Extending Problem Specifications for Plan-Space Planners

Frank Weberskirch and Héctor Munõz-Avila

University of Kaiserslautern, Dept. of Computer Science
P.O. Box 3049, D-67653 Kaiserslautern, Germany
E-mail: weberski@informatik.uni-kl.de

Abstract

Problem specifications for classical planners based on a STRIPS-like representation typically consist of an initial situation and a partially defined goal state. Hierarchical planning approaches, e.g., Hierarchical Task Network (HTN) Planning, have not only richer representations for actions but also for the representation of planning problems. The latter are defined by giving an initial state and an initial task network in which the goals can be ordered with respect to each other. However, studies with a specification of the domain of process planning for the plan-space planner CAPLAN (an extension of SNLP) have shown that even without hierarchical domain representation typical properties called goal orderings can be identified in this domain that allow more efficient and correct case retrieval strategies for the case-based planner CAPLAN/CBC.

Motivated by that, this report describes an extension of the classical problem specifications for plan-space planners like SNLP and descendants. These extended problem specifications allow to define a partial order on the planning goals which can interpreted as an order in which the solution plan should achieve the goals. These goal ordering can theoretically and empirically be shown to improve planning performance not only for case-based but also for generative planning. As a second but different way we show how goal orderings can be used to address the control problem of partial order planners. These improvements can be best understood with a refinement of Barrett's and Weld's extended taxonomy of subgoal collections.

LSA-Report LSA-98-02E

Centre for Learning Systems and Applications
Kaiserslautern, Germany

Contents

1 Introduction

Problem specifications for planners traditionally consist of an initial situation and the goals that have to be achieved. Classical planners as well as newer ones with STRIPS-like (Fikes and Nilsson, 1971) representation of situations and actions in most cases use this way to describe planning problems.

Hierarchical planning approaches, in particular, *Hierarchical Task Network (HTN)* (Erol, 1995; Erol et al., 1996) use extended representations for domains, problems, and plans.[1] The effect of this can be summarized by the fact that the HTN language is strictly more expressive than the STRIPS language (Erol et al., 1994): the domain writer has strictly more possibilities to represent elements of a planning domain and he can code more knowledge about domains and problems. HTN planning starts with a problem specification consisting of an initial state and an initial task network (instead of unordered goals). The initial HTN, that can be compared with the initial plan in partial-order planning, captures more information than the initial situation and the goals. In the initial HTN the goals of the problem are represented as so-called *complex tasks* and these tasks can be ordered with respect to each other. The initial task network then is refined gradually by the planner. Operators in the HTN approach refine complex tasks into more primitive ones. Primitive tasks are comparable to actions in partial-order planning. The solution to an HTN planning problem is a refinement of the initial task network in which we have only primitive tasks. This solution will achieve the goals in accordance with the ordering of the complex tasks in the initial task network.

Motivated by this possibility to represent additional knowledge about a planning problem and its solutions we describe a similar extension for the non-hierarchical planning algorithm SNLP (McAllester and Rosenblitt, 1991) which is implemented in the system CAPLAN (Weberskirch, 1995). The basic idea is to extend the standard problem specifications consisting of the initial situation and the goals by adding information about the order in which goals have to be achieved in a solution plan. The orderings, called *goal orderings*, are thought as additional knowledge that should help the planning system to find a solution. One interpretation of goal orderings is the same as in HTN planning when two initial tasks are ordered with respect to each other. This allows to take profit from ordering information for a certain problem without having to do planning with an hierarchical planner which also needs a hierarchical domain specification. Goal orderings may come from different sources: similar to the HTN approach, the user that formulates planning problems can give this additional knowledge or domain-dependent or domain-independent mechanisms may compute such goal orderings based on a problem specification.

Previous work in the area of case-based planning has shown that it is an important enhancement if there is additional information about goal orderings. In CAPLAN/CBC such goal orderings lead to significantly improved indexing and retrieval mechanisms for the case base (Muñoz-Avila and Weberskirch, 1996a; Muñoz-Avila, 1998). This work also demonstrates that in the domain of process planning we can identify such goal orderings based on geometrical analysis. The resulting goal orderings will apply to all solutions of the problem. More generally, in domains where the subgoals of a problem typically have a lot of interactions, these goal orderings have shown more accurate ways to handle the problem of case retrieval.

There is another aspect that is concerned by this report. Controlling a partial-order planner is a big challenge in generative planning even after all these years of research. SNLP does not

[1] Older HTN-like planners like SIPE (Wilkins, 1988) or O-Plan (Currie and Tate, 1991) also offer such extended representational possibilities but typical concepts of HTN planning and their relation to partial-order planning is formalized less clear there.

say anything about a good control strategy (i.e., goal selection and operator selection) for a plan-space planner, but there has been a lot of work in that area and several control strategies have been developed that show good performance in many situations. A good overview and comparison can be found in (Pollack et al., 1997). Most popular ones are DSep (Peot and Smith, 1993), LCFR (Joslin and Pollack, 1994), ZLIFO (Gerevini and Schubert, 1996) which are also available for CAPLAN in form of control plug-ins called control components.

With respect to goal orderings three important questions come up now:

1. Can a generative planner take profit from goal orderings similar to the case-based planner CAPLAN/CBC?
2. How can such a goal ordering $<_G$ be used to make the planner work more efficiently in term of the size of the search space? Can we define a control strategy based on $<_G$?
3. Where does $<_G$ come from? How can goal orderings be computed automatically?

In this report, we will concentrate on giving an answer to the first two questions.[2] In fact, not only case-based planning but also generative planning can be improved significantly using the additional knowledge about goal orderings. The planner CAPLAN on which the case-based planning system CAPLAN/CBC is based implements mechanisms for defining extended problem specifications and for improving generative planning performance. The improvement is achieved by two different starting-points: goal orderings are translated into ordering constraints between plan steps, and they are used to define a goal selection strategy for the planner. The work reported here is fully implemented in the CAPLAN system and is available from the author or the CAPLAN homepage at `http://wwwagr.informatik.uni-kl.de/~caplan`.

The next pages are organized as follows: Section 2 summarizes all notations and concepts that are important for describing the improvements of the SNLP-like planner CAPLAN. Section 3 defines the key extension of problem specifications by goal orderings. Section 4 shows why planning can be expected to be more efficient and how this can be implemented based on an SNLP-like planning algorithm. Section 5 empirically proves the gain of efficiency in various domains. Finally, some conclusions about this work and future work are made.

2 Notations

This section is intended to give an overview of notations and concepts that are important for the following sections. CAPLAN is an SNLP-like planner that implements these concepts. We will give a short review of the representation of domains, planning problems and plans.

2.1 Representation of Planning Domains

CAPLAN uses a STRIPS-like (Fikes and Nilsson, 1971) representation for planning domains. Domains typically consist of specifications of *types*, *predicates* and *operators*.

Predicates and types define the main language elements that can be used in the definition of atomic operations and situations in the world. Predicates define the possible sentences for the definition of preconditions, effects and situations. Only sets of literals (positive or negative atomic sentences) are allowed in these definitions. The explicit representation of

[2]The third question will be tackled in near future in some other report or paper.

types are one of the major differences between SNLP and CAPlan (Weberskirch, 1995; Weberskirch and Muñoz-Avila, 1997). Types can be organized in hierarchies with single or multiple inheritance. The current version of CAPlan (v2.15) allows types to occur in three places:

- they are used in operator specifications in form of *type constraints*,
- all predicates allow to specify a *type for each argument (argument types)*, which is interpreted as an implicit type constraint wherever the predicate is used,[3] and
- all *objects* of problems and situations are declared to be instances of a certain type.

Operator specifications contain all typical information for STRIPS-like planning, e.g., preconditions and effects. Preconditions and effects are positive or negative literals. These specifications are used to instantiate plan steps each of which represents an action that can be executed in the world. Other details have been described earlier in (Weberskirch, 1995).

2.2 Standard Problem Specifications

Problem specifications for planners based on the STRIPS paradigm traditionally consist of a description of the *initial situation* in the world and of the *planning goals*, i.e., a partial description of the goal situation.

Definition 1 (Problem Specification) *A problem specification* (I, G) *consists of a specification of the initial state* I *and of the planning goals* G*.*

As stated in Section 2.1, I and G both are sets of literals. Because of the Closed-World Assumption I typically contains only positive literals.

The goal of a planning process is to find a *solution plan* for a planning problem. This is a sequence of actions meeting the following condition:

Definition 2 (Solution) *A* ***solution*** *to a planning problem* (I, G) *is a (totally ordered) sequence of actions (plan steps), which when executed from the initial state* I*, results in a world state in which all goals* G *are satisfied.*

Different classical planning paradigms can be characterized by the means and representations to find solutions for a planning problem, e.g., the requirements and properties of the plan representation (see (Kambhampati et al., 1995; Srivastava and Kambhampati, 1998)).

2.3 Plan Representation

CAPlan is a partial-order planner that is based on the SNLP algorithm (McAllester and Rosenblitt, 1991; Barrett and Weld, 1994). It searches in the space of plans to find a solution plan for a given problem. An introduction to these ideas can also be found in (Weld, 1994).

[3] This is an extension with respect to the version reported in (Weberskirch and Muñoz-Avila, 1997). It can be understood as a way to simplify the definition of domains with type constraints in operators specifications because less type constraints have to be defined explicitly.

2.3.1 Partial Plan and Initial Plan

Plans are the basic structure on which plan-space planners work. They represent sequences of actions in the planning world. Actions are called *plan steps*, sequences of actions are *plans*.

Definition 3 (Partial Plan) *A **partial plan** (S, O, B) consists of a set S of plan steps, a set O of ordering constraints between plan steps, and a set B of variable binding constraints over variables occurring in plans steps.*

The transitive closure over the set of ordering constraints O induces a partial order $\prec_O$ on the plan steps. This implies transitivity and irreflexivity of $\prec_O$. Binding constraints occur during planning as new operator might add new variables. CAPLAN uses codesignation constraints (Chapman, 1987) and type constraints (Weberskirch, 1995; Weberskirch and Muñoz-Avila, 1997) to restrict the set of consistent bindings of such variables. It does not depend on ground instances of operators.

In partial-order planners that are based on the principle of refining partial plans (Kambhampati et al., 1995), a problem (I, G) is represented as a so-called *initial plan* or *null plan* (Weld, 1994). It consists of two plan steps, s_0 and s_∞. s_0 precedes all other plan steps, has no preconditions and its effects are the elements of the initial situation I, s_∞ is preceded by all other operators, has no effects and its preconditions are the planning goals G.

2.3.2 Plan Consistency

For the definition and implementation of the basic planning algorithm, the notion of plan consistency is important as CAPLAN always refines a consistent plan into another consistent plan using the typical plan modification operators of SNLP (Kambhampati et al., 1995): simple establishment, step addition, and threat resolutions operators. It backtracks chronologically if the plan is not a solution and no consistent refinement (i.e., a refinement that leads to a consistent plan) can be found.

Definition 4 (Plan Consistency) *A partial plan (S, O, B) is said to be **consistent** if the following conditions hold:*

- ***Ordering consistency:*** *s_0 is the first, s_∞ is the last step, irreflexivity holds, and the orderings must not contain cycles, i.e.,*
  - $\forall s \in S : s_0 \prec_O s \wedge s \prec_O s_\infty \wedge \neg(s \prec_O s)$ *and*
  - $\forall s, t \in S, s \neq t :$ *if $s \prec_O t$ then $t \prec_O s$ cannot be true.*
- ***Binding consistency:*** *The set of binding constraints B is consistent.*[4]

Consistency is tested each time a plan refinement operator has been applied and only consistent plans are considered for further refinement.

2.3.3 Solution Plan

The goal of the planning process is to find a solution plan, i.e., a plan satisfying the solution criterion (Definition 2). The planning process starts with the initial plan which is to be refined until a solution is found. SNLP defines two conditions for a solution plan:

[4] The problem of deciding whether a set of binding constraints B is consistent is not examined here in more detail. The rest of this reports assumes the existence of a mechanism to test constraint consistency.

Definition 5 (SNLP Solution) *A plan (S, O, B) is considered to be a* ***solution*** *to a planning problem $prb = (I, G)$ by SNLP if it is consistent and complete. $solutions(prb)$ denotes the set of all solutions obtained by SNLP for the problem prb.*

The required completeness of plans is defined such that a partial plan (S, O, B) is said to be **complete** (McAllester and Rosenblitt, 1991) for a planning problem (I, G) if every topological sort of it is a solution to (I, G) as given by Definition 2. For the implementation of this completeness criterion the structure of a partial plan is extended by some bookkeeping to make it become operational. SNLP uses the concept of *causal links* to represent causal dependencies between plan steps. Interactions between plan steps are called *threats* (see (McAllester and Rosenblitt, 1991)).[5]

Throughout the following pages we will use the following notation:

- $s_i \rightarrow p@s_j$ denotes a *causal link* between steps s_i and s_j to establish $p@s_i$ (precondition p of step s_i). It implies the ordering $s_i \prec_O s_j$ and s_i is called the *establisher of* $p@s_j$ (short: $establisher(p@s_j) = s_i$). If no causal link exists for a precondition p, it is called an *open goal.* Initially, the preconditions of s_∞ are open goals.
- A *threat* between a plan step s_t and a causal link is denoted as $s_t \leftrightarrow (s_i \rightarrow p@s_j)$.

Now we can give an operational definition of a complete plan that is implemented in the SNLP planning algorithm:

Definition 6 (Complete Plan) *A partial plan (S, O, B) is said to be* ***complete*** *for a planning problem (I, G) if for every precondition p of a plan step s, there exists a causal link $s' \rightarrow p@s$ and all threats to such causal links are resolved.*

This also gives an operational criterion for SNLP to find a solution plan which is implemented in the SNLP algorithm and descendants like CAPLAN: the algorithms tries to create a causal link for every precondition of a plan step. (McAllester and Rosenblitt, 1991; Barrett and Weld, 1994) described the base algorithm, (Kambhampati et al., 1995; Kambhampati et al., 1998) presented a kind of abstraction unifying partial-order, state-space and HTN plan refinements in one algorithm.

3 Extending Standard Problem Specifications

After the last section summarized notations and concepts used in partial-order planning based on SNLP, this section now will describe one extension of these concepts that is implemented in the planner CAPLAN. CAPLAN allows what we call *extended problem specifications*, i.e., problem specifications that contain some more information than the standard problem specifications of traditional planners with STRIPS-like specification languages.

Extended problem specifications have already been mentioned earlier (Weberskirch, 1995; Muñoz-Avila and Hüllen, 1995; Muñoz-Avila and Weberskirch, 1996a). In the case-based planning system CAPLAN/CBC, which is based on the planner CAPLAN, this extension plays a key role for being able to build an indexing structure for a case base that allows a more accurate and efficient retrieval of cases. Here we will especially focus on the advantages of extended problem specifications for pure generative planning with CAPLAN.

[5]If it is necessary to refer to elements of additional bookkeeping we will use the notation (S, O, B, L) for a plan (SNLP plan). It extends a normal plan by a set L of causal links. However, in most cases we will simply write (S, O, B) if we speak about plans but as far as CAPLAN is concerned we always mean SNLP plans.

3.1 Extended Problem Specifications

Section 2.2 defined the standard way of specifying a planning problem in planning that is based on the STRIPS formalism. The only information that is given besides an specification of the planning domain (see Section 2.1) is an initial situation of the world and a set of goals that have to be achieved. CAPLAN extends this by allowing to specify orderings between planning goals, called *goal orderings.*

Definition 7 (Extended Problem Specification) *Given a problem (I, G) then the triple $(I, G, <_G)$ is called an* ***extended problem description****. $<_G$ is a set of ordering constraints between goals from G, called* ***goal orderings****, and they induce a (partial) order on the set of planning goals G. If goals g_i, g_j are ordered, we write $g_i <_G g_j$.*[6]

This definition declares the basic syntax of extended problem specifications as used by CAPLAN. Now we have to define the semantics. The additional goal orderings are thought as extra information that should help to find a solution more quickly (if $<_G \neq \emptyset$). If we consider the typical differences between plan-space and state-space planning there are two possible interpretations of goal orderings that have not been distinguished explicitly in literature until now. Goal orderings can be interpreted as an order

- *in which goals are achieved in any plan* that is considered to be a solution, or
- *in which the planner should process goals* during a planning episode.

