A Specification of the Domain of Process Planning: Properties, Problems and Solutions

Héctor Muñoz-Avila and Frank Weberskirch
University of Kaiserslautern, Dept. of Computer Science
P.O. Box 3049, D-67653 Kaiserslautern, Germany
E-mail: {munioz|weberski}@informatik.uni-kl.de

Abstract

This report presents the properties of a specification of the domain of process planning for rotary symmetrical workpieces. The specification results from a model for problem solving in this domain that involves different reasoners, one of which is an AI planner that achieves goals corresponding to machining workpieces by considering certain operational restrictions of the domain. When planning with SNLP (McAllester and Rosenblitt, 1991), we will show that the resulting plans have the property of minimizing the use of certain key operations. Further, we will show that, for elastic protected plans (Kambhampati et al., 1996) such as the ones produced by SNLP, the goals corresponding to machining parts of a workpiece are *$\prec$-constrained trivial serializable*, a special form of trivial serializability (Barrett and Weld, 1994). However, we will show that planning with SNLP in this domain can be very difficult: elastic protected plans for machining parts of a workpiece are nonmergeable. Finally, we will show that, for sufix, prefix or sufix and prefix plans such as the ones produced by state-space planners, it is not possible to have both properties, being $\prec$-constrained trivial serializable and minimizing the use of the key operations, at the same time.

Contents

1 Introduction

Process planning has been defined by the Society of Manufacturing Engineers as "the systematic determination of the methods by which a product is to be manufactured economically and competively" (Alting and Zhang, 1989). During process planning for conventional machining operations, different tasks must be performed (Ham and Lu, 1988; Weill and Eversheim, 1982):

1. Identification of the processing areas,
2. Selection of machining tools,
3. Sequencing of tool sets,
4. Grouping of set-ups,
5. Selection of machining operations and their sequence,
6. Selection of cutting tools,
7. Selection of clamping devices and positions, and datum surfaces,
8. Calculation of cutting conditions,
9. Determination of tools paths,
10. Determination of cutting times, non machining times and costs,
11. NC part program generation.

A major difficulty for systems performing computer aided process planning, CAPP (Tönshoff and Andres, 1990; Eversheim and Schneewind, 1993; ElMaraghy and McMaster, 1993), are the interdependencies between the tasks mentioned above. For example, selecting the tool paths for removing layers of the piece of raw material depends on the clamping device and the clamping position from which the piece of raw material has been clamped. Thus, during the machining process these interdependencies must be considered.

Our model for problem solving involves the interaction between a geometrical reasoner, an AI planner and a numeric control (NC) reasoner. Each of them performs some of the tasks mentioned before. This integration of specific reasoners has been already proposed by different authors working in CAPP and fielded in both AI (Kambhampati et al., 1991) and in mechanical engineering (Ham and Lu, 1988; Giusti et al., 1989; Eversheim and Schneewind, 1993; ElMaraghy and McMaster, 1993). The advantage of using an AI planner is its recognized capability to perform the subgoaling process and to detect interactions between subplans, for example, for machining parts of a workpiece. The AI planner reasons about a specification of the machining process of the workpiece involving the particularities of the workpiece and the available resources. Our aim is not to provide a complete model, but rather, to investigate the possibilities and limitations of using a symbolic reasoner to handle those interactions. Thus, the machining process of certain selected features such as threads was carefully represented.

We found interesting properties about our specification of this domain: when planning with SNLP (McAllester and Rosenblitt, 1991), the resulting plans minimize the application of clamping the workpiece and holding cutting tools, which are key to the machining process. We also found that for elastic protected plans (Kambhampati et al., 1996), the goals corresponding to machining parts of a workpiece are $\prec$*-constrained trivial serializable*, a special form of trivial serializability (Barrett and Weld, 1994). Thus, planners like SNLP that produce elastic protected plans are a good choice for problem solving in this domain (Kambhampati et al., 1996). However, problem solving still may be very difficult with SNLP. A first factor showing this difficulty is that even for elastic protected plans for machining a workpiece are nonmergeable (Kambhampati et al., 1996). We identify the high number of interactions between subplans as the main reason for this difficulty. Finally, we will show that, for sufix, prefix or sufix and prefix plans such as the ones produced by state-space planners, it is not possible to have both properties, beeing $\prec$-constrained trivial serializable and minimizing the use of the clamping and holding operations, at the same time. We conclude that for planning in this domain, SNLP is a good choice, but, external mechanisms such as EBL or CBR are required to guide SNLP in treating those interactions, and, thus, to improve the performance of the problem solving process.

This report is organized as follows. Section 2 outlines the domain of process planning for manufacturing rotary symmetrical workpieces. Section 3 presents a model for problem solving in this domain. Section 4 explains the symbolic specification of this domain. Section 5 discusses how SNLP plans with this specification. Section 6 shows that the elastic protected plans resulting from this specification are $prec$-constrained trivial serializable and minimize the number of clamping and holding operations. Section 7 shows that elastic protected plans are nonmergeable. Section 8 presents some empirical results and analyses the reasons for the difficulty of planning in this domain. Section 9 discusses related work, and, section 10 discusses the aspects of $\prec$-constrained trivial serializable and mimimality of certain key operations for prefix, sufix, and prefix and sufix plans. Finally, section 11 makes concluding remarks about this work.

2 The Domain of Process Planning for Rotary Symmetrical Workpieces

From an operational point of view, process planning can be defined as a process of determining the methods and the sequence of machining operations to produce a finished component to design specifications (Weill and Eversheim, 1982). The available resources and their operational restrictions must be considered. Resources are the cutting tools and clamping material. Figure 1 shows different cutting tools and the parts of the workpiece that they can machine.

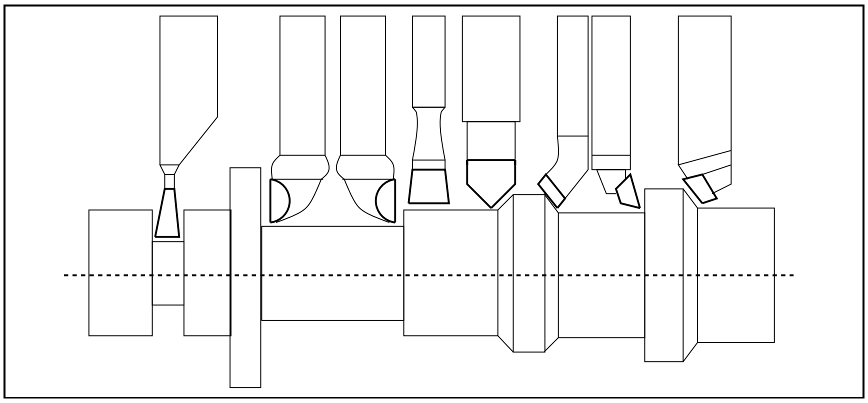

Figure 1: Cutting tools

Before we continue, the reader must take notice that the description of the domain presented in this section is neither complete nor detailfull, but, rather, it is to be introductive of the basic concepts that will be used the rest of this report.

2.1 The Rotary Symmetrical Workpieces

In this work, we concentrate on rotary symmetrical workpieces such as the one shown in figure 2. Rotary symmetrical workpieces consist of a sequence of volume elements such as cylinders, toroids, and cones. Depending on the way the volume elements are coupled, so-called *undercuts* such as the two shown in figure 2 may be formed.

Rotary symmetrical workpieces always contain two faces corresponding to the non coupled areas of the two extreme volume elements. These faces may contain *perforations* such as the one in the right face of the workpiece shown in figure 2. Additionally, different *features* may appear on the surface of the volume elements. There are different kinds of features including threats, drilled holes, fases, prick-outs and slopes (the first two are shown in figure 2).

Before the machining process begins, the descriptions of a piece of raw material and of a workpiece to be machined are given. The grid area in figure 3 corresponds to the

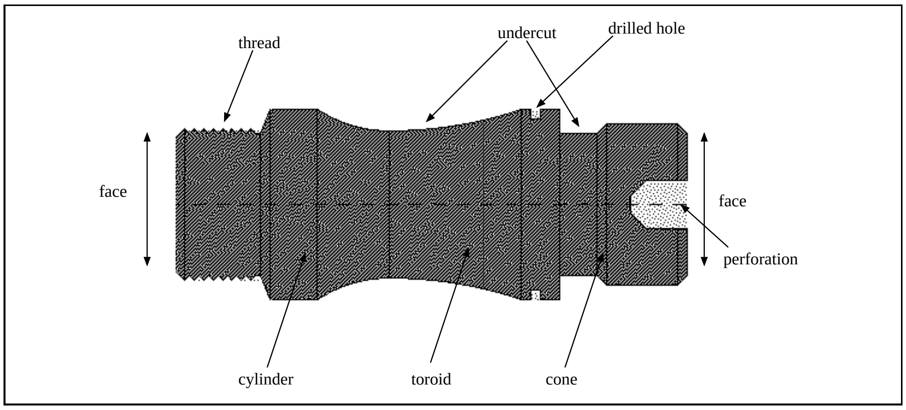

Figure 2: Example of a rotary symmetrical workpiece

section of raw material that must be removed to obtain the workpiece shown in figure 2. Based on the geometry of this workpiece, the grid area is decomposed into five *outlines*: the two sides, the left, right and center outlines. The area located between a face and the boundary of the grid area located in front of it is called a side. Rotary symmetrical workpieces always have two sides. The piece of raw material can be clamped from the two outlines neighbouring the two sides (for example, in figure 3, the left and right outlines). The center outline is the area between the two boundaries defined by extending the ends of the two volume elements that have the maximal diameter. If only one volume element has the maximal diameter, the area covering it is the center processing area. Features, undercuts, outlines and perforations are called *processing areas*. Machining processing areas different than features correspond to the removal of an area, whereas machining a feature corresponds to the removal of a substrate of an area (Karinthi et al., 1992).

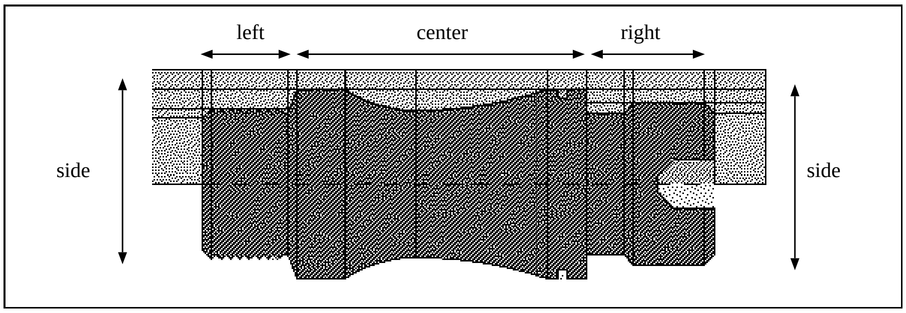

Figure 3: Example of a rotary symmetrical workpiece and of half of a raw material

2.2 Machining Workpieces

To machine a workpiece, the piece of raw material must be clamped from a certain position. Clamping positions are the two outlines neighbouring the faces and the perforations in the faces of the workpiece when they meet certain conditions. Once the piece is clamped, layers of raw material are removed by using a certain cutting tool. The process continues by changing the clamping position, changing the clamping status (i.e., rotating or not) or the cutting tool. For example, machining a thread requires the piece of raw material to be clamped to a lathe that rotates it and a cutting tool for broaching the surface, whereas machining a drilled hole requires the piece of raw material to be clamped but not to be rotated, and, a drill to be held.

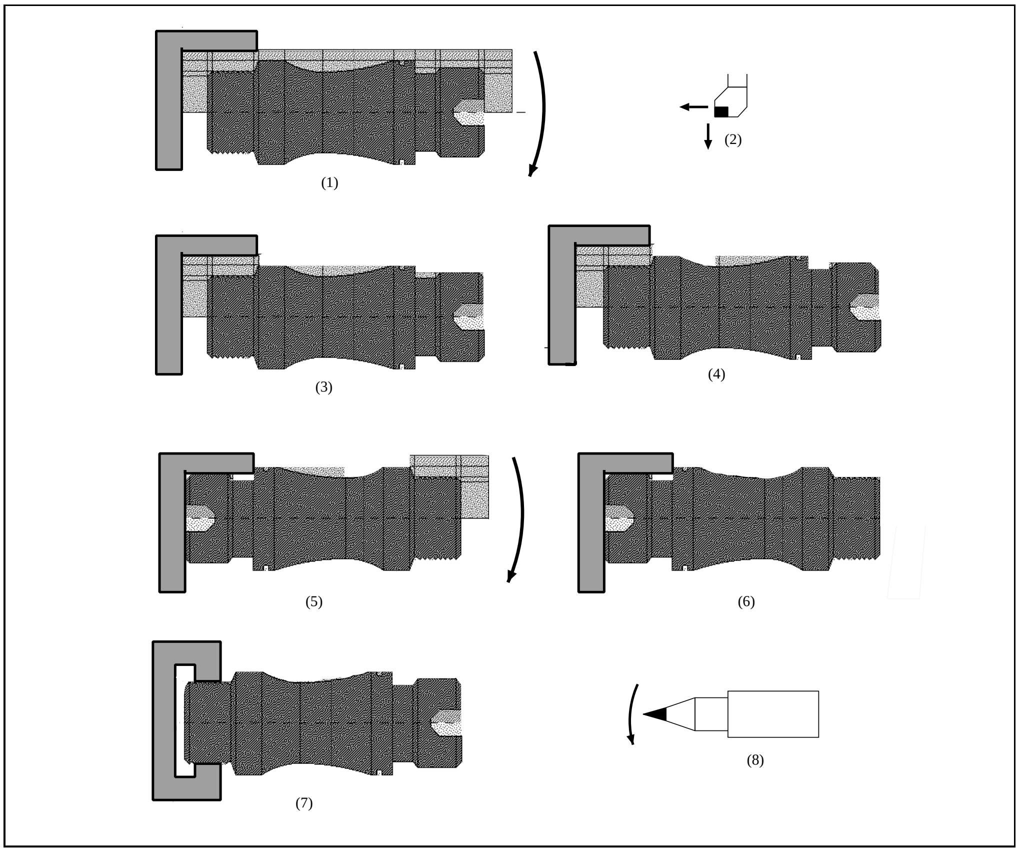

Figure 4: Parts of the machining process for the workpiece shown in figures 2 and 3.

Example of a machining process: figure 4 shows parts of a plan for machining the workpiece shown in figures 2 and 3:

(1) The piece of raw material is clamped from the clamping outline and rotated at a very high speed.

(2) A certain cutting tool is held that cuts in two directions.

(3) The center and the right outlines and the right side are removed.

(4) The left halfs of both undercuts are removed.

(5) The workpiece is clamped from the opposite outline (i.e., the right outline) to the one in operation 1 and rotated.

(6) The left outline and the other side are removed.

(7) The workpiece is clamped without rotating it from the same position as in operation 1.

(8) A rotary cutting tool is held.

To finish the machining process, the following operations are made (not shown in figure 4):

(9) The perforation is machined.

(10) A drill is held.

(11) The drilled hole is machined.

(12) The workpiece is clamped from the same position as in operaton 5 and rotated.

(13) A new tool (i.e., a broaching tool) is held.

(14) The thread is machined.

To perform this plan, four clamping and four tool holding operations are required. The clamping and holding operations are the most expensive ones during the machining process. The clamping operation is expensive because of the material required and the time needed to cool down the piece before the next clamping operation can be performed. For the same reasons, the operation of holding a cutting tool is very expensive. Thus, the number of clamping and holding operations must be minimized by maximizing the processing areas that can be machined after each clamping and holding operations have taken place, that is, the number of *setups* must be minimized.

