Korean J. Chem. Eng., **27**(6), 1654-1661 (2010)
DOI: 10.1007/s11814-010-0272-x

Automatic verification of operating schedules for batch processes using symbolic model checking: Latch model vs. real-time

Jinkyung Kim* and Il Moon,†**

*School of Chemical & Biomolecular Engineering, Georgia Institute of Technology,
311 Ferst Drive, NW, Atlanta, Georgia 30332, U.S.A.
**Department of Chemical Engineering, Yonsei University, 134 Shinchon-dong Seodaemun-gu, Seoul 120-749, Korea
(*Received 3 January 2010 • accepted 17 February 2010*)

Abstract−This study proposes two models for reading Gantt charts and finding embedded errors in the operating schedules of batch processes. Two automatic techniques for finding errors, a real-time model and a latch model, are developed using the symbolic model verifier (SMV) and are compared to verify that the schedules are error free and to represent the scheduling information and policies. These models are designed to automatically detect embedded errors relating to unavailability, superimpositions, and violation of intermediate storage policies in batch processes with various intermediate storage policies.

Key words: Operating Schedule, Real-time Model, Latch Model, SMV, Batch Process

INTRODUCTION

Batch processes are useful for producing value-added multi-products and can rapidly satisfy consumer demands and preferences. Research has been carried out into scheduling strategies and control techniques that reduce costs and increase profits. Increasing the efficiency of a batch process operation makes the scheduling of that process more complex, and the verification of the schedule plays an important role in process safety. Schedules include various policies that take into account the nature of the production recipes and the process units. As schedules become more complex, errors are likely to be embedded. However, finding scheduling errors by simply analyzing Gantt charts is difficult. The error-finding procedure must be automated in order to produce error-free operating schedules [1,2].

The automation of the safety verification process provides huge benefits compared with manual testing methods such as checklists, hazard and operability studies (HAZOP), and fault tree analysis (FTA). It becomes possible to avoid human error during the computation, to test more complex systems and many scenarios, to reduce the verification time and cost, and to address safety and operability problems more efficiently.

Therefore, this study adopts the symbolic model verifier (SMV) approach to automatically find errors in the operating schedules. SMV is a computer program that decides if given logic is true or false using CTL (computation tree logic) and BDD (binary decision diagrams) [3]. SMV is composed of a system model, assertions, and a model checker. The system model describes the control software, process equipment, and operating procedures. The assertions are questions about the behavior of the system in terms of safety and operability. The model checker determines whether the assertions are satisfied by the system and supplies counterexamples if an error is detected [4].

This study focuses on using SMV to develop algorithms that represent practical operating schedules and can find scheduling errors. The results provide information on the existence of scheduling errors and how to identify them. This method finds all errors because it checks all paths. The information reduces disturbances and the probability of errors embedded in complex industrial operating schedules. The approach is effective at creating error-free operating schedules for batch processes.

THEORY

1. Temporal Logic Model Checking

Simulators are often used to investigate the behavior of sequential chemical processes based on discrete models. However, examining the output of the simulation is usually time consuming, making exhaustive simulations rarely feasible. Although simulations are helpful, in practice they cannot be used to ensure the proper behavior of a system.

Temporal logic model checking is an alternative approach that has recently achieved significant results. Efficient algorithms can verify the properties of extremely large systems [5]. In these techniques, specifications are written as formulas in a temporal propositional logic and systems are represented by state-transition graphs. The verification is accomplished by an efficient breadth-first search procedure that views the transition system as a model for the logic, and determines whether the specifications are satisfied by the model.

There are several advantages to this approach; in particular, it is completely automatic. An overview of the model verification is given in Fig. 1. The model checker accepts a model description and specifications written as temporal logic formulas. It then determines whether the formulas are true for that model. Another advantage is that if the formulas are false, the model checker will provide a counterexample, i.e., an execution trace that shows why the formula is not true. Using temporal model checking algorithms, SMV verifies the

†To whom correspondence should be addressed.
E-mail: ilmoon@yensei.ac.kr

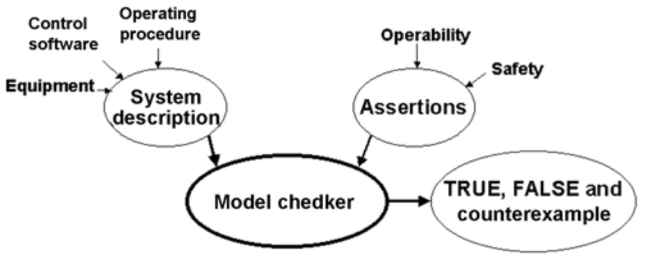


Fig. 1. Architecture of symbolic model verification.

model.