In this and the next section we will concentrate on the first one, which leads to the so-called ordered establisher condition that is implemented in CAPLAN. This idea has also been subject of previous planning research. For example, (Irani and Cheng, 1987; Cheng and Irani, 1989) already developed a definition of subgoal ordering constraints that is motivated by the first interpretation. The second interpretation addresses the control problem of partial-order planning can also be found in literature, for example (Korf, 1987; Drummond and Currie, 1989; Barrett and Weld, 1994). We will investigate it later.

3.1.1 The Ordered Establisher Condition

A basic idea is that the existence of goal orderings has effects on the notion of consistency of a plan for CAPLAN (not for SNLP). Goal orderings put additional constraints on a plan with respect to the ordering of establishers of goals occurring in goal orderings (called *ordered goals*). The following definition makes these additional constraints concrete as they lead to a more restricted notion of plan consistency.

Definition 8 (Plan Consistency modulo $<_G$) *Given a problem $(I, G, <_G)$, a partial plan (S, O, B) is said to be* ***consistent modulo*** *$<_G$ if the following conditions hold:*

- *(S, O, B) is a consistent plan as given by Definition 4*

[6]Throughout the report we treat $<_G$ as an transitive, irreflexive, binary relation and use it as follows: (1) if we write $(g_i, g_j) \in <_G$ then we mean the explicit ordering relation between two goals; (2) $g_i <_G g_j$ says that the two goals are ordered; (3) $<_G$ is transitive, i.e., if we define a certain ordering as $<_G = \{g_1 <_G g_2, g_2 <_G g_3\}$ also $g_1 <_G g_3$ holds.

- ***Ordered establisher condition:*** *The establishers of ordered goals are ordered in the same way as the goals, i.e.,* $\forall g_i, g_j \in G : g_i <_G g_j \Rightarrow establisher(g_i) \prec_O establisher(g_j)$

Figure 1 illustrates the important second condition of this definition. If there are two goals g_1, g_2 for which a goal ordering $g_1 <_G g_2$ is specified, the ordered establisher condition says that the plan steps that establish these two goals, s_1 and s_2, must be ordered in the same way, $s_1 \prec_O s_2$. This means that the ordering between goals is translated into an ordering between plan steps. A simple consequence is that consistency modulo $<_G$ is more restrictive than normal consistency.

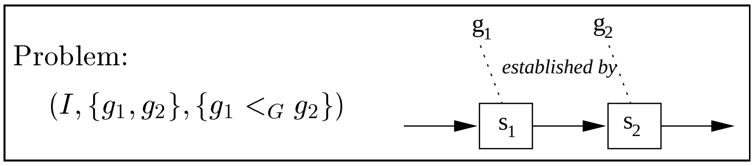


Figure 1: Additional Ordering Constraints because of Goal Orderings

As SNLP allows to identify the unique establisher for each goal in a certain solution plan (completeness of solutions, Definition 5 and 6), we can identify the set of ordering restrictions between plan steps which is induced by a goal ordering by the ordered establisher condition.

Definition 9 (Establisher Ordering) *Given a problem* $(I, G, <_G)$ *and a solution plan* $P = (S, O, B)$ *that is consistent modulo* $<_G$*. Then the* ***establisher ordering induced by*** $<_G$*,* $estabOrd(<_G, P) \subseteq \prec_O$*, is defined as the unique set of plan step orderings that are induced for* $<_G$ *in* P *by the ordered establisher condition:*

$$\forall g_i, g_j \in G : g_i <_G g_j \Rightarrow (establisher(g_i), establisher(g_j)) \in estabOrd(<_G, P)$$

Having these definitions we can state the consequences of goal orderings with an extended solution criterion as it is used in CAPlan:

Definition 10 (Solution modulo $<_G$) *Given a problem* $prb = (I, G, <_G)$*, then a plan* (S, O, B) *is considered to be a* ***solution modulo*** $<_G$ *if it is consistent modulo* $<_G$ *and complete. The notation* $solutions_{CAPlan}(prb)$ *is used for the set of all solutions modulo* $<_G$ *for a problem* prb*.*

The difference between SNLP and CAPlan with respect to extended problem specifications is now clear: CAPlan only considers a plan to be a solution if it doesn't violate any of the additional ordering constraints imposed by the given goal orderings.

A new class of plans. In terms of (Kambhampati et al., 1996) that will also be referred to in other sections, the definition of plan consistency modulo $<_G$ formalizes a new class of plans. Kambhampati categorizes the plans produced by SNLP to be *elastic protected plans* because key properties of plans created by SNLP are the partial order on plan steps, the fact that causal dependencies are recorded by causal links (alias IPCs) and protected against interacting steps.
The solution criterion of CAPlan when using the ordered establisher condition (Definition 10) describes a new subclass of plans, the class of ***elastic protected*** $<_G$***-consistent plans***, which the planner CAPlan is capable to create using goal orderings.

3.1.2 Comparison with Initial HTNs

The introduction motivated goal orderings and extended problem specifications by drawing comparisons with HTN planning where we have an initial task network being part of a problem specification (input task network, see (Erol, 1995)). Tasks in an initial task network are equivalent to goals of a standard problem specification as used in CAPLAN. As goals are not represented in partially ordered plans in a similar way as tasks in HTNs they are mapped to plan steps by the definition of plan consistency modulo $<_G$. Here we want to explain the differences between ordered tasks in a an input task network and ordered goals in an extended problem specification $(I, G, <_G)$ in more detail.

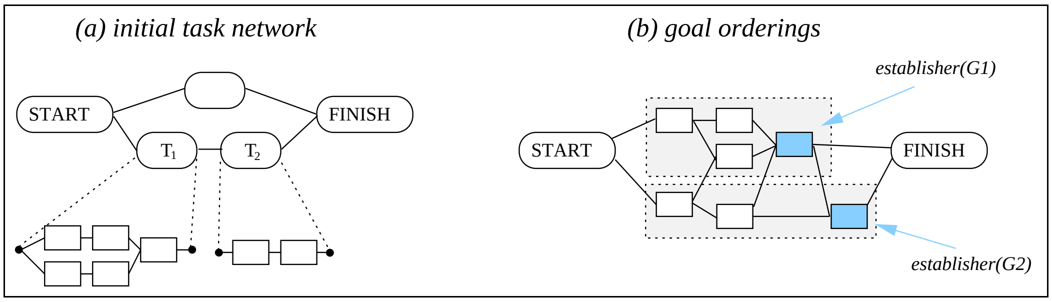


Figure 2: Difference between HTN planning and goal orderings of CAPLAN

Figure 2.(a) shows an initial task network for a problem with two tasks T_1, T_2, where T_1 is ordered before T_2. This is comparable to the initial knowledge that a goal G_1 must be achieved before a goal G_2 for the non-hierarchical planner CAPLAN (Figure 2.(b)). But there is a difference: an ordering between two tasks T_1, T_2 forces all steps in further decompositions of T_1 to be ordered before all steps of decompositions of T_2. A goal ordering $G_1 <_G G_2$ of CAPLAN is less restrictive because it only forces the establishers of the two goals to be ordered, but it doesn't say anything about the ordering of other steps that result from establishing preconditions of those establishers. These steps might for example still be interleaved.

3.2 Types of Goal Orderings

The definitions of plan consistency modulo $<_G$ states the effect of goal orderings on plan consistency for CAPLAN, but it does not say anything about which goal orderings are to be given for certain problem (I, G) or which types of them exist. It only says that, if goal orderings are given, the ordered establisher condition will affect what is considered to be a valid solution. Here we will look at different types of goal orderings in more detail.

3.2.1 Necessary and Possible Goal Orderings

With respect to the ordered establisher condition of Section 3.1.1 we distinguish between orderings that do not exclude solutions from the search space (called *necessary goal orderings* because they are true in all solutions) and orderings that have effects on the set of possible solutions that can be found by the planner (we call them *possible goal orderings*, because they are not valid for all solutions).

Definition 11 (Necessary Goal Orderings) *Given a problem (I, G), then a set of goal orderings $<_G$ is called a* ***necessary goal ordering****, written as necessary$((I, G), <_G)$, if it*

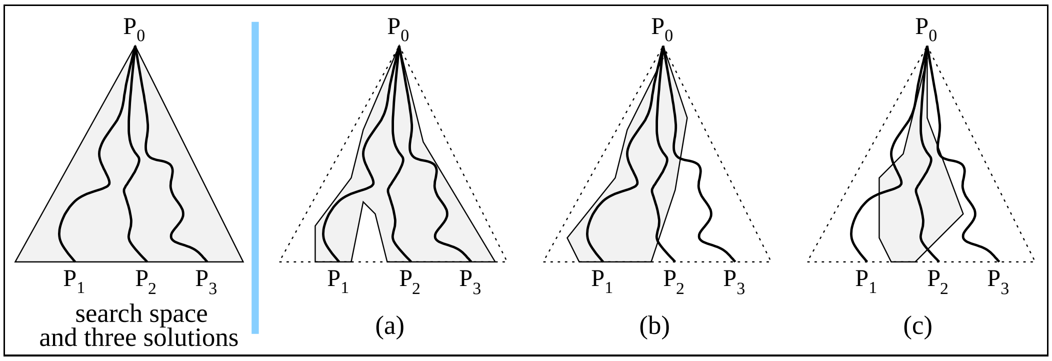


Figure 3: Effective search space for different types of goal orderings: (a) necessary goal orderings, (b) possible ordering w.r.t. P_1, (c) impossible goal orderings

satisfies the condition

$$solutions_{CAPlan}((I, G, <_G)) = solutions((I, G)).$$

Definition 12 (Possible Goal Orderings) *Given a problem (I, G) and a solution plan P, then a set of goal orderings $<_G$ is called a* ***possible goal ordering*** *w.r.t. P, written as $possible((I, G), P, <_G)$, if it satisfies the condition*

$$P \in solutions_{CAPlan}((I, G, <_G)) \subset solutions((I, G)).$$

Definition 13 (Impossible Goal Orderings) *Given a problem (I, G), then a set of goal orderings $<_G$ is called* ***impossible goal ordering****, written as $impossible((I, G), <_G)$, if it satisfies the condition*

$$solutions_{CAPlan}((I, G, <_G)) = \emptyset \neq solutions((I, G))$$

These definitions allow to categorize a goal ordering $<_G$ that is given by $(I, G, <_G)$: it is either a necessary, a possible or an impossible ordering. Figure 3 illustrates this by showing the characteristic difference of the search space:[7]

- **Necessary orderings:** These orderings naturally follow from each solution plan that can be found by SNLP for (I, G), i.e., there exists no solution plan that violates the ordering restrictions (Figure 3.(a)). So, the set of solutions doesn't change when using the necessary orderings, but the search space can be expected to get smaller.

- **Possible orderings:** Possible orderings define a certain *preference* to be used in searching for a solution plan. We can call them *useful* for finding at least that particular solution (Figure 3.(b)) as they reduce the search space to a part that at least will contain that solution. Such orderings typically will be given by a user.

- **Impossible orderings:** If a preference is too restrictive the set of solutions for CAPlan becomes empty (Figure 3.(c)). Such orderings can be characterized to induce establisher orderings that are inconsistent with the plan step orderings of any solution.

For CAPlan the distinction between these types of goal orderings is not important from an implementation point of view. They are treated in the same way as given by Definition 8.

[7]Figure 3 shows the search space for a problem with three solutions P_1, P_2, P_3 as a triangle and how it can be reduced with goal orderings. The path from P_0, the initial plan, to a P_i is indicated by a line and represents the process of refining partial plans until the solution is found.

Nevertheless, the distinction helps to understand the effects of using them. It makes no sense for CAPlan to use impossible orderings for planning as defined by the ordered establisher condition;in practice, CAPlan allows to recognize a special case of impossible orderings, i.e., sets of goal orderings containing cycles, and does not start planning at all in that case. Below we will however show, that impossible orderings as defined above also might be useful for other purposes than the ordered establisher condition. We will speak about possible orderings mainly in case these orderings reduce the set of possible solution plans, but still are not impossible orderings.

3.2.2 Extracting Goal Orderings A-Posteriori

Until now we only have a definition of what is considered to be a necessary or possible goal ordering but not at least a theoretical way to compute them.

Imagine the following steps to extract goal orderings a-posteriori for a problem (I, G):

1. Let the planner find an SNLP solution $P \in \mathit{solutions}((I, G))$ for the problem (I, G), i.e., without using any goal orderings.
2. Extract the goal orderings *a-posteriori* from the obtained solution $P = (S, O, B)$, i.e., look for ordering relations between establishers of top-level goals $g_i@s_\infty$ in plan P (an algorithm is specified in Figure 4).

```
extractOrderings((S,O,B), (I,G))
orderings:={}
for each g1 ∈ G do
   for each g2 ∈ G − {g1} do
      if establisher(g1) ≺O establisher(g2) then
         orderings:=orderings ∪ {g1 <G g2}
      endif
   done
done
return orderings
```

Figure 4: Algorithm for the extraction of goal orderings. Notice, that $\prec_O$ is the order on the plan steps S induced by O.

The algorithm `extractOrderings` computes the maximal set of possible goal orderings to obtain a certain solution plan P. It infers goal orderings from the plan step orderings found in a solution. This is the maximal amount of information that can be provided with goal orderings for a problem. In general, these orderings are only possible orderings because there can exist different solutions from which different sets of goal orderings can be extracted. But each set of orderings is maximal w.r.t. the plan they were extracted from.

Definition 14 (Maximal Possible Orderings) *Given a problem (I, G), then a goal ordering $<_G$ is called the maximal set of possible goal orderings w.r.t. a solution $P = (S, O, B)$, written as $<_G = \mathit{maxPossible}((I, G), P)$, if the following condition is satisfied:*

$$\forall g_i, g_j \in G : \mathit{establisher}(g_i) \prec_O \mathit{establisher}(g_j) \Rightarrow g_i <_G g_j$$

Notice, that $\mathit{maxPossible}((I, G), P)$ always contains all orderings of the transitive closure because it is based on the order $\prec_O$ between plan steps. Notice further the difference to the

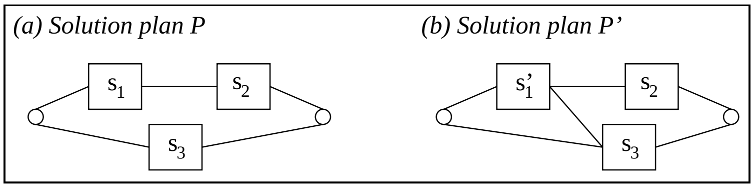


Figure 5: Maximal possible orderings w.r.t. different solutions P, P' for a problem (I, G), each goal g_i is established by the corresponding plan step s_i or s'_i. (a) $maxPossible((I,G),P) = \{g_1 <_G g_2\}$, (b) $maxPossible((I,G),P') = \{g_1 <_G g_2, g_1 <_G g_3 \supset maxPossible((I,G),P)\}$

ordered establisher condition (Definition 8), which forces orderings in a $<_G$-consistent plan because of goal orderings. Here we have the opposite direction: orderings between establishers define the elements of $maxPossible((I,G),P)$. This set is the maximal set of goal orderings such that the SNLP solution P can also found by CAPLAN. It still might happen that a superset of $maxPossible((I,G),P)$ allows to find a different solution plan $P' \neq P$ (Figure 5 gives a simple example). Further, the following is easy to see: if $<= maxPossible((I,G),P)$ and $< \subset <'$ then we have either $\exists P' : P' \in solution((I,G),<')$ or $impossible((I,G),<')$.

There are a number of other relations between necessary/possible goal orderings and maximal sets of possible goal orderings.

Lemma 1 *Given a problem $prb = (I,G)$ then for any goal ordering $<_G$ the following statements are true:*

1. *$maxPossible(prb,P)$ is unique for prb and P*
2. *$\forall P \in solutions(prb) : possible(prb,<_G,P) \Rightarrow (<_G \subseteq maxPossible(prb,P))$*
3. *$\forall P \in solutions(prb) : necessary(prb,<_G) \Rightarrow (<_G \subseteq maxPossible(prb,P))$*

Proof: See Appendix B.1, page 53.