Notice that, in the clamping operations 5 and 12, the piece is clamped from the same position and with the same status (i.e., the piece is rotating). In this situation, the clamping operations are said to be the same. In a similar way, two holding operations are said to be the same if they hold the same cutting tool.

The tolerance of the piece of raw material is another factor that must be taken into account during different tasks such as the determination of cutting times. Additionally, the tolerance within a workpiece may change. For example, the workpiece should not be clamped from a thread, because, otherwise, the thread will be damaged. Thus, if the piece of raw material must be clamped from the position where a thread is to be machine, the clamping operation must take place *before* the thread has been machined.

Other restrictions must be considered for the clamping operation which are based on the geometry of the workpiece. Depending on the form of an outline, the clamping operations can not be made when the outline has been machined. For example, a workpiece can not be clamped from a machined outline if the resulting part has a conical form.

At any time of the machining process, the workpiece is clamped from only one position and is rotating or not, and one cutting tool is held. The last restriction may be relaxed to several cutting tools at the time. However, in this report, we will restrict to one cutting tool for the sake of simplicity.

3 A Model for Problem Solving

Our model for problem solving involves a geometrical reasoner, an AI planner and a numeric control (NC) reasoner. Table 1 shows the tasks performed by each reasoner.

Task	**Geom.**	**AI**	**NC**
1. Identification of the processing areas	X		
2. Selection of machine tools		X	
3. Sequencing of tool sets		X	
4. Grouping of set-ups	X	X	
5. Sel. of machining operations and their sequence		X	
6. Sel. of clamping devices, positions, and datum	X	X	X
7. Calculation of cutting conditions			X
8. Determination of tools paths			X
9. Det. of cutting times, non mach. times, costs			X
10. NC part program generation			X

Table 1: Tasks performed by the geometrical reasoner, the AI planner and the NC-reasoner.

3.1 The Geometrical Reasoner

The geometrical reasoner identifies the processing areas (task 1 in table 1) by overlapping a model of the workpiece to be machined and of the given piece of raw material as shown in figure 3. It also selects the clamping positions and the clamping status (task 6), which determines a pregrouping of the set-ups (task 4). The geometrical reasoner produces a symbolic representation of the geometry of the processing areas. The symbolic representation includes information about positions from which the workpiece may be clamped, and, information about the occurences of certain key features, such as threads, that require an special treatment.

Additionally, the geometrical reasoner states a partial order for machining the processing areas (not directly shown in table 1 although it can be seen as a subtask of task 4). This partial order is obtained solely based on the geometrical relations between the processing areas. For example, in figure 2, the drilled hole is covered by the horizontal outline. Thus, the drilled hole can not be machined until the horizontal outline has been removed. Ordering restrictions for machining the processing areas are always stated by the geometrical reasoner in the following situations:

Feature. Every feature is covered by an outline, thus, an ordering restriction is introduced indicating that the outline should be machined before the feature. For example, for the workpiece shown in figures 2 and 3, machining the horizontal outline is ordered before the drilled hole and the undercut located under this outline.

Thread on clamping area. When a thread is covered by an outline from which the workpiece can be clamped, several ordering restrictions are introduced. As explained in section 2, the workpiece should not be clamped from this position when the thread has been machined. Thus, for the processing areas which may require the workpiece to be clamped from this position to machine them, ordering restrictions are introduced indicating that they must be machined before the thread. For example, for the workpiece shown in figures 2 and 3, machining the thread is ordered after machining the horizontal and the two ascending outlines, the right side, the two undercuts, the drilled hole and the perforation.

Unclampable area after machining. When the workpiece cannot be clamped from an outline that already has been machined, several ordering restrictions are introduced, one for each processing area which may require the workpiece to be clamped from the outline. The ordering restrictions indicate that the processing areas must be machined before the outline. Typically, these outlines have a conical form so the clamping claws are not able to grap it.

Perforations. When the workpiece contains a perforation from which the workpiece can be clamped, the machining of the perforation is ordered before all outlines excluding both sides.[1] For example, if the perforation in the workpiece shown in

[1]The workpiece is clamped from the perforation located in one of the sides and from the opposite side. This type of clamping causes an small perforation on the opposite side resulting from the rotation of the workpiece. For this reason, the opposite side should be machined after this clamping operation has taken place.

figures 2 and 3 has appropiate dimensions, machining the perforation is ordered before machining the ascending outlines, the horizontal outline, the undercuts, the thread and the drilled hole.

The symbolic description produced by the geometrical reasoner includes these ordering restrictions. But, *the domain is specified so that AI planner can always obtain plans meeting the ordering restrictions even if they are not given explicitly.* However, because this information is available before the planning process begins, it may be usefull to consider them to reduce the search effort of the AI planner. In general, although these ordering restrictions reduce the number of possible orderings to machine the processing areas, the number of orders remaining still corresponds to an exponential factor.

3.2 The AI Planner and the NC-Reasoner

The AI planner receives the symbolic representation obtained by the geometrical reasoner and the symbolic description of the available cutting tools. They constitute what we called *extended problem descriptions* (Muñoz-Avila and Hüllen, 1995; Weberskirch, 1995; Muñoz-Avila and Weberskirch, 1996), which can be seen as triples $< I, G, \prec >$, where:

1. The initial state, I, describes the geometrical relations between the processing areas obtained by the geometrical reasoner, and, the available cutting tools.
2. The goals, G, are the set of processing areas to be machined.
3. The partial order between the goals, $\prec$, obtained by the geometrical reasoner. $\prec$ *is taken by the AI planner as a partial order for achieving the goals.*

Traditionally, the AI planner receives problem descriptions consisting only of an initial state and the goals. By providing the AI planner with information restricting the possible solutions, the efficiency of the planning process can be improved. The output of the AI planner is a plan indicating

1. an ordered sequence of clamping operations,
2. the sequence of processing areas that can be machined after each clamping operation has taken place, and,
3. the cutting tools that are required to machine the processing areas in the setup defined in point 2.

This plan, which can be seen as an sketch, is completed by the NC-reasoner. The NC-reasoner, for example, computes the paths that the cutting tools must follow to machine the processing areas (task 8).

3.3 Discussion on our Model for Problem Solving

Other authors have proposed a more flexible integration between the domain specific reasoners as opposed to the sequence proposed here (Kambhampati et al., 1991). Under this approach, the three reasoners should perfom the same tasks as indicated in table 1, but, new requirements are given on the reasoners and complex interfaces between the reasoners are also required. Particularly, the AI planner should be capable of dealing with dynamic changes in the conditions of the problem.[2] However, in practice, performing such an integration is very difficult. Most CAPP systems achieve the tasks indicated in table 1 sequentially. By introducing an AI planner we are considering interactions between tasks 2 to 7, which we believe are key in the manufacturing process and are the most suited tasks to be performed by a symbolic planner.

Another important point is that the results on trivial serializability and mergeability of the plans are independent of the fact that the three reasoners work sequentially (see sections 6.3 and 7). As we will see later, these results depend solely on the tasks that, according to our model, the AI planner must perform and on the particular symbolic representation of these tasks.

4 The Symbolic Specification of the Domain

In our approach, the symbolic representation of the domain is decomposed into three parts: types, predicates and operators (Weberskirch, 1995). The complete specification is shown in appendix C. The basic representation units are the objects, which can be of different types.

Object Types In our domain specification, 33 types of objects are organized in hierarchies representing the type-subtype relation. At the top level, there are two categories:

Processing area. Its subtypes include outlines, features, undercuts, and internal processing areas. The internal processing areas are the perforations that may appear on the faces of the workpieces.

Tool. Its subtypes are tools for machining the different processing areas.

[2]In fact, the AI planer CAPlan (Weberskirch, 1995) developed as part of our project is capable of dealing with dynamic changes without having to recompute the plan from the scratch every time a condition is changed. The reason for CAPlan's capabilty to handle the so-called contingencies (Petrie, 1991) is that it is built on the REDUX architecture (Petrie, 1992) that was designed handle them.

Predicates The predicates indicate relations between objects. Our specification includes the following classes of relations, although, in contrast to the type hierarchy, this classification is not represented explicitly:

Geometrical relations. For example, the predicate *subarea(thr1,hor)* indicates that the thread *thr1* is covered by the horizontal outline *hor*.

State of an area. An area can be machined or not machined. This is represented by the unary predicates *processed* and *unprocessed* respectively.

Availability and state of a cutting tool. The former is represented by the unary predicate *available* and the latter by the unary predicate *toolHeld*. The 0-ary predicate *toolHolderFree* indicates that no cutting tool is held.

Clamping relations. There are predicates indicating that the workpiece can be clamped from a certain processing area (i.e., the unary predicate *isClampArea*). Other predicates precise whether the workpiece can be clamped from an area after it has been machined (i.e., the unary predicate *unrestrictedClamp*) or not (i.e., the unary predicate *restrictedClamp*). Finally, the predicate *clampTurn* indicates that the workpiece is clamped and that the workpiece is rotating. The predicate *clampNoTurn* indicates that the workpiece is clamped but that the workpiece is not rotating.

Operators In our domain specification, the 27 operators represent operations taking place during the machining process. They can be classified as follows (again, this classification is not represented explicitly):

Processing operators. They represent actions for machining parts of a workpiece, *MachineThread* for machining a feature, *MachineOutline* for machining an outline, *MachineUndercut* for machining an undercut, or *MachineInternalArea* for machining a perforation.

Clamping operators. They represent actions for clamping a workpiece. Our specification includes three kinds of clamping operations: clamping from an outline and rotating the workpiece, such as *ClampTurn*, clamping on an outline without rotating the workpiece, such as *ClampNoTurn*, and, clamping from a perforation, such as *ClampFromFace*, which always turns the workpiece.

Holding Operators. They rationalize the use of the cutting tools. We suppose that only one tool can be held at a time. To model this restriction, our specification includes two actions: *HoldTool* and *MakeToolHolderFree*.

Our STRIPS-like operators (Fikes and Nilsson, 1971) are constituted of arguments, constraints, preconditions and effects:

Arguments. The arguments list all variables used in the definition of the operator.

Constraints. The constraints are the binding constraints on the variables. We used four classes of constraints (Weberskirch, 1995): *IsOfType(<type>,<var>)*, *IsNotOfType(<type>,<var>)*, *Same(<var>,<var'>)* and the constraint *Not-Same(<var>,<var'>)* indicating that the binding of the variable *<var>* must (not) be of type *<type>* or that the variable *<var>* must (not) be instantiated to the same object as the variable *<var'>*, respectively.

Preconditions. The preconditions of an operator list the conditions that must hold between the arguments of the operator before the operator can be executed. A condition has the form: *+predicateName(<var-1>, ..., <var-n>).*

Effects. The effects of an operator list the changes to the world that the action causes. These changes have the form *+predicateName(<var-1>, ..., <var-n>)*, if the effect is in the add-list, and, *-predicateName(<var-1>, ..., <var-n>)*, if the effect is in the delete-list. That is, the effect is to be added or deleted from the current state.

Figure 5 shows the definition of the clamping operator $ClampTurn$ and of the holding operators $HoldTool$ and *MakeToolHolderFree*.

Important: At any time of the machining process the workpiece must be clamped, at most, from one place and, at most, one cutting tool must be held. However, the modelling of the clamping and holding operations is quite different in our specification:

- **Clamping operations are specified explicitly.** Workpieces always have a limited and predefined number of clamping operations that may be applicable.[3] Workpieces can be clamped at most from four clamping positions (i.e., the two outlines neighbouring each side of the workpiece, and, the perforations that may occur on the sides), and they are to be rotated or not. Thus, in our model, the clamping operators always achieve a certain clamping operation by stating it in the add-list and exclude the other operations by stating them in the delete list (see, for example, the operator *ClampTurn* in figure 5).

- **Holding operations are modelled implicitly.** In contrast to the clamping operations, it is not possible to state all the possible cutting tools that can be held because the cutting tools that are available vary from problem to problem. Even when considering the fact that, in our current model, only one tool can be held at a time, operators cannot exclude all other possibilities explicitly because several tools of the same type might be available. For this reason, an intermediate operator, *FreeToolHolder*, is introduced (see figure 5). In this

[3]Depending on the particular workpiece, some of the clamping operations are not be applicable.

Operator: *ClampTurn*
Arguments:
outl1, s1, outl2, s2
Constraints:
IsOfType(Outline,outl1)
IsOfType(Outline,outl2)
IsOfType(WpieceSide,s1)
IsOfType(WpieceSide,s2)
NotSame(outl1,outl2)
NotSame(s1,s2)
Effects:
+clampTurn(outl1)
−clampNoTurn(outl1)
−clampTurn(outl2)
−clampNoTurn(outl2)
−clampTurn(s1)
−clampTurn(s2)
Preconditions:
+noSubareaThread(outl1)
+isClampArea(outl1)
+isClampArea(outl2)
+unrestrictedClamp(outl2)

Operator: *HoldTool*
Arguments:
tool
Constraints:
IsOfType(Tool,tool)
Effects:
+toolHeld(tool)
−toolHolderFree()
Preconditions:
+toolHolderFree()

Operator: *MakeToolHolderFree*
Arguments:
tool
Constraints:
IsOfType(Tool,tool)
Effects:
+toolHolderFree()
−toolHeld(tool)
Preconditions:
+toolHeld(tool)

Figure 5: Definition of clamping and holding operators

way, we cope the inability to model the undefined number of tools available. Additionally, our specification can be easily extended to hold multiple tools working at the same time. Notice that this is not possible for the clamping operation: once a workpiece is clamped from a position and a clamping status is fixed (for example, rotating), it makes no sence, for example, to clamp the workpiece from other positions at the same time.

In our symbolic description of the domain of process planning, we have modelled both, resources that are known previously, such as the clamping operations, and, resources that are not, such as the available cutting tools. In this way we cope a wide spectrum of possibilities relative to the tasks that, according to our model for problem solving, the AI planner must accomplish.

5 Systematic Nonlinear Planning in the Domain of Process Planning

We will now explain how systematic nonlinear planning is done with our specification of the domain of process planning for rotary symetrical workpieces, but first the basic concepts are introduced.

5.1 Systematic Non Linear Planning (SNLP)

SNLP (McAllester and Rosenblitt, 1991) is a planning algorithm that refines partially ordered plans. A partially ordered plan can be viewed as a 3-tuple $<S, L, B>$, where S is the set of *plan* steps (each of which is associated with an operator in the domain), L is a set of *links* indicating an order for executing steps in S, and, B is a set of *constraints* on the variables bindings.[4] *The execution order L induces a partial order,* $<_L$*, by taking the transitive clousure of L.*

Once a planning problem is given, a so-called initial plan representing the planning problem is constructed. The initial plan contains two artificial plan steps, *start* and *finish*, a single link from *start* to *finish* and no variable bindings. The step *start* contains no preconditions and has as effects the initial state of the problem whereas the step *finish* contains no effects has as preconditions the final state of the problem.

Planning proceeds by *establishing open conditions* and *resolving conflicts*. An open condition is a precondition such that there is no link from an effect of an step to the precondition. We write $p@s$ to denote that p is a precondition of s. A link can be introduced when the effect of a plan step unifies the open precondition modulo the binding constraints in B. When such a link is introduced, the precondition is said to be established by the plan step. These kinds of links are called *causal links*. We write $s_1 \rightarrow p@s_2$ to denote that the precondition $p@s_2$ is established with s_1. We distinguish between the case in which s_1 was already in S and the case in which s_1 is a new step (i.e., not in S). In the first case, the establishment is said to be *simple* whereas, in the second, the precondition is said to be *established with a new step*. When the establishment of the precondition is simple, the constraints making the precondition and the effect unifiable are added to B. When a precondition is established with a new step, the constraints of the corresponding operator are added to B.