2. Symbolic Model Verifier (SMV)

Clarke [6] developed a model-based verification method. Symbolic model checking is an algorithmic means of verifying that a finite-state, sequential model satisfies temporal logic specifications [6]. The algorithms use symbolic representations to denote sets of states and legal transitions between states. The use of symbolic representations allows the routine verification of properties in models with as many as 10^{120} states [5]. Boolean formulas can be represented by binary decision trees. The nodes in the decision tree correspond to the variables of the formula. The descendants of each node are labeled true or false. The value of the formula for a given assignment of values to the variables can be found by traversing the tree from root to leaf. At each node, the descendant labeled with the value of that variable is chosen. Internal SMV algorithms eliminate redundant information in the structure and order the variables to compact the size of the BDD [3]. Counterexamples may be generated when certain classes of specifications are discovered to be false for the model, and the counterexample function of symbolic model checking is useful when analyzing and correcting faults.

Moon [7] applied the SMV technique to verify the control logic of a chemical plant. The modeling objects are discrete events such as valves, pumps, and the level of the tank. Later, Moon [8,9] researched SMV verification methods for a chemical process based on a PLC (programmable logic controller) [8,9]. Probst et al. (1996) proposed a method that combines unit modules to verify the entire process. It is applied to a solid transfer system, leak detection, and a general furnace system [10,11].

In an approach that is different from those of the above studies we develop verification algorithms for the scheduling of a batch process by modeling the time.

3. Scheduling of Batch Processes

Batch processes are used in the manufacture of specialty chemicals, pharmaceutical products, foods, and certain polymers [3]. Since the production volumes are usually low, batch plants are often multiproduct facilities in which the various products share the same equipment. This requires that the production in these plants be scheduled. Many algorithms have been introduced to optimize the scheduling of batch processes. Elaborate techniques are needed because batch processes have become increasingly complex and are now widely used for fine chemicals.

The classifications of schedules for batch processes vary. Plants can be classified as flowshop plants and jobshop plants. Flowshop plants follow the same processing sequence; jobshop plants have different sequences. Transfer policies can be classified as ZW (zero-wait), UIS (unlimited intermediate storage), NIS (no intermediate storage), FIS (finite intermediate storage), and MIS (mixed intermediate storage) [12].

In general, the Gantt chart is used to represent the schedule of a batch process. It is convenient to prepare but cannot express complex schedule characteristics. Furthermore, verifying that the schedule meets the needs of units and products is complex. Hence, automatic verification algorithms are required for efficiency, feasibility, and safety [13].

DEVELOPMENT OF SMV MODELS FOR GANTT CHART

SMV expresses the state with the state-transition. Every state is characterized by the state of its variables. The state-transition gives a convenient solution when every state switches with a specific relation to their variables. The state does not correspond to the time interval, because a transition may require more than one time interval. For example, a variable transition system can be expressed by the states $S=\{s_0, s_1, s_2, s_3, s_4 \cdots\}$ together with propositions for each state $s_i=\{t^i, p_1^i, p_2^i, p_3^i\}$. The subscript of s indicates the state number for the state transition. An algorithm can be developed to implement the transitions shown in Eq. (1).

$$\begin{array}{l} s_0=\{0,1,0,0\} \\ s_1=\{1,0,1,0\} \\ s_2=\{2,0,0,1\} \\ s_3=\{3,1,0,0\} \\ s_4=\{4,0,1,0\} \\ \vdots \end{array} \quad \text{or} \quad s_i=\begin{cases} (i,1,0,0) \text{ if } \mathrm{mod}(i/3)=0 \text{ for } \forall i\geq 0 \\ (i,0,1,0) \text{ if } \mathrm{mod}(i/3)=1 \text{ for } \forall i\geq 0 \\ (i,0,0,1) \text{ if } \mathrm{mod}(i/3)=2 \text{ for } \forall i\geq 0 \end{cases} \qquad (1)$$

An attempt at an algorithm for the transitions of Eq. (1) is given in Fig. 2.

In SMV, the above algorithm does not give the expected result. The actual result is illustrated in Fig. 3.

This is caused by the difference between state transition and time transition. In Fig. 3, the variable t^i is not matched with state subscript i.

```
For $n = 0$ to Upper Bound

    If $mod(i/3) = 0$ then
        $p_1^i = 1$ and $p_2^i = 0$ and $p_3^i = 0$
    Else If $mod(i/3) = 1$ then
        $p_1^i = 0$ and $p_2^i = 1$ and $p_3^i = 0$
    Else If $mod(i/3) = 2$ then
        $p_1^i = 0$ and $p_2^i = 0$ and $p_3^i = 1$
    End If

    $i = i + 1$
    Print $s_i(i, p_1^i, p_2^i, p_3^i)$
Next $n$
```

Fig. 2. Algorithm for variable transition.

$$
\begin{aligned}
s_0 &= \{0,0,0,0\}\\
s_1 &= \{0,1,0,0\}\\
s_2 &= \{1,0,1,0\}\\
s_3 &= \{2,0,0,1\}\\
s_4 &= \{3,1,0,0\}\\
&\vdots
\end{aligned}
$$

Fig. 3. Results of variable transition in SMV.

