Efficient Planning for a Miniature Assembly Line

Inger Klein

Department of Electrical Engineering
Linköping University, SE-581 83 Linköping, Sweden

WWW: <http://www.control.isy.liu.se>

Email: inger@isy.liu.se

Peter Jonsson and Christer Bäckström

Department of Computer and Information Science
Linköping University, SE-581 83 Linköping, Sweden

WWW: <http://www.ida.liu.se>

Email: {petej,cba}@isy.liu.se

August 18, 2000

Report no.: LiTH-ISY-R-2288

Technical reports from the Automatic Control group in Linköping are available by anonymous ftp at the address [ftp.control.isy.liu.se](ftp://ftp.control.isy.liu.se). This report is contained in the file 2288.pdf.

Efficient Planning for a Miniature Assembly Line

Inger Klein

Department of Electric Engineering
Linköping University
S-581 83 Linköping, Sweden
email: inger@isy.liu.se
phone: +46 13 281665

Peter Jonsson and Christer Bäckström

Department of Computer and Information Science
Linköping University
S-581 83 Linköping, Sweden
email: {petej,cba}@ida.liu.se
phone: +46 13 282415, +46 13 282429
fax: +46 13 282606

August 18, 2000

Abstract

This paper presents a provably correct and efficient, polynomial time, planning tool and its application to a miniature assembly line for toy cars. Although somewhat limited, this process has many similarities with real industrial processes. One of our previous polynomial-time planning algorithms has been extended and adapted to work for a larger class of planning problems, including this particular process. The plans produced by the planner are then translated into GRAFCET charts, which are compiled into code for a programmable logic controller. Although capable of producing ordinary assembly plans, the system is mainly intended for producing plans in error recovery situations.

Keywords: AI planning, Sequential control, Assembly planning, Error recovery, Polynomial-time planning

The motivation for interference-safeness is that if an operator changing the value of a variable v in a desired way also has the side effect of changing the value of some other variable v' , then we know that there is no way to make the desired change to v without also changing v' . This can be exploited in a planning algorithm to avoid search. That the reachability graph for the requestable values is acyclic is useful in the following way. If a variable v must have a certain value in order to make a change in some other variable possible, then this value is a requestable value for v . If there are several such necessary changes in other variables requiring v to pass through a number of its requestable values, then

there is at most one possible order in which to pass through these values. This can also be exploited to avoid search. Finally, if an instance is prevail-order-preserving, then it can never be better to choose the longer of two alternative paths in a domain-transition graph, since the operators along the longer path always require at least the same requestable values of other variables and in the same order as the shorter one does.

Whether a SAS^+ instance is a SAS^+ -IAO instance can be tested in polynomial time.⁸ Furthermore, a polynomial-time plan generation algorithm for the SAS^+ -IAO class has previously been presented^{10,8} and this algorithm will be referred to later in this article under the name *PlanIAO*.

1 THE PLANNING ALGORITHM

In this section we present a formally correct extension of the SAS^+ -IAO algorithm¹⁰ which is capable of generating plans for the LEGO car factory. The basic idea is to partition the original SAS^+ instance into two separate instances, both being SAS^+ -IAO instances. The first of these instances can then be solved in polynomial time and its solution constitutes a skeleton to be filled in by solving subproblems from the second instance. This process is referred to as *interweaving* and can be viewed as a restricted variant of the more general concept *refinement*, as used in hierarchical state abstraction.^{19,12} In fact, the whole method we use can be viewed as a restricted variant of two-level state abstraction. However, while the state abstraction method in general is not yet well understood—it can sometimes speed up planning exponentially¹² and sometimes slow down planning exponentially³—our, more restricted, method is provably correct and runs in polynomial-time.

First we define the restriction of a SAS^+ structure to a subset of its variables.

Definition 1 *Let $\langle \mathcal{V}, \mathcal{O} \rangle$ be a SAS^+ structure. Then, the restriction to a variable set $\mathcal{V}' = \{v_{i_1}, \dots, v_{i_n}\} \subseteq \mathcal{V}$ of a state $s \in S^+$, an operator $o \in \mathcal{O}$ or an operator set $\mathcal{O}' \subseteq \mathcal{O}$ is denoted $s \restriction \mathcal{V}'$, $o \restriction \mathcal{V}'$ and $\mathcal{O} \restriction \mathcal{V}'$ respectively, and is defined as:*

- $s \restriction \mathcal{V}' = \langle s[v_{i_1}], \dots, s[v_{i_n}] \rangle$,
- $o \restriction \mathcal{V}' = \langle \text{pre}(o) \restriction \mathcal{V}', \text{post}(o) \restriction \mathcal{V}', \text{prv}(o) \restriction \mathcal{V}' \rangle$,
- $\mathcal{O} \restriction \mathcal{V}' = \{o \restriction \mathcal{V}' \mid o \in \mathcal{O}\}$.