A conflict or *threat* to an established precondition $s_1 \rightarrow p@s_2$ is caused by a third plan step s_3 that has as effect p or $\neg p$ and that is parallel to $s_1 \rightarrow p@s_2$, that is, the following condition does not hold: $s_3 <_L s1$ or $s_2 <_L s3$ (see figure 6).

[4] Usually, a partially ordered plan is represented as a 4-tuple $<S, O, B, L>$, where O are the ordering constraints in the plan and L are the causal links, capturing causal dependencies between steps. The ordering constraints induced by L are also contained in O. In this paper, we consider the ordering constraints as links and unite them with the causal links in a single set, L.

If the effect of s_3 is p, in which case we write $s_3 \rightarrow p$, the threat is said to be *positive* and write $s_3 \overset{+}{\longleftrightarrow} (s_1 \rightarrow p@s_2)$, otherwise, it is said to be *negative* and write $s_3 \overset{-}{\longleftrightarrow} (s_1 \rightarrow p@s_2)$. SNLP solves negative and positive threats. Whereas negative threats are solved to ensure the consistency of the plan, positive threats are solved to ensure the systematicity of the planning process, which can reduce the size of the search space (Kambhampati, 1993). Threats are solved by

- introducing the link $s_3 \longrightarrow s_1$, called a *protection link*, in L (this operation is called a *demotion*),
- introducing the link $s_2 \longrightarrow s_3$, also called a *protection link*, in L (this operation is called a *promotion*), or,
- introducing bindings constraints in B such that the precondition p is not unifiable with the conflicting effect of s_3 (this operation is called a *separation*).

Additionally, no step can be ordered before *start* or after $finish$. Thus, for example, if s_1 is *start* demoting s_3 is not a valid alternative to solve the threat.

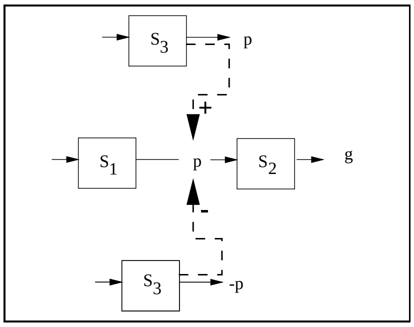

Figure 6: Graphical representation of the positive threat $s_3 \overset{+}{\longleftrightarrow} (s_1 \rightarrow p@s_2)$ and the negative threat $s_3 \overset{-}{\longleftrightarrow} (s_1 \rightarrow p@s_2)$

At every step of the planning process, partially ordered plans are refined by introducing new links, steps, or ordering constraints. The planning process is finished if the initial plan is refined to a plan containing no open conditions and no threats. This plan corresponds to a solution of the planning problem and we say that this plan *achieves the goals*. SNLP has been shown to be complete (McAllester and Rosenblitt, 1991), that is, a solution is found if there is one, otherwise, a failure is returned.

5.2 Classification of Plans

This section summarizes some notions that are used in the following sections and taken from (Kambhampati et al., 1996).

Plans in which the steps are ordered by precedence relations are said to be *elastic*. Plans including *interval preservation constraints* (IPC) are said to be *protected*. IPC are 3-tuples of the form $s_1 \stackrel{p}{-} s_2$ indicating that the condition p must be preserved between s_1 and s_2.

It is easy to see that the plans produced by SNLP are elastic and protected. They are elastic because their steps are partially ordered by $<_L$ and protected because causal links capture causal dependencies between steps. Thus, they correspond to the interval preservation constraints.

Plans produced by state-space planners can be *sufix*, *prefix*, or both. In a *sufix* plan, the plan steps are contiguous to each other in a chain whose last element is an dummy step representing the final state. In the same way, in a *prefix* plan, the plan steps are contiguous to each other in a chain whose first element is an dummy step representing the initial state.

5.3 Planning for Machining Workpieces with SNLP

The set of open conditions of the initial plan (i.e., the preconditions of the artificial plan step *finish*) is constituted by the processing areas to be machined. In the same way, the effects of the artificial step *start* correspond to the description of the geometrical relations between the processing areas and to the information concerning the availability of the cutting tools (see section 3).

For establishing an open condition corresponding to the goal of machining a processing area, a plan step indicating a machining operation is introduced (see section 4). Typically, these steps have two preconditions: to clamp the workpiece and to hold a certain tool. These preconditions are established by using clamping and holding steps as shown in figure 7. This plan contains three plan steps, to machine the center outline, ms_{ctr}, to hold a tool, hs, and to clamp the workpiece, cs, from the left outline *outl-L*. It also contains four causal links representing the following establishments: $ms_{ctr} \rightarrow machined(ctr)@finish$, $cs \rightarrow clampFrom(outl{-}L)@ms_{ctr}$, $hs \rightarrow toolHeld(t{-}ctr)@ms_{ctr}$, and, $start \rightarrow toolHolderFree@hs$.

In our specification, preconditions of a plan step can be:

- the processing areas to be machined, such as *processed(ctr)* in figure 7,
- the processing areas from which the workpiece is to be clamped, such as *clampTurn(outl-L)*, and,
- the cutting tools to be held, such as *toolHeld(t-ctr)*.

Although the scheme to machine a processing area is essentially the same as the one shown in figure 7, search must be done to bind different variables such as the tool and

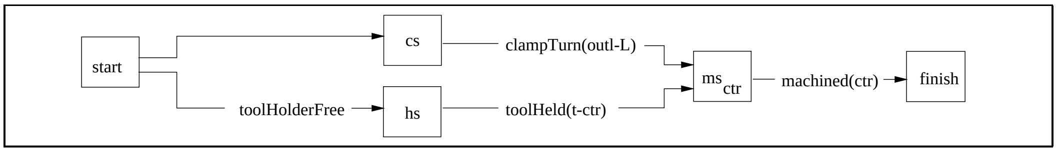

Figure 7: Plan for machining the center outline, *ctr*

the clamping position, and, to select the clamping operation. Thus, depending on the conditions of the problem, backtracking may be needed, not only to bind other values to the variables or to select other clamping operations, but, to select another machining operation if the selected one cannot be applied due to the particularities of the problem. Some processing areas, such as the undercuts, have several machining operators indicating different conditions that the cutting tools and the clamping positions must meet to machine an undercut.

Threats occur when two or more processing areas are to be machined and they result from the operations for clamping the workpiece and for holding a tool. For example, machining an undercut requires a left or right rotary cutting tool whereas machining a thread requires a broaching tool. In this situation, negative threats occur because only one tool can be held at the time. This situation is typical for real-world problems in which several processes require more resources than there are available. Thus, the use of the resources needs to be rationalized.

Another example for an occurrence of negative threats takes place when machining two processing areas, each of which requires the workpiece to be clamped from a different position. Notice, that these threats are not the result of the use of external resources, but, rather, of operational restrictions (i.e., in this situation, a workpiece can only be clamped from one position and is rotating or not at any time of the machining process).

5.4 Properties of Holding and Machining Steps

Although solution plans $<S, L, B>$ for machining a workpiece are partially ordered, the subsets of S formed by the clamping and holding steps are both totally ordered w.r.t. to the partial order $<_L$ induced by L as we will now see.

Two clamping operations are the same if they clamp the workpiece from the same position and with the same clamping status. Symbolically, two clamping steps represent the same clamping operation if the added effect of their corresponding clamping operator is the same (see figure 5). For the sake of simplicity, we will assume that the goals corresponding to clamp the workpiece require always the workpiece to be rotated.

Lemma 1 *Let* $<S, L, B>$ *be a plan were all the preconditions of the machining steps* *corresponding to clamping the workpiece have been established and all threats have been solved. Then, the following affirmations are valid:*

1. *The clamping steps in S are totally ordered in a chain of the form: $cs_1 <_L MS_1 <_L cs_2 <_L ... <_L cs_n <_L MS_n$, where cs_i is a clamping step and MS_i is the set of machining steps, ms_j, in S such that clampTurn(t_j)@ms_j is established with cs_i (with $1 \leq i \leq n$).*[5]

2. *New clamping steps can be ordered anywhere in the chain $cs_1 <_L cs_2 <_L ... <_L cs_n$.*

3. *A new pair of clamping-machining steps, cs, ms with $cs \rightarrow clampTurn(pos)@ms$, can be ordered between MS_i and cs_{i+1} (with $1 \leq i \leq n-1$), after MS_n, or before cs_1.*

Proof. See appendix A.

This lemma shows that our model of the clamping operations is correct because, at any time of the machining process, the workpiece is clamped at most from one position and with one clamping status. The total order of the clamping operations is due to the fact that clamping operators exclude explicitly the other clamping possibilities to occur at the same time.

As explained before, holding operations are modeled differently. Thus, the argument for showing that the holding steps are also totally ordered w.r.t. the order induced by L, $<_L$, is also different. In what follows, we will assume that the tool holder is initially free, that is, the condition *toolHolderFree* is an effect of *start*.

Two holding operations are the same if the same cutting tool is being held. Symbolically, two holding steps are recognized to represent the same clamping operation if the added effect of their corresponding holding operator is the same (see figure 5).

Lemma 2 *Let $<S, L, B>$ be a complete plan containing only holding and MakeToolHolderFree steps.*[6] *Then, the following affirmations are valid:*

1. *The steps in S are totally ordered in a chain of the form: $hs_1 <_L mthf_1 <_L hs_2 <_L ... <_L hs_n$, where $mthf_i, hs_i$ are MakeToolHolderFree and holding steps respectively. Further, $mthf_i$ establishes the precondition $toolHolderFree@hs_{i+1}$ and hs_i establishes the precondition $toolHeld(t_i)@mthf_i$ with $1 \leq i \leq n-1$.*

[5] $cs_i <_L MS_i$ indicates that $cs_i <_L ms_j$ for all ms_j in MS_i.

[6] This plan may be obtained by taking a complete plan to machine parts of a workpiece and removing all non *Holding* and non *MakeToolHolderFree* steps (with the exception of *start* and *finish*) and all links that are not from and to a holding step, a MakeToolHolderFree step, *start* or *finish*.

2. *If a new holding step is introduced then a new MakeToolHolderFree step must be introduced as well.*

3. *The new MakeToolHolderFree step is appended to the end of the chain of holding steps in S when the plan* $<S, L, B>$ *is completed to achieve all open conditions. Further, the precondition corresponding to hold a tool of the new MakeToolHolderFree step is achieved by the last holding step in* S, hs_n.

Proof. See appendix B.

This lemma shows that our specification correctly models the restriction of the domain that at most one cutting tool must be held at any time of the machining process.

5.5 Minimizing the Number of Clamping and Holding Steps

The standard strategy of SNLP for plan refinement pursues to establish open conditions first by performing simple establishments and only if this is not possible, by performing establishments with new plan steps (McAllester and Rosenblitt, 1991). This means that steps will only be introduced if there are no existing steps that establish the open condition, or, if using the existing steps results in an inconsistency. As a result, in the next section we will show that the plans produced by SNLP minimize the number of clamping and holding operations. For example, for the workpiece whose machining plan is sketched in figure 4, SNLP always produces a plan consisting of two clamping steps and two holding steps.

Notice that minimizing the number of clamping and holding steps does not exclude plans containing two different plan steps corresponding to the same clamping operation (i.e., the same clamping position and the same clamping status) or to the same holding operation (i.e., holding the same cutting tool). For example, the machining process of a workpiece presented in section 2 requires the same clamping operation twice (steps 5 and 12) although it minimizes the number of clamping and holding operations required to machine the workpiece.

The fact that the elastic protected plans contain non redundant clamping and holding operations is important for real-world solutions. This *suboptimality* of the machining plans is also the result of our model for problem solving, in which the AI planner must achieve tasks 2-6 (see table 1), and, supports our statement that these tasks are the most suited to be performed by a symbolic planner (see section 3), particularly, the selection of machining tools (task 3) and the selection of machining steps and their sequences (task 5).

6 $\prec$-Constrained Trivial Serializability of Machining Goals and Suboptimality of Plans

Related to the efficiency of the planning process, an important question to be answered is if plans in a certain class of plans are trivial serializable (Barrett and Weld, 1994; Kambhampati et al., 1996). If the answer is affirmative, the performance of the planning process will be improved when using algorithms planning in this class (Kambhampati et al., 1996). In this section, we will show that planning with SNLP and our specification of the domain of process planning for rotary symmetrical workpieces results in plans that have the property of being $\prec$-*constrained trivial serializable*, a special form of trivial serializability. Further, the resulting plans contain a minimal number of clamping and holding steps. Thus, we conclude that elastic protected plans for machining workpieces are $\prec$-*constrained trivial serializable* and minimize the number of clamping and holding operations.

6.1 Trivial Serializability

This section summerizes some definitions presented in (Kambhampati et al., 1996). Consider the plan for machining the center outline shown in figure 7, and, suppose that an additional processing area, for example a drilled hole, is to be machined. If the plan can be extended to machine both processing areas, the plan is said to be serially extensible with respect to the goal corresponding to machining the drilled hole.

Definition 1 (Serial Extensibility) *Given a class* $\mathbf{P}$ *of plans, a plan* P *in* $\mathbf{P}$ *achieving a goal* g_1 *is serially extensible with respect to a second goal* g_2 *if there is a refinement of* P *in* $\mathbf{P}$ *achieving both* g_1 *and* g_2*.*

Notice that backtracking may take place for achieving g_1 and g_2. What the definition of serial extensibility says is that no backtracking should take place in the plan refinement steps introducing steps, links or constraints in P.

In the same situation as before, if *any* plan for machining the center outline is serially extensible with respect to the goal corresponding to machining the drill, then, the order, first plan for machining the center outline and then for machining the drill, is called a serialization order.

Definition 2 (Serialization Order) *Given a class* $\mathbf{P}$ *of plans and a set of goals,* $g_1, g_2, ..., g_n$*, a permutation* π *on these goals is said to be a serialization order if every plan in* $\mathbf{P}$ *for solving* πg_1 *can be serially extended to* πg_2 *(modulo the class* $\mathbf{P}$*) and any resulting plan can be serially extended to* πg_3 *(modulo the class* $\mathbf{P}$*) and so on.*

Unfortunately, serial extensibility is not a commutative property (Barrett and Weld, 1994; Kambhampati et al., 1996). Thus, if the order, first plan for machining the center outline and then for machining the drill, is a serialization order, it does not necessarily mean that the opposite order is. If this is the case, the goals are said to be trivial serializable.

Definition 3 (Trivial Serializability) *A set of goals, $g_1, g_2, ..., g_n$, is said to be trivial serializable, if any permutation on these goals is a serialization order (modulo a class of plans* **P***).*

6.2 $\prec$-Constrained Trivial Serializability

As explained in section 3, the AI planner solves extended problem descriptions, that is, problems consisting of an initial state, a set of goals, and, optionally, $\prec$, a partial order for achieving the goals. $\prec$ is intended to be an order for achieving the goals in the plan. Thus, we will consider only permutations of goals that extend $\prec$.

Definition 4 ($\prec$-Consistent Permutation) *A permutation π on a partially ordered set of goals, $((g_1, g_2, ..., g_n), \prec)$, is said to be $\prec$-consistent, if, for any two goals, g_i, g_j, such that $g_i \prec g_j$, then, πg_i is ordered before πg_j.*

Accordingly, we will extend the definition of trivial serializability to consider only permutations that are $\prec$-consistent. In this way, we exclude from consideration permutation orders of the goals that are not to be followed by the AI planner.

