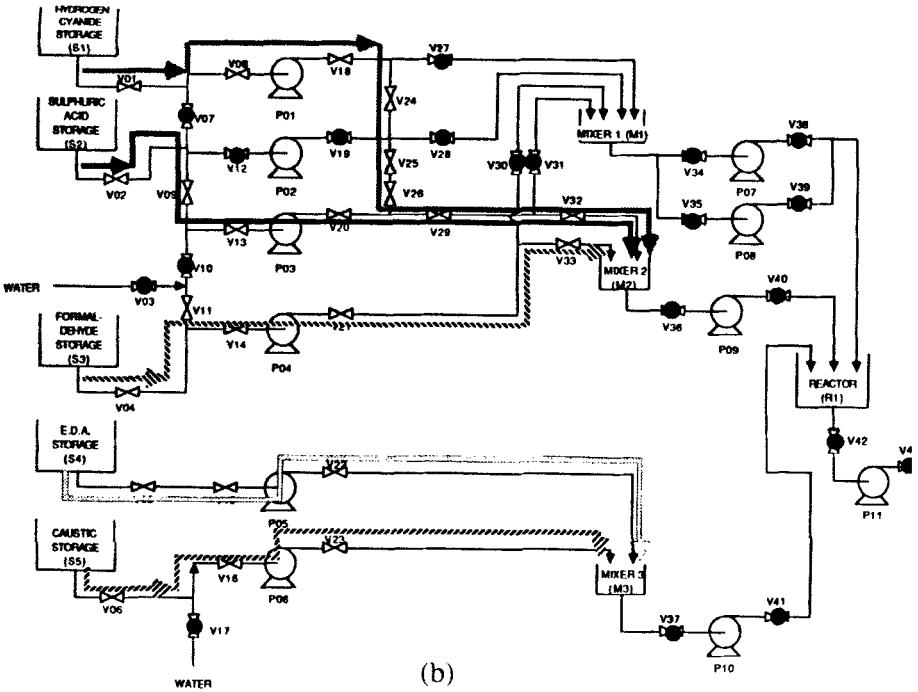
{Valves that are OPEN at the initial state}.
{Valves that are characterized as OPEN at the goal state}.

Then, we construct the influence graphs (IGs) of HCN and HCHO for the initial and goal states (Fig. 15), and from these we can identify the



(a)

FIG. 15. Influence graphs in the (a) initial and (b) goal states. (Reprinted from *Comp. Chem. Eng.*, **14**, Lakshmanan, R. and Stephanopoulos, G., Synthesis of operating procedures for complete chemical plants, Parts III, p. 301, Copyright 1990, with kind permission from Elsevier Science Ltd., The Boulevard, Langford Lane, Kidlington 0X5 1GB, UK.)



(b)

FIG. 15. *Continued.*

region of potential mixing between HCN and HCHO (intersection of IGs for HCN and HCHO) at the worst-case scenario (Fig. 16). From Fig. 16 we see that the IGs intersect at the valves V29 and V32. It is also clear that {V10} is the minimal separation valve set (MSVS; see Section III,C), i.e., the set of valves that, if closed, would prevent the mixing of HCN and HCHO. Thus, V10 must be closed before any valve in the initial state is opened. Looking at the elements of the set V in the previous paragraph, we derive the following temporal ordering of valve operations:

(CLOSE V10) before {OPEN V9, OPEN V24, OPEN V25, OPEN V26, OPEN V32, OPEN V14, OPEN V21, OPEN V33] (Ordering-1)

So the mixing constraint has been transformed into the above constraint on the temporal ordering of primitive operations.

6. *Generation of a Purge/Evacuation Procedure*

Valves V29 and V32 and the associated pipes contain leftovers of HCHO and must be purged with H_2SO_4 before the other chemical,

