Abstract

Of all hard- and software developed for industrial control purposes, the majority is devoted to sequential, or binary valued, control and only a minor part to classical linear control. The sequential parts of the controller are typically invoked during startup or shut-down phases to bring the system either into its normal operating region or into some safe standby region. Despite its importance, fairly little theoretical research has been devoted to this area, and sequential control programs are still created manually without much support for a systematic approach. We propose a method to create sequential control programs automatically and on-line upon request, for example when a plant fault has occurred. The main idea is to spend some effort off-line on modeling the process, and from this model generate the control strategy, i.e. the plan. Here we present a planning tool implemented in a real-time expert system called G2. The planning system contains algorithms for creating plans in form of minimal GRAFCET charts that show maximal parallelism. The algorithms can handle a restricted class of problems and for this class the complexity only increases polynomially with the number of state variables.

Keywords: Planning, GRAFCET, Sequential Control, Real-time Expert Tools.

1 Introduction

Amongst all control problems found in industry, the ones of sequential nature are especially common. Almost all process plants contain sequential parts; startup or shut-down phases are typical examples of this. Despite its importance fairly little theoretical research has been devoted to this problem, and sequential control programs are usually created manually without much support for a systematic approach. The sequential control problem can be divided into two parts: planning and implementation. Planning is the problem of finding a plan, i.e., a control sequence, which transforms a given initial state into a desired final (or goal) state. Such a plan may be implemented, e.g., by using the graphical language GRAFCET [6].

Normally when creating sequential control programs one must take into account not only the ordinary operation of the plant, but also what should be done when a plant fault occurs. For each possible fault situation we must construct a control strategy that eliminates the effects of, or at least reduces the damage caused by, the fault. This control strategy is traditionally constructed off-line.

When a fault occurs during operation it must be decided which plan to follow. To verify that such a program is correct is very costly, and in practice each case must be simulated, which due to time constraints often is impossible.

To resolve this problem, we propose an alternative approach, namely one allowing automatic creation of sequential control schemes on-line upon request. From the current fault situation and a prior defined “safe state” a sequential control program is automatically generated. Note that the “safe state” may be the same for many fault situations and that each plan is generated only if and when needed. We must demand two things from such a program. It must be correct and it must be generated fast enough to allow for on-line replanning.

Another situation where automatic plan generation might be wished-for is when adding new devices to the plant. It is common that these devices are added when the control program is already developed, and hence the program must be changed to take this into account. It may be difficult to realize how the changes affect different parts of the program and the plant. If the plant is described by a model, from which the control scheme is automatically generated, only the model must be changed when the process is modified.

On-line planning can also be used for operator support. A supervisor checks if the wanted operations are allowed. If, e.g., the operator wants to open a specific valve the supervisor should inform that such an action is not allowed unless he or she first opens another valve. Another possibility is to automatically generate and execute a sequence of actions resulting in the opening of the valve.

There is thus a need for automatic synthesis of sequential control schemes. Instead of working directly on the control program itself we suggest a different point of view. As mentioned above the idea is to spend some effort on modeling the process, and from this model generate the control scheme. From now on we will by planning mean automated creation and implementation of a plan, or a control program. Using, for instance, a finite state model the planning problem is always solvable in principle, but when the size of the problem increases it soon gives rise to complexity problems. Hence we propose to study subclasses where we retain feasibility. We formulate the model using Simplified Action Structures originally introduced by Sandewall and Rönquist [12].

We have previously presented algorithms for generating minimal plans (no plan containing fewer actions exists) for
The generated plans also show maximal parallelism, i.e., as many actions as possible are carried out simultaneously. One major advantage with this approach is that the complexity of these algorithms only increases polynomially with the number of state variables, whereas the complexity for algorithms based on searching the state graph increases exponentially.

In this paper we present a planning tool implemented in a real-time expert system environment called G2 [4]. The implemented system contains algorithms for creating plans in form of GRAF CET charts using a fairly general G2 package called GrafcetTool [11]. Based on this function chart it is straightforward to directly generate PLC (Programmable Logic Controller) code.

The paper is organized as follows. Section 2 is directed to those who are not familiar with G2 and GRAF CET. An informal description of the formalism used for planning and the planning algorithms themselves are given in section 3. In section 4 we show how GrafcetTool is connected to the planner. A simple example is then given in Section 5 and section 6 contains some concluding remarks.

