Automatic Synthesis of Control Programs in Polynomial Time for an Assembly Line

Inger Klein
Department of Electrical Engineering
Linköping University,
S-581 83 Linköping, Sweden
e-mail: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
e-mail:{petej,cba}@ida.liu.se
phone: +46 13 282415, +46 13 282429

Abstract

The industry wants provably correct and fast formal methods for handling combinatorial dynamical systems. One example of such problems is error recovery in industrial processes. We have used a provably correct, polynomial-time planning algorithm to plan for a miniature assembly line, which assembles toy cars. Although somewhat limited, this process has many similarities with real industrial processes. By exploring the structure of this assembly line we have extended a previously presented algorithm, thus extending the class of problems that can be handled in polynomial time. The planning tool presented here contains general-purpose algorithms that generate plans in the form of GRAFCET charts that are automatically translated into PLC code using a commercial PLC compiler.

Keywords: Planning, GRAFCET, sequential control, automated manufacturing

1 Introduction

The majority of all hardware and software developed for industrial control purposes is devoted to sequential control and only a minority to classical, linear control. Typically, the sequential parts of the controller are invoked during startup and shut-down to bring the system into its normal operating region and into some safe standby region, respectively. Despite its importance, not much theoretical research has been devoted to this area, and sequential control programs are therefore still created manually without much theoretical support to obtain a systematic approach.

We propose a method to create sequential control programs automatically and on-line upon request. The main idea is to spend some effort off-line *modeling* the plant, and from this model generate the control strategy, that is, the *plan*.

The process industry is one example of application areas where automatic generation of control programs can be useful. The problem is not primarily to find the plan for normal operation of the plant. This is usually done once and for all and can probably be done better manually, since time is not critical in this case. Automated planning is more likely to enter the scene when something goes wrong. Since there are many ways in which a large process may go wrong, we can end up in any of a very large number of states. It is not realistic to have pre-compiled plans for recovering from any such state so it would be useful to find a plan automatically for how to get back to a safe state, where normal operation can resume. It is important that such a plan be correct and we also want to find it fast since the costs accumulate very quickly when large-scale industrial processes are non-operational.

Automated plan generation is also important if the initial state is not fully specified until the plan is needed. As in the error recovery case it is important that the plan be correct and that it be found reasonably fast. Another situation where automated plan generation may be useful is for operator support, *ie.*, a form of semi-automated planning. In this case the operator request *OpenValveA* may result in an automatically generated plan which brings about the desired effect. Another possibility is that a supervisor checks if the wished-for action is allowed in the present state, and if not specifies why and how the operator can achieve what he or she wants. Sometimes new devices are added to the plant when the control program is already developed. It can be very difficult and time consuming to find out how such a change affects the original control program. If using a model-based synthesis approach, then only the model need to be changed to take the new devices into account.

Automated generation of plans, *action planning*, has been studied for over 25 years within the area of artificial intelligence. A number of languages for modeling planning problems and a number of algorithms for solving planning problems have been developed. The traditional planning algorithms are general and search based, thus suffering from the problems of combinatorial explosion. This is not only a problem with the algorithms, however, but also inherent in the problem since the representation languages allow formulating very difficult problems. A problem is usually considered practically solvable if there exists an algorithm whose running time is bounded in some polynomial in the size of the problem instance (the number of state variables, for instance). To the contrary a problem which can only be solved by algorithms with an exponential running time behavior are infeasible for all but extremely small problem instances. Action planning, even in very restricted formalisms like STRIPS [5] restricted to only propositional atoms, is very difficult, belonging to the class of PSPACE-complete problems, which are strongly believed to require exponential time. Real problems in the industry, on the other hand, are probably not that difficult in practice. This, however, does not imply that the results from action planning are useless, it only tells us that real problems have inherent structure and restrictions which we could exploit to plan efficiently.

Hence we propose to study subclasses where we retain feasibility by exploring the structure of the problems. We formulate the model using *Simplified Action Structures* [4, 7] based on action structures originally introduced by Sandewall and Rönnquist [12]. We have previously presented algorithms for generating *minimal plans* (no plan containing fewer actions exists) for some restricted classes of planning problems [2, 3, 8, 9]. One major advantage with this approach is that the complexity of these algorithms only increases polynomially with the number of state variables. A summary of the earlier results is presented in Bäckström and Nebel [4]. Lately we have concentrated on studying certain structural properties of planning problems that are closer to reality [7, 10].