Definition 5 ($\prec$-Constrained Trivial Serializability) *A partially ordered set of goals, $((g_1, g_2, ..., g_n), \prec)$, is said to be $\prec$-constrained trivial serializable, if any $\prec$-consistent permutation on these goals is a serialization order (modulo a class of plans* **P***).*

Notice that, *for deciding if a set of goals is $\prec$-constrained trivial serializable, the number of permutations required to be serially extensible is less than the number of permutations required for deciding if they are trivial serializable.* The reduction in the number of permutations to be considered is directly proportional to the degree of ordering defined by $\prec$. In the limit, if the set of goals is totally ordered, only one permutation, the one defined by $\prec$, must be inspected to decide if they meet this property. The following lemma can now be stated:

Lemma 3 *Let $((g_1, g_2, ..., g_n), \prec)$ be a partially ordered set of goals. If $g_1, g_2, ..., g_n$ is trivially serializable, then $((g_1, g_2, ..., g_n), \prec)$ is $\prec$-constrained trivially serializable.*

6.3 $\prec$-constrained Trivial Serializability of Machining Goals

Let $< I, G, \prec >$ be an extended problem description corresponding to the problem of machining a workpiece. Thus, the goals in G correspond to the processing areas of the workpiece to be machined. Consider a partially ordered plan $<S, L, B>$ achieving a proper subset of goals G' of G and a goal, g, in $G - G'$. Notice that there is no goal g' in G' with $g \prec g'$, as we defined $\prec$ to be a preference order for achieving the goals in the plan. We will now illustrate why SNLP can serially extend $<S, L, B>$ to solve $G' \cup \{g\}$.

For achieving g, a new machining step, ms, is introduced. No step in S can achieve g because, in our specification, a machining operator can only achieve one goal corresponding to machining a single processing area, and, because SNLP ensures that $<S, L, B>$ is non redundant, there are no "superfluous" plan steps in S. That is, there are no more machining steps in S than necessary (i.e., one for each goal in G'). Machining operators indicate the conditions that the clamping position must meet and the clamping status. They also indicate, the type of cutting tool that must be held. In what follows, we will suppose that the open conditions introduced by new machining step ms are to clamp the workpiece with status rotating, $clampTurn(pos)$, and, to hold a certain kind of cutting tool, $toolHeld(t)$. Notice that pos and t are not determined initially although their possible values may have been constrained.

We will suppose that the open condition for clamping the workpiece is established first, and then the one for holding a cutting tool. The argumentation when these open conditions are achieved in the opposite order is analogous.

6.3.1 Trivial Serializability of the Subgoals for Clamping the Workpiece

Suppose that there is a clamping step, cs_i, in S that can be used to establish *clamp-Turn(pos)@ms*. No threats can occur with clamping steps, cs_j, such that $cs_j <_L cs_i$. If $cs_i <_L cs_j$, then there is a threat $cs_j \overset{+/-}{\longleftrightarrow} (cs_i \rightarrow clampTurn(pos)@ms)$ (see figure 8). Separation cannot be performed. The only way of solving all these threats is to promote cs_{i+1} because, once cs_{i+1} is promoted, $ms <_L cs_{i+1} \leq_L cs_j$ for all cs_j with $cs_i <_L cs_j$.

If there is no step in S that can establish the open condition $clampTurn(pos)@ms$, or, if all clamping steps in S have been discarded after backtracking, a new clamping step, cs, is introduced. Lemma 1 shows that the cs and ms can be intercalated between the clamping steps in P'.

We can now state the following proposition:

Theorem 1 *Subgoals corresponding to clamping a workpiece are trivially serializable modulo the class of elastic protected plans for machining workpieces.*

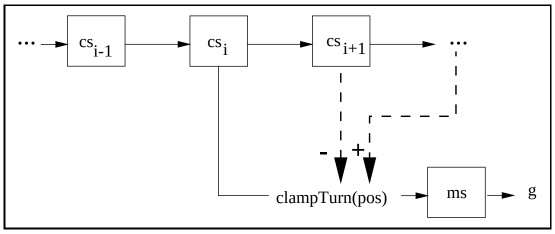

Figure 8: Threats occuring when establishing the open condition $clampTurn(pos)@ms$ by using an step in S

The reason for this result is that clamping operators exclude explicitly the other clamping possibilities to occur at the same time. As a result threats caused by establishing the precondition of a new machining step corresponding to clamping the workpiece are solved by introducing appropiate ordering constraints. These ordering constraints result in an intercalation of the new clamping step with respect to the existing clamping and machining steps in the plan.

As we will know see, the way the holding operations was modelled results in a different strategy for solving the threats caused when establishing the precondition corresponding to hold a tool, $toolHeld(t)$.

6.3.2 $\prec$-Constrained Trivial Serializability of the Subgoals for Holding Tools

Suppose now that the open condition corresponding to clamping the workpiece has been established by using a clamping step cs_j, possibly in S. To establish the open condition *toolHeld(t)*@ms, SNLP first tries to use an adequate holding step in S.

(a) Suppose that there is a holding step, hs_i, in S that establishes the open condition *toolHeld(t)*@ms. Thus, $hs_i <_L ms$. Additionally, we know that $cs_j <_L ms$ and that $ms <_L cs_{j+1}$, if cs_j is not the last clamping step. Two alternative situations with respect to the plan step $mthf_i$ can be distinguished:

- $mthf_i <_L cs_j$: If $mthf_i <_L cs_j$ then $mthf_i <_L ms$ because $cs_j <_L ms$. Thus, we have a threat $mthf_i \stackrel{-}{\longleftrightarrow} (hs_i \rightarrow toolHeld(t_i)@ms)$ because $hs_i <_L mthf <_L ms$. This threat can be solved neither by separation because $mthf_i \rightarrow \neg toolHeld(t_i)$ nor by promotion or demotion because it results in a contradiction. For example, if $mthf_i$ is promoted, then $ms <_L mthf_i$. Thus, the plan cannot be completed and the establishment with hs_i must be rejected.
- $mthf_i \not<_L cs_j$: For each $mthf_l$ such that $hs_i <_L mthf_l$ (i.e., $i \leq l$) and $cs_{j+1} \not<_L mthf_l$ if cs_j is not the last clamping step, $mthf_l \stackrel{+/-}{\longleftrightarrow} (hs_i \rightarrow toolHeld(t)@ms)$

because $hs_i <_L mthf_l$ and $ms \not<_L cs_{j+1}$ ($ms <_L cs_{j+1}$ and $cs_{j+1} \not<_L mthf_l$). This situation is illustrated in figure 9. These threats are solved by promoting $mthf_l$, that is, by introducing the ordering constraints $ms <_L mthf_l$. In particular, for $l = i$ we have $hs_i <_L ms <_L mthf_i$ and we know that for $l > i$ $mthf_i < mthf_l$. That is, these threats are solved by ordering the machining step ms before the step that dismounts the required tool, $mthf_i$.

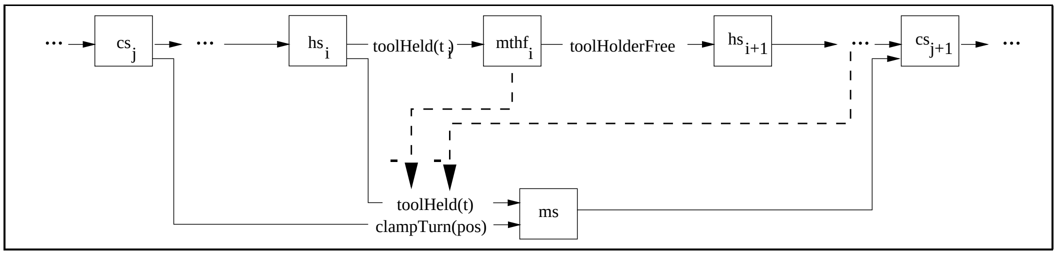

Figure 9: Threats occuring when establishing the open condition *toolHeld*(t)@ms by using an step in S

This means that if S contains adequate holding and clamping steps, the plan $<S, L, B>$ for achieving G' can be serially extended with respect to g.

(b) If there is no step in S that can establish the open condition *toolHeld*(t), a new holding step, hs, is introduced. Lemma 2 shows that a new *MakeToolHolderFree* step must be introduced and that the precondition *toolHeld*(*pos*)@$mthf$ must be established by the last holding step in S, hs_k. Two cases may be distinguished:

- ms is ordered between two clamping steps, cs_j and cs_{j+1}. There must be a machining step, ms_m, whose precondition *clampTurn*(*pos*$'$)@ms_m is established with cs_{j+1} because $<S, L, B>$ contains no redundant steps. Additionally, because $<S, L, B>$ is complete and lemma 2, toolHeld(t')@ms_m must be established with hs_k. Thus, $mthf \overset{+/-}{\longleftrightarrow} (hs_k \rightarrow toolHeld(t_k)@\mathrm{ms}_m)$ because $hs_k <_L mthf$ and $mthf$ is unordered relative to ms_m. This threat can be solved neither by separation because $mthf \rightarrow -toolHeld(t_k)$ nor by promotion or demotion because otherwise a contradiction occurs. For example, if $mthf$ is demoted, then $mthf <_L hs_k$.

- ms is ordered after the last clamping step, $cs_{n'}$ ($n' = n$, if no additional clamping step has been introduced, or, $n' = n + 1$, otherwise). Suppose, for the sake of simplicity, that only one machining step, ms_m, occur after hs_k. We have $hs_k <_L ms_m$ and $hs_k <_L mthf <_L hs$, thus, the threat $mthf \overset{+/-}{\longleftrightarrow} (hs_k \rightarrow toolHeld(t_k)@ms_m)$ occurs. No other threat occurs because ms is totally ordered w.r.t the clamping steps ($cs_{n'} <_L ms$) and $mthf$, hs and

ms are totally ordered w.r.t the holding steps ($hs_k <_L mthf <_L hs <_L ms$). This threat cannot be solved by separation because $mthf \rightarrow -toolHeld(t_k)$. Demoting $mthf$ (i.e., forcing $mthf <_L hs_k$) results in a contradiction because $hs_k <_L mthf$. Promoting $mthf$ solves the threat (i.e., forcing $ms_m <_k mthf$) and results in the following order: $hs_k <_L ms_m <_L mthf <_L hs <_L ms_m$.

6.4 Main Results on Trivial Serializability and Suboptimality of Plans

In *summary*, the strategy resulting from using SNLP to plan the additional goal, g, corresponding to machining a processing area can be described as follows: SNLP first tries to use adequate clamping and holding steps in S. If this fails, SNLP introduces a new clamping step and tries to intercalate it so that an existing holding step in S can be used. If this fails too, new clamping and holding steps that are required to achieve g are introduced and linked to the end of the plan. We can now state the following proposition:

Theorem 2 *Plans to machining workpieces produced by SNLP are in the class of machining plans that minimize the number of clamping and holding steps.*

The last alternative for solving the additional goal resulting from the use of SNLP, i.e., linking new clamping and holding steps to the end of the plan, supposes that planning for each processing areas can always be performed in any order. However, this is not always the case as the following example shows: suppose a problem consisting of machining a workpiece that contains a thread, thr, covered by the center outline, ctr. Suppose that SNLP begins by planning for achieving thr. A machining step, ms_{thr}, for the thread is introduced. This step contains contains three preconditions (see operator $ProcessThread$ in appendix 1): to hold a cutting tool for threads, *toolHeld(t-thr)*$@ms_{thr}$, to clamp the workpiece, $clampTurn(pos)@ms_{thr}$, and to machining the covering outline, $machined(ctr)@ms_{thr}$. SNLP can pursue to achieve these preconditions in any order. Figure 10 shows a partial solution when *toolHeld(t-thr)*$@ms_{thr}$ and $clampTurn(pos)@ms_{thr}$ have been achieved before $machined(ctr)@ms_{thr}$, which has not been completely achieved as the precondition $toolHolderFree@hs_{ctr}$ has not been established yet. Additionally, threats resulting from solving $machined(ctr)@ms_{thr}$ have not been solved. Notice that, for achieving $machined(ctr)@ms_{thr}$, a new holding step, hs_{ctr}, has been introduced.

Lemma 2 states that the only possible solution for the threats caused by the new holding step hs_{ctr} is to link hs_{ctr} to the end of the plan. However, because ms_{thr} is ordered after ms_{ctr} (i.e., $ms_{thr} <_L ms_{ctr}$), this solution introduces a cycle in the plan. Thus, SNLP backtracks on the subplan for achieving *toolHeld(t-thr)* and $clampTurn(pos)$. Particularly, it rejects the establishment of the open condition $toolHolderFree$ of

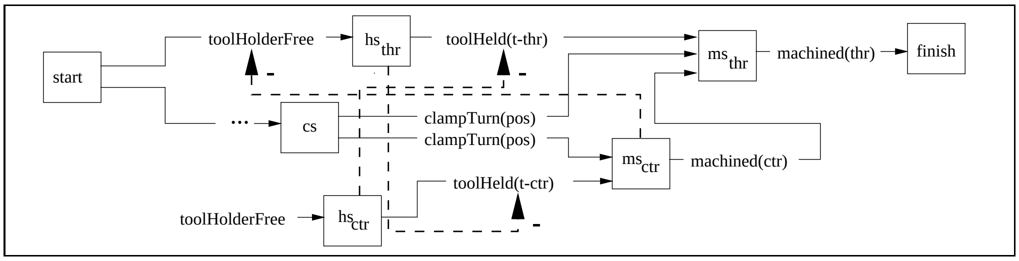

Figure 10: Not trivial serializability of the domain of process planning.

hs_{thr} with *start* and introduces a *MakeToolHolderFree* step instead.[7] Thus, we can now state the following proposition:

Theorem 3 *Goals corresponding to machining processing areas of workpieces are not trivially serializable modulo the class of elastic protected plans to machining workpieces.*

This negative result has been obtained under the supposition that the goals are achieved in any order. However, as stated before, we are dealing with extended problem descriptions, which include a partial order, $\prec$, for achieving the goals. For example, a feature is always machined after its covering outline. Thus, in the example illustrated before, *ctr* is always machined before $thr1$ so the situation illustrated in figure 10 cannot occur. Essentially, a plan, P', for machining a set of goals, G', corresponding to machining processing areas is not serially extensible with respect to a goal, g, corresponding to machining another processing area, if, for achieving g, a new holding step, hs_g, must be introduced and the machining step for achieving g must be ordered before the machining step for achieving a goal g' in G'. The partial order $\prec$ states precisely all processing areas for which the order of achievement is mandatory. As stated at the beginning of this section, P' was achieved by considering $\prec$, that is, there is no goal g' in G' with $g \prec g'$. Thus, it is not mandatory to intercalate g in P'. We conclude that P' can be serially extended to solve $G' \cup \{g\}$. The following theorem summarizes this result:

Theorem 4 *Let* $< I, G, \prec>$ *be an extended problem description corresponding to machining a workpiece. Then* G *is* $\prec$*-constrained trivially serializable modulo the class of elastic protected plans to machining workpieces that minimize the number of clamping and holding steps.*

[7] For establishing the precondition $toolHolderFree@hs_{thr}$, there are two possibilities: to establish it with *start* (i.e., no tool is being held initially), or, to establish it with a *MakeToolHolderFree* step. As explained before, SNLP always tries first to establish open conditions with existing steps, and, then with new steps. Thus, SNLP first establish the precondition $toolHolderFree@hs_{thr}$ with *start* as we assume that no tool is being held initially.