## 2 GRAF CET in G2

Inspired by the work presented in [1] we now turn to a discussion about how to handle sequences in a structured and clear way. By using the G2 real-time expert system [4] as a programming platform we have designed and implemented GrafcetTool [11], which is a fully graphical tool based on the GRAF CET formalism [6]. Before discussing GRAF CET and GrafcetTool we start with a short introduction to G2.

### The G2 real-time expert system

G2 is an advanced graphical tool for designing and running real-time knowledge-base systems for complex applications that require continuous and “intelligent” monitoring, diagnosis and control [4]. It is aimed at a wide range of real-time applications including process control, robotics, network management and financial analysis. In G2 the objects are ordered in an object-oriented class hierarchy, where each defined class can inherit various attributes from its superior class. Using the built-in inference engine, it is possible to reason about the current system state and perform actions through rules that are operating on the objects. G2 heavily rely on graphics and every object is represented by and manipulated through its icon. The objects can also be graphically connected to each other, thereby making G2 suitable for problems that have a graphical representation.

As indicated above G2 is intended for real-time applications and supports this in a number of ways, including rule scan intervals and priorities, asynchronous event handling, and more. Another quite important concept is the possibility to activate and deactivate subworkspaces. A workspace is just a kind of window upon which various knowledge components are placed. Workspaces can also be assigned to objects, or in other words, be owned by objects. Such a workspace is called a subworkspace and in a way it represents the internal structure of the object. If a subworkspace is activated during run-time, all its knowledge parts (rules, procedures etc.) are executables, but when it is deactivated, none of its knowledge pieces are available. Thus, a partial order on a set of actions, or a plan, can be realized by activating and deactivating object subworkspaces in the order prescribed by the plan. Guaranteeing this order is of course crucial. For a number of reasons (standardization, graphic representation, decomposition possibilities, syntax suitable for compilers) we have chosen to use GRAF CET for this purpose.

### GRAF CET and GrafcetTool

A GRAF CET function chart [6, 11] is mainly composed of steps and transitions, interconnected via directed links as in Fig. 1. The steps can either be active, which is illustrated by a highlighted dot called a token, or inactive. Once a step is activated its associated subworkspace also becomes active, and all commands or actions (e.g., open a valve, start a motor) on this workspace are enabled. Conversely, deactivating a step implies that the step’s subworkspace also is deactivated. The change from an active step to an inactive one is determined by the transition located between the steps in question. More precisely, a state change occurs when the so called transition condition becomes true (e.g., when a sensor is triggered).

Actions and transition conditions are specified using ordinary G2 rules. A stored action (the action is performed until another rule belonging to a subsequent step executes a reset action) is for instance specified by an initial rule with the following structure:

**initially conclude that** (<ACTION SPECIFICATION>)

A transition condition rule can be implemented as a scanned rule:

**when** (<TRANSITION CONDITION IS TRUE>)

**then start Fire-Transition(this workspace)**

![Fig. 1: Concepts and ideas found in GrafcetTool.](image-url)
fact the only ones needed.

Since most industrial processes are not strictly sequential, the GRAFCET standard also includes two control structures—simultaneous sequences (or parallel branches) and sequence selection (or alternative paths). As a plan is described by a partial order on a set of actions the resulting control chart lacks alternative branches. Parallel branches, see Fig. 1, will on the other hand be used whenever possible. Once a parallel branch is reached, all underlying sequences are activated simultaneously and thereafter executed independently of each other. To be able to continue from the convergence point of a branch, all incoming branches must be ready, i.e., all steps connected to the “closing” point must be active.

3 Planning for a class of sequential control problems

In this section we informally describe a formalism for a set of planning problems, and introduce some restrictions to form the specific classes we concentrate on. We will also intuitively outline a planning algorithm for one restricted class of planning problems. The formal definitions and proofs may be found in [2, 3, 7, 10].

A planning formalism

Let us assume that the plant is described by a state, and that the control action, or event, is chosen from a set of actions which transforms the state of the plant into a new state. To be able to state the planning problem more formally we need some basic definitions.

States: The state of the plant is described by a state vector $x$ of dimension $n$, i.e., $x = (x_1, \ldots, x_n)$. Each state variable belongs to a discrete, finite set $S_i$ and thus $x \in S = S_1 \times S_2 \times \ldots \times S_n$.

Action types and actions: Examples of actions could be MoveWorkpiece, where a robot moves a workpiece from a work-station to storage, OpenValveA where a valve called ValveA is opened and ReadInputChannel, where a computer reads an input channel. Each action is defined by its action type. The action type can be interpreted as a generic action description, and an action is a particular instantiation of an action type.