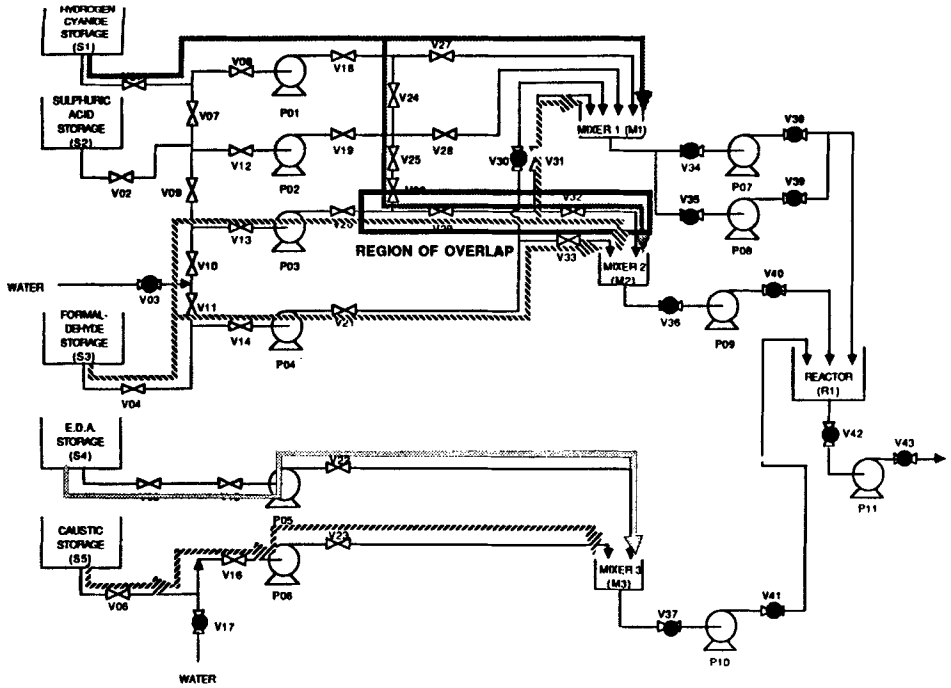


FIG. 16. Influence graphs in the worst-case state, showing the region of overlap. (Reprinted from *Comp. Chem. Eng.*, **14**, Lakshmanan, R. and Stephanopoulos, G., Synthesis of operating procedures for complete chemical plants, Parts III, p. 301, Copyright 1990, with kind permission from Elsevier Science Ltd., The Boulevard, Langford Lane, Kidlington 0X5 1GB, UK.)

i.e., HCN, is allowed to flow through. (Remember the required order of addition: H_2SO_4 before HCHO before HCN.) The path-finding algorithm (see Section III,C) identifies the following path from the storage of the purgative material (i.e., H_2SO_4) to mixer 2:

Purge Path = {PUMP PO3; VALVE V20; VALVE V29; VALVE V32}

This path represents a "planning island," i.e., the goals of an operating subprocedure, which must be completed after the MSVS has been closed, and before the operations that allow HCN in the line. Therefore, we have the following temporal constraints:

(CLOSE V10) before (PURGE OPERATIONS) before (OPEN V24) (Ordering-2)

where (PURGE OPERATIONS) is the sequence of operating steps re-

quired for the purging of HCHO and stands on its own as an independent operating procedure. The synthesis of primitive steps leads to

(PURGE OPERATIONS) = {OPEN V09, START PO3, OPEN V32, CLOSE V32, STOP PO3, CLOSE V09}

7. Transformation of Constraints-2 and -3

Suppose that we were to load mixer M2 with the following sequence of abstract operators (see earlier paragraph in this section for definition of abstract operators):

{(Start-Flow-HCN-M2), (Start-Flow-H_2SO_4-M2), (Start-Flow-HCHO-M2)}.

If the initial concentrations of the raw material were (in moles per liter), $[HCN]_0 = 1$, $[H_2SO_4]_0 = 1.5$, and $[HCHO]_0 = 2$, then the preceding sequence of operations would immediately have violated Constraint-2, since $[HCN] = 1 > 0.1$. Similarly, Constraint-3 is violated by the preceding sequence, since $[HSO_4] = 0 < 1.5$. In Section III,D, we discuss the notion of "demotion" of Clobberers. Let us see how it works here and leads to temporal ordering of operations:

(a) (Start-Flow-HCN-M2) is a Clobberer of $[HCN] \leq 0.1$. Demoting this Clobberer, we require that

(Start-Flow-HCN-M2) after (Start-Flow-H_2SO_4-M2; Start-Flow-HCHO-

(b) (Start-Flow-HCN-M2) and (Start-Flow-HCHO-M2) are Clobberers of $[H_2SO_4] \geq 1$. Demoting both Clobberers, we establish the following temporal ordering constraints:

(Start-Flow-H_2SO_4-M2) before
(Start-Flow-HCN-M2; Start-Flow-HCHO-M2)