Goal orderings and case-based planning: A-posteriori orderings also play a considerable role in the case-based extension of CAPLAN, where these orderings are important for indexing a case. Cases for CAPLAN/CBC, i.e., a problem (I,G), a solution P, and some information about causal dependencies and justifications of the solution (Muñoz-Avila and Weberskirch, 1996a), are indexed based on the maximal set of possible goal orderings, $maxPossible((I,G),P)$, as determined by the algorithm shown in Figure 4 (see (Muñoz-Avila, 1998) for details).

3.2.3 Maximal Set of Necessary Goal Orderings

Definition 14 introduced the notion of the unique set of maximal possible orderings for a certain solution. With respect to the more interesting necessary goal orderings a similar formation of a concept can be made, because necessary orderings are true in all solutions of a problem and naturally allow to distinguish a maximal set of necessary orderings based on the notion of maximal possible orderings.

Definition 15 (Maximal Necessary Orderings) *Given a problem* (I,G) *and the solution plans* $solutions((I,G)) = \{P_1, \cdots, P_k\}$,[8] *then we have*
$$maxNecessary((I,G)) = \bigcap_{i\in\{1,\cdots,k\}} maxPossible((I,G),P_i).$$

It is obvious that we have $necessary((I,G),<_G)$ for $<_G= maxNecessary((I,G))$ because each ordering from the intersection of the maximal possible orderings for each solution is true in all solutions (this is the requirement for necessary orderings). Further, the set $maxNecessary((I,G))$ is unique as the intersection of sets is unique. It is also worthy of note that the maximal necessary orderings can be empty.[9]

Unfortunately, Definition 15 is an non-operational definition of how to compute necessary orderings and we have to look for ways to get these orderings (or a subset of them) without having to compute every possible solution plan. One method that is based on Operator Graphs (Smith and Peot, 1993) will be documented in near future. Here we show how we can increase planning performance if we know about some goal orderings for a planning problem.

3.2.4 Properties and Examples

Now we will state some properties that clarify the relation between the different sets of goals orderings.

Lemma 2 *Given a problem* (I,G) *in a certain domain, then the following statements about any goal ordering* $<_G$ *are true:*

1. (a) $necessary((I,G),<_G) \Rightarrow (<_G \subseteq maxNecessary((I,G)))$
   (b) $impossible((I,G),<_G) \Rightarrow (\forall P \in solutions((I,G)):$
   $maxPossible((I,G),P) \subset <_G \vee (\exists g_j <_G g_i : g_i < g_j \in maxPossible((I,G),P)))$
2. (a) *There can exist more than one possible goal ordering for a fixed* P.
   (b) *If* $P, P' \in solutions((I,G))$ *are two different solutions, we might have a* $<_G$ *such that* $possible((I,G),<_G,P)$ *and* $possible((I,G),<_G,P')$ *are true.*
3. *If* P, P' *are two different solutions for* (I,G)
   (a) $maxNecessary((I,G)) \subseteq maxPossible((I,G),P) \cap maxPossible((I,G),P')$
   (b) *we might have* $maxPossible((I,G),P) \neq maxPossible((I,G),P')$.

Proof: See Appendix B.1, page 53.

In summary, Figure 6 illustrates graphically the spectrum of possibilities that can occur with goal orderings of different types: On the left side it shows the necessary ordering $maxNecessary((I,G))$ and subsets of it, which are independent of a certain solution plan. On the right side we have possible orderings, which focus on a certain solution plan, e.g., there might be two solution plans P and P'. The maximal sets are the sets of orderings on the right

[8] This set of solutions is finite if simple establishment is preferred to step addition in the planning algorithm, because then SNLP only generates minimal plans. Minimal plans are complete plans which have no complete subplans (Kambhampati, 1995).

[9] This is a typical effect in domains in which all actions are reversible, for example, the logistics transportation domain – we might have several solutions to a problem with completely different possible goal orderings.

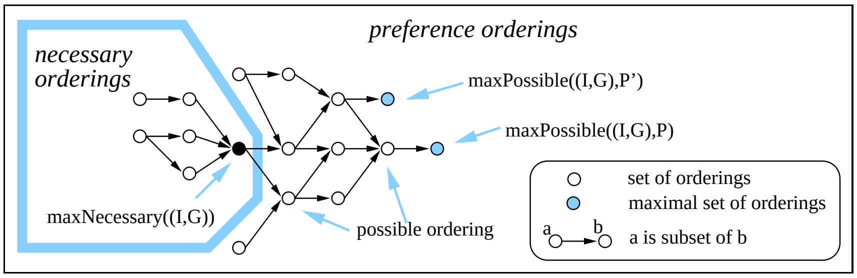


Figure 6: Types of goal orderings and their relations

side (which are not subset of some other set of orderings). For each solution plan P we have exactly one such maximal set, but these maximal sets do not need to be different for different solution plans. Figure 6 shows the case that $maxPossible((I,G),P) \neq maxPossible((I,G),P')$ for two solutions P, P'.

Necessary orderings seem to be most interesting because they apply to all solutions that can be found, i.e, they won't change the set of solutions. In particular, this is important if orderings are computed automatically and they are not intended to focus on a particular kind of solution plan. On the other hand, if a user gives orderings while defining a planning problem, he might think of a certain solution meeting this orderings, but maybe he doesn't know whether his orderings are necessary orderings. In both cases we can expect that the additional knowledge about ordering restrictions will help to find the solution faster because the search space gets smaller.

Example: Let us illustrate the definitions using the artificial domain $\Theta_n D^m S^1$ (Barrett and Weld, 1994) (see Appendix A.2 for details about the typical operators and problems). The only solution for a problem of size n is a plan with a linear sequence of steps s_0, $A_\alpha, A_1^\alpha, \cdots, A_n^\alpha$, s_∞. So we have

$$maxNecessary((I,G)) = \{G_\alpha <_G G_i|_{i=1..n}, G_i <_G G_{i+1}|_{i=1..n-1}\}.$$

All subsets of this set are necessary orderings. Here, the maximal set of necessary goal orderings always orders all goals totally, so, $maxNecessary((I,G)) = maxPossible((I,G),P)$ for P being the solution plan for (I,G).

$\{G_i <_G G_\alpha|_{i=1..n}\}$ is an example of an impossible goal ordering in this domain because there exists no solution meeting this condition.

3.3 Goal Orderings and Serializability

The concept of serializability (Korf, 1987; Barrett and Weld, 1994) has been studied to examine how complicated a planning domain is for a planner that is able to create plans of a certain class of plans (Kambhampati et al., 1996). Serializability is a criterion among many possible others that characterizes a domain by looking at interactions between different subgoals in that domain and counting the number of ways the subgoals can be selected for successfully solving the problem without too much backtracking. Based on this idea collections of subgoals in a domain can be categorized to be *independent*, *trivially serializable*, *laboriously serializable*, or *nonserializable* collections (Korf, 1987; Barrett and Weld, 1994). The classes of independent and trivially serializable subgoal collections are often understood as synonyms for *easy to solve* subgoal collection, the others as *difficult to solve*.

Unfortunately, we will later see that this is too simple. Further, this criterion is a very hard one and many interesting domains can, for example, be shown to be not trivially serializable.

Related to the concept of serializability, we will show some limitations of its expressiveness and we will further refine the extended taxonomy of subgoal collections as presented in (Barrett and Weld, 1993; Barrett and Weld, 1994). We define an extended criterion that is based on goal orderings, but that is weaker than trivial serializability. We will explain how it fits in the taxonomy, how it can be used to improve planning and what are the relations to necessary and possible goal orderings.

3.3.1 Trivial Serializability

We will first summarize some basic definitions around the concept of serializability as presented in (Kambhampati et al., 1996). Consider a plan that solves some planning goal g_1 and suppose there is an additional planning goal g_2 to be solved. A question that arises is whether that plan could be extended to solve both goals.[10]

Definition 16 (Serial Extensibility) *Given a class $\boldsymbol{P}$ of plans, a plan P in $\boldsymbol{P}$ achieving a goal g_1 is **serially extensible** with respect to a second goal g_2 if there is a refinement of P in $\boldsymbol{P}$ achieving both, g_1 and g_2.*

This definition does not say that the plan achieving g_1 and g_2 must be found without any backtracking at all, nor does it say that it is easy to find the plan for g_1 or for both goals. But it says that backtracking must not revise any refinement that has been made to find plan P. So, if a plan P is given for g_1 and we know that it is serially extensible the amount of backtracking is limited to the inference steps needed to solve g_2. The planner only has to solve the problem of combining the plan P with a solution for goal g_2 (knowing that there exists one) but it does not have to figure out whether a plan for both goals exists that is based on P.

Serial extensibility can be extended to a set of goals. The next step is to generalize it by looking at *all* plans for a certain set of goals instead of only one plan. If all plans achieving a certain goal g_1 are serially extensible to a plan for two goals g_1, g_2, then (g_1, g_2) is a suitable order to process the goals regardless of which plan for g_1 is selected by the planner. Further generalized to a set of goals the planner then can be sure that it is possible to find a plan for all goals if it already found a plan for a subset of them and selects goals in this order. In the following definitions we examine *permutations* π of the goals $G = \{g_1, \cdots, g_n\}$, where π is a bijective mapping $\pi : [1 \cdots n] \to [1 \cdots n]$ that orders the goals in a certain way. A permutation on G is denoted as $(g_{\pi(1)}, \cdots, g_{\pi(n)})$. For some permutations on a given set of goals we might have the following property.

Definition 17 (Serialization Order) *Given a class $\boldsymbol{P}$ of plans and a set of goals, $g_1, \cdots, g_n$, a permutation π of these goals is said to be a **serialization order** if every plan in $\boldsymbol{P}$ for solving $g_{\pi(1)}$ can be serially extended to $g_{\pi(2)}$ (modulo the class $\boldsymbol{P}$) and the resulting plan can be serially extended to $g_{\pi(3)}$ (modulo the class $\boldsymbol{P}$) and so on.*

[10]Following (Kambhampati et al., 1996) in the definitions we speak about classes of plans rather that the planning algorithms that are capable of creating such classes of plans. For CAPLAN as for SNLP we have the class of *elastic protected plans* that can be created by the algorithm. If CAPLAN uses the ordered establisher condition it create the subclass of elastic protected $<_G$-consistent plans (Section 3.1.1).

Unfortunately, serial extensibility is not a commutative property (Barrett and Weld, 1994; Kambhampati et al., 1996), so it makes a difference how many of the permutations on a set of goals are a serialization order. The easiest case is given if every permutation on a set of goals is a serialization order.

Definition 18 (Trivial Serializability) *A set of goals, $g_1, \cdots, g_n$, is said to be trivially serializable, if any permutation of these goals is a serialization order (modulo the class $\boldsymbol{P}$). A problem is trivially serializable if this is true for its goals. Similar, a domain is trivially serializable if all problems are trivially serializable.*

This means that it doesn't matter in which order the planner works on the goals, it will always be possible to extend the partial solution found so far for the next goal. This is a nice but hard restriction and many domains can be shown not to be trivially serializable, even for planners that are characterized to be able to create the class of elastic protected plans (Kambhampati et al., 1996) like SNLP. Examples are the artificial domains $\Theta_n D^m S^1$ (Barrett and Weld, 1994) or the Workpiece domain of CAPLAN and CAPLAN/CBC (Muñoz-Avila and Weberskirch, 1996b; Muñoz-Avila, 1998) and many others.

3.3.2 Limited Expressiveness of Serializability

Serializability allows to classify a domain or a problem to be, for example, trivially serializable. But if a permutation of goals is a serialization order (trivial serializability requires all permutations to be a serialization order) this does not imply that it is easy to find a solution in all cases we use this permutation. Serial extensibility only limits the amount of backtracking to the refinements that are necessary to extend a given subplan. Nevertheless, finding an extension (also the extension of the initial plan that solves the first goal) might still be difficult because of other reasons than the fact that we have to extend a given subplan for an new goal. So, if it is easy to solve an arbitrary goal, *'a problem is trivially serializable'* does allow to conclude that the collection of goals is easy to solve. If some individual goal is difficult to solve, a problem can be difficult and also trivially serializable!

Let us look at a simple example. Imagine a problem $prb = (I, G)$ with goals $G = g_1, \ldots, g_n$ which is not trivially serializable, because the permutations π in which $\pi(2) < \pi(1)$, i.e., g_2 is selected before g_1, are not a serialization order, all other permutations are a serialization order. Figure 7 shows one permutation for each case: it is easy to solve g_2 after g_1 is solved (Figure 7.(a)) but the opposite order is difficult because of exponential costs of backtracking (Figure 7.(b)). We can extend this domain by one new goal g^* and one simple new action: $\{g_1, g_2\}\ A^*\ \{g^*\}$.[11] In this extended domain the problem prb can be reformulated to $prb' = (I, G')$ with $G = g^*, g_3, \ldots, g_n$ and it is easy to see solving prb is almost the same as solving prb'. Surprisingly, prb' is trivially serializable, because serial extensibility disregards the complications of solving g^* and goal g^* is easy to solve on top-level because there exists only the one action A^* (Figure 7.(c)). The complications emerge from the preconditions of A^* and they are the same for which problem prb is not trivially serializable.

We can conclude that it does not make sense to characterize a domain by looking some problems but that it is important to look at all possible problems or at the characteristic problems covering all. If we have a permutation π on a set of goals $G = g_1, \ldots, g_n$, we can only say that G is comparable easy if all g_i are easy and π is a serialization order. If at least one goal is difficult to solve, the fact that π is a serialization order allows to conclude that no

[11] Notation: {preconditions} operator name {effects}

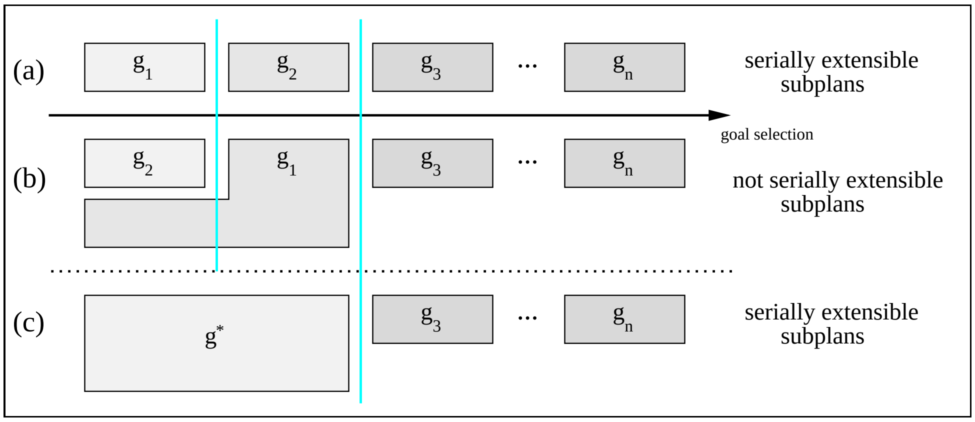


Figure 7: Limited Expressiveness of Serializability. Each area is a measure for the amount of planning effort to find a subplan for a certain goal.

new difficulty emerges because of having to solve the set of goals. If π is not a serialization order, then an additional difficulty is caused just by the fact that a set of goals is to be solved.

The Problem of Goal Subsumption. We found another criterion that might be true in a certain domain that allows to show that the expressiveness of the concept of serializability decreases. For that purpose, we need the notion of subsumption between goals.

Definition 19 (Subsumption of Goals) *Given a problem (I, G) in a certain domain and goals $g_i, g_j \in G$ then the goal g_i subsumes goal g_j, written as $g_i \triangleright g_j$, if every solution plan achieving g_i also achieves g_i, g_j, i.e.,*

$$\forall P : P \in solutions((I, \{g_i\})) \Rightarrow P \in solutions((I, \{g_i, g_j\}))$$

It is easy to see that, if we have $g_i \triangleright g_j$ then (g_i, g_j) is always a serialization order. But, this does not say anything about how easy it is to find a plan for g_j nor does it say anything about how easy it is to find the plan solving both goals. This means, if $g_i \triangleright g_j$ is true the it is only helpful to know whether (g_j, g_i) is a serialization order.