This concept is extended to SAS^+ structures and SAS^+ instances by component-wise application, that is $\langle \mathcal{V}, \mathcal{O} \rangle \restriction \mathcal{V}' = \langle \mathcal{V}', \mathcal{O} \restriction \mathcal{V}' \rangle$ and $\langle \mathcal{V}, \mathcal{O}, s, t \rangle \restriction \mathcal{V}' = \langle \mathcal{V}', \mathcal{O} \restriction \mathcal{V}', s \restriction \mathcal{V}', t \restriction \mathcal{V}' \rangle$.

The concept of restriction can be used to split a SAS^+ instance into smaller subinstances by partitioning the set of variables and take the restrictions to each partion of the original instance. In particular, we are interested in partitioning the variable set $\mathcal{V}$ of a SAS^+ structure $\Phi = \langle \mathcal{V}, \mathcal{O} \rangle$ into two sets $\mathcal{V}_1$ and $\mathcal{V}_2$ such that $\mathcal{V}_1$ is *independent* of $\mathcal{V}_2$. This means that a plan for the restriction of Φ to $\mathcal{V}_1$ does not affect any variables in $\mathcal{V}_2$ and can be executed regardless of the values of these variables, while a plan for the restriction of Φ to $\mathcal{V}_2$ does likewise not affect any variables in $\mathcal{V}_1$ but may require such variables to take

on requestable values in order to be executable. That is, there is a one-way dependence between the two partitions.

Definition 2 Let $\Phi = \langle \mathcal{V}, \mathcal{O} \rangle$ be a SAS^+ structure and let $\mathcal{V}_1, \mathcal{V}_2$ be disjoint subsets of $\mathcal{V}$. Then, $\mathcal{V}_1$ is independent of $\mathcal{V}_2$ iff

- every operator $o \in \mathcal{O}$ affecting some variable in $\mathcal{V}_1$ satisfies $\text{pre}(o) \cap \mathcal{V}_2 = \text{post}(o) \cap \mathcal{V}_2 = \text{prv}(o) \cap \mathcal{V}_2 = \langle u, \dots, u \rangle$.
- every operator $o \in \mathcal{O}$ affecting some variable in $\mathcal{V}_2$ satisfies $\text{pre}(o) \cap \mathcal{V}_1 = \text{post}(o) \cap \mathcal{V}_1 = \langle u, \dots, u \rangle$.

We further adopt the term *reachability* from automatic control, with an analogous meaning. A SAS^+ structure is reachable if there exist plans for going from any state to any other state in the state space of this instance.

Definition 3 Let $\Phi = \langle \mathcal{V}, \mathcal{O} \rangle$ be a SAS^+ structure. Then Φ is reachable iff for any two states $s, t \in \mathcal{S}$ the planning problem instance $\langle \mathcal{V}, \mathcal{O}, s, t \rangle$ is solvable.

A planning problem instance $\langle \mathcal{V}, \mathcal{O}, s_0, s_* \rangle$ is reachable iff the corresponding structure $\langle \mathcal{V}, \mathcal{O} \rangle$ is reachable.

It is an immediate consequence that a reachable instance has a strongly connected state-transition graph.

We can now show that if a SAS^+ instance Π is split into two independent instances with certain properties, then it is sufficient to solve these subinstances in order to solve the original instance.

Theorem 1 Let $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_* \rangle$ be a SAS^+ instance and let $\mathcal{V}_1, \mathcal{V}_2$ be a partitioning of $\mathcal{V}$. If $\mathcal{V}_1$ is independent of $\mathcal{V}_2$, $\Pi \cap \mathcal{V}_1$ is reachable and $\Pi \cap \mathcal{V}_2$ is solvable, then Π is solvable.

Proof Suppose there exists a plan $\omega = \langle o_1; \dots; o_n \rangle$ solving $\Pi \cap \mathcal{V}_2$. Since $\Pi \cap \mathcal{V}_1$ is reachable, there must be some plan ω_0 solving the SAS^+ instance $\langle \mathcal{V}_1, \mathcal{O}, s_0, \text{prv}(o_1) \rangle \cap \mathcal{V}_1$. Obviously, o_1 is executable in the state $\text{Result}(\omega_0, s_0)$. Furthermore, recursively define the plans ω_k , where $1 \leq k \leq n-1$, such that ω_k is a solution to the instance

$$\langle \mathcal{V}_1, \mathcal{O}, \text{Result}((\omega_0; \dots; \omega_{k-1}), s_0, \text{prv}(o_{k+1})) \rangle \cap \mathcal{V}_1.$$