Taking (a) and (b) together, we see that Constraints-2 and -3 have been transformed to the following temporal ordering:

(Start-Flow-H_2SO_4-M2) before (Start-Flow-HCHO-M2) before (Start-Flow-HCN-M2) (Ordering-3)

8. Transformation of Constraint-4

Let the temperature of the three materials in the storage tanks be 35°C. Then, all three operations, (Start-Flow-HCN-M2), (Start-Flow-H_2SO_4-M2)

and (Start-Flow-HCHO-M2), are Clobberers of the Constraint-4, since $T_{M2} = 35°C > 30°C$. No demotion of any of the three operators would restore feasibility, and we are led to the need for a White Knight. Searching the plant for relevant operations we identify the following White Knight:

(OPEN VALVE-CW) ⇒ Start-Cooling-Water-through-Jacket

and we constrain the White Knight to be carried out before any of three Clobberers, i.e.,

(OPEN VALVE-CW) before
[(Start-Flow-HCN-M2), (Start-Flow-H_2SO_4-M2), (Start-Flow-HCHO-M2)] (Ordering-4)

9. Synthesis of the Complete Switchover Operating Procedure

In the previous paragraphs we saw how the ideas of nonmonotonic planning have transformed the required operational constraints into constraints ordering the temporal sequencing of operations. These temporal constraints define partial segments of the overall operating procedure. Now, let us see how they can be merged into a complete operating procedure. Let us recall the set V of operations, which was defined earlier through the means–ends analysis. Any complete plan must carry out the abstract operators AO-1 through AO-6. From each temporal ordering we take the following implications:

Ordering-1 ⇒ (AO-3: Stop-Flow-HCHO-M1) before (AO-4; AO-5; AO-6).

Ordering-2 ⇒ (AO-3) before (PURGE OPERATIONS) before (AO-4).

Ordering-3 ⇒ (AO-5) before (AO-6) before (AO-4).

Ordering-4 ⇒ (OPEN VALVE-CW) before (AO-4, AO-5, AO-6).

The following three sequences satisfy all the preceding constraints on the temporal ordering of operations:

Procedure-A. (AO-3) > (PURGE OPERATIONS) > (OPEN VALVE-CW) > (AO-5) > (AO-6) > (AO-4).

Procedure-B. (AO-3) > (OPEN VALVE-CW) > (PURGE OPERATIONS) > (AO-5) > (AO-6) > (AO-4).

Procedure-C. (OPEN VALVE-CW) > (AO-3) > (PURGE OPERATIONS) > (AO-5) > (AO-6) > (AO-4).

Each of the abstract operators, AO-3, AO-4, AO-5, AO-6, is composed of

more than one operation. No constraints have been stated or generated to determine their relative order. For example, (AO-3) could be implemented as (CLOSE V10) > (CLOSE V31) or (CLOSE V31) > (CLOSE 10).

It is also important to note that abstract operators (AO-1) and (AO-2) are not participating in any temporal ordering constraint. Consequently, they could be placed anywhere in the chain of operations of a procedure. For example, *Procedure-A* can be completed with (AO-1) and (AO-2) in any of the following ways:

Procedure A-1. (AO-1) > (AO-2) > (AO-3) > (OPEN VALVE-CW) > ....

Procedure A-2. (AO-1) > (AO-3) > (AO-2) > (OPEN VALVE-CW) > ....

Procedure A-3. (AO-2) > (AO-1) > (AO-3) > (OPEN VALVE-CW) > ....

Procedure A-4. (AO-2) > (AO-3) > (AO-1) > (OPEN VALVE-CW) > ....

Procedure A-5. (AO-3) > ··· > (AO-5) > (AO-6) > (AO-1) > (AO-2).

Procedure A-6. (AO-3) > ··· > (AO-5) > (AO-6) > (AO-2) > (AO-1) etc.

V. Revamping Process Designs to Ensure Feasibility of Operating Procedures

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 chapter, the worst-case scenario 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 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 can be found to counter at least one of the Clobberers. In these instances a modification must be made to the process flowsheets so as to introduce structures that will serve as White Knights. In this section we

describe algorithms that 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 scenario. The White Knights that are needed to counter this Clobberer are (a) an acceptable separation valve set (ASVS), and (b) a purgative and a purge route. In this analysis, we shall assume that the absence of White Knights is due to one of these two situations. Thus the modifications to the flowsheet must provide either an ASVS or a purgative and purge route (or both) depending on the needs of the situation at hand. The remainder of this section provides a methodology for accomplishing both these tasks.