This paper reports how we have applied a polynomial time algorithm to plan for a semi-realistic miniature version of an industrial process, the LEGO[1] car factory [13]. This is a realistic miniature version of real industrial processes in many respects. Our planning algorithm is very fast, running in polynomial time, which is made possible by exploiting typical structural properties of the assembly line. Furthermore, the algorithm is also provably correct, that is, we have proven mathematically that the algorithm will always find a solution if one exists at all and that it will never return a faulty solution [10]. This is important for the quality aspects of production processes. Here we present the practical results stemming from the LEGO car factory project. In fact the study of the LEGO car factory also provided feedback for modifying the theory, and the class of problems that can be handled is thus extended. The presented planning tool generates a plan as a GRAFCET [6] chart, that is automatically loaded into a commercial tool [1] that translates the GRAFCET chart into PLC code. The PLC code is loaded into the PLC which then controls the factory. Note that neither our algorithm nor the planning tool is specifically tailored for this assembly line, but is a general-purpose device.

The rest of the paper is organized as follows. Section 2 presents the LEGO car factory, while the modeling is given in section 3. The planning tool is described in section 4 and the algorithm is given in section 5. Section 6 contains the conclusion.

2 The LEGO Car Factory

Our application example is an automated assembly line for LEGO cars [13], which is used for undergraduate laboratory sessions in digital control at the Department of Electrical Engineering at Linköping University. The students are faced with the task of writing a program to control this assembly line using the graphical language GRAFCET [6]. Being an IEC standard, GRAFCET is becoming increasingly popular in industrial applications. There is commercially available software for editing GRAFCET charts and compiling them to PLC (Programmable Logical Controller) programs. The LEGO car factory is a realistic miniature version of a real industrial process in many respects. Hence, being able to model and automatically generate control programs for this factory has been one of our goals.

The main operations for assembling a LEGO car are shown in Figure 1. The assembly line consists of two similar halves, the first mounting the chassis parts on the chassis (see Figure 2) and the second mounting the top (see Figure 3).

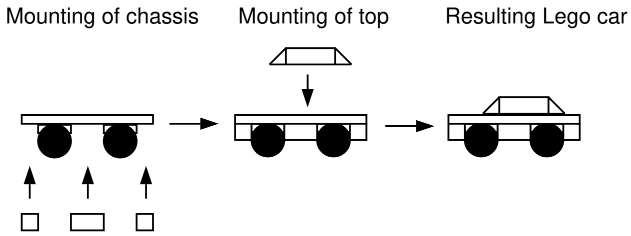

Figure 1: Assembling a LEGO car.

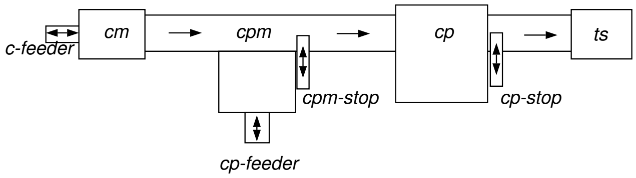

Figure 2: The first half of the LEGO car factory.

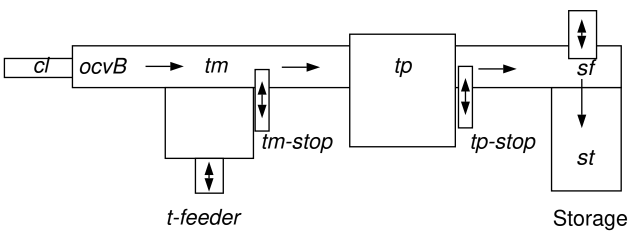

Figure 3: The second half of the LEGO car factory.

[1]LEGO is a trademark of the LEGO company.

The first half of the LEGO car factory is presented in Figure 2. The chassis is initially stored up-side down in the chassis magazine (*cm*). It enters the conveyor belt by using the chassis feeder (*c-feeder*), and is transported to the chassis parts magazine (*cpm*) where the chassis parts are fed onto the chassis using the chassis parts feeder (*cp-feeder*). The chassis is then transported to the chassis press (*cp*), where the chassis is pressed together. It is then transported to the turn station (*ts*) where the chassis is turned upright and enters the second half of the factory (Figure 3) where it is placed on the chassis lift (*cl*). It is lifted up, placed on the conveyor belt (*ocvB*) and transported to the top magazine (*tm*) where a top is fed onto the chassis by the top feeder (*t-feeder*). The chassis is then transported to the top press (*tp*) where the top is pressed tight onto the chassis. From there it is transported to the end of the conveyor belt (*sf*) and placed so that the storage feeder (*st-feeder*) can push the chassis into a buffer storage (*st*).