Finally, let ω_n be a solution to the instance

$$\langle \mathcal{V}_1, \mathcal{O}, \text{Result}((\omega_0; \dots; \omega_{n-1}), s_0, s_*) \rangle \cap \mathcal{V}_1.$$

Since $\Pi \cap \mathcal{V}_1$ is reachable, also the plans $\omega_1 \dots \omega_n$ must exist. Now, let us consider the plan $\omega' = (\omega_0; o_1; \omega_1; \dots; \omega_{n-1}; o_n; \omega_n)$. By the construction of $\omega_0, \dots, \omega_{n-1}$, all pre- and prevail-conditions of $o_1, \dots, o_n$ are satisfied because $\omega = (o_1; \dots; o_n)$ is a valid plan solving $\Pi \cap \mathcal{V}_2$ by assumption. Furthermore, all pre- and prevail-conditions of the operators in $(\omega_0; \dots; \omega_n)$ are satisfied because $(\omega_0; \dots; \omega_n)$ is a valid plan and $\mathcal{V}_1$ is independent of $\mathcal{V}_2$. It remains to show that $s_* \sqsubseteq \text{Result}(\omega', s_0)$. This follows immediately since we know that $(o_1; \dots; o_n)$ is a valid plan for $\Pi \cap \mathcal{V}_2$ and $(\omega_0; \dots; \omega_n)$ is a valid plan for $\Pi \cap \mathcal{V}_1$ and $\mathcal{V}_1$ is independent of $\mathcal{V}_2$. $\square$

Figure 1 shows a planning algorithm based on the proof of Theorem 1. The sets $\mathcal{V}_1$ and $\mathcal{V}_2$ are such that both $\Pi \cap \mathcal{V}_1$ and $\Pi \cap \mathcal{V}_2$ satisfy the IAO restriction in

```

1 procedure Plan( $\langle \mathcal{V}, \mathcal{O}, s_0, s_* \rangle$ );
2  $\langle o_1, \dots, o_n \rangle \leftarrow \text{PlanIAO}(\langle \mathcal{V}_2, \mathcal{O}, s_0, s_* \rangle \mathbin{\text{\textcircled{R}}} \mathcal{V}_2)$ 
3  $\omega_0 \leftarrow \text{PlanIAO}(\langle \mathcal{V}_1, \mathcal{O}, \text{prv}(o_1) \rangle \mathbin{\text{\textcircled{R}}} \mathcal{V}_1)$ 
4 for  $k = 1, \dots, n-1$  do
5    $\omega_k \leftarrow \text{PlanIAO}(\langle \mathcal{V}_1, \mathcal{O}, \text{Result}(\langle \omega_0; \dots; \omega_{k-1} \rangle, s_0, \text{prv}(o_{k+1})) \rangle \mathbin{\text{\textcircled{R}}} \mathcal{V}_1)$ 
6 end for
7  $\omega_n \leftarrow \text{PlanIAO}(\langle \mathcal{V}_1, \mathcal{O}, \text{Result}(\langle \omega_0; \dots; \omega_{n-1} \rangle, s_0, s_*) \rangle \mathbin{\text{\textcircled{R}}} \mathcal{V}_1)$ 
8 return  $\langle \omega_0; o_1; \omega_1; o_2; \dots; \omega_{n-1}; o_n; \omega_n \rangle$ 

```

Figure 1: Planning algorithm

the previous section, and $\mathcal{V}_1$ is independent of $\mathcal{V}_2$. The procedure *PlanIAO* used in this algorithm is the planning algorithm previously presented for the SAS⁺-IAO planning problem.¹⁰ It is obvious from the proof of Theorem 1 that the algorithm is correct, and since *PlanIAO* is polynomial the resulting algorithm is polynomial.

The first step in the algorithm is to find a plan $\langle o_1, \dots, o_n \rangle$ from s_0 to s_* when only state variables from the set $\mathcal{V}_2$ and only operators affecting such variables are taken into account (line 2 in Figure 1). This results in an incomplete plan that may not be executable due to unsatisfied prevail-conditions. The second step, referred to as *interweaving*, attempts to fill in these ‘gaps’ with subplans satisfying the remaining prevail-conditions (lines 3–7 in Figure 1). The interweaving process tests for each operator in the incomplete plan whether its prevail-conditions are satisfied at this point in the plan, *ie.* whether the prevail-condition for the operator o_{k+1} is satisfied in the state s_k which is the state reached when executing the operators in the plan $\langle \omega_0; o_1; \omega_1; o_2; \dots; \omega_{k-2}; o_{k-1}; \omega_{k-1}; o_k \rangle$. If the prevail-condition is not satisfied, then a plan ω_k achieving the desired prevail-condition is developed using only operators in the set $\mathcal{O} \mathbin{\text{\textcircled{R}}} \mathcal{V}_1$. That such a plan exists is obvious since $\Pi \mathbin{\text{\textcircled{R}}} \mathcal{V}_1$ is reachable. In the last step (line 8 in Figure 1) the plans constructed during the interweaving process are spliced into the original incomplete plan, resulting in a complete plan that solves the original problem and is executable.