A. Algorithms for Generating Design Modifications

In this subsection, we present the algorithms that 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 ASVS, and those that are used for generating a purge source and a purge route.

1. Flowsheet Modifications for Generating an ASVS

The methodology for proposing flowsheet modifications to ensure the existence of an ASVS is described in this section.

a. Stage 1. Determine a Minimal Separation Pipe Set

1. Construct the reduced pipe network corresponding to the topological graph of the worst-case state.
   (a) If the current node represents a PIPE or a "supersource," do nothing.
   (b) If the current node represents any other type of processing equipment, do the following:
      (i) Delete the current node from the topological graph.
      (ii) Let U be the set of nodes incident to the in-edges of the current node.
      (iii) Let V be the set of nodes incident to the out-edges of the current node.
      (iv) 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.

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 MSPS may not be an acceptable separation pipe set (ASPS) 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.

b. Stage 2. Determine an Acceptable Separation Pipe Set

1. Let the current separation pipe set (SPS) of edges be the MSPS already determined.
2. Identify the pipes (if any) in the current SPS that 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 step 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 that 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 ASPS.
4. Remove all "unsatisfactory" pipes from the reduced pipe network. Go to stage 1, step 2.

Once an ASPS has been found, the final stage of the methodology can be started.

c. Stage 3. Generation of an ASVS. For each pipe in the ASVS, locate a corresponding pipe in the topological graph. For each pipe located in the topological graph, place a valve on the pipe. The set of valves that are added to the flowsheet in this step will form an ASVS 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 ASVS was found, the preceding algorithm may be used to propose flowsheet modifications (in the form of extra valves) to create an appropriate ASVS.

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, address only the structural modifications that 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 scenario are defined. This may not necessarily be the case, but as yet no algorithm is known that 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 the valves that are members of the ASVS. 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 that 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″ that consists of these nodes and their incident edges. For each of the graphs G′ and G″, do the following:
   (a) Find all the minimal and maximal nodes in G′ (or G″) (see Lakshmanan, 1989).
   (b) Create a supersource of the chemical species that was identified (by either the user or 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 supersource to the minimal node.
   (c) 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.

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

3. *Case Studies*

In this subsection we examine two modifications of the EDTA plant discussed earlier, in Section IV. The purpose of the illustration is to demonstrate the working of the algorithms that generate flowsheet modifications when (1) no MSVS or ASVS can be found and (2) no purge source or purge route is available.

In this example the flowsheet of the EDTA process has been modified by removing the following valves: v10, v11, v13, and v20. The result of the modifications is shown in Fig. 17.

When the planning algorithms are run on this plant no MSVS is found since, in the worst-case state, there are no valves that will prevent hydrogen cyanide and formaldehyde from mixing in the region of overlap. Thus the computer will try to use the methodology described in the

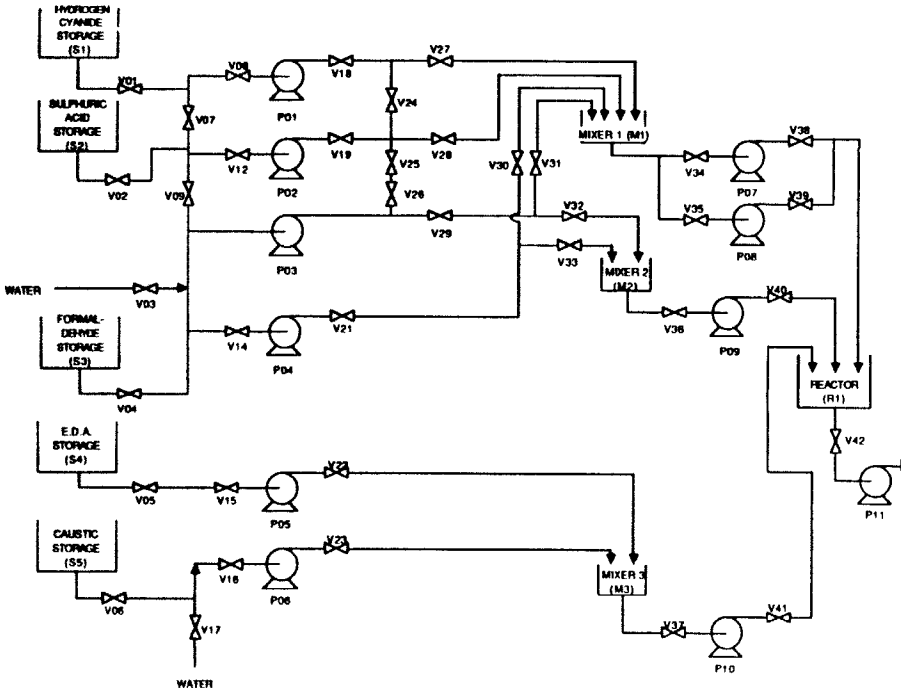


FIG. 17. Flowsheet of the modified EDTA plant.