The conveyor belt used to transport the chassis runs continously. Hence, at each work-station a stopper bar is pushed out in front of the chassis holding the car fixed, sliding on the belt (see see Figure 2 and Figure 3). The car cannot pass the work-station until the stopper bar is withdrawn.

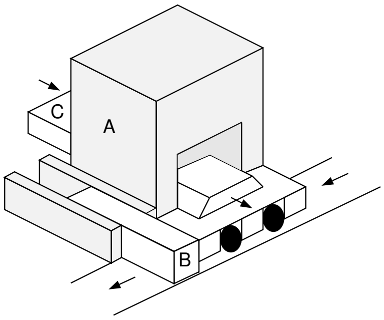

Figure 4: Putting the top onto the chassis.

Figure 4 shows one of the work-stations in more detail, namely the one where the top is put onto the chassis (*tm* in Figure 3). The chassis is held fixed at the top storage (A) by the stopper bar (B). The tops are stored in a pile 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.

3 Modeling the LEGO car factory

The main idea is to spend some effort off-line modeling the plant, and from this model generate the control strategy automatically and on-line upon request. To model the plant we use the SAS$^+$ planning formalism [4, 7] which can be viewed as a variation on the propositional version of the STRIPS [5] formalism. The process is modeled by a state and a set of operators that can change the system state. Each operator specifies how the operator affects the process (*pre-* and *post-condition*) and a constraint that must be satisfied when executing the operator (*prevail-condition*). More specifically, the pre-condition is a condition that must be satisfied before executing the operator, and the post-condition is a condition that will hold after execution. For example, the operator *MoveFromTopMagazineToTopPress* has as pre-condition that the chassis is at the top magazine, and as post-condition that it is at the top press. The prevail-condition is that the stopper bar at the top magazine and the top feeder must be retracted, the stopper bar at the top press extended and the top press in its upper position.

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

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

We continue by modeling the LEGO car factory as a SAS$^+$ instance. The state variables are shown in Table 1. The variable *pos* gives the position of the chassis, and the corresponding positions are given in Figure 2 and Figure 3. The stopper bars and the corresponding variable names are also marked in these figures, as well as the variable names for the feeders. For the feeders and the stopper bars the value *ext* means that the feeder (or stopper bar) is extended, while *rtr* means that it is retracted. The variable *turner* tells if the turner (*ts* in Figure 2) is turned towards the first half of the factory (A) or towards the second half of the factory (B). The two variables *cp-status* and *t-status* give the 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. The other variables should be obvious from the table and the figures.

Using the variables defined in Table 1 we can define operators as in Tables 2 and 3. Additionally there are two operators for each feeder and each stopper bar for retracting and extending the feeder or stopper bar. The operators corresponding to the chassis feeder are denoted *extend-c-feeder* and *retract-c-feeder*. For the operator *extend-c-feeder* the pre-condition is that *c-feeder* = *rtr*, the post-condition is that *c-feeder* = *ext* and there is no prevail-condition. The other operators corresponding to the feeders and stopper bars are denoted in a similar manner.

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

Table 2: Operators with prevail-conditions.

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

Table 3: Operators without prevail-conditions.

4 Automatic Generation of Control Programs

In figure 5 the basic parts of the planning tool as well as the information flow is shown.

The base concept used in the planning tool is the *model* of the process to be controlled as described in section 3. The model is developed off-line and is stored in a database ready for use (see figure 5). Together with the actual state of the process and the goal state the model is used by the planner to develop a plan. Depending on the properties of the problem, different algorithms should be used. Which algorithm to use can be decided off-line by analyzing the model. The algorithm used for the LEGO car factory is described in section 5. The plan delivered by the planner is a partially ordered set of actions. This plan is then automatically translated to a GRAFCET chart, which in turn is translated to PLC-code and loaded into the PLC, which controls the real-world process.

The planner and the module translating the partial order plan into a GRAFCET chart work with different views of the model. The planner needs to know the definition of a state and the pre-, post- and prevail-conditions for each action as described in section 3. The translation module on the other hand must know the actual implementation of each action, i.e., which actuator implements a specific action, and how the pre-, post- and prevail-conditions are to be translated into sensor readings. This second view of the model will not be described in detail here, but an example is shown in Figure 6. A more thourough description is given in [11].