Summary. It is not correct to use the term *trivial serializability* simply as a synonym for *easy to solve* because the concept of serializability does nothing else than saying whether the combination of goals can be responsible for making a problem difficult to solve. If a problem is trivially serializable, this means that the combination of the goals of that problem does not cause an additional kind of difficulty that is not encountered if we only look at the individual goals. Similar, if a problem is laboriously serializable, exactly the problem of combining the solutions for the goals causes a new difficulty regardless of whether individual goals are difficult or not. We gave a simple example: it shows how a problem that is not trivially serializable can be converted into a problem that is trivially serializable simply by adding a new goal and a new operator which moves all complications away from the top-level goals to the preconditions of the new action.

Thus, we should always speak about all problems or the most difficult problems of a domain and be aware that serializability only concerns the question of new complications caused by trying to find a plan for several goals.

3.3.3 $<_G$-Constrained Trivial Serializability

As said above, trivial serializability puts hard constraints on a domain or problem by requiring all permutations of the goals to be a serialization order. The idea new is to weaken the condition of trivial serializability using goal orderings. Similar to serializability this weakened concept characterizes the difficulty of a domain, a problem, or a set of goals relative to the individual goals under the assumption that certain goal orderings are known. This leads to a second way to interpret goal orderings (already mentioned in Section 3.1) where we consider only those permutations of the goals that are an extension of $<_G$.

Definition 20 ($<_G$-Consistent Permutation) *A permutation π on a set of goals, $g_1, \cdots,$ g_n partially ordered by $<_G$, is said to be $<_G$-**consistent** if $\forall g_i, g_j : g_i <_G g_j \Rightarrow \pi(i) < \pi(j)$*

If two goals are ordered then a $<_G$-consistent permutation will preserve this ordering. As the permutations represent orders in which the planner can select the goals, g_i will be selected before g_j in this case. Now we can extend the criterion of trivial serializability to consider only those permutations that are $<_G$-consistent.

Definition 21 ($<_G$-Constrained Trivial Serializability) *A set of goals, $g_1, \cdots, g_n$, partially ordered by $<_G$ is said to be $<_G$-**constrained trivially serializable**, if any $<_G$-consistent permutation of these goals is a serialization order (modulo the class **P**).*

Notice, we have to check less permutations for $<_G$-constrained trivial serializability than for trivial serializability, if $<_G \neq \emptyset$. So, trivial serializability is a harder criterion than $<_G$-constrained trivial serializability and we have the property that, if a set of goals is trivially serializable, it is also $<_G$-constrained trivially serializable (Muñoz-Avila and Weberskirch, 1996b). The opposite direction does not hold.

The restrictions with respect to expressiveness as illustrated in Section 3.3.2 also apply to this new criterion. So, it formalizes the fact that the combination of goals that are processed in an order consistent with $<_G$ does not cause a new difficulty for the planner compared to solving individual goals.

3.3.4 Refined Taxonomy of Subgoal Collections

The notion of $<_G$-consistent trivial serializability allows to refine the extended hierarchy of subgoal collections as given by (Barrett and Weld, 1993), where we have a distinction between independent, trivially serializable, laboriously serializable and nonserializable collections. Barrett & Weld's main contribution lies in the refinement of the class of serializable subgoals (Korf, 1987) into *trivially serializable* and *laboriously serializable* subgoals. These can be seen as formalizing the borderline between subgoal collections that are *easy* to solve (in terms of serializability) for a planner capable of creating a certain class of plans and those that are *difficult* (in terms of serializability) because of exponential costs of backtracking on a small but significant number of subgoal orderings (which in average dominate the planning time).

We add the class of *$<_G$-constrained trivially serializable* subgoal collections to this taxonomy as shown in Figure 8. This new class contains all trivially serializable collections and, additionally, some of the laboriously serializable collections. Laboriously serializable subgoal collections are separated into two subclasses, one of those collections for which a $<_G$ exists

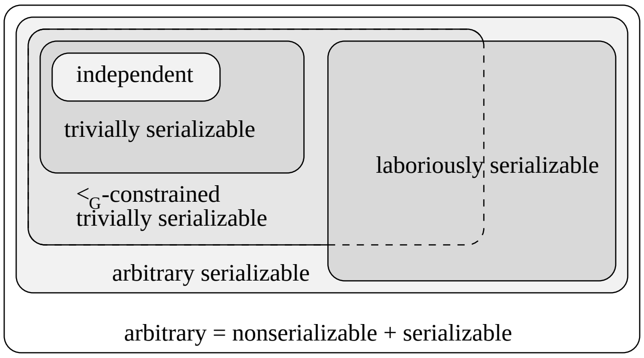


Figure 8: Refined Hierarchy of Subgoal Collections

such that the collection is $<_G$-constrained trivially serializable and one of those for which no such $<_G$ exists.

3.3.5 $<_G$-Consistent Goal Selection Strategy

The notions of $<_G$-consistent permutations of goals and $<_G$-constrained trivial serializability motivate a second way to use goal orderings that is also implemented in CAPLAN besides forcing the establisher to be ordered accordingly: if we have a goal ordering $<_G$, we use it to address the control problem of goal selection and determine an order in which goals are selected during the planning process such that we have a $<_G$-consistent permutation of the goals. For example, if $g_i <_G g_j$ holds then CAPLAN will select goal g_i before goal g_j, thus creating a subplan achieving g_i before extending it to achieve g_j also.[12] We will call this a *$<_G$-consistent goal selection strategy*. Section 4.2.2 will explain in detail why this can be expected to reduce the size of the explored part of search space.

We claim that, if a domain or at least a certain problem is $<_G$-constrained trivially serializable but not trivially serializable, performance improvements of using goals orderings should be observable because $<_G$-constrained trivially serializable problems are for CAPLAN like trivially serializable problems for SNLP. Further, the definition of $<_G$-constrained trivial serializability states a property of a collection of goals on a certain class of plans, e.g., the class of plans produced by SNLP. If we here use the new class of *elastic protected $<_G$-consistent plans*, which the planner CAPLAN is capable to create, we have a combination of the ordered establisher condition and the $<_G$-consistent goal selection strategy. However, the next section will show that a combination of the ordered establisher condition and a $<_G$-consistent goal selection strategy not always means that the same $<_G$ should be used for both purposes, i.e., ordering establishers and goal selection, but that this is a domain-dependent property that the same can be used for both.

3.3.6 Relations to Necessary or Possible Goal Orderings

Section 3.2.1 distinguished between necessary and possible goal orderings; both are based on the ordered establisher condition that translates goal orderings into plan step orderings. We emphasized that necessary goal orderings are of special interest because they do not exclude any solutions for the planner. Section 3.3.3 and Section 3.3.5 proposed a second

[12] This assumes a LIFO strategy for adding/selecting new goals for open preconditions of plan steps.

way to understand goal orderings as a means of providing direct guidance for the planner by specifying in which order goals should be processed by the planner.

At first glance one could think that knowing about the necessary goal orderings is the key information that makes it easy to solve a problem. Unfortunately, this is not true. Knowing about necessary goal ordering is, of course, a big advantage, but it is neither enough nor necessary to be able to guarantee that a problem is $<_G$-constrained trivially serializable in a certain domain. The following Lemmas and their proofs give examples for the best and the worst cases with respect to the relation of necessary or possible goal orderings and $<_G$-consistent trivial serializability.

Lemma 3 *Given a problem (I, G) and a goal ordering $<_G$, if $<_G$ is equal to the set $maxNecessary((I, G))$ or even $maxPossible((I, G), P)$ for a certain solution P then the problem does not need to be $<_G$-constrained trivially serializable.*

In other words: even the maximal amount of information that can be specified using necessary or possible goal orderings does not always guarantee that the problem moves to the class of easy problems (in terms of serializability) for CAPLAN.

Proof: See Appendix B.1, page 54.

This result is in some sense discouraging but it helps a lot to understand the different amounts of performance improvements that can be obtained by using goal orderings (see Section 5). It shows that it is possible to find a domain (we gave a simple example in the proof of Lemma 3) where necessary or possible goal orderings will never make things become easier in terms of serializability. This observation will be further generalized below.

But there is another, more positive and encouraging relation between necessary goal orderings and the property of $<_G$-constrained trivial serializability that can be stated for a domain:

Lemma 4 *Given a problem $(I, G, <_G)$, if the problem is $<_G$-constrained trivially serializable but not trivially serializable, then $maxNecessary((I, G)) \subseteq <_G$ does not necessarily hold.*

In other words: we do not always need to know all necessary goal orderings to make a problem that is not trivially serializable (i.e., not easy for SNLP) become $<_G$-constrained trivially serializable (i.e., easy for CAPLAN).

Proof: See Appendix B.1, page 54.

The proof shows that the main difficulty of the artificial domain $\Theta_n D^m S^1$ already mentioned in Section 3.2.4 is to solve G_α first. If this is guaranteed by goal orderings and a $<_G$-consistent goal selection strategy, then the domain is as easy for CAPLAN as the simpler version $D^m S^1$ is. But there is no need to know the maximal set of necessary goal orderings.

3.3.7 Generalization of Extended Problem Specifications

The results of Lemma 3 and 4 can be explained in a more general way. So far, they show extreme examples for the relation of necessary or possible goal ordering and their use for goal selection. This lets us suspect that using goal orderings as defined by the ordered establisher condition and for a $<_G$-consistent goal selection strategy are two different ideas. In particular, it is a property of the domain whether a goal ordering that leads to improvements with the ordered establisher condition also is an improvement when used in the context of a $<_G$-consistent goal selection strategy.

Theorem 1 *Given an arbitrary problem (I, G) in a certain domain that is not trivially serializable, then it is a domain-specific property whether a necessary or possible goal ordering $<_G$ also leads to performance improvements when using it in a $<_G$-consistent goal selection strategy.*

Domain-specific means that we can find all possible relations depending on the domain. To prove this we will enumerate the possibilities. Parts of this theorem already have been proved above. In other words, the theorem says that, in general, necessary/possible goal orderings and the idea of $<_G$-consistent permutations for goal selection are two independent ideas and that we can give examples of domains and problems where

1. a certain necessary or possible ordering $<_G$ can make a problem become $<_G$-constrained trivially serializable (see Lemma 4),

2. no necessary or possible ordering $<_G$ can make a problem become $<_G$-constrained trivially serializable (see Lemma 3),

3. a certain necessary or possible ordering $<_G$ can make a problem become $<_G$-constrained trivially serializable but there exists another ordering $<'$ that allows to find the solution with less inferences steps than $<_G$,

4. an impossible ordering $<_G$ (one that cannot be used w.r.t. ordered establisher condition) can also allow to find a solution when using it for a $<_G$-consistent goal selection strategy.

Proof: The artificial domain LinkRepeat ((Veloso and Blythe, 1994), see also Appendix A.5) is an example showing the last two points; details can be found in Appendix B.2 on page 55.

Motivated by the observations of the domain LinkRepeat we created another artificial domain, $\Theta_2^2 D^m S^1$, where the effect found in LinkRepeat is much more clear. This domain is an example showing that a goal ordering that has been obtained by an analysis of possible solution plans based on the ordered establisher condition should not be (mis-)interpreted as an order for goal selection. Details can be found in Appendix A.3 and the experiments in Section 5.3.

Generalizing extended problem specifications: This theorem has the consequence that we have to redefine extended problem descriptions to be a 4-tuple $(I, G, <_e, <_a)$ where we distinguish between goal orderings $<_e$ that are interpreted as given by the ordered establisher condition and goal orderings $<_a$ that are used for a $<_a$-consistent goal selection strategy. In many domains we find that $<_e = <_a$ leads to good or the best results. In such a case, even a misinterpretation of goal orderings in terms of a goal selection strategy will work. But this is a domain-specific property that, for example, does not hold in the domain LinkRepeat.

3.4 Summary: Effects of Goal Orderings in CAPlan

In order to benefit from goal orderings in CAPLAN it has to be defined exactly how to get a goal ordering $<_G$ for a problem (I, G) of a certain domain and it has to be guaranteed that $<_G \neq \emptyset$. Then the planner CAPLAN might be able to solve problems of a certain domain more efficiently because of two reasons: (1) ordered establisher condition (Section 3.1.1, Definition 8), (2) $<_G$-consistent goal selection strategy (Section 3.3.5).

The first leads to a more constrained class of plans, the class of *elastic protected* $<_G$*-consistent plans*, compared to the class of plans created by SNLP. Section 4.2.1 will show in detail why this also reduces the size of the search space. This gain of efficiency depends on the fact that the additional orderings might automatically resolve threats in the correct way.

The second point addresses the control problem of partial-order planning by using goal orderings to define a goal selection strategy that makes it easy to solve a problem. Then a domain or problem *might* not be trivially serializable but it *might* be $<_G$-constrained trivially serializable where $<_G$ is computed based on (I, G). This is a second way to gain advantage from goal orderings as the degree of difficulty is reduced dramatically in case the difficulty is a result of having to solve a collection of goals instead of individual goals. Unfortunately, not in all domains we can compute or give goal orderings and there also exist domains in which no necessary goal orderings exist that could be used to improve planning performance without a loss of completeness when used for the ordered establisher condition.
Even if $<_G \neq \emptyset$, the spectrum of possible efficiency gains with respect to serializability is wide: A domain or problem does not need to be $<_G$-constrained trivially serializable even if we have maximal sets of necessary or possible orderings (see Section 3.3.6). On the other hand, we do not always need the maximal set of necessary orderings to have significant performance improvements. More general, it is a property of the domain whether a certain necessary or possible goal ordering also can improve planning with respect to serializability. If we know that a certain problem or domain is $<_G$-constrained trivially serializable for a certain mechanism to compute $<_G$ but not trivially serializable, we can however expect a significant increase in performance (and a reduction of the size of the explored part of the search space) by using a $<_G$-consistent goal selection strategy.

If we compare both ways to gain advantage of goal orderings the goal selection strategy has the potential to shift the domain from the class of laboriously serializable to the class of $<_G$-constrained trivially serializable domains for CAPLAN (not for SNLP). However, even if goal orderings exist a domain does not need to be $<_G$-constrained trivially serializable. On the other hand, applying the ordered establisher condition because of necessary goal orderings, i.e., using a planner that works on the class of *elastic protected* $<_G$*-consistent plans*, will always save some amount of work and improve efficiency.

4 Improving Planning Performance with Goal Orderings

How and why can $<_G$ be used to make the planner work more efficiently? We will summarize how CAPLAN uses goal orderings and how these ideas are implemented in the CAPLAN architecture. Then we will proof that goal orderings have positive effects in reducing the size of the explored search space. Finally, we will examine a number of planning domains known from literature and some variants of them; we will illustrate the characteristic difficulty and the expected effect of goal orderings in these domains.

4.1 Implementation of Goal Orderings in CAPlan

Section 3 illustrated the idea to use goal ordering in two situations:

1. The **ordered establisher condition** (see Definition 8) leads to a more constrained class of plans where ordering relations between establishers of ordered goals are required to make a plan consistent modulo $<_G$.

2. A $<_G$**-consistent goal selection strategy** (Section 3.3.5) addresses the problem of goal selection in partial-order planning. It guarantees that only $<_G$-consistent permutations of the goals are considered for goal selection.

We will now explain how this is implemented in the CAPLAN architecture. The planning algorithm of CAPLAN and its extensions with respect to SNLP already has been described earlier (Weberskirch, 1995) and will not be revisited again. We will only summarize the points that are important to support goal orderings and extended problem specifications.

Basically, standard problem specifications are extended by two slots for goal orderings and there are some GUI components available in the CAPLAN system for defining such extended problem specifications. This allows the generalized form of extended problem specifications $(I, G, <_e, <_a)$ (see Section 3.3.7). CAPLAN interprets the simpler variant $(I, G, <_G)$ as $(I, G, <_G, <_G)$. More interesting is the point how goal orderings are handled by the planning algorithm itself.

4.1.1 Implementing the Ordered Establisher Condition

CAPLAN takes care of goal orderings in case it works on a top-level goal $g \in G$ of a problem $(I, G, <_G)$ for which there exists an ordering restriction, i.e., $\exists g' \in G : g <_G g'$ or $g' <_G g$. Normally, CAPLAN will select one alternative out of the so-called conflict set of a goal, i.e., the set of concrete plan modification operators applicable to the goal. For top-level goals there are two alternatives: step addition and simple establishment.