previous paragraphs to come up with flowsheet modifications that will allow the computer to find an MSVS.

The algorithms described earlier 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 Fig. 18, and this valve serves as the MSVS in the planning problem.

The second example that 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 Section IV. The only change to the problem is the statement that sulfuric acid and formaldehyde should not be allowed to come into contact with each other.

It may be recalled that in the initial analysis sulfuric acid was used as a purge species. It is obvious that this is not possible anymore since one of the dangerous components, namely formaldehyde, cannot be brought into contact with sulfuric 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 earlier is run under the worst-case scenario of the original EDTA plant with the added

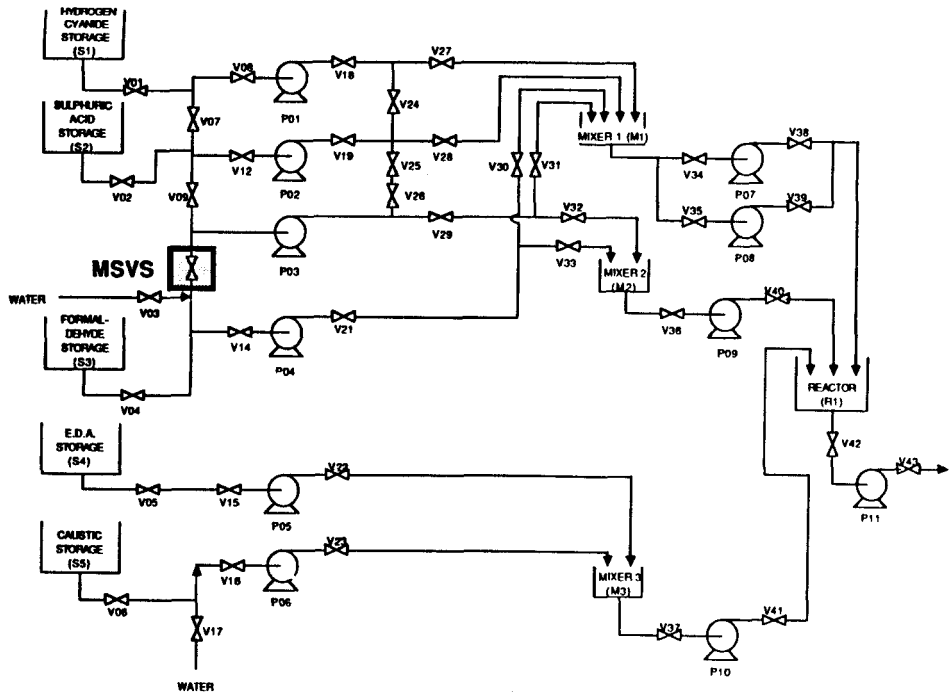


FIG. 18. Flowsheet of the EDTA plant with MSVS.