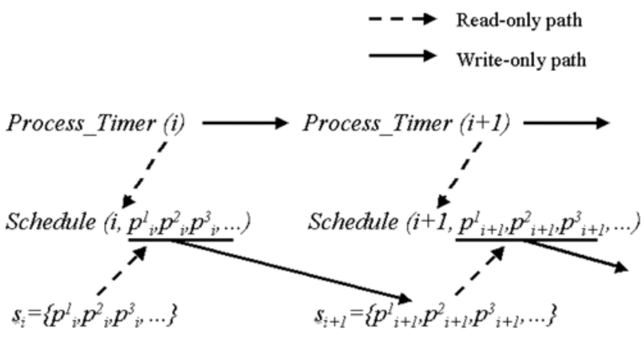

Read-only path
Write-only path
Process_Timer (i)
Process_Timer (i+1)
Schedule (i, p^1_i, p^2_i, p^3_i, ...)
Schedule (i+1, $p^1_{i+1}, p^2_{i+1}, p^3_{i+1}$, ...)
$s_i=\{p^1_i, p^2_i, p^3_i, ...\}$
$s_{i+1}=\{p^1_{i+1}, p^2_{i+1}, p^3_{i+1}, ...\}$

Fig. 4. Real-time algorithm.

Synchronizing the state transition and the time transition is the key to expressing the real-time characteristics of the schedule for a batch process. We developed two algorithms to solve this synchronization problem. The real-time algorithm has a timer in each unit module and the latch algorithm has a switch in the main module to match the state with the time.

1. Real-time Algorithm

The first algorithm is developed for the real-time model. For this model, the total makespan is divided into time units of the same length. In each time unit, the state of variables is maintained and not renewed. To renew the variables, another module is introduced. Because of limitations on the input/output of variables in SMV, read-only and write-only modules are separated. Two modules, the process timer and the schedule, realize this. Fig. 4 represents path transfers and integrations of variables for two modules. The process module is a timer that operates throughout the process and the schedule module is a unit timer that counts the time related to each product and unit. The schedule module refers to the process timer to guarantee the exact transition time.

Modules for each unit should be related to the process module and the schedule module. The algorithm based on the real-time model is more complex than the latch algorithm described in the next section.

2. The Latch Algorithm

The second algorithm, the latch algorithm, is based on the latch variable, which is the variable that maintains the current state of the variable until it is initialized by the variable to be defined as the next state. Fig. 5 illustrates this using RLL (relay ladder logic).

In Fig. 5, each rung represents a time interval. The serial transition of the TRUE value of variables indicates the transition of time. However, the condition that sets the next state to TRUE can occur at different times. The solution for this problem is the introduction of precondition variables that distinguish the same condition that

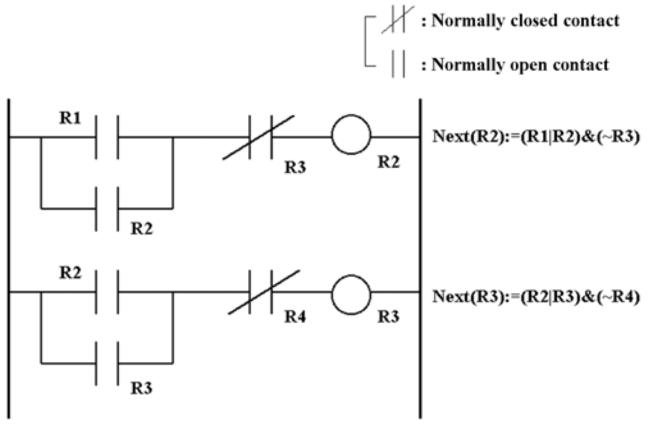


Fig. 5. Latch algorithm.

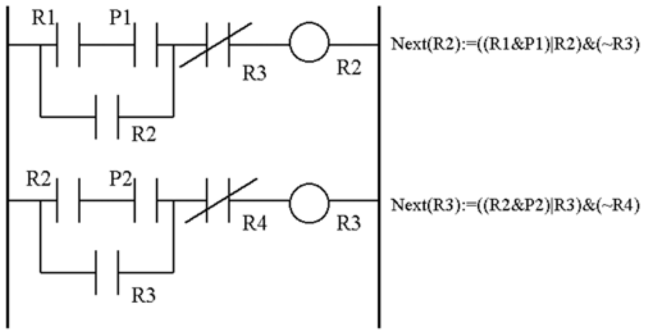


Fig. 6. Latch algorithm with precondition variables.

should be achieved at different times as a different job. Fig. 6 uses RLL to illustrate the introduction of precondition variables.

In the latch model, the time intervals are not the same length. The input and output time of each unit is the standard for optimized number of time unit. Furthermore, all the algorithms for each unit and product can be developed in the same module.