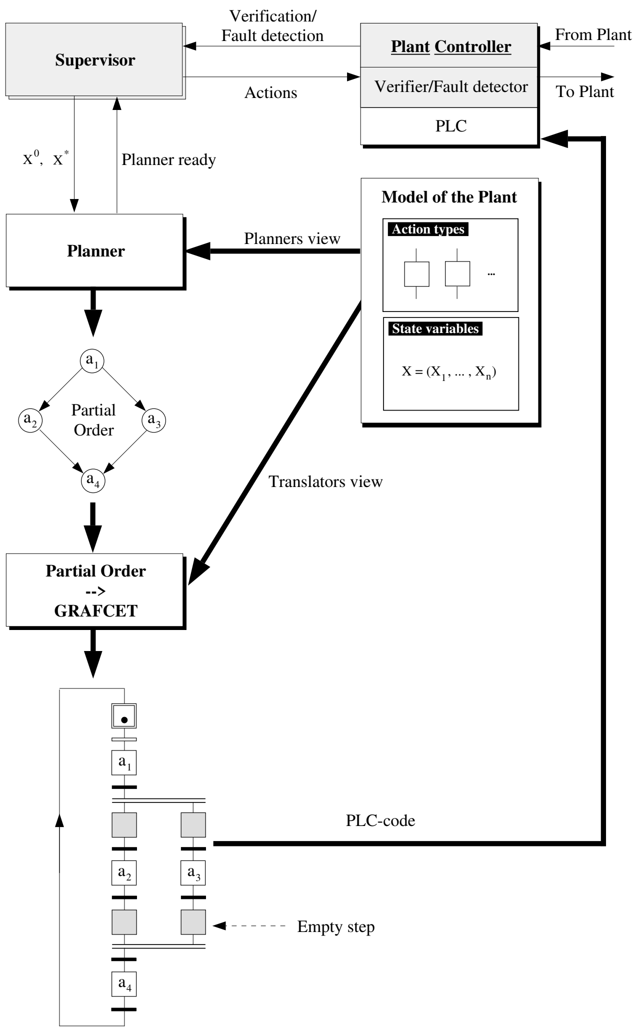

Figure 5: Sketch over the planning system. Grey boxes indicate modules not yet implemented.

5 The planning algorithm

Without putting any further restrictions on the modeled process we still have a complexity problem, and several subclasses show an exponential complexity. We have studied this problem for about five years, and have introduced restrictions to reduce the complexity. This has resulted in several subclasses of problems where we have constructed provably correct algorithms with polynomial complexity, see for example [7]. Recently we have focussed on structural restrictions on the state transition graph. By allowing a simple type of loops in the state transition graph, we have extended the algorithm given in [7] so that the LEGO car factory

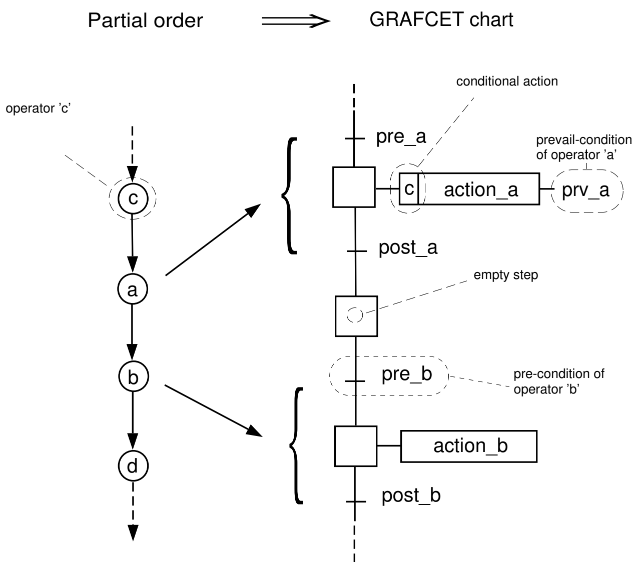

Figure 6: The translation from partial order plan to GRAFCET chart. Operator a is an operator with prevail-condition, while operator b has no prevail-condition.

problem above can be solved in polynomial time [10]. The resulting plan can be automatically translated into a GRAFCET chart, which in turn can be compiled to PLC code which actually controls the plant as described above.

The main idea in the algorithm is to split the set of operators into two sets, thereby splitting the planning problem into two parts. The result is a process where several simple planning problems are solved, and the combined solution is a solution to the original problem. These simple planning problems fit into the SAS$^+$-IAO class defined in [7], and hence the algorithm given there can be used. The complexity of the SAS$^+$-IAO algorithm increases polynomially with the number of state variables, and thus each subproblem can be solved in polynomial time. Since the number of subproblems is limited by the number of available operators, the resulting complete algorithm is polynomial. The over-all complexity figure only requires that the algorithm that is used to solve the subproblems is polynomial, that is, any polynomial-time algorithm can be used depending on the problem to be solved.