This positive result summarizes this section and is the main result of this report. It also supports the use of SNLP for achieving the tasks that, according to our model for problem solving, must be accomplished by the AI planner.

Lemma 1 shows that the new clamping steps can be intercalated anywhere in the chain of clamping steps whereas lemma 2 shows that new holding steps must always be appended to end of the chain of holding steps. Thus, there may be situations in which the new machining step ms can be intercalated in P': when the precondition $clampTurn(pos)@ms$ is established with a new clamping step, cs, and cs is intercalated in P' such that $toolHeld(t)@ms$ can be achieved with a holding step in P'. This means that there are some permutations on machining goals that are serialization orders although they are not $\prec$-consistent. However, as we saw, the way $\prec$ is defined ensures that the new machining steps can always be appended to the end of P'. Thus, if the clamping operation is modeled in the same way as the holding operation (i.e., by introducing an operator "MakeClampingFree"), theorem 4 still holds. In this situation, a permutation, π, on a set of machining goals is a serialization order if and only if the permutation π is consistent with $\prec$.

7 Nonmergeability of Subplans for Machining Parts of a Workpiece

One could imagine the following strategy to obtain machining plans: first, plan for obtaining machining plans for parts of the workpiece and then combine these plans to obtain a plan for the whole workpiece. In this section we will show that, although, this strategy may appear reasonable, it is in fact contraproductive for SNLP. But, first, we recall the definition of mergeability (Kambhampati et al., 1996):

Definition 6 (Mergeability of plans) *Given a class* $\mathbf{P}$ *of plans and a plan,* P_1*, in* $\mathbf{P}$ *for achieving a goal* g_1*.* P_1 *is mergeable with respect to a plan* P_2 *in* $\mathbf{P}$ *for achieving a goal* g_2 *if there is a plan* P *in* $\mathbf{P}$ *extending* P_1 *and* P_2 *achieving both* g_1 *and* g_2*.*

As in the definition of serial extensibility, the definition of mergeability does not exclude that backtracking takes place in finding P. The point is that no backtracking should take place in the plan refinement steps that introduced plan steps, links, or constraints in P_1 or in P_2.

Subplans achieving processing areas of a workpiece are not mergeable. Lemma 2 shows that holding steps are ordered in a chain of the form *HoldTool*, *MakeToolHolderFree*, *HoldTool* and so on and that new holding steps can only be appended to the end of this chain. Thus, when a holding step of one of the subplans is to be intercalated in other subplan, an establishment of the precondition *toolHolderfree* of the holding step of one of the subplans must always be revised. For example, consider the plan for achieving the center outline shown in figure 7. Suppose that this plan is to be merged with a plan to achieve the right outline, *outl-R*, which has the same form as the plan for achieving the center outline. That is, this plan uses the same clamping and holding operations (steps cs' and hs' respectively). In this situation, Either, $start \rightarrow toolHolderFree@hs$ or $start \rightarrow toolHolderFree@hs'$ must be revised so that one can be ordered after the other one. We can now state the following theorem:

Theorem 5 *Plans for achieving a processing area of a workpiece are not mergeable with respect to plans for achieving other processing areas of the same workpiece modulo the class of elastic protected plans to machining workpieces.*

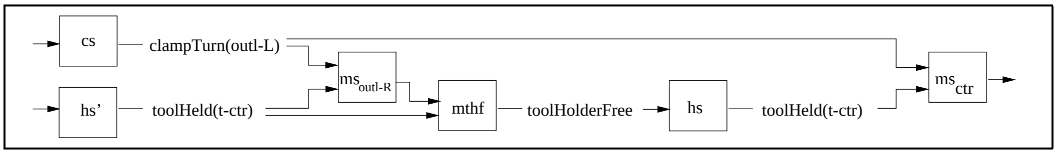

Figure 11: Plan for machining the center and the right outlines

Figure 11 shows a solution obtained by SNLP to machine the center and the right outline when starting with the two subplans mentioned before. Notice that the establishment of the precondition *toolHolderfree@hs* with *start* was revised and, instead, a new make-tool-holder-free step, *mthf*, was introduced. Notice also that cs' has also been rejected. Another negative effect of merging subplans is that the plans obtained do not necessarily minimize the number of clamping and holding operations. For example, in the plan shown in figure 11, the same holding operation, to hold the tool *t-hor*, is performed two times although this is clearly unnecessary.

8 Experiments on $\prec$-Constrained Trivial Serializability and Mergeability

We performed two experiments to evaluate the practical impact of our theoretical findings. In the first experiment, a representative problem was selected and the performance of the planning process was evaluated goal by goal when a $\prec$-consistent permutation on the goals was given. In the second experiment, several problems were given and the overall performance was meassured when $\prec$-consistent and non $\prec$-consistent permutations on the goals were given. The control strategy for SNLP used in these experiments is the standard one: prefer solving threats to establishing open preconditions, and, prefer performing simple establishments to performing establishments with new plan steps.

8.1 First Experiment

In the first experiment, the problem corresponding to machining the workpiece shown in figures 2 and 3 was given. Six goals were selected from the set of all goals for this workpiece: to machining (1) the left outline, (2) the center outline, (3) the undercut covered by the horizontal outline, (4) the drill, (5) the right outline and (6) the thread. The following ordering restrictions are generated by the geometrical reasoner: $(2) \prec \{(3),(4)\}$, $(5) \prec (6)$, and $\{(1),(2),(4)\} \prec (6)$.

In this experiment, we compared the performance of the planning process when:

1. The following, non $\prec$-consistent permutation on the goals was given: (5), (3), (4), (2), (6), (1).
2. The following, $\prec$-consistent permutation on the goals was given: (1), (2), (3), (4), (5), (6).
3. A plan for each goal was generated and then a solution was found by merging the plans.

Non $\prec$-consistent permutation. Table 2 summarize the results when the non $\prec$-consistent permutation on the goals was given. Each of the first six lines shows the effort in achieving a plan for the corresponding goals by starting with a plan achieving the previous goals in the order. For example, the third line shows the effort for achieving the first three goals when starting with a plan that achieved the first two goals. The last line shows the totals for achieving all six goals. The first column shows the time for planning in seconds. The second column shows the number of plan refinements that were performed by the planner. The third column shows the number of correct plan refinements performed by the planner. The difference between the second and third colummns measures the amount of backtracking that was necessary to achieve the solution. Finally, the fourth column shows the size of the plan for the corresponding goal (i.e., number of steps, causal links and protection links).

Notice that the fourth and sixth goals were solved by simple establishments as only one causal link was introduced. The principal aspect to observe in this table is the considerable effort to achieve the fifth goal (i.e., the thread). This is principally due to the fact that the order given is not $\prec$-consistent. Thus, a major planning effort was necessary in finding a solution by pursuing to extend the plan achieving the first five goals.

$\prec$-consistent permutation. Table 3 shows the results when $\prec$-consistent permutation on the goals was given. This table has the same structure as table 2. Notice that, beginning from the second goal, the effort for extending the next goal grows

Goal	Planning	Refinements	Correct Ref.	Step+CL+PL
(5)	5	15	10	3+5+3
(3)	15	35	24	5+9+12
(4)	25	40	26	5+11+9
(2)	1	1	1	0+1+0
(6)	113	75	24	5+9+17
(1)	1	2	2	0+1+1
Total	160	168	87	18+36+42

Table 2: Results with the non $\prec$-consistent permutation

steadily. Again when planning for machining the thread (the sixth goal) a high planning effort was necessary, although, not comparable with the effort when the thread was ordered inconsistently with $\prec$ (see line five of table 2). Notice also that sometimes a considerable effort must be made to extend the plan. For example, to achieve the first three goals (i.e., the right outline, the center outline and the undercut), more than 95% of the time was expend in extending the plan achieving the first two goals, which only took less than 5% of the time. Another example can be seen when achieving six first goals for which almost 50% of the time was spended for extending the plan achieving the first five goals. The reason for this amount of effort is the high number of interactions or more precisely *threats* that occur when pursuing achieving the additional goal as we saw in section 6.3.

Goal	Planning	Refinements	Correct Ref.	Step+CL+PL
(1)	5	15	10	3+5+3
(2)	2	4	4	1+3+1
(3)	23	41	20	4+7+11
(4)	17	35	27	5+11+9
(5)	11	6	6	1+3+3
(6)	50	36	23	4+7+13
Total	108	137	90	18+36+40

Table 3: Results with the $\prec$-consistent permutation

Merging subplans. Table 4 shows the results when the planner pursues to merge the plans achieving single goals. The goals were ordered with the same $\prec$-consistent permutation as before. First, a plan for the first goal was generated (see line 1 of table 4). Second, a plan for the second goal was generated and, then, the planner generated a plan for achieving the first two goals by pursuing to merge the plans achieving each goal. The cost of generating the plan for the two goals is shown in line 2. The third line shows the effort of merging the plan achieving the first two goals that was obtained by pursuing to merge their corresponding plans with the plan achieving the third goal. The process was repeated with the first four goals. The last line shows the totals for achieving the first four goals.

Goal	Planning	Refinements	Correct Ref.	Step+CL+PL
(1)	5	15	10	3+5+3
(1-2)	4	26	9	3+5+3
(1-3)	73	63	32	4+7+12
(1-4)	268	57	38	8+14+17
Total (1-4)	350	161	89	18+31+35

Table 4: Results when pursuing to merge the plans for single goals

The experiment in which the subplans were merged was stopped after four goals because the time for merging them, 350 seconds, was almost the double of the time the planner needed to achieve the six goals, 160 seconds, when the non $\prec$-consistent permutation was given and the triple of the time, 108 seconds, when the $\prec$-consistent permutation was given. Additionally, the number of steps for achieving the four goals when plans were merged is the same as the number of steps for achieving the six goals when the non $\prec$-consistent permutation was given (i.e., 18 steps) and five more than the number of plan steps for achieving the same four goals when the $\prec$-consistent permutation was given. Thus, merging the subplans had a worse performance than solving the goals with a non $\prec$-consistent permutation.

8.2 Second Experiment

In the second experiment, two collections of problems were formed: one containing permutations of the six goals mentioned before that are $\prec$-consistent, and, the other one containing permutations of the six goals that are not. As before, for each problem, a plan achieving the first goal was obtained, then, a plan achieving the first two goals was obtained by pursuing to extend the plan achieving the first goal and so on.

Table 5 summarize the results. The first line summarizes the results for the first collection and the second line summarizes the results for the other one. Notice that achieving the goals with $\prec$-consistent permutations results in an improvement of the performance of approximately 30% compared with achieving the goals with non $\prec$-consistent permutations. Additionally, with $\prec$-consistent permutations, almost 60% of the refinements were correct, whereas, with non $\prec$-consistent permutations, less than 50% of the refinements were correct.

Permutations	Planning	Refinements	Correct Ref.	Step+CL+PL
$\prec$-consistent	146	152	91	18+36+41
Non $\prec$-consistent	197	173	84	18+36+51

Table 5: Comparing the performance of planning with $\prec$-consistent permutations and of planning with non $\prec$-consistent permutations of the goals

9 Related Work

Process planning has been the subject of several studies in AI (Hayes, 1987; Kambhampati et al., 1991; Gil, 1991; Karinthi et al., 1992; Nau et al., 1995; Bergmann and Wilke, 1995), involving different AI techniques and different problem solving criteria.

(Hayes, 1987) already observed that the domain of process planning is characterized by the high number of interactions and uses a domain dependent mechanism to detect interactions before they occur. This idea is similar to our use of the geometrical reasoner to state the partial order, $\prec$, between the processing areas. The domain knowledge is submerged in a domain-specific reasoner.

To solve problems in this domain, (Kambhampati et al., 1991) proposed the interaction between several reasoners including NONLIN (Tate, 1977), a general purpose planner. These approach has been proposed by researches fielded in mechanical engineering as well (Ham and Lu, 1988; Giusti et al., 1989; Eversheim and Schneewind, 1993; ElMaraghy and McMaster, 1993).

(Gil, 1991) provides an extensive specification of this domain for the nonlinear, state-space planner Prodigy (Stone and Veloso, 1995). This specification is more detailed than ours including, among other things, several types of drilling operations and the representation of the numerical dimensions of the workpiece. As mentioned before, we do not pretend to model all aspects of the machining process but only those that are suitable for the AI Planner to handle. Thus, for example, no numerical information is provided in our specification of the domain. Another difference is that clamping operations in (Gil, 1991) are modelled with the same principle as our holding operations: there is an operator called *release-from-holding-device* that is used between clamping steps in the way our *MakeToolHolderFree* operator is used. In this situation lemma 2 applies for the clamping operations. As mentioned before, because of the way we modelled the clamping operation, there are some permutations on machining goals that are serialization orders although they are not $\prec$-consistent. However, if the holding operator was modelled as in (Gil, 1991), only the $\prec$-consistent permuations on the goals are serialization orders.

UCPOP (Penberthy and Weld, 1992) is an extension of SNLP supporting -among other things- universal quantification of variables. Thus, in UCPOP, the holding operations can be modelled in the same way as the clamping operations: to exclude all other holding possibilities by stating them in the delete list. In this situation lemma 1 applies for the holding operations. This means that machining goals are trivially serializable in the class of elastic protected plans for machining workpieces that minimize the number of clamping and holding steps (theorem 1). An interesting test would be to evaluate the performance of UCPOP when the holding operator is modelled in this way, and, to compare its performance with the model used here when $\prec$ is taken into consideration. Another interesting observation is that even with universal quantification, machining plans are nonmergeable because the resulting plans do not minimize the use of clamping and holding steps (see section 7). Thus, they are not in the required class.

In (Nau et al., 1995) a study of this domain is presented from the perspective of mechanical engineering as opposed to AI planning. This work explains the difficulty of using general purpose planners to represent the problematic of the domain, for example, the overlapping between features which could be very difficult to handle for the generic planner. Our model of problem solving is based precisely on the fact that representing the whole machining process in a symbolic representation is either impossible or intractable. The geometrical reasoner, for example, handles the interactions between features (see table 1).

The problem of merging plans achieving parts of a workpiece was carefully studied in (Karinthi et al., 1992; Yang et al., 1992). However, the notion of mergeability presented there is different to the one used here: each step has assigned costs, and, two steps are said to be mergeable if they can be replaced by a single step with less cost than the sum of the costs of the two steps. For example, if two plans contain the same clamping steps corresponding to the same clamping operation, they can be merged into a single clamping step. For obtaining an optimal solution (for example, minimizing the number of clamping and holding steps), all possible solution plans must be computed for each part of the workpiece, which can be very expensive. In general, merging subplans is NP-complete (Yang et al., 1992). Our experiments supports this result as the execution time increased exponentially with the number of subplans being merged.

In (Bergmann and Wilke, 1995), a case-based planner guiding a linear, state-space planner is presented. This system provides levels of abstraction and is developed with the idea that plans differing in details have the same abstract representation. In addition, a specification for the domain of process planning is given. However, key features such as threads are not modelled, which, as we saw, influence the overall planning process. An interesting question to be answered is if the specification presented there can be extended to consider key features. In this situation, two workpieces differing in a key feature are expected to have different abstract representations.