3. Preparation of Assertions

The real-time model and the latch models are developed to express the Gantt chart using SMV. To verify a schedule, assertions

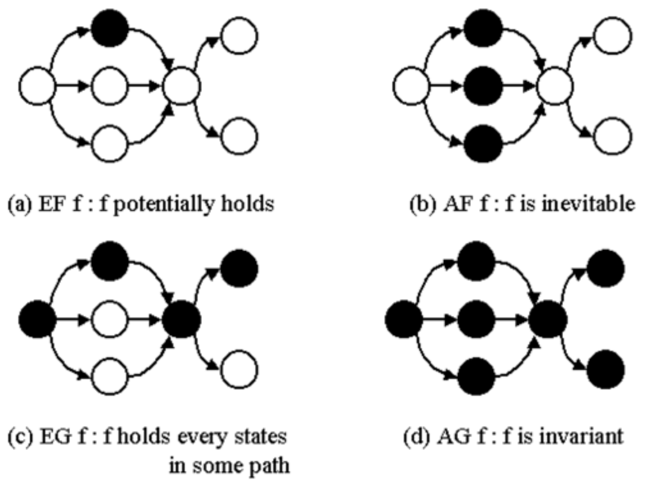

(a) EF f : f potentially holds
(b) AF f : f is inevitable
(c) EG f : f holds every states in some path
(d) AG f : f is invariant

Fig. 7. Assertions in CTL (●: f, ○: ~f).

must be developed. SMV provides CTL (computational tree logic) for this purpose. The simplest CTL formula consists of just an atomic proposition. If p is an atomic proposition, then p is TRUE for a state s if and only if p labels s: that is, p is an element of P(s). Formulas can be built upby using the standard operators of negation (written ~), and (written &), and or (written |). CTL is distinguished from elementary propositional logic by the model operators. Fig. 7 illustrates the representation of assertions in CTL.

With CTL, assertions are developed to verify that a given schedule meets the needs in characteristics of unit, product, and other recipes.

CASE STUDY

1. Multi-product Batch Process

One of the easiest ways to compare the effectiveness of the two models is to apply the same batch-process schedule. The given Gantt chart describes the schedule for three products (A, B, C) and three stages (Stage 1, Stage 2, Stage 3). The processing time for each product is given in Table 1. The target process makes three products, A, B, and C, in the sequence A→B→C→A→B→C. The process uses FIS intermediate storage policy. FIS policy assumes that the batch can be stored in a finite intermediate-storage tank in order to shorten the processing time. The intermediate-storage tank of the target process is located at Stage 1 and has the same capacity as the unit of Stage 1.

If there is an error in the given Gantt chart, an assertion is needed to detect the error. For this case study, an error is intentionally introduced. The final Gantt chart with the error is shown in Fig. 8.

1-1. Real-time Model for Given Gantt Chart

To describe the transfer procedure, a module for the storage tank is required. The state of the storage tank module has a variable that indicates whether it is full or empty. The relation between the state of Stage 1 and that of Stage 2 is the basis of the value of the variable. Figs. 9 and 10 show the algorithm for these sequences.

For the entire program, the process timer module, the schedule module, the storage tank module, and the modules for each stage are integrated in a main module.

The assertion to verify whether the schedule is feasible is:

Assertion 1.

While Stage 1 is processing, it is not possible for the storage tank to be full when Stage 1 still contains the intermediate products.

CTL 1.

AG(!(!(stage01_schedule.condition=off) & !(storage.condition= empty) & !(reactor.condition=0)))

1-2. Latch Model for Given Gantt Chart

In the latch model, the makespan is divided into sections and all the unit states including the storage tank are expressed with latch variables. All the unit states are changed by the value of the latch variable without a CASE statement. This is easier to realize than the real-time model because the algorithm reads only the Gantt chart. However, the number of variables including the latch variables exceeds that of the real-time model; this increases the number of BDD

Table 1. Processing time for each product

Stage \ Product	A	B	C
Stage 1	3	1	2
Stage 2	4	5	2
Stage 3	2	4	5

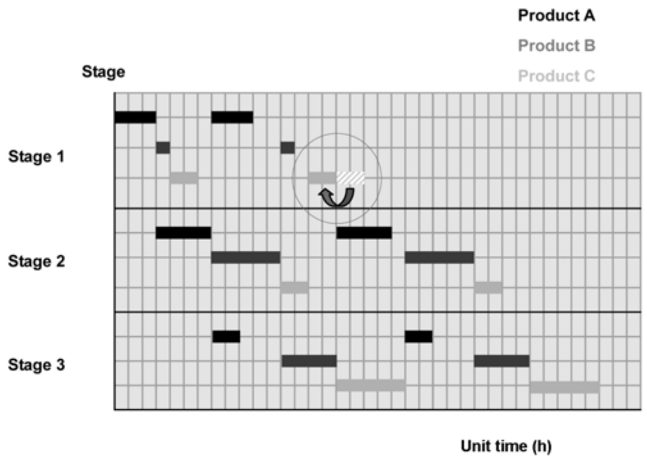


Fig. 8. Gantt chart with error.

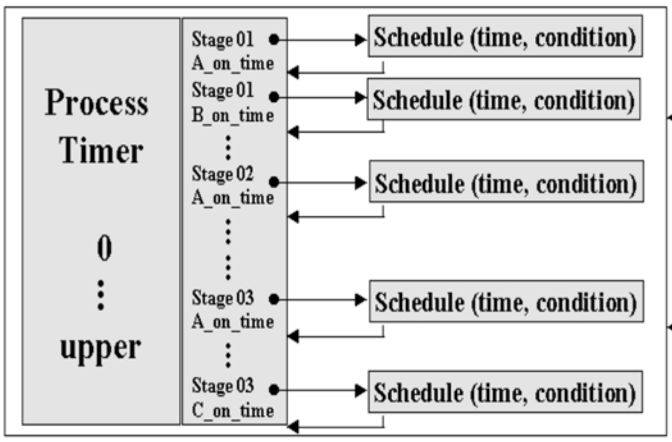


Fig. 9. Structure of schedule in SMV.