As pointed out in (Weberskirch, 1995) the extended consistency test of CAPLAN (plan consistency modulo $<_G$) has been reduced to a normal consistency test on an extended set of orderings constraints in the implementation. For the goal ordering $g <_G g'$ CAPLAN automatically adds the ordering constraint $establisher(g) \prec_O establisher(g')$ as soon as the planner has committed to both establishing steps. This means that the plan modification operators applicable to top-level goal, *step addition* and *simple establishment*, have been replaced by slightly extended versions in CAPLAN which guarantee that a $<_G$-consistent plan is refined into another $<_G$-consistent plan. Let $g \in G$ be the current goal and $g <_G g'$ the ordering restriction. Then the two plan modification operators applicable to ordered top-level goals are modified like that: (we only point out the differences)

- $<_G$**-consistent step addition:** Like normal step addition this one adds a new plan step s_{new} to the current plan. CAPLAN will also add $s_{new} \prec_O establisher(g')$ if $establisher(g')$ exists.
- $<_G$**-consistent simple establishment:** Like the normal version of simple establishment, this one uses an existing plan step s_{exist} to establish the goal. CAPLAN will also add $s_{exist} \prec_O establisher(g')$ if $establisher(g')$ exists.

So, the plans used by CAPLAN are always augmented by some additional ordering constraints compared to the plans that are created by SNLP in the same situation. But this difference between concept and implementation (see Figure 9) has the important advantage that a plan is consistent modulo $<_G$ whenever it's augmented version as generated by the extended plan modification operators of CAPLAN is consistent. Further, these new plan modification operator guarantee that CAPLAN only creates plans from the class of elastic protected $<_G$-consistent plans.

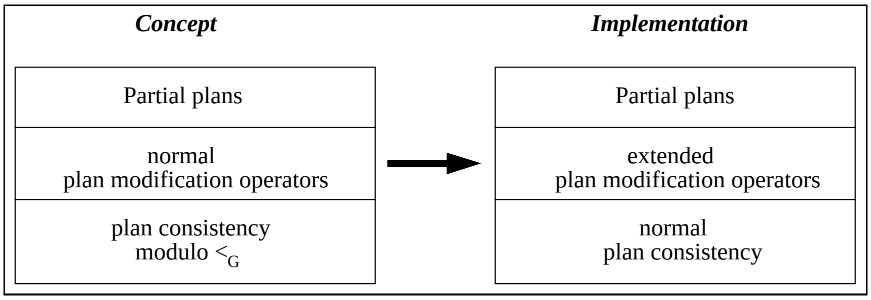


Figure 9: Concept and Implementation of the Ordered Establisher Condition

TMS justification for added orderings. One point about the implementation of goal orderings has not been documented in (Weberskirch, 1995): The fact is that ordering constraints between establishers, that have been added to a plan by the extended plan modification operators as described above, need a slightly different TMS justification than normal other orderings that are added to the plan, for example, by demotion or promotion refinement.

Let us recall from (Weberskirch, 1995): CAPLAN is based on REDUX (Petrie, 1992) and the implementation of REDUX is based on a JTMS (Doyle, 1979). All elements of the plan are represented as so-called assignments of REDUX decisions. Each decision basically represents one instance of a plan modification operator. If the decision is valid, it has been applied to the current plan. If the decision is valid then all its assignments are valid. This and all other dependencies are part of the REDUX model. They are maintained and propagated using a Truth Maintenance System (TMS) in CAPLAN. All typical justification structures have been described earlier (Petrie, 1992; Weberskirch, 1995).

Normally, for each assignment we can determine exactly one decision and all assignments are valid as long as their decision is valid. This gets a little more complicated when orderings (i.e., assignments) are added that depend on the validity of two other plan steps like orderings that are translated from goal orderings. If we have a goal ordering $g_1 <_G g_2$ this will be translated into an ordering between the establishers. Let us assume that the decision that added $establisher(g_2)$ already exists. Then CAPLAN adds the ordering $establisher(g_1) \prec_O establisher(g_2)$ and it will justify this assignment with the decision that added $establisher(g_1)$ **and** the decision that added $establisher(g_2)$. So, the validity of the extra assignment depends on both decisions (see Figure 10): if either the one or the other decision is rejected (e.g., because of backtracking or by a user modifying a solution interactively) the TMS is able to recognize that the validity of this extra orderings also changes.

In summary, this is a second conceptual extension to the justification structure of REDUX. As already reported in (Weberskirch, 1995, p. 32) a first modification is necessary for a correct representation of threats in the REDUX terminology: protection goals represent the need to resolve a threat and they also depend on two other decisions. For goal orderings a similar extension is necessary with respect to the correct justification of the assignment representing the additional orderings constraints. In particular, the correct TMS justification is needed for a full integration of goal orderings in the interactive planning environment of CAPLAN where a user can participate in the solution process.

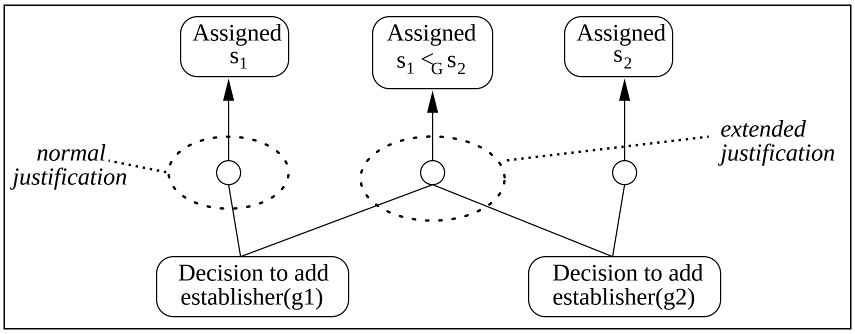


Figure 10: TMS justification of an ordering assignment

4.1.2 $<_G$-Consistent Goal Selection Strategy

In (Weberskirch, 1995) we also did not document the fact that it can be very helpful for planning performance if we use goal orderings to determine $<_G$-consistent permutations of the goals of a planning problem as done by a $<_G$-consistent goal selection strategy.

CAPLAN is an agenda-driven system. It uses two agendas of open goals:

- one agenda collects open establishment goals (task agenda),
- the other collection open protection goals (meta task agenda).

The selection strategy from each agenda is determined by control plug-ins called *control components*, which are the common interface in CAPLAN to define control strategies for goal and/or operator selection. Several well known control strategies are implemented for CAPLAN in form of a control component, e.g., DSep (Peot and Smith, 1993), LCFR (Joslin and Pollack, 1994), ZLIFO (Gerevini and Schubert, 1996),[13] or also more sophisticated control components like the case-based control component CbC of CAPLAN/CBC (Muñoz-Avila and Weberskirch, 1996a; Muñoz-Avila, 1998).

The implementation of a $<_G$-consistent goal selection strategy is done by ordering the task agenda such that it always contains a $<_G$-consistent permutation of the planning goals. A LIFO strategy for adding/selecting goals then guarantees that the planner always starts with the minima w.r.t. $<_G$.

4.2 Effect on the Size of the Search Space

We claim, that when using goal orderings in the way as defined in Section 3, we will be able to speed up planning. In this section we will prove that this claim holds. We will first examine the effect of the ordered establisher condition. Then we will look at the effect of using a $<_G$-consistent goal selection strategy.

For a certain problem we will show two things:

- The ordered establisher condition has effects on the *size of the search space.*

[13](Pollack et al., 1997) gave a good overview and comparison of the most interesting control strategies for partial-order planning.

- The $<_G$-consistent goal selection strategy addresses the control problem, i.e., the *navigation in the search space.*

In general, if we do not know about properties of a domain (e.g., whether it is $<_G$-constrained trivially serializable), a combination of both is the most promising way to gain advantage of the knowledge about goal orderings. However, this cannot always guarantee that the performance is improved as we have explained in Section 3.3.7.

4.2.1 Effect of the Ordered Establisher Condition

The ordered establisher condition forces CAPLAN to consider plans to be inconsistent in which the establishers of ordered goals are not ordered as given by Definition 8. The planning algorithm of CAPLAN never will refine a partial plan that is not consistent modulo $<_G$. This property can be used to show that the search space of CAPLAN differs from the search space of SNLP.

We first have to explain what the search space is (see also (Weld, 1994)). Partial-order planning can be understood as search in a directed graph: nodes represent partial plans (including the initial plan and the solutions) and edges denote plan refinement operators; together they form the search space. We will also use the term *inference step* for the application of a plan refinement operator. Typical refinements of partial-order planning are step addition, simple establishment, and the threat resolution operators (Kambhampati et al., 1995). The size of the search space can be measured by looking at the nodes and edges. If we want to compare two search spaces, one is smaller than another one if the first one consists of less nodes, or, if the number of nodes is the same, if the first has at least less edges between nodes. However, the fact that the search space gets smaller does not necessarily mean that planning gets easier. In worst case, there are less plan states and/or less possible refinements connecting the plan states, but the paths to a solution are longer than in the original search space. To get a statement about the average case we also look at the number of inference steps to obtain a certain solution, i.e., we check the paths from the initial plan to a certain solution.

With respect to the effect of the ordered establisher conditions on the search space of CAPLAN, we can state and prove the following theorem.

Theorem 2 *Given a problem (I, G) and necessary goal orderings $<_G \neq \emptyset$ for that problem, then the following conditions are true:*

(a) The size of the search space of CAPLAN *for finding a solution for $(I, G, <_G)$ is smaller than the size of the search space of SNLP for finding a solution for (I, G).*

(b) If solutions$((I, G)) \neq \emptyset$, the minimal path from the initial plan to a fixed solution in the search space of CAPLAN *is never longer than in the search space of SNLP, but it can be shorter.*

In other words: (a) In worst case, if the complete search space has to traversed, it is not more difficult to plan with goal orderings than to plan without them because the search space never grows when using goal orderings. (b) Finding a solution will never necessarily require more inference steps with CAPLAN than with SNLP. There can even exist shorter paths in the search space of CAPLAN. However, there is no guarantee that, given a certain goal and operator selection strategy, that finds the shortest path in the search space of SNLP, does also find the shortest path in the search space of CAPLAN. We only guarantee that *it is possible to find* the shortest SNLP path or a shorter one.

Proof: See Appendix B.3, page 56.

Finally, if we drop the restriction of the Theorem 2 that $<_G$ is a necessary goal ordering and allow possible orderings instead, part (a) and the proof will still be true and part (b) only applies to the paths to the solutions P' for which we have $possible((I, G), <_G, P')$.

4.2.2 Effect of $<_G$-Consistent Goal Selection Strategy

The previous section has shown that adding ordering constraints as done by the $<_G$-consistent plan refinement operators implementing the ordered establisher condition reduces the size of the search space without making it more difficult to find a solution. A similar question is why a $<_G$-consistent goal selection strategy can make planning easier. If we interpret goal selection in terms of the search space, it does not affect the size of the search space but it is one factor that defines how the planner navigates through the search space. Section 3.3.5 has already said that this should only be expected if the domain or problem is $<_G$-constrained trivially serializable but not trivially serializable.

Now we will explain in more detail the advantage of a $<_G$-consistent goal selection strategy. Let us compare the following two situations: we have two goals g_1 and g_2 which are ordered by $g_1 <_G g_2$. So, the permutation (g_1, g_2) is $<_G$-consistent, whereas the permutation (g_2, g_1) is not. What is the difference between both permutations for the planner? Let us analyze both situations assuming that the problem is $<_G$-constrained trivially serializable but not trivially serializable and that the planner uses a LIFO strategy for goal addition/selection.

Permutation (g_1, g_2)**:** First, g_1 is established, $establisher(g_1)$ is added to the plan (i.e., g_1 is established with a new plan step) and its preconditions are established. This results in a subplan for the goal g_1. Then $establisher(g_2)$ is added (Figure 11.(a)).

- (i) After $establisher(g_2)$ is added it is parallel to $establisher(g_1)$ and the complete subplan for g_1.
- (ii) We can be sure that the subplan for achieving g_1 can be serially extended to a plan achieving both goals because the permutation is $<_G$-consistent and the problem is $<_G$-constrained trivially serializable.

Permutation (g_2, g_1)**:** Here, g_2 is established first and the $establisher(g_2)$ is added to the plan. Then $establisher(g_1)$ is added (Figure 11.(b)).

- (i) The permutation (g_2, g_1) is not $<_G$-consistent. This means that there exists at least one possibility for refinements to establish g_2 that lead to a plan that is not serially extensible to a plan for both goals.
- (ii) After $establisher(g_1)$ is added it is parallel to $establisher(g_2)$ and the complete subplan for g_2. But if the planner has chosen a subplan for g_2 that is not serially extensible, there will occur at least one threat t between some causal link from the subplan for g_2 and either $establisher(g_1)$ or another step that is added to achieve the preconditions of $establisher(g_1)$, or in the opposite direction, a threat caused by a step of the subplan for g_2. This threat t is not resolvable, so the planner has to backtrack. This is exactly one reason (there might be more) for which this permutation is not $<_G$-consistent.

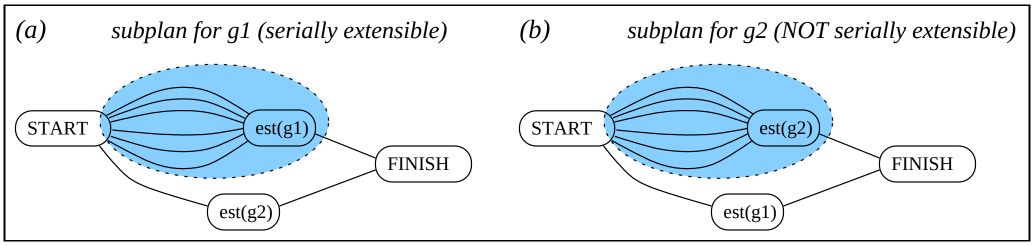


Figure 11: Extending a subplan for both permutations. est(g1) and est(g2) are the establishers of goals g_1 and g_2.

Comparison. The explanation of the possible consequences of both permutations shows, that with the $<_G$-consistent goal selection strategy the planner will always be able to solve the two goals without having to backtrack over decisions for achieving g_1 and it preconditions. This is an effect of the fact that each possible subplan is serially extensible. On the other hand, with the non-$<_G$-consistent permutation the planner has two subproblems and solves g_2 first, but there must be more than one alternative for a subplan solving g_2, and at least one subplan exists that cannot be serially extended to a solution. In that case, the planner has to perform backtracking and revise parts of the solution of the goal g_2. For n goals we have in worst case the exponential cost of backtracking over all decisions for a subplan solving $g_1, \cdots, g_i$ when trying to extend it for the next goal g_{i+1}.

Figure 12 illustrates this for three goals. If subplans are serially extensible the planner spends a certain amount of planning effort to solve g_1, then it can concentrate on g_2 and extend the subplan, after that it plans for g_3. If the subplan for g_1 is not serially extensible, the planner has to backtrack when it tries to solve g_2 with respect to the plan found for g_1. The same might happen if it starts planning for g_3. The size of the area for a certain goal in Figure 12 symbolizes the increasing effort to solve the next goal and the need to backtrack over decisions that were made for previous goals.

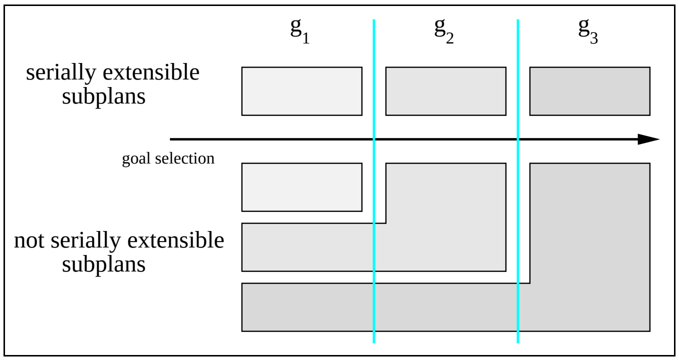


Figure 12: Planning effort if subplans are serially extensible or not

If the problem is not $<_G$-constrained trivially serializable the $<_G$-consistent goal selection strategy is nothing more than a heuristic that might help to find the solution faster. The reason is, that here even the right goal selection order might produce plans that cannot be serially extended to solve all goals. Then the planner has to backtrack although it selected the goals in the right order unless the planner always selects the right operator for each goal.