2 THE PLANNING TOOL

Figure 2 shows the basic modules of the planning tool and the information flow between these. The two main modules are the planner, which generates operation plans, and the translator, which translates the plans into GRAFCET charts. GRAFCET is an IEC-standard⁷ graphical language for process control based on Petri nets.⁵ The GRAFCET chart produced is then compiled into code for a programmable logic controller (PLC) using a commercial compiler¹ (not shown in the figure) and loaded into the PLC which controls the process. Both the planner and the translator are based on a model of the process to be controlled, but the model provides different views of the process for the two modules. The planner view of the model describes the process in the SAS⁺ language, that is, in terms of states and state-changing operators modelled with pre-, post- and prevail-conditions, while the translator view provides information on how to translate each operator in the planner view into a corresponding GRAFCET fragment. The first of these views is, thus, a more abstract model

Figure 2: Sketch over the planning system. Grey boxes indicate modules not yet implemented.

which ties together the functioning of the various parts of the process, while the second view provides fragmented knowledge of how to translate the abstract operator descriptions into actual sensor readings and actuator signals.

In principle one could plug in any planning module as long as it can be

Figure 3: The translation from partial order plan to GRAFCET chart. Operator *a* is an operator with prevail-condition, while operator *b* has no prevail-condition.

supported by a model which can also provide a corresponding view for the translator. In particular, any planner designed for the full SAS⁺ language or some sublanguage thereof would work. Since our main concern is efficient planning, we focus on using efficient planners for sublanguages, which means that the choice of planner must depend on the characteristics of the process at hand. In the LEGO-car-factory example, to be presented shortly, we use the planner described in Section 1 and the details on the implementation of the planner are described by Kvarnström.¹³ The translator is straightforward and will not be described in detail here, but a graphical example is provided in Figure 3 (Russian¹⁸ gives a more detailed account of the translator module).

3 THE LEGO CAR FACTORY

Our application example is an automated assembly line for LEGO cars,²⁰ which is used for undergraduate laboratory sessions in digital control at the De-

Figure 4: Assembling a LEGO car.

Figure 5: The first half of the LEGO car factory.

partment of Electrical Engineering at Linköping University. The students are faced with the task of writing a program to control this assembly line using GRAFCET. The LEGO car factory is a realistic miniature version of a real industrial process in many respects. Finding a polynomial-time planning algorithm capable of planning for the LEGO-car factory has, thus, been a long-standing goal in our research.

The main assembly operations for building a LEGO car are shown in Figure 4. However, this is an abstracted view from the point of the workpiece, *ie.* the car, and the actual operation plan to achieve the effect of these two abstract assembly operations is much more complex. The assembly line consists of two similar halves, the first mounting the chassis parts on the chassis and the second mounting the top.

The first half of the LEGO car factory is presented in Figure 5. The chassis are initially stored up-side down in a stack in the chassis magazine (*cm*), and the lowermost chassis can be pushed out onto the conveyor belt by the chassis feeder (*c-feeder*). The chassis is then transported to the chassis-parts magazine (*cpm*) where, analogously to the chassis magazine, the chassis parts are stored in a stack such that the lowermost set of chassis parts can be pushed out onto the current chassis by the chassis-parts feeder (*cp-feeder*). The conveyor belt runs continuously, so to stop a chassis one does not stop the belt, but pushes out a stopper bar in front of the chassis to hold it fixed at the desired position, the belt thus sliding under the chassis. At the chassis-parts magazine, for instance, the

Figure 6: The second half of the LEGO car factory.

stopper bar *cpm-stop* must be pushed out across the conveyor belt before the chassis reaches the chassis-parts magazine. When the chassis parts have been pushed out onto the chassis, the feeder is first retracted and then the stopper bar is retracted, allowing the chassis to move on. The chassis is thus transported to the chassis press (*cp*), where it is once again stopped by a stopper bar (*cp-stop*) and the chassis parts, put onto the chassis at the previous workstation, are pressed tight onto the chassis by the chassis press (*cp*). The chassis is then once again released and moves on to the end of the first conveyor belt and moves into the second half of the factory, which is shown in Figure 6.