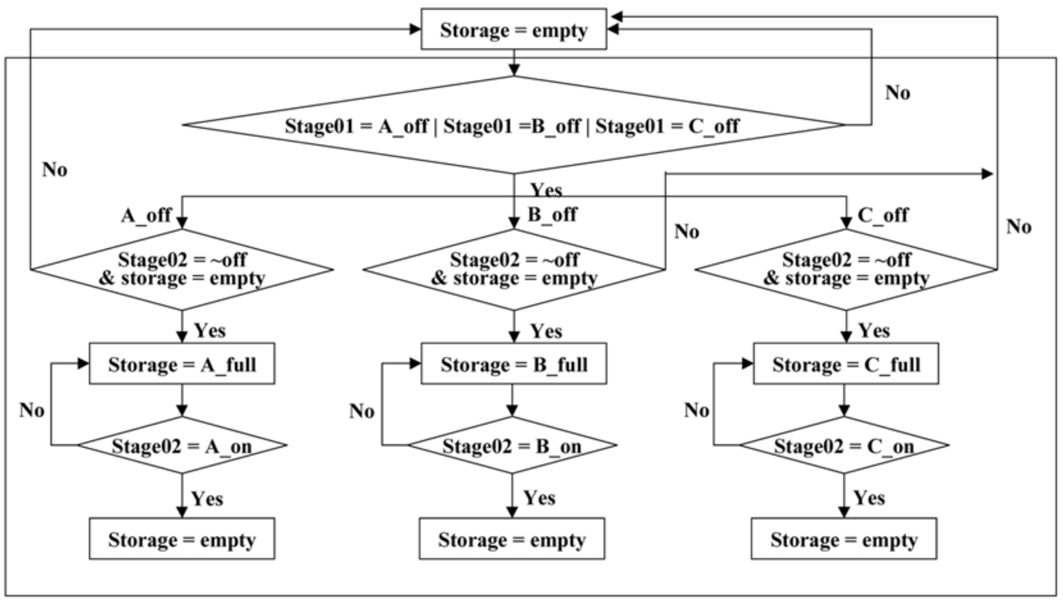


Fig. 10. Algorithm for FIS storage module.

Table 2. Results of comparison

	Real-time model	Latch model
BDD nodes allocated	88002	11815120
Bytes allocated	2359296	189857792
BDD nodes representing transition relation	36078	6008560

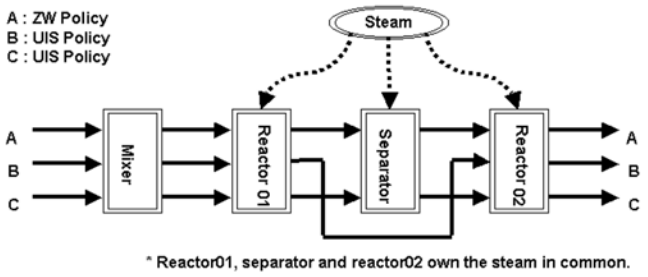


Fig. 11. Multipurpose plant with steam generator.

nodes and the detection time.

The assertion is similar to that for the real-time model.

1-3. Result Comparison

The latch model can be prepared more easily than the real-time model. Because of the process timer module and the schedule module, the real-time model has three distributed modules. In the latch model, all variables can be expressed as latch functions in one main module.

Both models find errors successfully. However, the real-time model uses fewer nodes and less memory than does the latch model, because the larger the verification time, the more variables are needed in the latch model. If the makespan of the Gantt chart is larger, the difference increases. Table 2 indicates that the real-time model is more efficient than the latch model.

2. Multi-purpose Batch Process

The real-time model can more effectively realize the real-time characteristics in a batch-process schedule. However, its application depends on the assertions for the given system. With multiple assertions, the analyzer can detect all the possible errors simultaneously. In addition, several specific algorithms should be introduced to detect all possible paths.

The given Gantt chart indicates the wide application of this method. It has three processing stages for three types of products, as shown in Fig. 11. Each product has its own production path and the heater supplies steam to stages 2 and 3 simultaneously. The different path for each product and the limit on the capacity of the steam increase the complexity of estimating the safety and feasibility.

2-1. Detection Algorithms for All Possible Paths

In this case, all possible models are verified by introducing an algorithm with a CASE statement. The schedule that will be verified is already designed in general cases. To detect all the paths that can be followed by the input raw materials, the concepts of an undetermined variable and exclusive OR are introduced. The undetermined variable is a feature of SMV; it gives the probability of multiple paths. The order of the input raw materials changes the path that should be followed to finish the processing because of the varied processing times and procedures for different productions.

Another approach for expressing a multi-path system is the method of exclusive OR. The exclusive OR is defined as a logical notation that is TRUE if exactly one of the propositions in the current state is TRUE. The exclusive OR proposition can be composed using the *and* and *or* propositions.