10 Discussion

Another interesting aspect to consider is to examine if $\prec$-constraint trivial serializability holds for sufix plans. Suppose that a $\prec$-consistent permutation on a set of goals corresponding to machining the processing areas of a workpiece is given. A sufix plan achieving the first goal, g_1, is generated containing a clamping, a holding and a machining step, $cs_1 * hs_1 * ms_1$, where $*$ indicates the contiguity constraints (Kambhampati et al., 1996). Notice that the holding and the clamping steps may be exchanged (i.e., $hs_1 * cs_1 * ms_1$). For achieving the second goal, g_2, a new machining step, ms_2, and new clamping and/or holding steps are introduced if cs_1 and/or hs_1 are not adequate. By continuing in this way, when planning to achieve the goal g_k, the current state, which is obtained after the last step of the sufix plan is applied, includes the effects of the last clamping and holding operations, cs_i and hs_j respectively (with $i, j \leq k$), if other clamping and/or holding steps are needed the sufix plan can always be extended by introducing the clamping and/or holding steps needed and chaining the machining step of g_k, ms_k, as the last step in the plan. This extension can always be done because a $\prec$-consistent permutation on the goals was given.

A similar analysis can be done for sufix or prefix and sufix plans. Thus, we can state the following theorem:

Theorem 6 *Let $< I, G, \prec >$ be an extended problem description corresponding to machining a workpiece. Then G is $\prec$-constrained trivial serializable modulo the class of prefix, sufix, or prefix and sufix plans to machining workpieces.*

Another aspect to be consider is if the minimality on the application of clamping and holding operations is preserved. Consider, for example, the workpiece shown in figures 2 and 3. Suppose that the problem is to achieve the following goals: to machine (1) the horizontal outline, (2) the drilled hole, (3) the right side, (4) the undercut under the horizontal outline, (5) the perforation, and (6) the right ascending outline. Notice that (1), (2), ..., (6) is a $\prec$-constrained permutation on the goals because machining the horizontal outline (1) is ordered before machining the drilled hole (2) and the undercut (4), and, machining the right side (3) is ordered before machining the perforation (5). When achieving these goals by extending each sufix plan generated, the sufix plan obtained will contain five clamping operations (one for each of the first four goals and another one for the last two) and six holding operations (one for each goal). However, a plan achieving these goals that contains three clamping and three holding operations can be obtained. Thus extending sufix plans may result in plans that does not minimize the number of clamping and holding steps. We can now formulate the following theorem.

Theorem 7 *Let $< I, G, \prec >$ be an extended problem description corresponding to machine a workpiece. Then, G is not $\prec$-constrained trivial serializable modulo the* *class of prefix, sufix, or prefix and sufix plans for machining the workpieces that minimize the number of clamping and holding steps.*

11 Final Remarks

In this report we have presented a specification motivated by a model for problem solving in the domain of process planning that integrates several reasoners including an AI planner which must perform certain tasks. We have analyzed the tasks that the AI planner must perform and motivated the particular specification representing these tasks. We studied the properties resulting of the use of SNLP to plan with this specification and found that goals corresponding to machining processing areas of a workpiece are $\prec$-constrained trivial serializable modulo the class of elastic protected plans to machining workpieces that minimize the number of clamping and holding steps. In addition, we found that subplans achieving processing areas of a workpiece are not mergeable in this class.

We also found that goals corresponding to machining processing areas of a workpiece are $\prec$-constrained trivially serializable modulo the class of prefix, sufix, or prefix and sufix plans, but, the resulting plans may contain unnecessary clamping and holding steps which are precisely the ones to be minimized.

We performed some experiments to evaluate the practical impact of our theoretical findings, and concluded that:

1. Planning by extending plans that achieve goals corresponding to machine processing areas of a workpiece results in a significant increase in the performance when a $\prec$-consistent permutation on the goals was given.

2. Planning by merging subplans for machining parts of a workpiece is prohibitive expensive and results in plans containing redundant holding and/or clamping operations.

3. Even when a $\prec$-consistent permutation on the goals was given, planning can be very expensive due to the high number of interactions occuring.

The last point motivates the necessity of mechanisms to guide the threat resolution process of SNLP. Either introducing in SNLP a more elaborate strategy to handle threats (Yang et al., 1992; Peot and Smith, 1993), or, using external control for SNLP such as CBR (Muñoz-Avila and Weberskirch, 1996; Ihrig and Kambhampati, 1996) or EBL (Kambhampati et al., 1995).

References

Alting, L. and Zhang, H. (1989). Computer aided process planning: the state-of-the-art survey. *International Journal of Productive Resources*, 27(4):553–585.

Barrett, A. and Weld, D. (1994). Partial-order planning: Evaluating possible efficiency gains. *Artificial Intelligence*, 67(1):71–112.

Bergmann, R. and Wilke, W. (1995). Building and refining abstract planning cases by change of representation language. *Journal of Artificial Intelligence Research*, 3:53–118.

ElMaraghy, H. A. and McMaster (1993). Evolution and future perspectives of CAPP. In *Annals of CIRP*, volume 42.

Eversheim, W. and Schneewind, J. (1993). Computer-aided process planning - state of the art and future development. *Robotics and Computer-Integrated manufacturing*, 10(1/2):65–70.

Fikes, R. and Nilsson, N. (1971). Strips: A new approach to the application of theorem proving in problem solving. *Artificial Intelligence*, 2:189–208.

Gil, Y. (1991). A specification of manufacturing processes for planning. Technical Report CMU-CS-91-179, School of Computer Science, Carnegie Mellon University, Pittsburg.

Giusti, F., Santochi, M., and Dini, G. (1989). Kaplan: a knowledge-based approach to process planning of rotational parts. In *Annals of the CIRP*, volume 38.

Ham, I. and Lu, S. C.-Y. (1988). Computer-aided process planning: The present and the future. In *Annals of the CIRP*, volume 37.

Hayes, C. (1987). Using goal interactions to guide planning. In *Proceedings of AAAI-87*, pages 224–228.

Ihrig, L. and Kambhampati, S. (1996). Design and implementation of a replay framework based on a partial order planner. In Weld, D., editor, *Proceedings of AAAI-96*. IOS Press.

Kambhampati, S. (1993). On the utility of systematicity: Understanding tradeoffs between redundancy and commitment in partial-order planning. In *Proceedings of IJCAI-93*, pages 116–125.

Kambhampati, S., Cutkosky, M., Tenenbaum, M., and Lee, S. (1991). Combining specialized reasoners and general purpose planners: A case study. In *Proceedings of AAAI-91*, pages 199–205.

Kambhampati, S., Ihrig, L., and Srivastava, B. (1996). A candidate set based analysis of subgoal interactions in conjunctive goal planning. In *Proceedings of the 3rd International Conference on AI Planning Systems (AIPS-96)*, pages 125–133.

Kambhampati, S., Katukam, S., and Qu, Y. (1995). Failure driven dynamic search control for partial order planners: An explanation-based approach. *Artificial Intelligence*. (submitted, ASU-CSE-TR-95-010).

Karinthi, R., Nau, D., and Yang, Q. (1992). Handling feature interactions in process-planning. *Applied Artificial Intelligence*, 6:389–415.

McAllester, D. and Rosenblitt, D. (1991). Systematic nonlinear planning. In *Proceedings of AAAI-91*, pages 634–639.

Muñoz-Avila, H. and Hüllen, J. (1995). Retrieving relevant cases by using goal dependencies. In Veloso, M. and Aamodt, A., editors, *Proceedings of the 1st International Conference on Case-Based Reasoning (ICCBR-95)*, number 1010 in Lecture Notes in Artificial Intelligence. Springer.

Muñoz-Avila, H. and Weberskirch, F. (1996). Planning for manufacturing workpieces by storing, indexing and replaying planning decisions. In *Proceedings of the 3rd International Conference on AI Planning Systems (AIPS-96)*. AAAI-Press.

Nau, D., Gupta, S., and Regli, W. (1995). AI planning versus manufacturing-operation planning: A case study. In *Proceedings of IJCAI-95*.

Penberthy, J. and Weld, D. (1992). Ucpop: A sound, complete, partial order planner for adl. In *Proceedings of KR-93*.

Peot, M. and Smith, D. (1993). Threat-removal strategies for partial-order planning. In *Proceedings of AAAI-93*, pages 492–499.

Petrie, C. (1991). Context maintenance. In *Proceedings of AAAI-91*, pages 288–295.

Petrie, C. (1992). Constrained decision revision. In *Proceedings of AAAI-92*, pages 393–400.

Stone, P. and Veloso, M. (1995). User-guided interleaving of planning and execution. In *Proceedings of the 3rd European Workshop on Planning (EWSP-95)*.

Tate, A. (1977). Project planning using a hierarchical non-linear planner. Research report no. 25, Dept. of Artificial Intelligence, Edinburgh University.

Tönshoff, H. K. and Andres, N. (1990). Survey on state of development and trends in capp researchsurvey on state of development and trends in CAPP research within cirp within cirp. In *Annals of CIRP*, volume 39.

Weberskirch, F. (1995). Combining SNLP-like planning and dependency-maintenance. Technical Report LSA-95-10E, Centre for Learning Systems and Applications, University of Kaiserslautern, Germany.

Weill, R. and Eversheim, W. (1982). Survey of computer-aided planning systems. In *Annals of the CIRP*, volume 31.

Yang, Q., Nau, D., and Hendler, J. (1992). Merging separately generated plans with restricted interactions. *Computational Intelligence*, 8(2):648–676.

A Proof of Lemma 1

Lemma 1 *Let $<S, L, B>$ be a plan were all the preconditions of the machining steps corresponding to clamping the workpiece have been established and all threats have been solved. Then, the following affirmations are valid:*

1. *The clamping steps in S are totally ordered in a chain of the form: $cs_1 <_L MS_1 <_L cs_2 <_L ... <_L cs_n <_L MS_n$, where cs_i is a clamping step and MS_i is the set of machining steps, ms_j, in S such that $clampTurn(t_j)@ms_j$ with is established with cs_i (with $1 \le i \le n$).*[8]
2. *New clamping steps can be ordered anywhere in the chain $cs_1 <_L cs_2 <_L ... <_L cs_n$.*
3. *A new pair of clamping-machining steps, cs, ms with $cs \to clampTurn(pos)@ms$, can be ordered between MS_i and cs_{i+1} (with $1 \le i \le n-1$), after MS_n, or before cs_1.*

Proof. Let ms, cs be new machining and clamping step (i.e., not in S), such that $cs \to clampTurn(pos)@ms$. We will show by induction on the number, n, of clamping steps the third affirmation, which implies the other two.

Case $n = 1$. By hypothesis, *clampTurn(pos)@*ms_j can only be established by cs_1 for all ms_j in MS_1 because cs_1 is the only clamping step in S and all clamping preconditions has been established. Because cs is unordered w.r.t. cs_1 and ms_j for all ms_j in MS_1, and, by definition, clamping operators exclude the other clamping possibilities explicitly (see figure 5), $cs \overset{+/-}{\longleftrightarrow} (cs_1 \to clampTurn(pos_1)@ms_j)$ for all ms_j in MS_1. In the same way, $cs_1 \overset{+/-}{\longleftrightarrow} (cs \to clampTurn(pos)@ms)$. Figure 12 depictes this situation when $MS_1 = \{ms_1\}$.

Neither threat can be solved by separation because clamping operators always exclude explicilty the other possibilities (see figure 5). These threats can only be solved solved by promoting either cs for all ms_j in MS_1 or cs_1, that is, they are solved by introducing the protection link $ms_j \to cs$ for all ms_j in MS_1 or the protection link $ms \to cs_1$. If cs is promoted for all ms_j in MS_1, the resulting order is $cs_1 <_L MS_1 <_L cs <_L ms$. If cs_1 is promoted, the resulting order is $cs <_L ms <_L cs_1 <_L MS_1$.

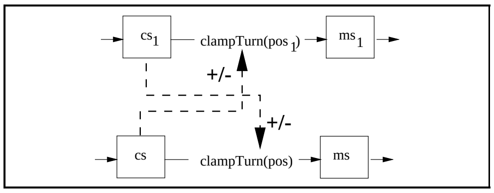

Figure 12: Basic inductive case for the clamping operation

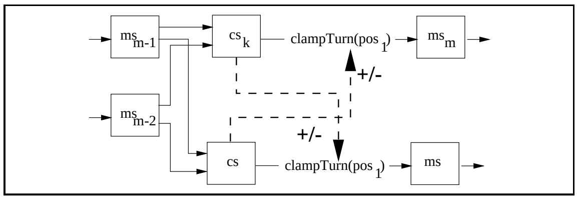

Figure 13: General inductive case for the clamping operation

[8] $cs_i <_L MS_i$ indicates that $cs_i <_L ms_j$ for all ms_j in MS_i.

Case $n = k + 1$. Let $cs_1 <_L ... <_L MS_{k-1} <_L cs_{k-1} <_L MS_k$ be the ordered chain of clamping and machining steps in $<S, L, B>$. By induction, cs can be ordered either before cs_1, between MS_i and cs_{i+1} (with $1 \leq i \leq k-2$) or after MS_{k-1}. If any of the first two operations is performed there are no interactions to be consider with cs_{k-1} or MS_k because $cs <_L ms <_L cs_{i+1} <_L cs_{k-1} <_L MS_k$. So, we only have to consider the case in which cs is ordered after MS_{k-1}, that is, $MS_{k-1} <_L cs <_L ms$. Thus, $cs \overset{+/-}{\longleftrightarrow} (cs_k \rightarrow clampTurn(pos_k)@ms_j)$ for all ms_j in MS_k because cs is unordered relative to cs_k and ms_j for all ms_j in MS_k. In the same way, $cs_k \overset{+/-}{\longleftrightarrow} (cs \rightarrow clampTurn(pos)@ms)$. Figure 13 depictes this situation when $MS_k = \{ms_m\}$ and $MS_{k-1} = \{ms_{m-1}, ms_{m-2}\}$.

As in the basis case ($n = 1$), the threats cannot be solved by separation. Thus, the threats can only be solved by either promoting cs for all ms_j in MS_k or cs_k. If cs is promoted for all ms_j in MS_k, the resulting order is $cs_k <_L MS_k <_L cs <_L ms$. If cs_k is promoted, the resulting order is $MS_{k-1} <_L cs <_L ms <_L cs_k <_L MS_k$. ■

B Proof of Lemma 2

Lemma 2 *Let <S, L, B> be a complete plan containing only holding and MakeToolHolderFree steps.*[9] *Then, the following affirmations are valid:*

1. *The steps in S are totally ordered in a chain of the form: $hs_1 <_L mthf_1 <_L hs_2 <_L \ldots <_L hs_n$, where $mthf_i, hs_i$ are MakeToolHolderFree and holding steps respectively. Further, $mthf_i$ establishes the precondition $toolHolderFree@hs_{i+1}$ and hs_i establishes the precondition $toolHeld(t_i)@mthf_i$ with $1 \leq i \leq n-1$.*

2. *If a new holding step is introduced then a new MakeToolHolderFree step must be introduced as well.*

3. *The new MakeToolHolderFree step is appended to the end of the chain of holding steps in S when the plan <S, L, B> is completed to achieve all open conditions. Further, the precondition corresponding to hold a tool of the new MakeToolHolderFree step is achieved by the last holding step in S, hs_n.*

Proof. Let hs be a new holding step (i.e., not in S). We will show by induction on the number, n, of holding steps the last two properties, which imply the first one.