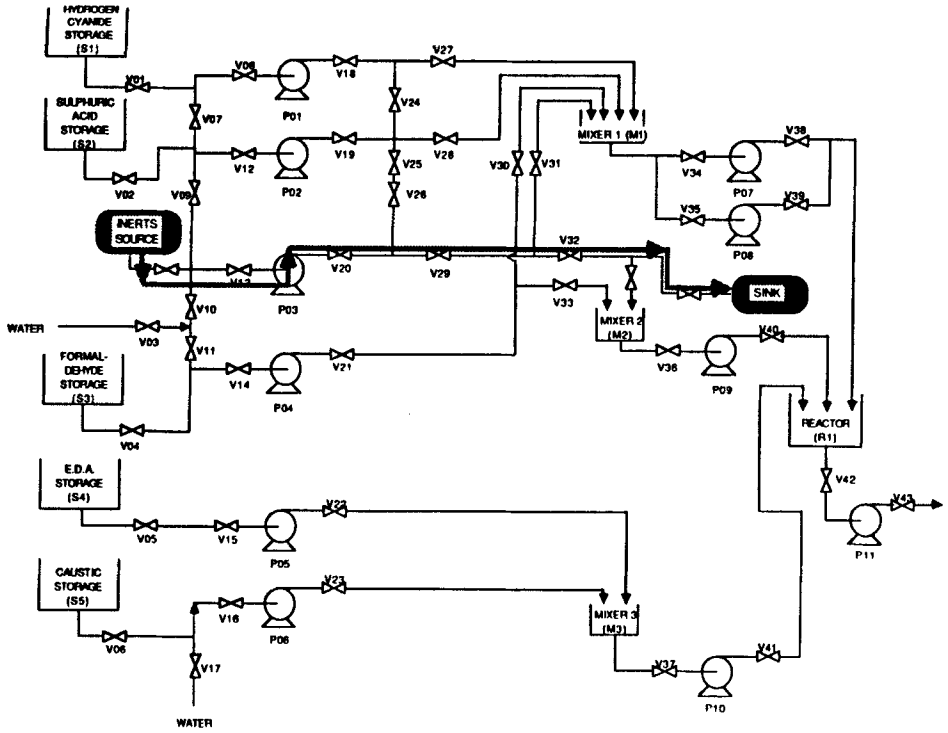


FIG. 19. Flowsheet of the EDTA plant showing purge route.

mixing constraint, and piping modifications are generated in the flowsheet. The final flowsheet with the purge route is shown in Fig. 19.

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 of explosive mixtures is needed to accomplish this task. The construction of such a knowledge base, while useful, is assumed to be outside the scope of the planning problem. The program is designed only to propose the piping and valving modifications to the flowsheet.

VI. Summary and Conclusions

The synthesis of operating procedures for continuous chemical plants can be represented as a mixed-integer nonlinear programming problem, and it has been addressed as such by other researchers. In this chapter we have attempted to present a unifying theoretical framework, which ad-

dressed the logical (or integer) components of the problem, while allowing a smooth integration of optimization algorithms for the optimal selection of the values of the continuous variables. Nonmonotonic reasoning has been found to be superior to the traditional monotonic reasoning approaches, and capable to account for all integer decision variables. Critical in its success is the modeling approach, which is used to represent process operational tasks at various levels of abstraction.

Nevertheless, a series of interesting issues arise, which must be resolved before these ideas find their way to full industrial implementation:

1. *Dynamic simulation with discrete-time events and constraints.* In an effort to go beyond the integer (logical) states of process variables and include quantitative descriptions of temporal profiles of process variables one must develop robust numerical algorithms for the simulation of dynamic systems in the presence of discrete-time events. Research in this area is presently in full bloom and the results would significantly expand the capabilities of the approaches, discussed in this chapter.

2. *Design modifications for the accommodation of feasible operating procedures.* Section V introduced some early ideas on how the ideas of planning operating procedures could be used to identify modifications to a process flowsheet, which are necessary to render feasible operating procedures. More work is needed in this direction. Clearly, any advances in dynamic simulation with discrete-time events would have beneficial effects on this problem.

3. *Real-time synthesis of operating procedures.* Most of the ideas and methodologies, presented in this chapter, are applicable to the a priori, off-line, synthesis of operating procedures. There is a need though to address similar problems during the operation of a chemical plant. Typical examples are the synthesis of operational response (i.e., operating procedure) to process upsets, real-time recovery from a fallback position, and supervisory control for constrained optimum operation.

References

Barton, P. I., The modeling and simulation of combined discrete/continuous processes. Ph.D. Thesis, University of London (1992).

Chakrabarti, P. P., Ghose, S., and DeSarkar, S. C., Heuristic search through islands. *Artif. Intell.* **29**, 339 (1986).

Chapman, D., "Planning for Conjunctive Goals," MIT AI Lab. Tech. Rep. AI-TR-802. Massachusetts Institute of Technology, Cambridge, MA, 1985.

Crooks, C. A., and Macchietto, S., A combined MILP and logic-based approach to the synthesis of operating procedures for batch plants. *Chem. Eng. Commun.* **114**, 117 (1992).

Even, S., "Graph Algorithms." Computer Science Press, Rockville, MD, 1979.

Fikes, R. E., and Nilsson, N. J., STRIPS: A new approach to the application of theorem proving to problem solving. *Artif. Intell.* **2**, 198 (1971).