2-2. Development of Unit Module

After the initial value of processing material is determined by the undetermined variable or the exclusive OR proposition, the rest of the system can be developed by using the real-time model, which is more efficient than the latch model. However, the schedule module does not know the production sequence for the various initial raw materials; it can work from the previous state using a CASE

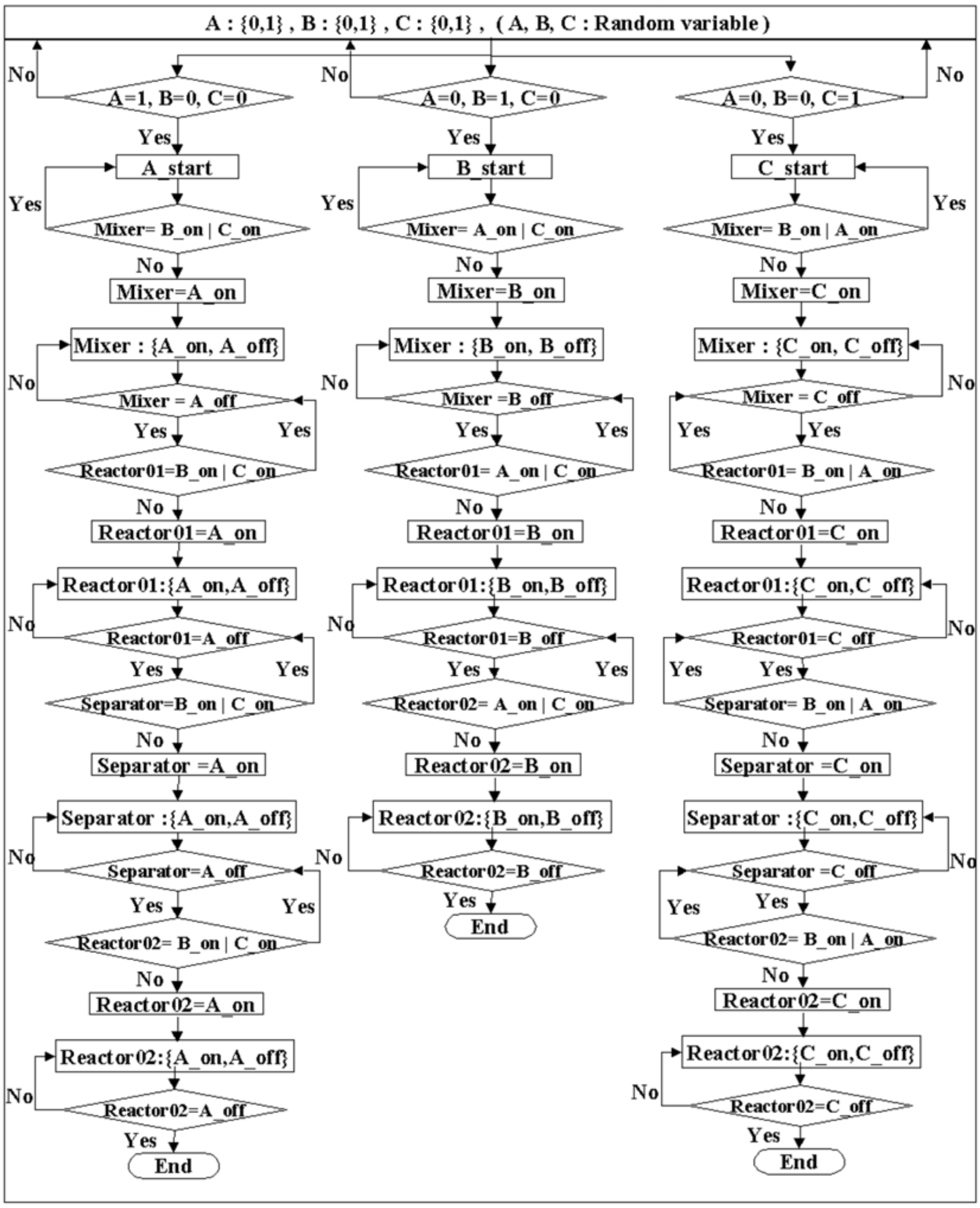


Fig. 12. Algorithm for multipurpose batch plant.

statement. Fig. 12 illustrates the CASE statement and the algorithm for all schedules.

Each unit module is developed in the real-time model. For feasibility, only one product can exist in a unit at the same time. Thus, this model cannot violate feasibility.

2-3. Assertions

An assertion must be prepared to give the limitation imposed by the common use of the steam.

Assertion 1

It cannot simultaneously occur that reactor01 is processing B, reactor02 is processing A, and the separator is processing C.

CTL 1

AG(!(Reactor02=A_on & Reactor01=B_on & Separator=C_on)))

In addition, product A should follow the ZW policy. Three assertions are therefore added.

Assertion 2

It cannot occur that reactor01 is processing directly after the mixer finishes processing A.