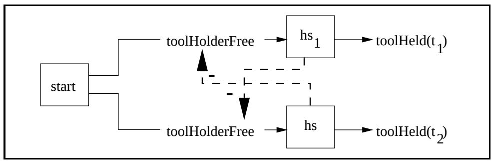

Figure 14: Establishing *toolHolderFree@hs* with *start*

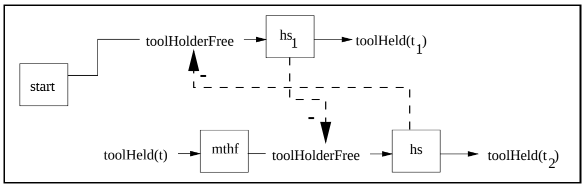

Figure 15: Threats occuring by introducing new *Holding* and *MakeToolHolderFree* steps

Case $n = 1$. When $n = 1$, the precondition *toolHolderFree*@hs_1 is achieved by $start$. Suppose that $start$ is also used to establish *toolHolderFree@hs* (see figure 14). Two threats occur: $hs \overset{-}{\longleftrightarrow} (start \to toolHolderFree@hs_1)$ and $hs_1 \overset{-}{\longleftrightarrow} (start \to toolHolderFree@hs)$. The threats cannot be solved by separation because of $hs \to -toolHolderFree$ and $hs_1 \to -toolHolderFree$. Demoting hs or hs_1 is not permitted because no step can be ordered before $start$. Thus, the only remaining possibility to solve the threats is to promote hs or hs_1. However, if hs is promoted, then $hs_1 <_L hs$ and if hs_1 is promoted, then $hs <_L hs_1$, which is clearly a contradiction. Thus, performing a simple establishment with $start$ to establish $toolHolderFree@hs$ results in a dead-end. We conclude that the only way to establish this precondition is to introduce a new MakeToolHolderFree step.

[9] This plan may be obtained by taking a complete plan to machine parts of a workpiece and removing all non holding and non MakeToolHolderFree steps (with the exception of $start$ and $finish$) and all links that are not from and to a holding step, a MakeToolHolderFree step, $start$ or $finish$.

Let $mthf$ be a new MakeToolHolderFree step establishing $toolHolderFree@hs$ which has *toolHeld(t)*@$mthf$ as precondition. Because $mthf$ and hs are unordered relative to S, the following threats occur: $hs \stackrel{-}{\longleftrightarrow} (start \rightarrow toolHolderFree@hs_1)$ and $hs_1 \stackrel{-}{\longleftrightarrow} (mthf \rightarrow toolHolderFree@hs)$ (see figure 15). Separation cannot be performed. The only way of solving these threats is to demote hs_1, that is, to introduce the protection link $hs_1 \rightarrow mthf$. Demoting hs_1 results in the following order: $hs_1 <_L mthf <_L hs$. Thus, no threats can occur. To complete the plan, *toolHeld(t)*@$mthf$ must be established. There are two alternatives: to introduce a new holding step or to make a simple establishment with hs_1. The first alternative is discarded because it results in a goal-loop. Thus, *toolHeld(t)*@$mthf$ can only be established with hs_1.

Case $n = k + 1$. By the inductive hypothesis, we can assume that $hs_1, mthf_1 <_L h_2 <_L ... <_L h_k$ is a chain of holding steps in S where the precondition $toolHeld(t_i)@mthf_i$ is achieved by hs_i and the precondition *toolHolderFree*@hs_{i+1} is achieved by $mthf_i$ with $1 \leq i \leq k-1$.

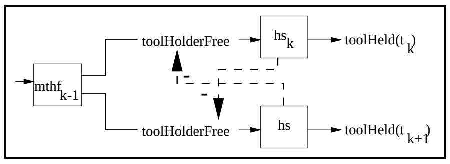

Figure 16: Establishing *toolHolderFree* with *start*

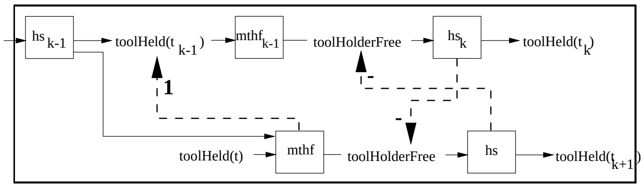

Figure 17: Threats occuring by introducing a new holding step

By induction, we know that none of the MakeToolHolderFree steps $mthf_i$ with $1 \leq i \leq k-2$ can be used to establish the precondition *toolHolderFree*@hs. Thus, there are only two alternatives to establish this precondition: to perform a simple establishment with $mthf_{k-1}$ or to introduce a new MakeToolHolderFree step. Suppose that *toolHolderFree*@hs is established with $mthf_{k-1}$. Then, two threats occur: $hs \stackrel{-}{\longleftrightarrow} (mthf_{k-1} \rightarrow toolHolderFree@hs_k)$ and $hs_k \stackrel{-}{\longleftrightarrow} (mthf_{k-1} \rightarrow toolHolderFree@hs)$ (see figure 16). These threats cannot be solved by separation. Demoting hs_k or hs results in a contradiction of the form $hs_k <_L hs_k$ and promoting hs_k and hs results in a contradiction of the form $hs <_L hs$. For example, if hs_k is demoted then $hs_k <_L mthf_{k-1}$, but, because $mthf_{k-1} \rightarrow toolHolderFree@hs_k)$, $mthf_{k-1} <_L hs_k$. Thus, performing a simple establishment with $mthf_{k-1}$ results in a dead-end. We conclude that the only possibility of establishing the precondition *toolHolderFree*@hs is to introduce a new MakeToolHolderFree step.

Let $mthf$ be a new MakeToolHolderFree step with precondition *toolHeld(t)*@$mthf$. By induction, we know that $mthf$ must be ordered after hs_{k-1} because $hs_1 <_L mthf_1 <_L h_2 <_L ... <_L hs_{k-1}$ is a chain of $k-1$ holding steps. Thus, following threats must occur: $hs \stackrel{-}{\longleftrightarrow} (mthf_{k-1} \rightarrow toolHolderFree@hs_k)$, $hs_k \stackrel{-}{\longleftrightarrow} (mthf \rightarrow toolHolderFree@hs)$ and $mthf \stackrel{-}{\longleftrightarrow} (hs_{k-1} \rightarrow mthf_{k-1}@hs)$ (see figure 17). The first two threats can be solved by demoting either hs or hs_k. Suppose that hs is demoted, then, $hs_{k-1} <_L mthf <_L hs <_L mthf_{k-1}$ and $hs_{k-1} <_L mthf_{k-1}$, thus, the threat $mthf \stackrel{-}{\longleftrightarrow} (hs_{k-1} \rightarrow mthf_{k-1}@hs)$ still occurs (labelled *1* in figure 17). In this situation, the only possible solution of this threat is by performing a separation, that is, by constraining t to be different than t_{k-1}. However, this is a contradiction to the inductive hypothesis because the only way to achieve *toolHeld(t)*@$mthf$ is with a simple establishment with hs_{k-1} because $hs_1 <_L mthf_1 <_L h_2 <_L$ ... $<_L hs_{k-1}$ is a chain of $k-1$ holding steps and, thus, the precondition *toolHeld(t)mthf* must be established by the last holding step in the chain, hs_{k-1}. We conclude that the only possible solution to the threats is to demote hs_k (i.e., ordering $mthf$ after hs_k). Thus, the resulting ordering is: $hs_{k-1} <_L mthf_{k-1} <_L hs_k <_L mthf <_L hs$.

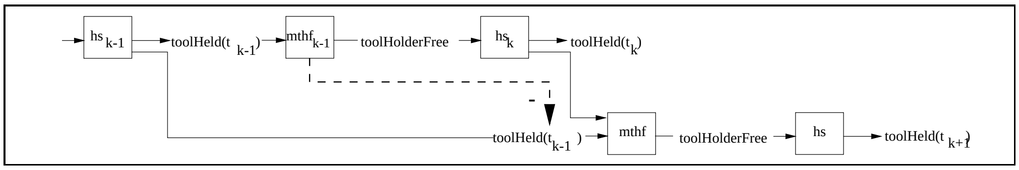

Figure 18: Wrong establishment of *toolHeld(t)*@$mthf$.

To achieve *toolHeld(t)*@$mthf$, there are three alternatives: to introduce a new holding step, to perform a simple establishment with hs_{k-1} and to perform a simple establishment with hs_k. The first one is discarded because it results in a goal loop. If *toolHeld(t)*@$mthf$ is established with hs_{k-1}, then $mthf_{k-1} \overset{-}{\longleftrightarrow} (hs_{k-1} \rightarrow toolHeld(t_{k-1})@mthf)$ because $hs_{k-1} <_L mthf_{k-1} <_L hs_k <_L mthf$ (see figure 18).This threat cannot be solved by separation because t is instantiated with t_{k-1}. Promotion and demotion cannot be performed because they result in a contradiction. For example, if $mthf_{k-1}$ is demoted, then $mthf_{k-1} <_L hs_{k-1}$. Thus, the only possibility to achieve *toolHeld(t)*@$mthf$ is to perform a simple establishment with hs_k. ■

C Specification of the Domain of Process Planning

An electronic version of this specification can be downloaded from
http://wwwagr.informatik.uni-kl.de/~caplan/PP.html.

Object types: (33)

CenterOutline
Superclass: Outline

DrillTool
Superclass: HoleTool

ExtRotaryTool
Superclass: Tool

Feature
Superclass: ProcArea

FeatureTool
Superclass: Tool

Groove
Superclass: Feature

GrooveTool
Superclass: FeatureTool

Hole
Superclass: Feature

HoleTool
Superclass: FeatureTool

IntProcArea
Superclass: ProcArea

IntRotaryTool
Superclass: Tool

LeftOutline
Superclass: Outline

LeftRTool
Superclass: ExtRotaryTool

Outline
Superclass: ProcArea

PrickOut
Superclass: Feature

PrickOutTool
Superclass: FeatureTool

ProcArea
Superclass: None.

RightOutline
Superclass: Outline

RightRTool
Superclass: ExtRotaryTool

RoundOff
Superclass: Feature

RoundOffTool
Superclass: FeatureTool

Slope
Superclass: Feature

SlopeTool
Superclass: FeatureTool

TappingTool
Superclass: HoleTool

Thread
Superclass: Feature

ThreadTool
Superclass: FeatureTool

Tool
Superclass: None.

Undercut
Superclass: Feature

UndercutHalf1
Superclass: Undercut

UndercutHalf2
Superclass: Undercut

WpieceSide
Superclass: ProcArea

WpieceSide1
Superclass: WpieceSide

WpieceSide2
Superclass: WpieceSide

Predicates: (17)

available
Arguments: tl

clampNoTurn
Arguments: pos

clampTurn
Arguments: pos

isClampArea
Arguments: area

neighbour
Arguments: area1 area2

noSubareaThread
Arguments: outl

processed
Arguments: area

processedUcutHalf1
Arguments: ucut

processedUcutHalf2
Arguments: ucut

restrictedClamp
Arguments: area

subarea
Arguments: feat outl

toolHeld
Arguments: tl

toolHolderFree
Arguments: None.

unprocessed
Arguments: area

unprocUcutHalf1
Arguments: area

unprocUcutHalf2
Arguments: area

unrestrictedClamp
Arguments: outl

Operators: (27)

ClampFromFace
Arguments: s1 s2 outl1 outl2 inn
Constraints: IsOfType(Outline, outl1)
IsOfType(Outline, outl2)
NotSame(outl2, outl1)
IsOfType(WpieceSide, s2)
IsOfType(WpieceSide, s1)
NotSame(s2, s1)
IsOfType(IntProcArea, inn)
Effects: +clampTurn(s1)
-clampTurn(outl1)
-clampNoTurn(outl1)
-clampTurn(outl2)
-clampNoTurn(outl2)
-clampNoTurn(s2)
Preconditions: +subarea(inn, s1)
+processed(inn)
+unprocessed(s2)
+isClampArea(s1)

ClampNoTurn
Arguments: outl s2 s outl2
Constraints: IsOfType(Outline, outl2)
IsOfType(Outline, outl)
IsOfType(WpieceSide, s2)
IsOfType(WpieceSide, s)
NotSame(outl, outl2)
NotSame(s2, s)
Effects: +clampNoTurn(outl)
-clampTurn(outl2)
-clampNoTurn(outl2)
-clampNoTurn(s)
-clampNoTurn(s2)
-clampTurn(outl)
Preconditions: +noSubareaThread(outl)
+isClampArea(outl2)
+isClampArea(outl)
+unrestrictedClamp(outl)

ClampNoTurn-NotFree

Arguments:	outl s2 s outl2
Constraints:	IsOfType(Outline, outl2)
	IsOfType(Outline, outl)
	IsOfType(WpieceSide, s2)
	IsOfType(WpieceSide, s)
	NotSame(outl, outl2)
	NotSame(s2, s)
Effects:	+clampNoTurn(outl)
	-clampTurn(outl2)
	-clampNoTurn(outl2)
	-clampNoTurn(s)
	-clampNoTurn(s2)
	-clampNoTurn(outl)
Preconditions:	+unprocessed(outl)
	+isClampArea(outl2)
	+isClampArea(outl)
	+restrictedClamp(outl)

ClampNoTurn-Thread

Arguments:	outl s2 s outl2 thr
Constraints:	IsOfType(Outline, outl2)
	IsOfType(Outline, outl)
	IsOfType(WpieceSide, s2)
	IsOfType(WpieceSide, s)
	NotSame(outl, outl2)
	NotSame(s2, s)
	IsOfType(Thread, thr)
Effects:	+clampNoTurn(outl)
	-clampTurn(outl2)
	-clampNoTurn(outl2)
	-clampNoTurn(s)
	-clampNoTurn(s2)
	-clampTurn(outl)
Preconditions:	+unprocessed(thr)
	+subarea(thr, outl)
	+isClampArea(outl2)
	+isClampArea(outl)

ClampTurn

Arguments:	outl s2 s outl2
Constraints:	IsOfType(Outline, outl2)
	IsOfType(Outline, outl)
	IsOfType(WpieceSide, s2)
	IsOfType(WpieceSide, s)
	NotSame(outl, outl2)
	NotSame(s2, s)
Effects:	+clampTurn(outl)
	-clampNoTurn(s)
	-clampNoTurn(s2)
	-clampTurn(outl2)
	-clampNoTurn(outl2)
	-clampNoTurn(outl)
Preconditions:	+noSubareaThread(outl)
	+isClampArea(outl2)
	+isClampArea(outl)
	+unrestrictedClamp(outl)

ClampTurn-NotFree

Arguments:	outl s2 s outl2
Constraints:	IsOfType(Outline, outl2)
	IsOfType(Outline, outl)
	IsOfType(WpieceSide, s2)
	IsOfType(WpieceSide, s)
	NotSame(outl, outl2)
	NotSame(s2, s)
Effects:	+clampTurn(outl)
	-clampTurn(outl2)
	-clampNoTurn(outl2)
	-clampNoTurn(s)
	-clampNoTurn(s2)
	-clampNoTurn(outl)
Preconditions:	+unprocessed(outl)
	+isClampArea(outl2)
	+isClampArea(outl)
	+restrictedClamp(outl)

ClampTurn-Thread

Arguments:	outl s2 s outl2 thr
Constraints:	IsOfType(Outline, outl2)
	IsOfType(Outline, outl)
	IsOfType(WpieceSide, s2)
	IsOfType(WpieceSide, s)
	NotSame(outl, outl2)
	NotSame(s2, s)
	IsOfType(Thread, thr)
Effects:	+clampTurn(outl)
	-clampTurn(outl2)
	-clampNoTurn(outl2)
	-clampNoTurn(s)
	-clampNoTurn(s2)
	-clampNoTurn(outl)
Preconditions:	+unprocessed(thr)
	+isClampArea(outl2)
	+isClampArea(outl)
	+subarea(thr, outl)