An action type is defined by its pre-, post- and prevail-conditions. The pre-condition specifies what must hold when the action starts, the post-condition what holds when the action ends, and the prevail-condition what must be true during the execution of the action. Notice that the state variables in the prevail-condition are not affected by the action. Consider, e.g., the action type OpenValveA where a valve called ValveA is opened. Suppose that in order to open ValveA we require that a valve called ValveB is already open to avoid overflow. Here, the pre-condition is that ValveA is closed, and the post-condition that it is open. Thus the pre- and post-conditions describe what is changed by the action. An action of type OpenValveA can only be started when ValveA is closed (the pre-condition is fulfilled) and when ValveB is open.

Planning: A plan from $x^0$ to $x^*$ is a set of actions $\Psi$ and a partial order $\rho$ on the set $\Psi$. The order $\rho$ is the execution order specifying the order in which the actions in the set $\Psi$ should be performed so that the initial state $x^0$ is transformed into the final state $x^*$. We say that a plan is minimal if there is no plan from the given initial state to the final state containing fewer actions. Any partial order can be implemented using GRAFCET and the plan (the partial order) might therefore be represented by a GRAFCET chart. The reader who is not familiar with relations and partial orders may think of a plan as a GRAFCET chart without alternative paths.

Classes of planning problems

The class of planning problems defined so far is called the SAS class, where SAS stands for Simplified Action Structures. This formalism was first introduced by Sandewall and Rönquist [12], but is here somewhat simplified. To form the class of problems we focus on in this paper let us introduce some additional restrictions:

- all state variables are binary,
- each action affects only one state variable (unary),
- no two different action types can change a particular state variable to the same value (post-unique), and
- no two different action types have different but defined prevail-conditions for the same state variable (single-valued).

This is the so-called SAS-PUBS (Post unique, Unary, Binary, Single-valued) class [3, 7]. If we generalize to non-binary state variables then the class is called the SAS-PUS class [2, 10]. An example of a problem in the SAS-PUS class is a plant where some fluid is transported in pipes. In such a plant the typical action types would be to open or to close a specific valve. However, the restriction to single-valued sets of action types means that if there is one action type whose prevail-condition is that a specific valve is open, then there can be no other action type whose prevail-condition is that this valve is closed.

SAS-PUBS planning

Our approach is to develop specialized planners tailored for different classes of planning problems. The main idea is to use the structure of the SAS-PUBS class and the SAS-PUS class respectively to find and order the actions without having to explicitly construct the state graph. Here we will only give an intuitive description of the process of finding SAS-PUBS plans.

For planning problems in the SAS-PUBS class, planning can be divided into two parts: finding the set of necessary actions and finding the execution order named precedes. The set of necessary actions ($\Delta$) consists of two sets:

- The set of primarily necessary actions is found by checking the state variables that are different in the
The next step is to find the execution order \( \text{precedes}(\delta) \) which is a relation defined on the set of necessary actions \( \Delta \). The relation \( \text{precedes} \) is constructed from the relations \( \text{enables} \) and \( \text{disable} \):

- If \( a_1 \) ‘enables’ \( a_2 \) then \( a_1 \) provides some part of the prevail-condition of \( a_2 \).
- If \( a_2 \) ‘disables’ \( a_1 \) then \( a_2 \) destroys some part of the prevail-condition of \( a_1 \).

In both these cases \( a_1 \) should be performed before \( a_2 \). Putting these two relations together and taking the transitive closure \([5]\) gives the execution order \( \delta \).

If the set \( \Delta \) exists and \( \delta \) is a partial order, then they form a minimal plan from the initial state \( x^0 \) to the final state \( x^* \) where the unordered actions can be executed in parallel. Otherwise there is no plan transforming \( x^0 \) into \( x^* \) using the available actions. In \([3, 7]\) we give an algorithm for finding \( \Delta \) and \( \delta \) according to the scheme outlined above. The algorithm is proven to be correct and its complexity to increase polynomially with the number of state variables. If the available actions is to close or to open a specific valve then each state variable corresponds to one valve, and the number of state variables is the number of valves in the plant.

It can also be shown that if we do not demand single-valued sets of actions the algorithm will still be correct, i.e., if the algorithm generates a plan this plan will solve the stated planning problem. However, if the algorithm fails a plan can still exist. The planning problem can then be split into a number of polynomial planning problems each of which can be solved using a slightly modified version of the SAS-PUBS algorithm \([8]\).