CTL 2

AG(!(Mixer=A_off & (Reactor01=B_on | Reactor01=C_on))))

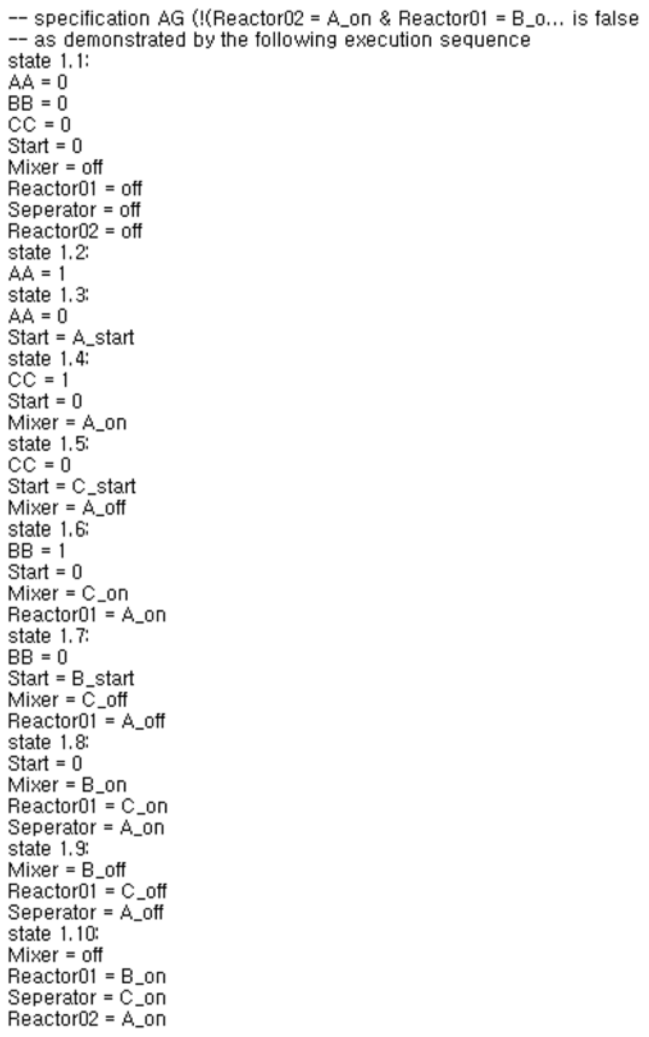
Assertion 3

It cannot occur that the separator is processing directly after reactor01 finishes processing A.

CTL 3

AG(!(Reactor01=A_off & Separator=C_on)))

Assertion 4



```
-- specification AG (!(Reactor02 = A_on & Reactor01 = B_o... is false
-- as demonstrated by the following execution sequence
state 1.1:
AA = 0
BB = 0
CC = 0
Start = 0
Mixer = off
Reactor01 = off
Seperator = off
Reactor02 = off
state 1.2:
AA = 1
state 1.3:
AA = 0
Start = A_start
state 1.4:
CC = 1
Start = 0
Mixer = A_on
state 1.5:
CC = 0
Start = C_start
Mixer = A_off
state 1.6:
BB = 1
Start = 0
Mixer = C_on
Reactor01 = A_on
state 1.7:
BB = 0
Start = B_start
Mixer = C_off
Reactor01 = A_off
state 1.8:
Start = 0
Mixer = B_on
Reactor01 = C_on
Seperator = A_on
state 1.9:
Mixer = B_off
Reactor01 = C_off
Seperator = A_off
state 1.10:
Mixer = off
Reactor01 = B_on
Seperator = C_on
Reactor02 = A_on
```

Fig. 13. Result of verification for common use of steam.

It cannot occur that reactor02 is processing directly after the separator finishes processing A.

CTL 4

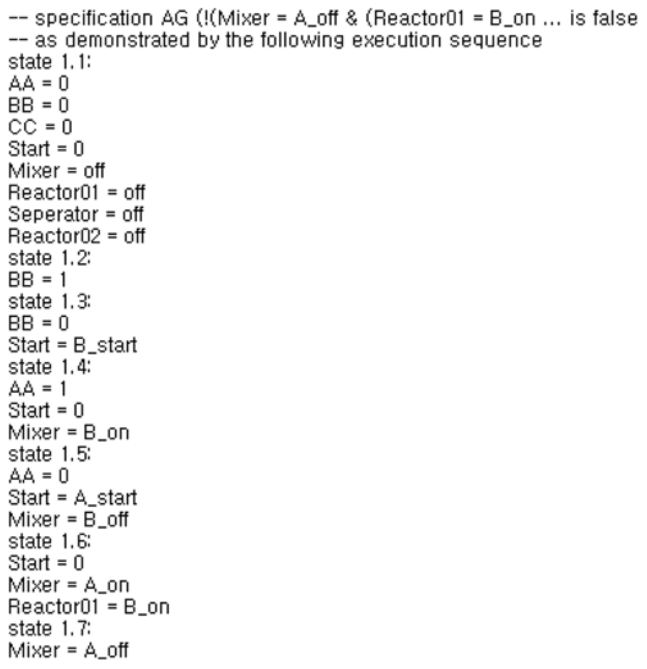
AG((!(Separator=A_off & (Reactor02=B_on | Reactor02=C_on))))

Assertion 1 expresses the unit limitation and Assertions 2 to 4 express the product limitations.