An outline of the complete algorithm is given in Figure 7. The algorithm is described in more detail in [10], while only an informal description is given here. First the set of operators $\mathcal{O}$ is split into two sets $\mathcal{O}_1$ and $\mathcal{O}_2$ where the operators in the set $\mathcal{O}_2$ are independent of the operators in the set $\mathcal{O}_1$. For the LEGO car factory, the first set ($\mathcal{O}_1$) contains all operators with prevail-conditions, *ie.* the operators in Table 2, and the second set ($\mathcal{O}_2$) contains all operators without prevail-conditions, *ie.* the operators in Table 3 and all operators for extending and retracting feeders and stopper bars. The set of variables $\mathcal{V}$ is split in a similar manner. The set $\mathcal{V}_1$ contains all variables affected by an operator in the set $\mathcal{O}_1$, and $\mathcal{V}_2$ contains all variables affected by an operator in the set $\mathcal{O}_2$. Thus $\mathcal{V}_1 \cup \mathcal{V}_2 = \mathcal{V}$ and $\mathcal{O}_1 \cup \mathcal{O}_2 = \mathcal{O}$. Furthermore s_0 is the initial state and s_* the desired goal state.

The first step in the algorithm is to find a plan $(o_1, \ldots, o_n)$ from s_0 to s_* when only operators from the first set and their corresponding state variables are taken into account (line 2 in Figure 7). This will result in an incomplete plan that cannot be executed due to unfulfilled prevail-conditions. The second step is an interweaving process making the plan executable (lines 3–7 in Figure 7). In the interweaving process each operator in the incomplete plan is checked to see if its prevail-conditions are fulfilled when the operator is to be executed, *ie.*if 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 $(\omega_0; o_1; \omega_1; o_2; \ldots; \omega_{k-2}; o_{k-1}; \omega_{k-1}; o_k)$. If the prevail-condition is not satisfied a plan ω_k achieving the desired prevail-condition is searched for in the second set of operators ($\mathcal{O}_2$). In the last step (line 8 in Figure 7) the plans constructed during the interweaving process are merged into the original incomplete plan, resulting in a complete plan that solves the original problem and is executable.

Since only a part of the problem is considered at each time point, only a part of the state is considered as well. In the algorithm in Figure 7 this is not formally stated, but when planning with variables from $\mathcal{V}_1$ ($\mathcal{V}_2$) the state s_k is actually a restriction taking only variables from $\mathcal{V}_1$ ($\mathcal{V}_2$) into account.

1 **procedure** *Plan*$(\mathcal{V}, \mathcal{O}, s_0, s_*)$;
2 $(o_1, \ldots, o_n) \leftarrow \mathit{PlanIAO}(\mathcal{V}_1, \mathcal{O}_1, s_0, s_*)$
3 $\omega_0 \leftarrow \mathit{PlanIAO}(\mathcal{V}_2, \mathcal{O}_2, s_0, \mathbf{prevail}(o_1))$
4 **for** $k = 1, \ldots, n-1$ **do**
5 $\quad \omega_k \leftarrow \mathit{PlanIAO}(\mathcal{V}_2, \mathcal{O}_2, s_k, \mathbf{prevail}(o_{k+1}))$
6 $\quad$ {s_k *is the state reached when executing the operators in the plan* $(\omega_0; o_1; \omega_1; o_2; \ldots; o_{k-1}; \omega_{k-1}; o_k)$}
7 $\omega_n \leftarrow \mathit{PlanIAO}(\mathcal{V}_2, \mathcal{O}_2, s_n, s_*)$
8 **return** $(\omega_0; o_1; \omega_1; o_2; \ldots; \omega_{n-1}; o_n; \omega_n)$

Figure 7: Planning algorithm. *PlanIAO* is a procedure realizing the algorithm described in [7].

Depending on how we choose the initial state and the goal state we can plan for different cases. Here we show a plan for normal operation, *ie.* the goal is to assemble a LEGO car. It is straightforward to modify this to plan for error recovery or for an initial state unknown before execution. The goal state is that the chassis should be in the buffer storage (*pos = st* and *c-status = not-prepared*) and the top and chassis parts should be pressed onto the chassis (*cp-status = pressed* and *t-status = pressed*). All other state variables are undefined and can have any value. Suppose that the initial state is given as follows. The chassis is placed in the chassis magazine (*pos = cm*), there is no chassis parts on the chassis (*cp-status = off*) and there is no top on the chassis (*t-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.