Here the chassis enters a chassis lift (*cl*) which moves the chassis onto the second conveyor belt and at the same time turns it into upright position. The second half of the factory is analogous to the first half, consisting of two workstations, the first one placing a top on the car from the top magazine (*tm*) and the second one pressing the top tight onto the car. Finally, the finished car is pushed off the belt, sideways, into the buffer storage (*st*).

Figure 7 shows one of the work-stations in more detail, namely the one where the top is put onto the chassis (*tm* in Figure 6). The chassis is held fixed at the top storage (A) by the stopper bar (B). The tops are stored in a stack and the feeder (C) is used to push out the lowermost top onto the chassis. When the top is on the chassis, the feeder is withdrawn and then the stopper bar is withdrawn, thus allowing the chassis to move on to the next work-station.

4 THE PLANNER-VIEW OF THE FACTORY MODEL

To build the planner view of the planning-tool model for this process, we first introduce the state variables shown in Table 1 together with their corresponding value domains.

Most of these variables correspond to stopper and feeder bars *etc.* and have the obvious names and values (*ext* for extended and *rtr* for retracted). The variable *pos* gives the position of the chassis, with discrete values as indicated in Figures 5 and 6, and the variable *turner* tells if the turner (*ts* in Figure 5) is turned towards the first half of the factory (A) or towards the second half of the factory (B). Furthermore, the two variables *cp-status* and *t-status* give the

Figure 7: Putting the top onto the chassis.

variable	values
<i>pos</i>	<i>cm, cpm, cp, ts, cl, ocvB, tm, tp, sf, st</i>
<i>turner</i>	<i>A, B</i>
<i>cp-status</i>	<i>off,on,presed</i>
<i>t-status</i>	<i>off,on,presed</i>
<i>c-status</i>	<i>prepared,not-prepared</i>
<i>cp-press</i>	<i>down,up</i>
<i>t-press</i>	<i>down,up</i>
<i>clift</i>	<i>down,up</i>
<i>c-feeder,cp-feeder,t-feeder,st-feeder</i>	<i>ext,rtr</i>
<i>cpm-stop,cp-stop,tm-stop,tp-stop</i>	<i>ext,rtr</i>

Table 1: State variables $\mathcal{V}$ and their associated domains of values $\mathcal{D}_v$.

status of the chassis parts and the top, respectively, while the variable *c-status* denotes the status of the chassis and is mainly needed since we have no sensor detecting if the chassis is just outside the chassis magazine.

Using these variables we define SAS⁺ operators corresponding to the possible actions that can be executed in the factory. Table 2 shows the workpiece-related actions, that is, the actions changing the position of the chassis or changing its status, and Table 3 shows the operators for controlling the turner, the lift and the presses. Finally, there must be operators for extending and retracting the stopper and feeder bars. These are not shown explicitly here, but are named systematically. For instance, the operators corresponding to the chassis feeder

are denoted *extend-c-feeder* and *retract-c-feeder*. The pre-condition is that *c-feeder = rtr*, the post-condition is that *c-feeder = ext* and there is no prevail-condition.

Operator	Pre	Post	Prevail
<i>cm2cpm</i>	<i>pos = cm</i>	<i>pos = cpm</i>	<i>c-feeder = ext, cp-feeder = rtr, cpm-stop = ext</i>
<i>cpm2cp</i>	<i>pos = cpm</i>	<i>pos = cp</i>	<i>cpm-stop = rtr, cp-stop = ext, cp-press = up</i>
<i>cp2ts</i>	<i>pos = cp</i>	<i>pos = ts</i>	<i>turner = A, cp-stop = rtr, cp-press = up</i>
<i>ts2cl</i>	<i>pos = ts</i>	<i>pos = cl</i>	<i>turner = B, clift = down</i>
<i>cl2ocvB</i>	<i>pos = cl</i>	<i>pos = ocvB</i>	<i>clift = up</i>
<i>ocvB2tm</i>	<i>pos = ocvB</i>	<i>pos = tm</i>	<i>tm-stop = ext, t-feeder = rtr</i>
<i>tm2tp</i>	<i>pos = tm</i>	<i>pos = tp</i>	<i>tm-stop = rtr, tp-stop = ext, t-press = up</i>
<i>tp2sf</i>	<i>pos = tp</i>	<i>pos = sf</i>	<i>tp-stop = rtr, st-feeder = rtr, t-press = up</i>
<i>sf2st</i>	<i>pos = sf</i>	<i>pos = st</i>	<i>st-feeder = ext</i>
<i>prepare-chassis</i>	<i>c-status = not-prepared</i>	<i>c-status = prepared</i>	<i>c-feeder = rtr</i>
<i>put-cp</i>	<i>cp-status = off</i>	<i>cp-status = on</i>	<i>pos = cpm, cp-feeder = ext</i>
<i>press-cp</i>	<i>cp-status = on</i>	<i>cp-status = pressed</i>	<i>cp-press = down, pos = cp</i>
<i>put-top</i>	<i>t-status = off</i>	<i>t-status = on</i>	<i>pos = tm, t-feeder = ext</i>
<i>press-top</i>	<i>t-status = on</i>	<i>t-status = pressed</i>	<i>pos = tp, t-press = down</i>

Table 2: Operators with prevail-conditions.