2-4. Results of Verification

Figs. 13 and 14 show the results of verification for each assertion, and they represent false assertions. At state 1.10 in Fig. 13, reactor01 processes product B, the separator processes C, and reactor02 processes A simultaneously. This result indicates that there is a shortage of steam when the schedule follows the sequence A→C→B. The designer of this process should therefore remove the sequence A→C→B from consideration.

Fig. 14 represents Assertion 2, about ZW policy. In Fig. 14, reactor01 continues processing product B at state 1.7 and the mixer finishes processing product A. Product A should next be processed by reactor01, but reactor01 is busy with B. This violates the assertion that product A should follow a ZW policy. This error occurs when the sequence is B→A. The designer should therefore avoid the sequence that produces A after producing B.



```
-- specification AG (!(Mixer = A_off & (Reactor01 = B_on ... is false
-- as demonstrated by the following execution sequence
state 1.1:
AA = 0
BB = 0
CC = 0
Start = 0
Mixer = off
Reactor01 = off
Seperator = off
Reactor02 = off
state 1.2:
BB = 1
state 1.3:
BB = 0
Start = B_start
state 1.4:
AA = 1
Start = 0
Mixer = B_on
state 1.5:
AA = 0
Start = A_start
Mixer = B_off
state 1.6:
Start = 0
Mixer = A_on
Reactor01 = B_on
state 1.7:
Mixer = A_off
```

Fig. 14. Result of verification for ZW policy.

CONCLUSIONS

For complex batch-processing scheduling, SMV is an appropriate technique for verifying safety and feasibility. It can be applied after the schedule has been developed. After verification, a counterexample aids the redesign of the scheduling. This procedure is introduced in Fig. 15. In Fig. 15, the procedure for choosing an object for the schedule provides the assertion and it is converted to CTL. The Gantt chart that can be readily handled but not easily applied to complex batch processes is converted to an SMV algorithm. The real-time model is proposed for this conversion because it has higher computational efficiency than the latch model. Complex characteristics such as different unit capacities and different transfer policies can be verified with the real-time model. An advantage of this technique is that SMV is fully optimized. In addition, the computational requirements are low, reducing the computing resources needed. If a FALSE value is produced for any verification, the counterexample created can provide a hint for the redesign of the schedule. The

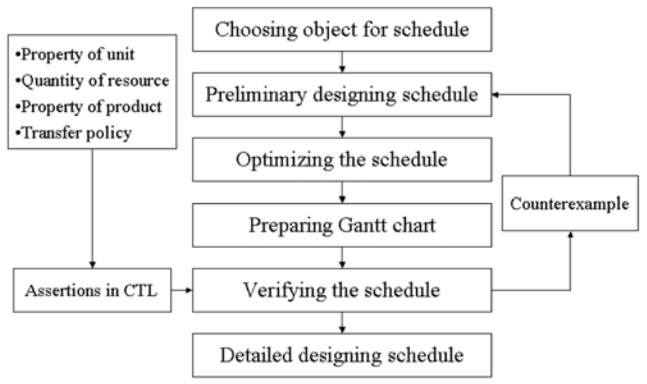


Fig. 15. Algorithm for verification of schedule of batch plant.

application of the algorithm with this verification loop can improve the safety of batch-processing plants.

REFERENCES

1. J. Kim, H. Lee and I. Moon, *The 8th APCChE Congress*, Seoul, Korea (1999).
2. H. Lee, J. Kim and I. Moon, *AIChE Annual Meeting*, Dallas, Texas (1999).
3. A. Reeve, *Process Engineering*, **73**, 33 (1992).
4. J. Kim and I. Moon, *Comp. Chem. Eng.*, **24**, 385 (2000).
5. J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill and J. Hwang, In *Proceedings of the Fifth Annual IEEE Symposium on Logic in Computer Science* (1990).
6. E. M. Clarke, E. A. Emerson and A. P. Sistla, *ACM Trans. on Programming Lang. Sys.*, **8**, 244 (1986).
7. I. Moon, G. J. Powers, J. R. Burch and E. M. Clarke, *AIChE J.*, **38**, 66 (1992).
8. I. Moon, D. Ko, S. T. Probst and G. J. Powers, *J. Chem. Japan*, **30**, 13 (1997).
9. I. Moon, *IEEE Control Systems*, **14**, 53 (1994).
10. S. T. Probst, G. J. Powers, D. E. Long and I. Moon, *Comp. Chem. Eng.*, **21**, 417 (1997).
11. S. T. Probst and G. J. Powers, *AIChE Annual Meeting*, San Francisco, CA (1994).
12. L. T. Biegler, I. E. Grossmann and A. W. Westerberg, *Systematic Methods of Chemical Process Design*, Prentice Hall, Englewood-cliffs, NJ, 187 (1997).
13. D. Ko and I. Moon, *Comp. Chem. Eng.*, **21**, 1067 (1997).