DrillHole

Arguments:	hole outl aTool clampArea
Constraints:	IsOfType(DrillTool, aTool)
	IsOfType(Outline, outl)
	IsOfType(ProcArea, clampArea)
	IsOfType(Hole, hole)
	NotSame(outl, clampArea)
Effects:	+processed(hole)
	-unprocessed(hole)
Preconditions:	+processed(outl)
	+clampNoTurn(clampArea)
	+toolHeld(aTool)
	+subarea(hole, outl)
	+available(aTool)

HoldTool

Arguments:	tool1
Constraints:	None.
Effects:	+toolHeld(tool1)
	-toolHolderFree()
Preconditions:	+toolHolderFree()

MachineGroove

Arguments:	groove outl tool clampArea
Constraints:	IsOfType(GrooveTool, tool)
	IsOfType(Outline, outl)
	IsOfType(ProcArea, clampArea)
	IsOfType(Groove, groove)
	NotSame(outl, clampArea)
Effects:	+processed(groove)
	-unprocessed(groove)
Preconditions:	+processed(outl)
	+clampTurn(clampArea)
	+toolHeld(tool)
	+subarea(groove, outl)
	+available(tool)

MachineInternalArea

Arguments:	inn s tool outl clampArea
Constraints:	IsOfType(IntProcArea, inn)
	IsOfType(WpieceSide, s)
	IsOfType(IntRotaryTool, tool)
	IsOfType(Outline, outl)
	IsOfType(Outline, clampArea)
	NotSame(outl, clampArea)
Effects:	+processed(inn)
	-unprocessed(inn)
Preconditions:	+available(tool)
	+processed(s)
	+clampNoTurn(clampArea)
	+toolHeld(tool)
	+subarea(inn, s)
	+neighbour(outl, s)

MachineOutline

Arguments:	outl tool clampArea
Constraints:	IsOfType(Outline, outl)
	IsOfType(ProcArea, clampArea)
	NotSame(outl, clampArea)
	IsOfType(ExtRotaryTool, tool)
Effects:	+processed(outl)
	-unprocessed(outl)
Preconditions:	+clampTurn(clampArea)
	+toolHeld(tool)
	+available(tool)

MachinePrickOut

Arguments:	prickout outl tool clampArea
Constraints:	IsOfType(PrickOutTool, tool)
	IsOfType(Outline, outl)
	IsOfType(ProcArea, clampArea)
	IsOfType(PrickOut, prickout)
	NotSame(outl, clampArea)
Effects:	+processed(prickout)
	-unprocessed(prickout)
Preconditions:	+processed(outl)
	+clampTurn(clampArea)
	+toolHeld(tool)
	+subarea(prickout, outl)
	+available(tool)

MachineRoundOff

Arguments:	roundoff outl tool clampArea
Constraints:	IsOfType(RoundOffTool, tool)
	IsOfType(Outline, outl)
	IsOfType(ProcArea, clampArea)
	IsOfType(RoundOff, roundoff)
	NotSame(outl, clampArea)
Effects:	+processed(roundoff)
	-unprocessed(roundoff)
Preconditions:	+processed(outl)
	+clampTurn(clampArea)
	+toolHeld(tool)
	+subarea(roundoff, outl)
	+available(tool)

MachineSide

Arguments:	s clampArea tool s1
Constraints:	IsOfType(Outline, clampArea)
	IsOfType(WpieceSide, s)
	IsOfType(WpieceSide, s1)
	IsOfType(LeftRTool, tool)
	NotSame(s, s1)
Effects:	+processed(s)
	-unprocessed(s)
Preconditions:	+clampTurn(clampArea)
	+toolHeld(tool)
	+unprocessed(s)
	+neighbour(clampArea, s1)
	+available(tool)

MachineSlope

Arguments:	slope outl tool clampArea
Constraints:	IsOfType(SlopeTool, tool)
	IsOfType(Outline, outl)
	IsOfType(ProcArea, clampArea)
	IsOfType(Slope, slope)
	NotSame(outl, clampArea)
Effects:	+processed(slope)
	-unprocessed(slope)
Preconditions:	+processed(outl)
	+clampTurn(clampArea)
	+toolHeld(tool)
	+subarea(slope, outl)
	+available(tool)

MachineThread

Arguments:	thr outl tool clampArea
Constraints:	IsOfType(ThreadTool, tool)
	IsOfType(Outline, outl)
	IsOfType(Thread, thr)
	IsOfType(ProcArea, clampArea)
	NotSame(outl, clampArea)
Effects:	+processed(thr)
	-unprocessed(thr)
Preconditions:	+processed(outl)
	+clampTurn(clampArea)
	+toolHeld(tool)
	+subarea(thr, outl)
	+unprocessed(thr)
	+available(tool)

MachineUndercutH1LeftToolOutline

Arguments:	ucut outl tool clampArea s1
Constraints:	IsOfType(LeftRTool, tool)
	IsOfType(ProcArea, clampArea)
	NotSame(outl, clampArea)
	IsOfType(Outline, outl)
	IsOfType(Undercut, ucut)
	IsOfType(WpieceSide1, s1)
Effects:	+processedUcutHalf1(ucut)
	-unprocUcutHalf1(ucut)
Preconditions:	+processed(outl)
	+toolHeld(tool)
	+clampTurn(clampArea)
	+subarea(ucut, outl)
	+available(tool)
	+neighbour(s1, clampArea)

MachineUndercutH1LeftToolSide

Arguments:	ucut outl tool clampArea s1
Constraints:	IsOfType(LeftRTool, tool)
	IsOfType(ProcArea, clampArea)
	NotSame(outl, clampArea)
	IsOfType(Outline, outl)
	IsOfType(Undercut, ucut)
	IsOfType(WpieceSide1, s1)
Effects:	+processedUcutHalf1(ucut)
	-unprocUcutHalf1(ucut)
Preconditions:	+processed(outl)
	+toolHeld(tool)
	+clampTurn(s1)
	+subarea(ucut, outl)
	+available(tool)
	+neighbour(s1, clampArea)

MachineUndercutH1RightToolOutline

Arguments:	ucut outl tool clampArea s2
Constraints:	IsOfType(RightRTool, tool)
	IsOfType(ProcArea, clampArea)
	NotSame(outl, clampArea)
	IsOfType(Outline, outl)
	IsOfType(Undercut, ucut)
	IsOfType(WpieceSide2, s2)
Effects:	+processedUcutHalf1(ucut)
	-unprocUcutHalf1(ucut)
Preconditions:	+processed(outl)
	+toolHeld(tool)
	+clampTurn(clampArea)
	+subarea(ucut, outl)
	+available(tool)
	+neighbour(s2, clampArea)

MachineUndercutH1RightToolSide

Arguments:	ucut outl tool clampArea s2
Constraints:	IsOfType(RightRTool, tool)
	IsOfType(ProcArea, clampArea)
	NotSame(outl, clampArea)
	IsOfType(Outline, outl)
	IsOfType(Undercut, ucut)
	IsOfType(WpieceSide2, s2)
Effects:	+processedUcutHalf1(ucut)
	-unprocUcutHalf1(ucut)
Preconditions:	+processed(outl)
	+toolHeld(tool)
	+clampTurn(s2)
	+subarea(ucut, outl)
	+available(tool)
	+neighbour(s2, clampArea)

MachineUndercutH2LeftToolOutline

Arguments:	ucut outl tool clampArea s2
Constraints:	IsOfType(LeftRTool, tool)
	IsOfType(ProcArea, clampArea)
	NotSame(outl, clampArea)
	IsOfType(Outline, outl)
	IsOfType(Undercut, ucut)
	IsOfType(WpieceSide2, s2)
Effects:	+processedUcutHalf2(ucut)
	-unprocUcutHalf2(ucut)
Preconditions:	+processed(outl)
	+toolHeld(tool)
	+clampTurn(clampArea)
	+subarea(ucut, outl)
	+available(tool)
	+neighbour(s2, clampArea)

MachineUndercutH2LeftToolSide

Arguments:	ucut outl tool clampArea s2
Constraints:	IsOfType(LeftRTool, tool)
	IsOfType(ProcArea, clampArea)
	NotSame(outl, clampArea)
	IsOfType(Outline, outl)
	IsOfType(Undercut, ucut)
	IsOfType(WpieceSide2, s2)
Effects:	+processedUcutHalf2(ucut)
	-unprocUcutHalf2(ucut)
Preconditions:	+processed(outl)
	+toolHeld(tool)
	+clampTurn(s2)
	+subarea(ucut, outl)
	+available(tool)
	+neighbour(s2, clampArea)

MachineUndercutH2RightToolOutline

Arguments:	ucut outl tool clampArea s1
Constraints:	IsOfType(RightRTool, tool)
	IsOfType(ProcArea, clampArea)
	NotSame(outl, clampArea)
	IsOfType(Outline, outl)
	IsOfType(Undercut, ucut)
	IsOfType(WpieceSide1, s1)
Effects:	+processedUcutHalf2(ucut)
	-unprocUcutHalf2(ucut)
Preconditions:	+processed(outl)
	+toolHeld(tool)
	+clampTurn(clampArea)
	+subarea(ucut, outl)
	+available(tool)
	+neighbour(s1, clampArea)

MachineUndercutH2RightToolSide

Arguments:	ucut outl tool clampArea s1
Constraints:	IsOfType(RightRTool, tool)
	IsOfType(ProcArea, clampArea)
	NotSame(outl, clampArea)
	IsOfType(Outline, outl)
	IsOfType(Undercut, ucut)
	IsOfType(WpieceSide1, s1)
Effects:	+processedUcutHalf2(ucut)
	-unprocUcutHalf2(ucut)
Preconditions:	+processed(outl)
	+toolHeld(tool)
	+clampTurn(s1)
	+subarea(ucut, outl)
	+available(tool)
	+neighbour(s1, clampArea)

MakeToolHolderFree

Arguments:	tool
Constraints:	IsOfType(Tool, tool)
Effects:	+toolHolderFree()
	-toolHeld(tool)
Preconditions:	+toolHeld(tool)

TapHole

Arguments:	hole outl aTool clampArea
Constraints:	IsOfType(TappingTool, aTool)
	IsOfType(Outline, outl)
	IsOfType(ProcArea, clampArea)
	IsOfType(Hole, hole)
	NotSame(outl, clampArea)
Effects:	+processed(hole)
	-unprocessed(hole)
Preconditions:	+processed(outl)
	+clampNoTurn(clampArea)
	+toolHeld(aTool)
	+subarea(hole, outl)
	+available(aTool)

D Specification of a Problem

The following description corresponds to the symbolic specification of the workpiece shown in figures 2 and 3.

Objects:

WpieceSide1(s1)
LeftOutline(outlL)
CenterOutline(ctr)
Undercut(hint1, hint2)
Hole(drill1)
Thread(thr1)
RightOutline(outlR)
WpieceSide2(s2)
IntProcArea(ipa)
LeftRTool(lrt)
RightRTool(rrt)
ThreadTool(tt)
DrillTool(dt)
IntRotaryTool(irt)

Finish:

1: +processed(thr1)
2: +processedUcutHalf1(hint2)
3: +processedUcutHalf1(hint1)
4: +processed(ctr)
5: +processed(outlL)
6: +processed(drill1)
7: +processed(outlR)
8: +processed(ipa)
9: +processed(s1)
10: +processed(s2)
11: +processedUcutHalf2(hint2)
12: +processedUcutHalf2(hint1)

Orderings:

$(2 \prec 1)$
$(3 \prec 1)$
$(4 \prec 1)$
$(4 \prec 3)$
$(4 \prec 6)$
$(4 \prec 12)$
$(5 \prec 1)$
$(6 \prec 1)$
$(7 \prec 1)$
$(7 \prec 2)$
$(7 \prec 11)$
$(8 \prec 1)$
$(8 \prec 7)$
$(10 \prec 8)$
$(10 \prec 1)$
$(11 \prec 1)$
$(12 \prec 1)$

Start:

+isClampArea(outlR)
+isClampArea(outlL)
+unrestrictedClamp(outlL)
+unrestrictedClamp(outlR)
+noSubareaThread(outlR)
+noSubareaThread(ctr)
+subarea(thr1, outlL)
+subarea(drill1, ctr)
+subarea(hint1, ctr)
+subarea(hint2, outlR)
+subarea(ipa, s2)
+neighbour(s1, outlL)
+neighbour(outlL, s1)
+neighbour(outlL, ctr)
+neighbour(ctr, outlL)
+neighbour(ctr, outlR)
+neighbour(outlR, ctr)
+neighbour(outlR, s2)
+neighbour(s2, outlR)
+available(tt)
+available(lrt)
+available(dt)
+available(irt)
+available(rrt)
+toolHolderFree()
+unprocessed(thr1)
+unprocUcutHalf1(hint2)
+unprocUcutHalf2(hint2)
+unprocUcutHalf1(hint1)
+unprocUcutHalf2(hint1)
+unprocessed(ctr)
+unprocessed(outlL)
+unprocessed(drill1)
+unprocessed(outlR)
+unprocessed(ipa)
+unprocessed(s1)
+unprocessed(s2)

E A Complete Solution Plan

The following figure shows a complete solution plan for machining the workpiece shown in figures 2 and 3.

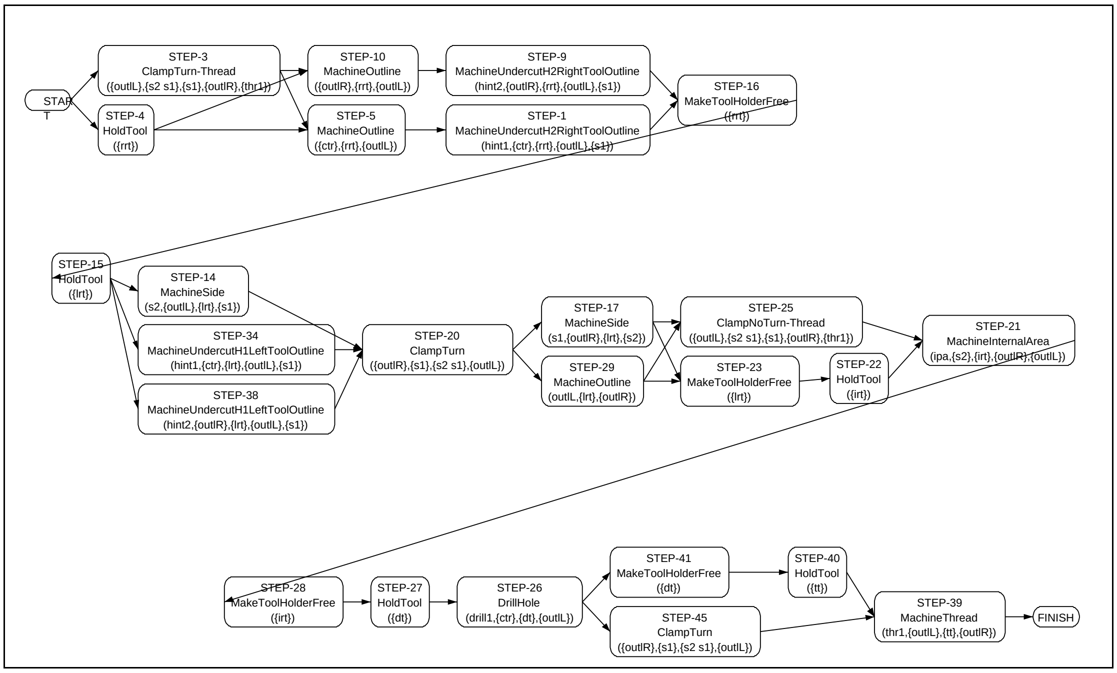

Figure 19: A complete solution plan