Operator	Pre	Post	Prevail
<i>A2B</i>	<i>turner = A</i>	<i>turner = B</i>	-
<i>B2A</i>	<i>turner = B</i>	<i>turner = A</i>	-
<i>cl-down</i>	<i>clift = up</i>	<i>clift = down</i>	-
<i>cl-up</i>	<i>clift = down</i>	<i>clift = up</i>	-
<i>cp-press-down</i>	<i>cp-press = up</i>	<i>cp-press = down</i>	-
<i>cp-press-up</i>	<i>cp-press = down</i>	<i>cp-press = up</i>	-
<i>t-press-down</i>	<i>t-press = up</i>	<i>t-press = down</i>	-
<i>t-press-up</i>	<i>t-press = down</i>	<i>t-press = up</i>	-

Table 3: Operators without prevail-conditions.

It should be noted that while some operators correspond directly to real actions in the factory, like the operator for extending a particular feeder bar, other operators correspond to actions which should rather be viewed as *events*. For instance, since the conveyor belt runs continuously it is not possible to directly execute the operator *cm2cpm* which moves the chassis from the chassis magazine to the chassis-parts magazine. In the real world this is rather an event which is automatically initiated as soon as the chassis is pushed out onto the conveyor belt and terminates successfully only if the stopper bar at the chassis-parts magazine is extended when the chassis reaches it. For this particular process, it is sufficient to view such events as actions and model them as operators, but some care must be taken in the modelling. For instance, this particular operator has as prevail-condition that the stopper bar is extended, that is, we must extend the stopper bar already before we push out the chassis onto the conveyor belt.

Although this means extending the stopper bar somewhat earlier than is strictly necessary, it is not likely to degrade performance in practice; even if we want to pipe-line the production of cars, we would, for safety reasons, probably not schedule two chassis so that they appear on the same belt at the same time without at least one extended stopper bar between them.

5 PLANNING FOR THE LEGO CAR FACTORY

The model of the LEGO-car factory presented in the previous section is a SAS⁺ structure $\Phi = \langle \mathcal{V}, \mathcal{O} \rangle$. In order to choose a suitable planning algorithm for this structure we have to examine the characteristics of it. It can be verified that the model does not satisfy the restrictions of the SAS⁺-IAO class¹⁰ or any of the other previously presented tractable SAS⁺ subclasses.⁴ However, it is easily seen that only the operators in Table 2 have defined prevail-conditions. We can thus divide the state variables into two sets: $\mathcal{V}_2$ containing those variables affected by operators with defined prevail-conditions, *ie.* the operators in Table 2, and $\mathcal{V}_1$ containing those variables affected by other operators. Obviously, these two sets form a partitioning of the set of state variables having the following values:

$$\begin{aligned}\mathcal{V}_1 &= \{turner, cp\text{-}press, t\text{-}press, clift, c\text{-}feeder, cp\text{-}feeder, t\text{-}feeder, \\ &\quad st\text{-}feeder, cpm\text{-}stop, cp\text{-}stop, tm\text{-}stop, tp\text{-}stop\} \\ \mathcal{V}_2 &= \{pos, cp\text{-}status, t\text{-}status, c\text{-}status\}\end{aligned}$$

These variables thus define two restricted substructures $\Phi \upharpoonright \mathcal{V}_1$ and $\Phi \upharpoonright \mathcal{V}_2$ of Φ and it can be further verified that the first of these structures is reachable and that $\mathcal{V}_1$ is independent of $\mathcal{V}_2$. Furthermore, both these substructures are SAS⁺-IAO structures, so the planning algorithm presented in Figure 1 can be used to plan for the LEGO-car factory.