Applying the algorithm in Figure 7 results in the plan in Figure 8. The solid arrows denote the incomplete plan resulting from the first step in the algorithm (line 2 in Figure 7). This plan cannot be executed due to unfulfilled prevail-conditions. For example the operator *cm2cpm* cannot be executed because the chassis feeder is retracted in the initial state, but according to Table 2 it must be extended when executing *cm2cpm*. The result of the interweaving process (lines 3-7 in Figure 7) is shown as dashed arrows in Figure 8.

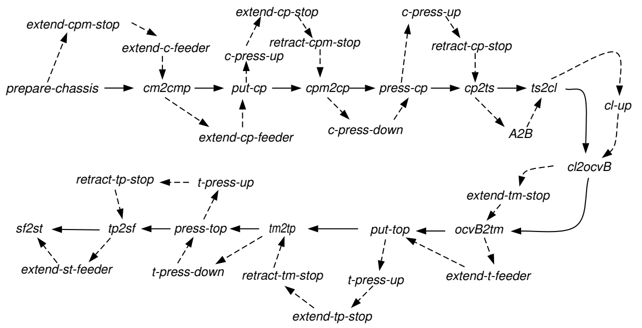

Figure 8: Resulting plan. The solid arrows are the output from solving the first planning problem, and the dashed arrows are the result from the interweaving process.

6 Conclusions

Tsatsoulis and Kayshap [14] 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 few attempts have been made at such applications.

We have applied our previous results on polynomial-time planning to an application example in automatic control—an assembly line for LEGO cars, and have presented a planning tool that can be used to control the LEGO car factory in reality. The planning tool contains algorithms for generating plans for a restricted class of problems in polynomial time. The plans are generated as GRAFCET charts which are automatically translated into PLC code and loaded to the PLC that controls the LEGO car factory. The result is an integrated system able to perform planning in reality.

This paper presents the practical results stemming from the LEGO car factory project. In fact the studying of the LEGO car factory also provided feedback for modifying the theory [10], and the class of problems that can be handled is thus extended.

References

[1] Actron AB. *ActGraph+ GrafCet Programming for HITACHI PLC*, 1991. Manual for the ActGraph+ development system.

[2] C. Bäckström and I. Klein. Parallel non-binary planning in polynomial time. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, pages 268–273, Sydney, Australia, Aug 1991.

[3] C. Bäckström and I. Klein. Planning in polynomial time: the SAS-PUBS class. *Computational Intelligence*, 7:181–197, August 1991.

[4] C. Bäckström and B. Nebel. Complexity results for SAS$^+$ planning. *Computational Intelligence*, 11(4):625–655, 1995.

[5] R. E. Fikes and N. J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. 2:189–208, 1971.

[6] IEC. Preparation of function charts for control systems - IEC 848. Technical Report 848:1988, IEC, Geneve, 1988.

[7] 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)*, Seattle, WA, USA, July–August 1994.

[8] I. Klein and C. Bäckström. On the planning problem in sequential control. In *Proceedings of the 30th Conference on Decision and Control*, pages 1819–1823, Brighton, England, 1991. IEEE.

[9] I. Klein and C. Bäckström. Planning in polynomial time: The SAS-PUS class. Technical Report LiTH-ISY-I-1229, Department of Electrical Engineering, Linköping University, Linköping, Sweden, 1991.

[10] I. Klein, P. Jonsson, and C. Bäckström. Tractable planning for an assembly line. In M. Ghallab and A. Milani, editors, *New Directions in AI Planning: EWSP'95—3rd European Workshop on Planning*, Frontiers in AI and Applications, Assissi, Italy, September 1995. IOS Press.

[11] F. Russian. Automatic generation of control programs for an assembly line. Technical Report LiTH-ISY-EX-1620, Department of Electrical Engineering, Linköping University, Linköping, Sweden, 1995. ISY.

[12] E. Sandewall and R. Rönnquist. A representation of action structures. In *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI-86)*, pages 89–97, Philadelphia, Pennsylvania, August 1986. Morgan Kaufman.

[13] J-E. Strömberg. Styrning av lego-bilfabrik. Technical report, Department of Electrical Engineering, Linköping University, Linköping, Sweden, 1991. Manual for control laboratory session.

[14] C. 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.