Foulkes, N. R., Walton, M. J., Andow, P. K., and Galluzzo, M., Computer-aided synthesis of complex pump and valve operations. *Comput. Chem. Eng.* **12**, 1035 (1988).

Fusillo, R. H., and Powers, G. J., A synthesis method for chemical plant operating procedures. *Comput. Chem. Eng.* **11**, 369 (1987).

Fusillo, R. H., and Powers, G. J., Operating procedure synthesis using local models and distributed goals. *Comput. Chem. Eng.* **12**, 1023 (1988a).

Fusillo, R. H., and Powers, G. J., Computer aided planning of purge operations. *AIChE J.* **34**, 558 (1988b).

Garrison, D. B., Prett, M., and Steacy, P. E., Expert systems in process control. A perspective. *Proc. AAAI Workshop Control*, Philadelphia (1986).

Holl, P., Marquardt, W., and Gilles, E. D., Diva—a powerful tool for dynamic process simulation. *Comput. Chem. Eng.* **12**, 421 (1988).

Ivanov, V. A., Kafarov, V. V., Kafarov, V. L., and Reznichenko, A. A., On algorithmization of the startup of chemical productions. *Eng. Cybern. (Engl. Transl.)* **18**, 104 (1980).

Kinoshita, A., Umeda, T., and O'Shima, E., An approach for determination of operational procedure of chemical plants. *Proc. PSE'82, Kyoto* 114 (1982).

Lakshmanan, R., and Stephanopoulos, G., "Planning and Scheduling Plant-Wide Startup Operations," LISPE Tech. Rep. (1987).

Lakshmanan, R., and Stephanopoulos, G., Synthesis of operating procedures for complete chemical plants. I. Hierarchical, structured modeling for nonlinear planning. *Comput. Chem. Eng.* **12**, 985 (1988a).

Lakshmanan, R., and Stephanopoulos, G., Synthesis of operating procedures for complete chemical plants. II. A nonlinear planning methodology. *Comput. Chem. Eng.* **12**, 1003 (1988b).

Lakshmanan, R., Synthesis of operating procedures for complete chemical plants. Ph.D. Thesis. Massachusetts Institute of Technology, Dept. Chem. Eng., Cambridge, MA (1989).

Lakshmanan, R., and Stephanopoulos, G., Synthesis of operating procedures for complete chemical plants. III. Planning in the presence of qualitative, mixing constraints. *Comput. Chem. Eng.* **14**, 301 (1990).

Marquardt, W., Dynamic process: Simulation-recent progress and future challenges. *In* "Chemical Process Control, CPC-IV" (Y. Arkun, and W. H. Ray, eds.) CACHE, AIChE Publishers, New York, 1991.

Modell, M., and Reid, R. C., "Thermodynamics and its Applications," 2nd ed., Prentice-Hall, Englewood Cliffs, NJ, 1983.

Newell, A., Shaw, J. C., and Simon, H. A., Report on a general problem solving program. *Proc. ICIP*, 256 (1960).

Pantelides, C. C., and Barton, P. I., Equation-oriented dynamic simulation. Current status and future perspectives. *Comput. Chem. Eng.* **17**, S263 (1993).

Pavlik, E., Structures and criteria of distributed process automation systems. *Comput. Chem. Eng.* **8**, 295 (1984).

Rivas, J. R., and Rudd, D. F., Synthesis of failure-safe operations. *AIChE J.* **20**, 311 (1974).

Sacerdoti, E. D., The non-linear nature of plans. *Adv. Pap. IJCAI, 4th*, 206, (1975).

Steele, G. L., "The Definition and Implementation of a Computer Programming Language Based On Constraints," AI Lab. Tech. Rep. AI-TR-595. Massachusetts Institute of Technology, Cambridge, MA, 1980.

Tomita, S., Hwang, K., and O'Shima, E., On the development of an AI-based system for synthesizing plant operating procedures. *Inst. Chem. Eng. Symp. Ser.*, 114 (1989a).

Tomita, S., Hwang, K., O'Shima, E., and McGreavy, C., Automatic synthesizer of operating procedures for chemical plants by use of fragmentary knowledge. *J. Chem. Eng. Jpn.* **22**, 364 (1989b).

Tomita, S., Nagata, M., and O'Shima, E., Preprints IFAC Workshop, Kyoto, 66 (1986).