CAPlan always uses $<_G$-consistent permutations of the goals for goal selection so this will guarantee that no obviously wrong ordering of ordered goals for goal selection can occur. But as we cannot know for all domains whether they are $<_G$-constrained trivially serializable and how to get $<_G$, we only *might* profit from a $<_G$-consistent goal selection strategy and there is no guarantee that the performance will increase.

Section 3.3.2 has also shown that it might not help much to know the properties of a problem w.r.t. serializability as this only looks at the difficulties that are caused simply by the fact that a set of goals has to be solved. Further, there are domains in which planning problems have to be given as $(I, G, <_e, <_a)$ with $<_e \neq <_a$ in order to improve performance when using goal orderings for a goal selection strategy.

4.2.3 Combination of Ordered Establishers and $<_G$-Consistent Goal Selection

So far, we explained the possible advantages of the ordered establisher condition and of the $<_G$-consistent goal selection strategy independent from each other. We now will look at the advantages of combining both.

If we have $g_1 <_G g_2$ for two goals, the establishers of both are ordered by CAPLAN with $establisher(g_1) \prec_O establisher(g_2)$ (see Figure 13 in comparison with Figure 11.(a)). This means that in any case the planner does not have to find the right ordering relation between those two establishers. As an effect of the $<_G$-consistent goal selection strategy (and the LIFO strategy for goal addition/selection) the subplan for g_1 is completed before the planner adds $establisher(g_1)$.

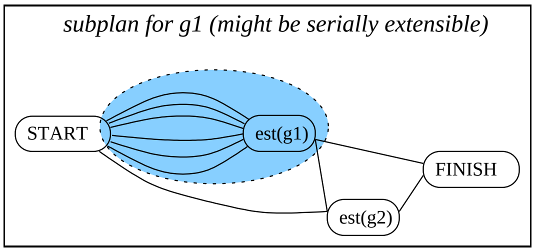


Figure 13: Combination of $<_G$-consistent goal selection and ordered establishers

Under the assumption that a domain is not trivially serializable, we can distinguish two situations:

- If the problem is $<_G$-constrained trivially serializable we can expect the biggest performance improvement because: (1) $<_G$-constrained trivially serializable means that the subplan for g_1 can always be extended to a subplan for both goals; (2) the ordering $establisher(g_1) \prec_O establisher(g_2)$ guarantees that the planner cannot select the wrong ordering between the establishers. Each refinement producing such an ordering, $establisher(g_2) \prec_O establisher(g_1)$, is inconsistent for CAPLAN and will never be considered.

- If the problem is not $<_G$-constrained trivially serializable the $<_G$-consistent goal selection strategy is only a heuristic that does not prevent the planner from choosing the wrong subplan for g_1, i.e., one that is not serially extensible to a plan for both goals. But CAPLAN also adds the ordering constraint between the establishers of both goals, thus, correctly constraining the plan compared with the situation without the goal orderings. This can, in the best case, reduce the set of alternatives for subsequent planning decisions so that the need for backtracking is recognized in less inference steps (no matter whether the subplan for g_1 is serially extensible or not). In the worst case, this does not change anything for the planner.

This argumentation leads to the following conclusion:

> The combination of the addition of orderings between establisher based on a necessary goal ordering $<_G$ and the $<_G$-consistent goal selection strategy in average reduces the number of mistakes the planner can make without reducing the solution space.

The reason for that can be summarized as follows: the ordered establisher condition always reduces the size of the search space whereas the $<_G$-consistent goal selection strategy typically can help in case the domain is $<_G$-constrained trivially serializable but not trivially serializable and such a $<_G$ is given. As already explained in Section 3.3.7, we cannot expect that the same goal ordering is useful for both purposes, ordering establishers and defining a goal selection strategy. The proof of Theorem 1 also has shown that there exist domains (in particular, LinkRepeat or $\Theta_2^2 D^m S^1$) where the number of inference steps differs dramatically between different goal selection orders and that a $<_G$-consistent goal selection strategy not always must lead to the minimal number of inference steps. In both domains, the best goal selection strategy is defined by an impossible ordering in terms of the ordered establisher condition.

4.3 Characteristic Difficulties of Domains and Goal Orderings

We will now illustrate the expected effect of goal orderings in some example domains which differ in their degree of difficulty. Typically, *subsumption of goals* (a goal is achieved as a side effect of achieving another one) or *interactions between plan steps* can cause simplifications or complications and they are the main reasons for goal orderings. Section 3.3.2 already emphasized that subsumption of goals reduces the expressiveness of serializability. In domains with goal subsumption serializability is not the best, sometimes not a suitable way to characterize the difficulty. We observed that domains might be difficult or simple for various reasons and we characterized some of these reasons. Many domains known from literature combine several or all of these elementary difficulties but it helps a lot to understand why a domain is difficult if we can identify such difficulties.

We will concentrate on difficulties produced by interactions. The difficulty of a domain can be characterized by at least two factors: (1) interactions between establishers of planning goals, (2) selection of the right establisher from several possibilities. The first factor can be found in many mouldings: in general, we might have threats between the establishers of different goals and other goals or preconditions, which force the establishers to be ordered. We identified one special case which we called the resource allocation problem. Here the interactions model the allocation and release of a resource. That seems to create the most complicated domains. In all cases we will investigate the possible effects of goal orderings and we will give example domains from literature with this difficulty.

4.3.1 The Establisher Ordering Problem

A general type of difficulty in a STRIPS-like planning domain – interactions between the plan steps – is present in every non-trivial domain, in which goals are not independent. With respect to goal orderings and their use in the ordered establisher condition (see Definition 8) the subset of threats produced by establishers of top-level goals are important and allow to define a general difficulty of a domain.

> The *Establisher Ordering Problem* is the difficulty that is characterized by the situation that, even if we select the correct plan steps achieving the goals of a planning problem these establishers interact – at least one deletes another goal or preconditions of the establisher of an other goal.

The consequence of these interactions is that the establishers need to be ordered with respect to each other in a certain way so that all threat are resolved. The artificial domains D^mS^1 or D^mS^2 (Barrett and Weld, 1994) are examples of domains illustrating the difficulty of the establisher ordering problem. However, as reported and proved by Barrett and Weld, these domains are comparable easy for SNLP and can be shown to be trivially serializable.[14] Although goal orderings help to increase the performance of the planner here (see empirical results in Section 5), the amount of improvement is not very well-developed and it can only be explained with the ordered establisher condition:

- The $<_G$-consistent goal selection strategy has no positive effect because the threat resolution problem can be solved by SNLP without backtracking and independent of a certain goal selection strategy.[15]
- The ordered establisher condition is responsible for improvements because each extra ordering constraints can resolve one threat without letting a choice for the planner.

Barrett and Weld also gave an example of a domain that is more difficult, D^mS^{2*}, although the main difficulty can also be characterized to be an establisher ordering problem. D^mS^{2*} is laboriously serializable, even for SNLP. Here we can show that there exist typical goal orderings for each problem which make it $<_G$-constrained trivially serializable, i.e., (in terms of serializability) easy for CAPLAN. This means that goal orderings help to improve performance in domains which are difficult because of establisher ordering problems, especially, if the domain is not trivially serializable.

4.3.2 The Operator Selection Problem

Domains get more complicated if the number of possibilities how a goal can be established is greater than one. This is a typical situation encountered in most real planning domains.

> We speak about an *Operator Selection Problem* (see also (Barrett and Weld, 1994)) if there are multiple (n) establishers for a goal but only some of them will work in an overall solution.

The artificial domain $\Theta_n D^m S^1$ (Barrett and Weld, 1994) is an example of a domain that has explicitly been constructed based on the D^mS^n versions to add the difficulty of an operator selection problem. In fact, this domain is laboriously serializable for SNLP and

[14] The main objective of (Barrett and Weld, 1994) was to investigate the different performance of three planning algorithms in various domains and it has shown some advantages of plan-space over state-space planning approaches.

[15] A $<_G$-consistent goal selection strategy may only causes a little computational overhead for the planner.

the operator selection problem is responsible for that. Nevertheless, if goal orderings $<_G = maxNecessary((I,G))$ of a problem are known, it can be shown that the domain is $<_G$-constrained trivial serializability.[16] Here, the $<_G$-consistent goal selection strategy solves the main difficulty because it forces the planner to use a permutation of the goals in which it is easy to find the right establisher. The ordered establisher condition also improves the performance because it addresses the establisher ordering subproblem which is the same as in the D^mS^1 domains.

The operator selection problem can be investigated separated from the establisher ordering problem in $\Theta_n D^0 S^1$ (see Appendix A.2). The difficulty of both domains is almost the same because $\Theta_n D^0 S^1$ is also not trivially serializable, but we can also find a necessary goal ordering $<_G$ such that it is $<_G$-constrained trivially serializable.

4.3.3 Resource Allocation Problems

The Workpiece domain of CAPLAN and CAPLAN/CBC (Muñoz-Avila and Weberskirch, 1996a; Muñoz-Avila, 1998) is laboriously serializable but there also exists a goal ordering $<_G$ such that problems in that domain are $<_G$-constrained trivially serializable.[17] But this domain is difficult for additional reason compared to $\Theta_n D^m S^1$: the operators `HoldTool` and `MakeToolHolderFree` (Muñoz-Avila and Weberskirch, 1996b) encode what we call a *resource allocation subproblem*. Here, the operators concerning the tool holder never act as establishers of top-level goals but they are needed to achieve some of the preconditions of establishers of top-level goals.

We can find a similar difficulty in the simpler artificial domain ART-1D-RD (Kambhampati, 1994) (see Appendix A.4 for the definition). This domain encodes the allocation and release of resources (like the tool holder of the lathe machine in the workpiece domain). This makes the domain significant more difficult that the simpler version ART-1D.[18] The main difficulty is not caused by interacting steps achieving the goals (i.e., an establisher ordering problem) but by the steps having to allocate resources while achieving the goals (i.e., a resource allocation problem). Giving goal orderings does not really address the resource allocation problem.

Characterization. It is not straightforward to give a definition that exactly describes all situations in which we have a resource allocation problem encoded in preconditions and effects of actions. We can, however, give some properties that can be found in certain domains which allow to conclude that we have one.
STRIPS-like representations only have a single way to represent a resource: the state of the resource allocation is encoded by certain predicates which also appears in preconditions and effects of actions. The simplest case is that we have a predicate like *res-released()* and some operators have the preconditions *+res-released()* and the effect *-res-released()*, others do the opposite. If we don't want to have negated preconditions we can use an equivalent representation with two predicates *res-released()* and *res-allocated()* instead (as, for example, done in the in the domain ART-1D-RD, see below). We can observe that such resource predicates have the property that they are frequently achieved and again destroyed during the execution of a plan (high-frequency condition, (Kambhampati et al., 1995)).

[16] The proof of a Lemma in Section 3.3.6 has shown that even a certain subset of the maximal necessary goal orderings is enough.

[17] For this domain there exists a goal ordering $<_G$ that reflects geometrical relations between different processing areas of a workpiece. This goal ordering can be computed for every problem (I,G) by a domain-specific geometrical reasoner. It has been shown, that the domain is $<_G$-constrained trivially serializable for this $<_G$ (Muñoz-Avila and Weberskirch, 1996b).

[18] ART-1D is just another name for D^1S^1 which is trivially serializable (Barrett and Weld, 1994).

There are a lot of variants how to model even a simple case of resource allocation. We might have an *explicit modeling* by actions doing nothing else than the allocation and/or release of the resource, or, resource allocation can be modeled by certain additional preconditions and effects of other actions (*implicit modeling*). We can also have mixed versions where allocation is modeled explicit but release is modeled implicit. This is the reason why it is so difficult to give a comprehensive definition of resource allocation. We will not distinguish these alternative representations here in more detail although we suspect that the way how resource allocation is modeled can have influence on the resulting planning performance. However, experiments show that domains in which we find the difficulty of resource allocation (no matter how it is represented) tend to be harder for planners than domains without this difficulty. This also leads to approaches where planning and scheduling tasks of a planning problem are handled in separated phases and by different algorithms (e.g., see (Bergmann and Kott, 1998)).

Goal orderings and resource allocation. In many cases goal orderings only accidentally solve a resource problem which occurs in combination with another problem. Let us examine this by having a closer look at ART-1D-RD and the variant ART-0D-RD which exclusively focuses on the resource allocation problem.

ART-1D-RD: We can summarize the following facts about this domain: (1) problems of size n have exactly one solution, (2) the domain is laboriously serializable, (3) with $<_G = maxNecessary((\text{I,G})) \neq \emptyset$ a problem is $<_G$-constrained trivially serializable.
This looks similar to $\Theta_n D^m S^1$. Both are laboriously serializable (although individual goals are easy to solve) and there exists $<_G$ such that it is $<_G$-constrained trivially serializable. However, the characteristic difficulties are different: ART-1D-RD is combination of an establisher ordering problem and an resource allocation problem (not an operator selection problem). We found that the establisher ordering problem is comparably easy in both domains (this can be seen by the fact that ART-1D/$D^m S^1$ is trivially serializable and individual goals are easy to solve) but neither the operator selection problem nor the resource allocation problem. This becomes clearer if we look at simplified variant.

ART-0D-RD: We constructed a variant of ART-1D-RD by eliminating that part of that is responsible for the establisher ordering problem (for the definition see Appendix A.4). At first glance, the variant looks easier because less threats will occur. Appearances are deceptive! ART-0D-RD also is not trivially serializable and each problem has a lot of solutions but not a single necessary goal ordering exists for any problem. The domain can be interpreted to model a resource allocation process: one half of the actions (implicitly) allocates the resource, the other half (implicitly) releases it; no two such actions must be parallel. The consequence is that it is not possible to find a necessary ordering $<_G$ such the domain is $<_G$-constrained trivially serializable. Of course, there are a lot of possible goal orderings for each solution plan and a problem becomes easy if such orderings are known.

Conclusion. The only difficulty of ART-0D-RD is a resource allocation problem which forces the establishers of the goals not to be parallel. This difficulty can only be tackled by possible goal orderings. But whenever we use possible goal orderings the planner gets incomplete by definition. This domain shows that goal orderings that are used for the ordered establisher condition don't allow to formulate necessary restrictions on all solutions, e.g, *actions of ART-0D-RD must not be parallel.* With goal orderings you can enumerate the possible restrictions and a set of atomic orderings means that the conjunction of these restrictions has to be true in a solution.

This also allows to understand the various difficulties of the workpiece domain. Here we have a combination of at least the three difficulties. The establisher ordering problem and the operator selection problem can be tackled by giving necessary goal orderings, not so the resource allocation problem. We tested a modified version of the workpiece domain in which the planner did not have to find the correct sequence of resource allocations. Here we found that the planning performance increases significantly when goal orderings are given compared to the performance when no goal orderings were given. The combination of several difficulty is the real source that make some domains difficult.

4.3.4 Summary: Domains and Difficulties

We summarized the critical difficulty of various domains known from literature and mentioned above in Table 1. For each domain or a set of domains the table shows the *characteristic difficulties* which are responsible for a certain degree of difficulty. As most domains are combinations of such difficulties we also list *secondary difficulties* which, however, do not change the overall difficulty significantly. This classification works well for artificial domain because they are in most cases explicitly constructed to show a certain difficulty. More realistic domains, e.g., the Workpiece domain (Muñoz-Avila and Weberskirch, 1996b), the FlatTyre domain (Russell, 1992), or the Logistics Transportation domain (Veloso, 1994) are a combination of all difficulties. In some we even can find, for example, multiple different resource allocation problems encoded.