Depending on how we choose the initial state and the goal state we can plan for different cases. As an example we show a plan for normal operation where the goal is a fully assembled LEGO car and the initial state is a factory in ‘resting’ state, that is, the chassis is placed in the chassis magazine ($pos = cm$, $c\text{-}status = not\text{-}prepared$), there are no chassis parts on the chassis ($cp\text{-}status = off$) and there is no top on the chassis ($t\text{-}status = off$). Furthermore the turner is turned towards the first half of the factory ($turner = A$), all feeders and stopper bars are retracted and the chassis press, the top press and the chassis lift are in their down position. The goal state is that the chassis should be in the buffer storage ($pos = st$) and the top and chassis parts should be pressed onto the chassis ($cp\text{-}status = pressed$ and $t\text{-}status = pressed$). All other state variables are left undefined in the goal states, meaning that they are allowed to have any value at the end of the plan, that is, we do not care about their final values.

Applying the algorithm in Figure 1 results in the plan shown as a directed graph in Figure 8. The subgraph consisting of those operators that are connected by solid arrows is the plan skeleton resulting from the first step in the algorithm (line 2 in Figure 1). This plan cannot be executed due to unsatisfied prevail-conditions. For example the operator $cm2cpm$ cannot be executed in its current position in the plan since the chassis feeder is retracted in the initial state and not affected by the previous operator $prepare\text{-}chassis$, while it has to be extended in order to execute the operator $cm2cpm$. In order to solve such problems, the interweaving process (lines 3-7 in Figure 1) adds those operators connected by

6 DISCUSSION

Tsatsoulis and Kashyap²¹ call planning “one of the most underused techniques of AI” in the context of manufacturing. They list a number of areas within industry where planning could be applied, but where no or very little attempts have been made at such applications. There are some examples of such work in the recent literature, however, including the following examples. Nau *et. al.*¹⁷ have studied the computational complexity of the domain-specific problem of machining parts, and also developed a planner for this problem. However, their planner is not based on standard AI planning techniques, but is a specialised algorithm tailored to the needs of the particular application. Kis and Váncza¹¹ analyze the computational complexity of manufacturing planning modelled with STRIPS operators, but their planning system is chiefly based on a combination of expert systems and genetic programming.¹⁴ Munõs-Avila and Weberskirch¹⁶ present a system for planning the manufacturing of workpieces and they seem to use STRIPS-style operators. Their planner uses standard techniques from AI planning, but it is the technique of case-based planning and plan adaptation rather than the techniques for producing plans ‘from scratch’.

Our work differs from these examples in two aspects. First, we use a general-purpose AI planner, designed for restricted languages, but not specifically tailored to manufacturing and assembly problems. Second, although applicable for generating assembly plans, our system is rather primarily intended for error-recovery planning. It is further important to note that although the interweaving planner used in this application was developed with the purpose of planning efficiently for the LEGO car factory, we have developed also other polynomial-time planners capable of planning for this application, *eg.* the 3S planner.⁹ This planner was not designed with any particular application in mind, but stems from purely theoretical considerations.

The reason that we could not apply the SAS⁺-IAO planner directly is that it does not allow cycles involving more than one requestable value. This limitation is to some extent overcome in the new planner, since the interweaving process allows certain restricted types of such cycles. However, although this is sufficient for modelling the LEGO car factory and probably also for modelling a number of other interesting processes, it should not be expected to suffice for the large share of real applications. This means that further extensions and/or modification will be required, and it is, thus, a topic of future research to investigate how to handle other, more complex forms of cycles. Further, it should not be forgotten that the planning tool is not dependent on planners being based on the SAS⁺-IAO class, so other planners could also be used whenever required.

Although the planner for the SAS⁺-IAO problem produces partial-order plans, having some potential for parallelizing operators, the subsequent interweaving process generates a total-order plan as output. Hence, the current planning tool can only produce plans where all operations must be executed in sequence. This is not an inherent limitation in the factory, however, so our plans for future research include modifying the planning algorithm and planning tool to produce plans which allow for parallel execution of operators and thus increase the throughput by pipe-lining. Although the problem of producing optimal parallel plans is known to be very difficult in the general case,² we believe this particular application to have sufficient structure to allow for efficient parallelization of plans. The future plans also include closing the feedback

loop by continuously monitoring the assembly line to detect deviations from the expected state when executing plans. In such cases plan execution should be stopped and the current actual state of the process be fed back to the planner, which can then produce a plan for recovering from this state.

7 CONCLUSIONS

We have applied our previous results on polynomial-time planning to an application example in automatic control—an assembly line for LEGO cars. Since none of the restricted planning languages used by our previous polynomial-time planners was sufficiently expressive for modelling this application, we had to modify one of the planners. This was done by using one of the previous planners, the planner for the SAS⁺-IAO class, as a subprocedure in a provably correct and efficient two-level hierarchical planner. Although developed for this application, the planner is a general-purpose planner, not specifically tailored to the particular application. This planner is used in a planning tool which works in the following way. The planning algorithm produces a plan, based on a model of the factory, and then a translation module converts this plan into an equivalent GRAFCET chart. Finally, a GRAFCET compiler is used to compile this chart into code for the programmable logic controller that controls the factory.