Here we have only considered planning problems in the SAS-PUBS class. In \([2, 10]\) an algorithm for dealing with the SAS-PUS class, where non-binary state variables are allowed, is presented. This algorithm is proven to be correct and complete, and again the complexity is proven to increase polynomially with the number of state variables.

### 4 Putting the pieces together

Now it is time to explain how the pieces discussed so far can be integrated. The components and the relationships amongst them are depicted in Fig. 2, and throughout this section we recommend the reader to keep this figure fresh in mind. To begin with, the general GrafcetTool package is fit into the framework of the planner by adding some attributes to the ordinary step definition (pre-, post- and prevail-conditions). With these alterations the steps may be treated as actions or action types.

Having access to the plant model along with the initial and the final states, the planner first creates the necessary actions by copying action type objects. The action instantiations as well as the creation of other GRAF CET elements are entirely handled via special interface procedures found in GrafcetTool. Second, the planner produces the relation ‘precedes’ on the set of necessary actions according to the SAS algorithms.

Next the partial order is converted to a GRAF CET function chart. To get a syntactically correct chart the steps must be separated by transitions. For that reason a transition with an associated transition condition is automatically created for each step. Observe that the transitions do not belong to the database, and that the transition condition is derived from the post-condition of the current step. Another syntactically motivated move is the adding of empty steps, which will not change the state. Empty steps are only situated just before parallel branches converge and their purpose is to guarantee a correct synchronization. After adding the syntactically necessary objects the graph is completed.

Finally, it is possible to translate the outgoing graph to ordinary PLC-code. Thus the cycle is closed and the result is an integrated system able to perform planning in reality.

### 5 Example

In this section we will use a simple example to illustrate how the system works. The example is a puzzle consisting of 16 squares according to Fig. 3. The squares can be either black or white, and the available action types change the color of each square from black to white or vice versa. For each square we introduce a state variable \( x_{ij} \), where \( i \) is the row and \( j \) the column the square is placed in. The state variable \( x_{ij} \) is interpreted as:

\[
x_{ij} = \begin{cases} 
0 & \text{if the color of the square is white} \\
1 & \text{if the color of the square is black}
\end{cases}
\]

There are two action types for changing the color of each square. For the square associated with the state variable \( x_{ij} \) the action types are called White-To-Black\( i_j \) and Black-To-White\( i_j \). From now on these will be abbreviated as \( W2B_i \) and \( B2W_i \), respectively. The pre- and post-conditions for the action type \( W2B_i \) are

- Pre-condition: \( x_{ij} = 0 \)
- Post-condition: \( x_{ij} = 1 \)
and for the action type $B2Wi_j$ the pre- and post-conditions are

- pre-condition: $x_{ij} = 1$
- post-condition: $x_{ij} = 0$

Notice that no other state variable will be changed when performing an action of this type. Recall also that an action only can be performed when both the pre-condition and the prevail-condition are satisfied. The prevail-conditions for $W2B_i,j$ are given in Fig. 4, and for $B2W_i,j$ in Fig. 5. For each action type a state of the puzzle is shown. For example, the state in the upper left corner in Fig. 4 shows the prevail-condition for the action type $W2B_{1,1}$ which is that $x_{21} = 1$. The initial and the final
states are given in Figure 6. In Fig. 7 a minimal plan of maximal parallelity is given in terms of a GRAFCET chart. Each step corresponds to an action as described in section 4 except the ones colored grey, which are empty steps.

6 Conclusions

We have presented a planning tool implemented in the G2 real-time expert system. It contains algorithms for generating plans for a restricted class of problems in polynomial time. The plans are generated as minimal GRAFCET charts showing maximal parallelism. The function charts can be translated into PLC code, which can be downloaded and executed in any modern control system. The result is an integrated system able to perform planning in reality.

Although we have exemplified with a toy-like problem it should be clear that this kind of tool can be successfully used in real applications. All real problems can of course not be fit into the SAS-PUBS or SAS-PUS framework, but a careful modeling extends the number of problems which end up in complexity difficulties. For example, if we let go of the single-valued requirement the worst-case complexity increases exponentially with the number of state variables. Even for the binary case it is easy to construct planning problems where the size of a minimal plan is exponential in the number of state variables. Naturally it is then impossible to find a plan in polynomial time. Fortunately, this is not likely to occur for a real-world planning problem. In reality it turns out that a SAS-PUB problem can be split into a number of simpler planning problems. These may be solved in polynomial time using a somewhat modified version of the SAS-PUBS algorithm [8].

7 References