Domain	Characteristic difficulties	Secondary	Subsumption
D^mS^1, D^mS^2, D^mS^{2*}	**Establisher ordering problem:** threats between establishers of different goals	—	—
$\Theta_n D^0 S^1$	**Operator selection problem:** Multiple (n) establishers for most of the goals, only one correct	—	—
$\Theta_n D^m S^1$, $\Theta_n^2 D^m S^1$	Operator selection problem	Establisher ordering	—
ART-0D-RD	**Resource allocation problem:** actions model the allocation of a resource that cannot be used by several consumers in parallel.	—	—
ART-1D-RD	Resource allocation problem	Establisher ordering	—
Link-Repeat	Resource allocation	—	$g_i \triangleright g_j \|_{j<i}$
Link-Chain	Operator selection	Establisher ordering	$g_j \triangleright g_i \|_{j<i-1}$
Workpiece	Combination of: Operator selection , Resource allocation	Establisher ordering	between some goals
FlatTyre	Combination	Establisher ordering	between some goals
Blocksworld, Transportation	Combination	Establisher ordering	—

Table 1: Characteristic difficulties of some planning domains

Table 2 summarizes the gain of efficiency when using goal orderings in terms of the classification using the refined taxonomy of subgoal collections. Nearly all domains are laboriously serializable for SNLP. It further shows whether a set of necessary or possible goal orderings can be used to make this domain $<_G$-constrained trivially serializable for CAPLAN. It also shows in which domain we have the property that the goal orderings following from solution plans (necessary or possible orderings) can also be used to gain advantage in a $<_G$-consistent goal selection strategy. Not in all domains this is clear at the moment.

Domain	SNLP	CAPLAN: $<_G$-constrained trivially serializable	$(I, G, <_e, <_a)$
D^mS^1, D^mS^2	trivially serializable	$<_G = \emptyset$	$<_e = <_a$
D^mS^{2*}	laboriously serializable	$necessary((I, G), <_G)$	$<_e = <_a$
$\Theta_n D^0 S^1$	laboriously serializable	$necessary((I, G), <_G)$	$<_e = <_a$
$\Theta_n D^m S^1$	laboriously serializable	$necessary((I, G), <_G)$	$<_e = <_a$
$\Theta_n^2 D^m S^1$	laboriously serializable	$necessary((I, G), <_G)$	$<_e \neq <_a <'$
ART-0D-RD	laboriously serializable	$possible((I, G), <_G, P)$	$<_e = <_a$
ART-1D-RD	laboriously serializable	$necessary((I, G), <_G)$	$<_e = <_a$
Link-Repeat	laboriously serializable	—	$< \neq <'$
Link-Chain	laboriously serializable	$necessary((I, G), <_G)$	$<_e = <_a$
Workpiece	laboriously serializable	$<_G \neq \emptyset$	
FlatTyre	laboriously serializable	$necessary((I, G), <_G)$	$<_e = <_a$ (?)
Blocksworld, Transportation	laboriously serializable	$possible((I, G), <_G, P)$	

Table 2: Possible effects of goal orderings in some planning domains

5 Empirical Evaluation

This section reports results of an empirical evaluation conducted with CAPLAN in order to validate the theoretical results of the previous sections. In CAPLAN all ideas presented in this report can be found implemented. We investigated the possible performance improvements of using extended problem descriptions $(I, G, <_G)$ with knowledge about goal orderings as done in CAPLAN over standard problem description (I, G) without any other problem-specific or domain-specific knowledge. We concentrated on $<_G$ being necessary goal orderings or subsets of that. Only some experiments use possible or impossible goal orderings. In some domains we performed tests with two different goal orderings, i.e., with problems of the form $(I, G, <_e, <_a)$ as explained in Section 3.3.5.

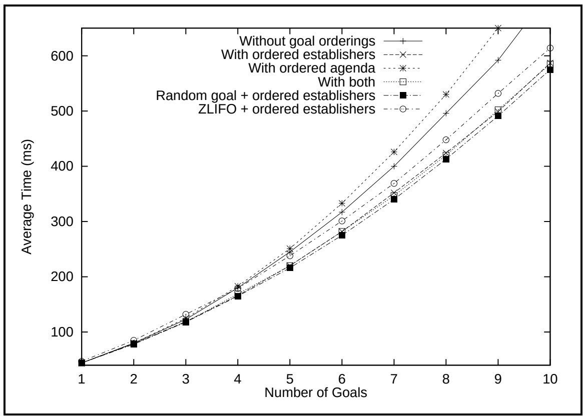
For the experiments we selected well known artificial domains already mentioned in previous sections: D^mS^2, D^mS^{2*}, $\Theta_n D^m S^1$ (Barrett and Weld, 1994), Link-Repeat, Link-Chain (Veloso and Blythe, 1994), ART-1D-RD (Kambhampati, 1994) and some new artificial domains focusing on certain special difficulties. Finally, some experiments were performed with the workpiece domain (Muñoz-Avila and Weberskirch, 1996b; Muñoz-Avila, 1998) and the FlatTyre domain (Russell, 1992). Both can be seen as domains combining the difficulties of several of the artificial domains (see Section 4.3.3). We conducted the experiments using well known control strategies like DSep, LCFR, ZLIFO and some with random goal selection.

As explained before, goal orderings are used for two purposes in CAPLAN: (1) ordered establisher condition (Section 3.1.1, Definition 8), (2) $<_G$-consistent goal selection strategy (Section 3.3.5). Both can be responsible for better performance of CAPLAN when using goal orderings, but the extent of improvement depends on properties of the domain.

We organized the experiments as follows:[19] each experiment concentrates on domains having a certain degree of difficulty in common. In all domains we conducted tests and compared four cases, where planning had to be done (a) without goal orderings, (b) with the ordered establisher condition, (c) with $<_G$-consistent goal selection strategy, (d) both. The general setup of all experiments was such that the planner had to solve each problem 30 or more times and we measured the average time per problem and the average number of inference steps within some reasonable maximal time limit varying between 30s and 50s. To be sure that the order in which the goals are defined in the planning problem has no effect on the results, we added the goals using random permutations.[20]

5.1 Experiment 1: Trivially Serializable Domains

The first experiment examines the effect of goal orderings in a domain that is trivially serializable and in which we only have the difficulty of an establisher ordering problem. Section 4.3.1 has shown that we cannot gain advantage of the $<_G$-consistent goal selection strategy in that case. But the ordered establisher condition, which allows CAPLAN to search in a more restricted class of plans, will lead to a smaller search space.



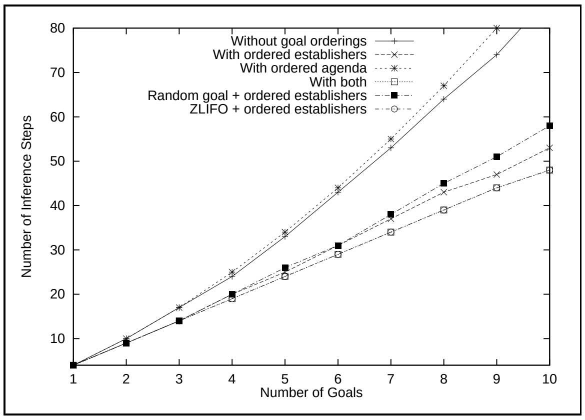


Figure 14: Total planning time and number of inference steps in the D^1S^2 domain

Figure 14 show the average planning time and the average number of inference steps with an increasing number of goals in the artificial domain D^1S^2. A $<_G$-consistent goal selection strategy decreases performance as it only creates computational overhead when sorting the goal in a certain way. On the other hand, the values for the ordered establisher condition and both are nearly the same, so in this domain the combination of both is nevertheless very good. This can be explained by the fact that ordering establishers as done by the ordered establisher condition is exactly the right thing to do in case of an establisher ordering problem. Also a random goal selection strategy or ZLIFO combined with the ordered establisher condition produce nearly the same results. ZLIFO cannot be better because if performs more time consuming tests when selecting a goal that SNLP or random goal selection. For domain D^mS^2 the results are qualitatively the same but the times are a little higher as there are more threats to resolve (not shown here).

[19]All experiments were performed with CAPlan v2.15 on a Pentium II/266 PC under Windows NT 4.0.
[20]This can be configured in CAPLAN by setting the flag 'Shuffle finish' to on.

5.2 Experiment 2: Laboriously Serializable Domains

The second experiment focuses on more difficult domains in which at least a part of the overall difficulty can be explained by the fact that the combination of several goals causes an additional difficulty for the planner.

5.2.1 Difficulty: Establisher Ordering Problem

The first test was performed in domain $D^m S^{2*}$ that is laboriously serializable only because of an establisher ordering problem. But we can find necessary goal orderings which make it $<_G$-constrained trivially serializable for CAPLAN.

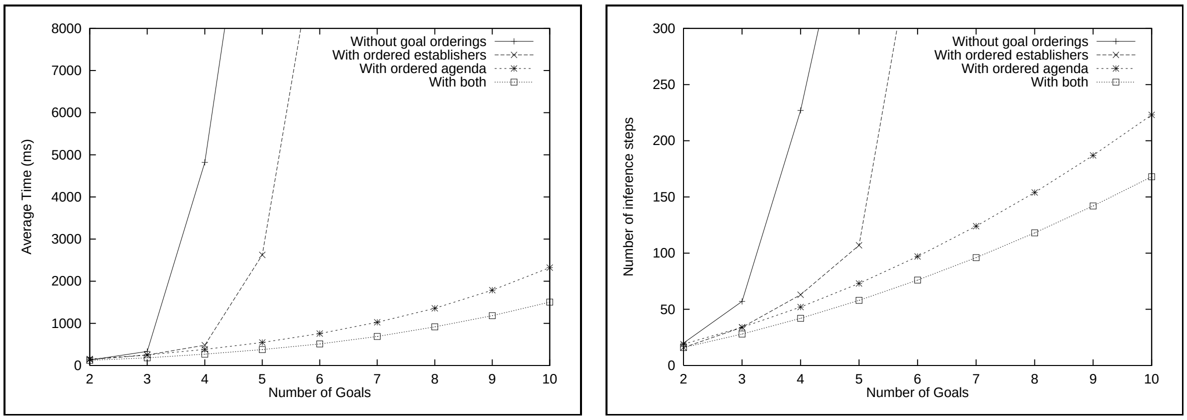


Figure 15: Total planning time and number of inference steps in the $D^m S^{2*}$ domain

Figure 15 shows that using goal orderings to order establishers improves the planning time and the number of inference steps, but the improvement is much bigger when using the necessary goal ordering for goal selection. The improvement by the $<_G$-consistent goal selection strategy is very clear in this domain.

5.2.2 Difficulty: Operator Selection Problem

Although the characteristic difficulty of the domain $\Theta_2 D^1 S^1$ is not the same a for the domain $D^m S^{2*}$ (see Section 4.3.2), both are not trivially serializable and the experiments (Figure 16) show results very similar to the ones for $D^m S^{2*}$. The right goal selection strategy is the key information for solving the problems.

We ran the same problems in $\Theta_2 D^0 S^1$ in which we don't have an establisher ordering problem (see Section 4.3.2) and found similar results (Figure 17). This shows that the operator selection problem in Θ_2 transformed domains dominates the establisher ordering problem.

In both domains we also tested whether it always is the best to have as much (necessary) goal ordering as possible. The answer is no (see also Lemma 4 on page 22). There is a trade-off between the number of additional orderings to be used by the ordered establisher condition and the extra effort created by them when searching the plan. For goal selection the same is not observed because they are used just once when the goals are added to the agenda of goals. We performed a test in $\Theta_2 D^1 S^1$ with using goal orderings for both purposes but where we only give the minimal set of necessary goal orderings such that the problems still are $<_G$-constrained trivially serializable. We get almost the same times per problem

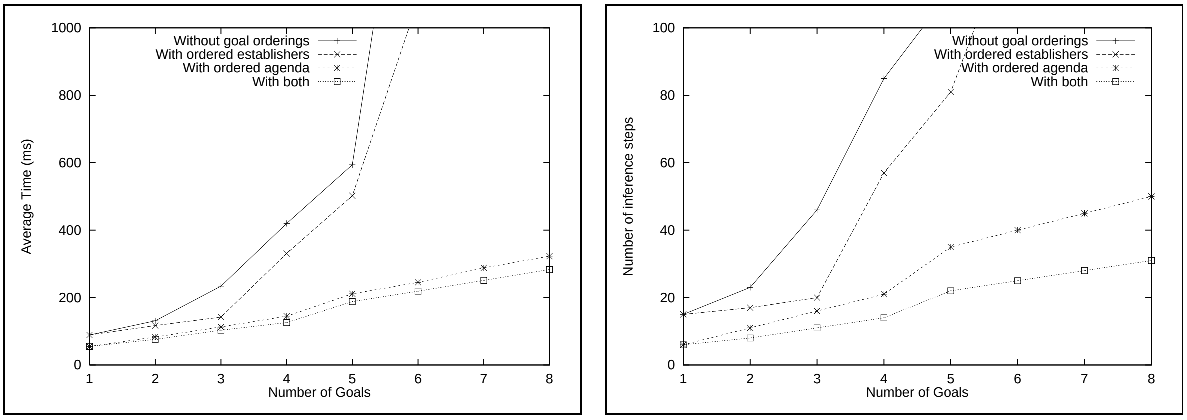


Figure 16: Total planning time and number of inference steps in the $\Theta_2 D^1 S^1$ domain

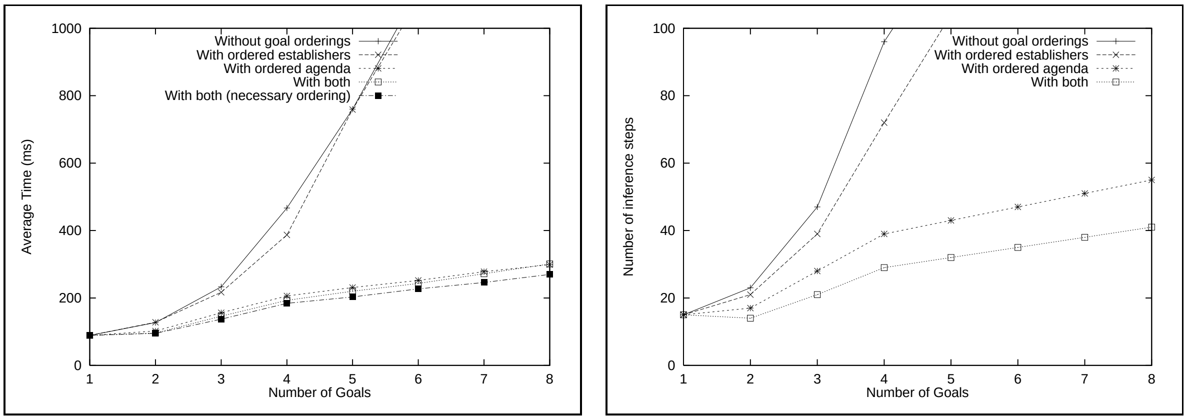


Figure 17: Total planning time and number of inference steps in the $\Theta_2 D^0 S^1$ domain

but the numbers of inferences steps per problem might be higher. This indicates that less orderings improve the average time per inference step and sometimes also the overall time. But the difference is very small. If we use the same set of orderings $\Theta_2 D^1 S^1$ we get a better performance (average time per problem) that in the first four test shown in Figure 17. The reason is that the first four tests use a possible ordering, which focuses on one special solution plan. The reduced set, however, is a necessary ordering in this domain which makes it much easier to find one of the many possible solutions.

5.2.3 Difficulty: Resource Allocation Problem

The difficulty of resource allocation seems to create the hardest class of domains. Figure 18 shows, however, that the right goal selection strategy is the key information that makes problems become easy. For ART-1D-RD a goal selection strategy using the maximal set of necessary orderings takes away the difficult task of finding a sequence of resource allocations. The fact that this optimal ordering is a necessary ordering is produced by the additional establisher ordering problem also encoded in ART-1D-RD because it is based on $D^1 S^1$.