This paper is an example where the attempt to apply originally theoretical results to an application provided feedback for modifying the theory in a way that allowed for solving the application as well as extending the theoretical results in a non-application-dependent way.

ACKNOWLEDGEMENTS

Jonas Kvarnström implemented the planning algorithm and Fabio Russian implemented the program for translating plans into GRAFCET charts. The research presented in this article was sponsored by the Swedish Research Council for Engineering Sciences (TFR) under grants dnr. 92-143, 93-291 and 95-731.

References

- [1] Actron AB. *ActGraph+ GrafCet Programming for Hitachi PLC*, 1991.
- [2] C. Bäckström. Finding least constrained plans and optimal parallel executions is harder than we thought. In Christer Bäckström and Erik Sandewall, editors, *Current Trends in AI Planning: EWSP'93—2nd European Workshop on Planning*, pages 46–59, Vadstena, Sweden, December 1993. IOS Press.
- [3] C. Bäckström and P. Jonsson. Planning with abstraction hierarchies can be exponentially less efficient. In Mellish,¹⁵ pages 1599–1604.
- [4] C. Bäckström and B. Nebel. Complexity results for SAS⁺ planning. *Computational Intelligence*, 11(4):625–655, 1995.
- [5] R. David and H. Alla. *Petri Nets and Grafcet: Tools for modelling discrete event systems*. Prentice Hall, New York, 1992.

- [6] Malik Ghallab and Alfredo Milani, editors. *New Directions in AI Planning: EWSP'95—3rd European Workshop on Planning*, Frontiers in AI and Applications, Assisi, Italy, September 1995. IOS Press.
- [7] IEC. Preparation of function charts for control systems. Publication 848:1988, IEC, 1988.
- [8] P. Jonsson and C. Bäckström. Tractable planning with state variables by exploiting structural restrictions. In *Proceedings of the 12th (US) National Conference on Artificial Intelligence (AAAI-94)*, pages 998–1003, Seattle, WA, USA, July–August 1994. American Association for Artificial Intelligence.
- [9] P. Jonsson and C. Bäckström. Incremental planning. In Ghallab and Milani,⁶ pages 79–90.
- [10] P. Jonsson and C. Bäckström. State-variable planning under structural restrictions: Algorithms and complexity. *Artificial Intelligence*, 100(1–2):125–176, 1998.
- [11] Tamás Kis and József Váncza. Computational complexity of manufacturing process planning. In Ghallab and Milani,⁶ pages 299–311.
- [12] Craig A. Knoblock. Automatically generating abstractions for planning. *Artificial Intelligence*, 68:243–302, 1994.
- [13] Jonas Kvarnström. Implementation of a tractable planner. Master thesis report, Department of Computer and Information Science, Linköping University, Linköping, Sweden, 1996.
- [14] András Márkus and József Váncza. Inference and optimization methods for manufacturing process planning. In Anthony G. Cohn, editor, *Proceedings of the 11th European Conference on Artificial Intelligence (ECAI-94)*, pages 595–599, Amsterdam, Netherlands, August 1994. Wiley.
- [15] Chris Mellish, editor. *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95)*, Montréal, PQ, Canada, August 1995. Morgan Kaufmann.
- [16] Héctor Munõs-Avila and Frank Weberskirch. Planning for manufacturing workpieces by storing, indexing and replaying planning decisions. In Brian Drabble, editor, *Proceedings of the Third International Conference on Artificial Intelligence Planning Systems (AIPS'96)*, pages 150–157, Edinburgh, Scotland, May 1996. AAAI Press.
- [17] Dana S. Nau, Satyandra K. Gupta, and William C. Regli. AI planning versus manufacturing-operation planning: A case study. In Mellish,¹⁵ pages 1670–1676.
- [18] Fabio Russian. Automatic generation of control programs for an assembly line. Master Thesis Report LiTH-ISY-EX-1620, Department of Electrical Engineering, Linköping University, Linköping, Sweden, 1995.
- [19] Earl D Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5(2):115–135, 1974.

- [20] Jan-Erik Strömberg. Styrning av LEGO-bilfabrik. Andra omarbetade upplagan. Department of Electrical Engineering, Linköping University, 1991.
- [21] Costas Tsatsoulis and R L Kayshap. Planning and its application to manufacturing. In Soundar T Kumara, Rangasami L Kashyap, and Allen L Soyster, editors, *Artificial Intelligence, Manufacturing Theory and Practice*, chapter 7, pages 193–223. Institute of Industrial Engineers, 1988.