If we take away the establisher ordering problem as done in ART-0D-RD we can observe that problems get much easier even for SNLP without any additional help (Figure 19). This can be explained by the fact that each problems has much more than only one correct solution in

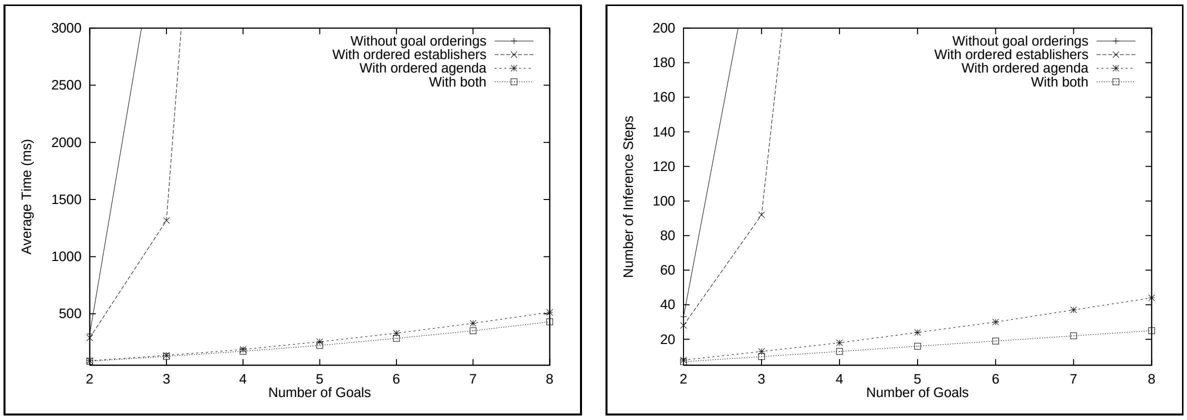


Figure 18: Total planning time and number of inference steps in the ART-1D-RD domain

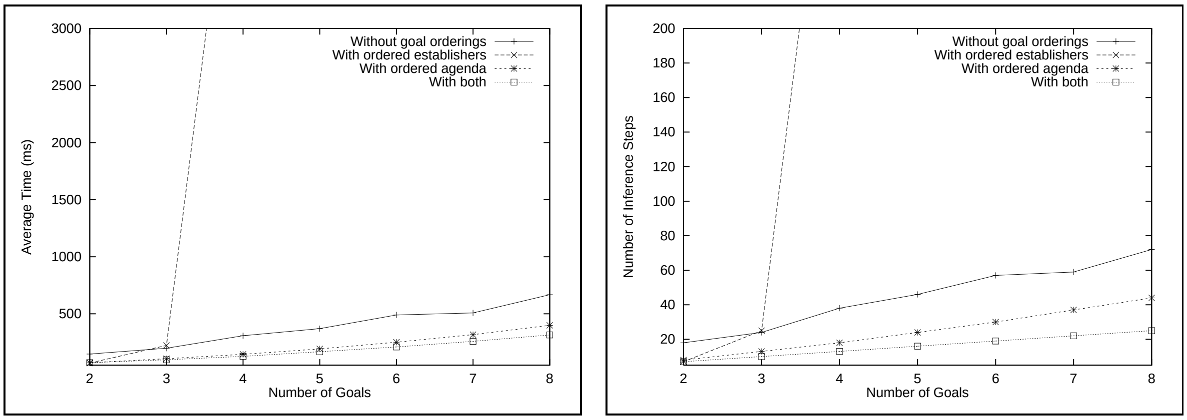


Figure 19: Total planning time and number of inference steps in the ART-0D-RD domain

ART-0D-RD. The test used the same set of goal orderings as in ART-1D-RD which, however, is only a possible goal ordering for problems in ART-0D-RD. This explains why using goal orderings for the ordered establisher condition does decrease performance.

These results indicate that a good control strategy is more important than the knowledge about necessary or possible goal orderings, especially, if the main difficulty of the domain is a resource allocation problem. In case such a resource allocation problem is combined with another difficulty, e.g., an establisher ordering problem, the relative performance improvements of using goal orderings is much bigger.

5.3 Experiment 3: A Single Goal Ordering Is Not Always Enough

In Section 3.3.7 we realized that there are domains in which we need different goal orderings to be used as defined by the ordered establisher condition and for defining a goal selection strategy. We mentioned two artificial domains in which we find this phenomenon, LinkRepeat (see Appendix A.5) and $\Theta_2^2 D^m S^1$ (see Appendix A.3).

Link-Repeat: As explained in the proof of Theorem 1, the use of maximal necessary goal orderings for a goal selection strategy in this domain produces a larger search space than not using goal ordering at all. This can also be observed in our experiments (Figure 20). Using

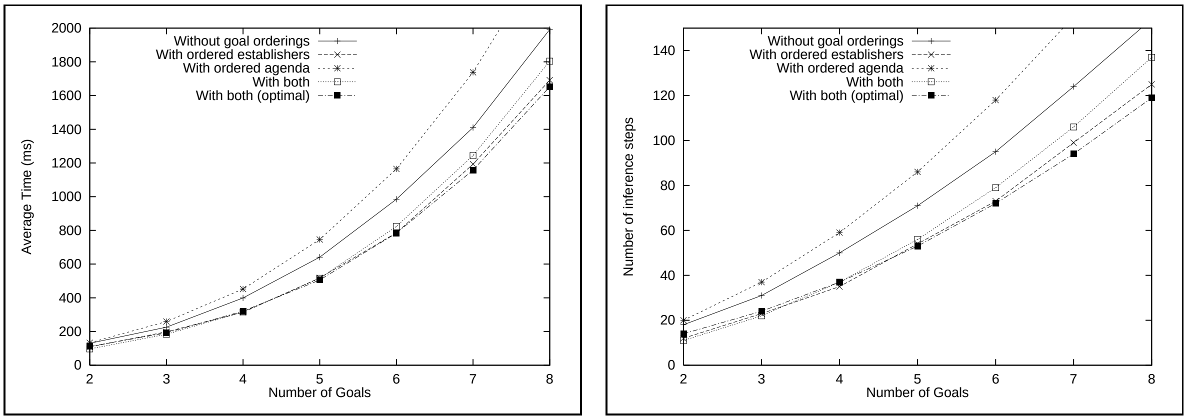


Figure 20: Total planning time and number of inference steps in the LinkRepeat domain with necessary goal orderings

the same ordering to order establishers still improves performances, but the combination with the goal selection strategy based on the same ordering partially ruins this improvement. The best results are achieved if we use the generalized version of extended problem specifications, $(I, G, <_e, <_a)$, with two different goal orderings: $<_e = \{(g_i, g_{i-1})|_{\forall i}\}$ for ordering establishers and the reverse ordering $<_a = \{(g_{i-1}, g_i)|_{\forall i}\}$ for the goal selection strategy. This effect however becomes visible only for larger problems with more than 5 goals (Figure 20).

$\Theta_2^2 D^m S^1$: The Θ_n^2 transformation of domains has been designed to make the effect observed in LinkRepeat more clear. Surprisingly, only a very little modification is necessary to achieve this objective by reusing Barrett and Weld's Θ_n transformation (Barrett and Weld, 1994). The difficulty of Θ_n^2 transformed $D^m S^1$ domains is the nearly the same than the difficulty the Θ_n transformation, in particular, both domains are laboriously serializable. We will illustrate this in more detail with some experiments in $\Theta_2^2 D^1 S^1$. In the experiments we compare the performance (planning time and number of inference steps) for a set of typical problems of size 2 to 8 (see also Appendix A.3) with the same general setup as described above. We solved such sets of problems and compared the performance with different sets of goal orderings being used for different purposes. We tested the problem sets with combinations of the following goal orderings:

$< = \{(G_{i-1}, G_i)|_{i=2\ldots n}\}$ with $necessary(prb, <))$ in $D^m S^1$ and all Θ_n/Θ_n^2 variants,

$<' = \{(G_\alpha, G_i)|_{\forall i}\}$ with $necessary(prb, <))$ in Θ_n variants but $impossible(prb, <))$ in Θ_n^2,

$<'' = \{(G_i, G_\alpha)|_{\forall i}\}$ the same vice versa.

With these basic sets of orderings, we tested the problems $(I, G, <_e, <_a)$ with three combinations of goal orderings:

(a) $<_e = <_a = <$, **(b)** $<_e = <_a = < \cup <''$, **(c)** $<_e = < \cup <''$, $<_a = < \cup <'$

Table 3 shows the results of this series of experiments when using no goal ordering, when using goal orderings only as defined by the ordered establisher condition (OEC), when using goal ordering only for defining a goal selection strategy (GSS), or when using goal orderings for both (OEC+GSS). Unlike $\Theta_2 D^1 S^1$, in $\Theta_2^2 D^1 S^1$ the maximal necessary orderings, combination (b), produces the worst goal ordering w.r.t. a $<_G$-consistent goal selection strategy although it is still the optimal one w.r.t. the ordered establisher condition. But the

Goal orderings /usage of goal orderings	none	OEC	GSS	OEC+GSS
none	1105 (94)	—	—	—
set (a)	—	840 (71)	366 (55)	328 (38)
set (b)	—	1017 (71)	1076 (140)	646 (61)
set (c)	—	1057 (70)	329 (50)	230 (22)

Table 3: Results for $\Theta_2^2 D^m S^1$ domain. The values show the average time in ms for the complete problem set and all iterations, the values in parentheses are the average numbers of inference steps.

ordered establisher condition only solves the secondary difficulty of the domain which does not dominate the performance. Even the subset $<$, combination (a), solving only the establisher ordering problem of $D^m S^1$ produces better results. The optimal strategy is to use combination (c). Surprisingly, the latter is also an impossible ordering.

5.4 Experiment 4: A-Posteriori Goal Orderings

In another experiment we extracted maximal possible orderings from solution plans and tried to solve the problem with this maximal possible goal orderings. Figure 21 show such results that were performed in the LinkChain (see Appendix A.5 for the definition) which also has been designed to be hard to solve for causal-link planners.

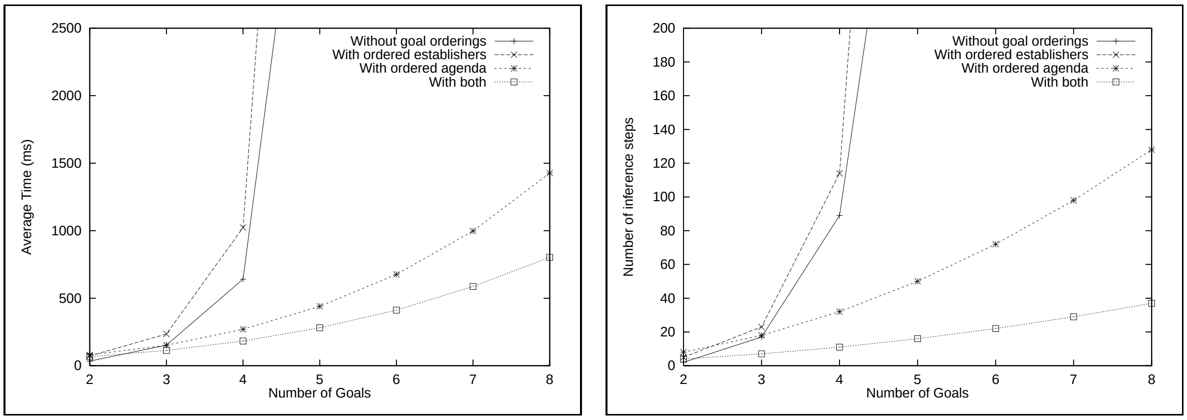


Figure 21: Total planning time and number of inference steps in the LinkChain domain with maximal possible goal orderings

5.5 Experiment 5: Non-Artificial Domains

Finally, we performed some experiments in two non-artificial domains, the Workpiece domain (Muñoz-Avila and Weberskirch, 1996b; Muñoz-Avila, 1998) and the FlatTyre domain (Russell, 1992). In both we selected some representative problems and compared the problem solving time with goal orderings and without goal orderings.

Workpiece domain: We selected problems with 2 to 10 goals. The overall result was that goal orderings helped to reduce the problem solving time to about 70% to 80% of the problem solving time without goal orderings.

FlatTyre domain: We tested the typical problem of that domain, `fixit`, which consists of eight goals and is solved by a plan containing 19 actions. We created a copy of this problem, `fixit-ord`, and added a set of eight orderings between these goals which is a necessary orderings and compared the solution time and the number of inference steps. Another variant,`fixit-x-ord` was created by adding two more (intermediate) goals and also six more goal orderings. The results can be found in Table 4. The show that the problem solving time is improved and also the number of inference steps. On the other hand we see, that although we have much less inference steps for the last problem, the time per inference step is increased because of the additional orderings in the plan.

Problem	Time (ms)	Inference steps
fixit	4462	114
fixit-ord	4434	82
fixit-x-ord	4102	68

Table 4: Results for FlatTyre domain

In both domains we have to take into account that we have a combination of all characteristic difficulties. In particular, we find several resource allocation problems in the Workpiece domain. This make both domains significantly more difficult that artificial domains which focus on one difficulty. Additionally, we typically have variables and objects which are used in nearly every operator and have to be propagated by the planner. This increases the average time per inference step in these domains.

6 Conclusion

In this report we investigated the problem of using goal orderings for a partial-order, plan-space planner. We presented two ways how planners based on the SNLP algorithm (McAllester and Rosenblitt, 1991) might use goal orderings. Both can be found implemented in the planner CAPLAN:

1. The **ordered establisher condition** (see Section 3.1.1) leads to the notion of necessary, possible, and impossible orderings, and to the class of elastic protected $<_G$-consistent plans in which goal orderings are translated into orderings between establishers of goals.
2. A **$<_G$-consistent goal selection strategy** (Section 3.3.5) addresses the control problem of goal selection and guarantees that only $<_G$-consistent permutations of the goals are considered for goal selection.

Basically, the ideas to use goal orderings in that way have been mentioned in literature separately before. For example, Cheng and Irani gave a definition of goal orderings that is comparable with the first one (Cheng and Irani, 1989), while others define goal orderings as the problem of deciding on which goal to work next (Drummond and Currie, 1989). However, until now both mechanisms have not been explicitly distinguished as two completely different ideas, in particular, for plan-space planners. It is also a common wisdom that using goal orderings can improve planning performance. But suprisingly, most of research about the use and the automatic computation of goal orderings can be found in the area state-space planner (Minton, 1988; Ryu and Irani, 1992; Etzioni, 1993a; Etzioni, 1993b).

We have now shown that both interpretations of goal orderings can be very different for plan-space planners based on SNLP. Basically, this is a consequence of the fact that there is no necessary relation between the order in which goals are processed and the ordering in which they are achieved in a plan. Ordering establishers addresses the problem of reducing the size of the search space while a $<_G$-consistent goal selection strategy addresses the control problem of planning. We proved that the ordered establisher condition effectively reduces the size of the search space while a $<_G$-consistent goal selection strategy only addresses the problem of navigating in search space.

If we compare both ways to gain advantage of goal orderings the goal selection strategy has the potential to shift the problems from the class of laboriously serializable to the class of problems that are as easy for CAPLAN as trivially serializable problems for SNLP. We introduced the notion of $<_G$-constrained trivial serializability which allows to classify problems/domains to be comparably easy ($<_G$-constrained trivially serializable) for CAPLAN. This class contains all trivially serializable problems/domains. Additionally, those laboriously serializable problems are included for which we find a goal ordering $<_G$ such that all $<_G$-consistent permutations of the goals are serialization orders. In that context we also pointed out that serializability is not meaningful enough to characterize a domain or problem as easy or difficult. But it allows to judge whether problems with collections of subgoals can be significantly more difficult to solve than the individual goals. We have also shown that is is a property of the domain whether a necessary or possible goal orderings that represent conditions that are valid in all or at least some solution plans might also be useful for defining a goal selection strategy. This is one of the most important results of this report. In worst case, we need totally different goal orderings that are then used for ordering establishers and for defining the goal selection strategy.

Finally, we searched for sources of goal orderings in various domain. In particular, in artificial domains we identified three characteristic difficulties that are responsible for the number of existing goal orderings:

- the establisher ordering problem,
- the operator selection problem, and
- the resource allocation problem.

We gave examples of domains in which such characteristic difficulties can be found separately and how such elementary difficulties are typically combined in more realistic planning domains. All theoretical results have also been validated in a large number of experiments that have been performed during a period of several months.

Acknowledgement

We would like to thank Jochem Hüllen for his very helpful comments and discussions on earlier versions of this report.

References

Barrett, A. and Weld, D. (1993). Characterizing subgoal interactions for planning. In *Proceedings of IJCAI-93*, pages 1388–1393.

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

Bergmann, R. and Kott, A., editors (1998). *Integrating Planning, Scheduling and Execution in Dynamic and Uncertain Environments*, AAAI Technical Report WS-98-02. AAAI Press.

Chapman, D. (1987). Planning for conjunctive goals. *Artificial Intelligence*, 32:333–377.

Cheng, J. and Irani, K. (1989). Ordering problem subgoals. In *Proceedings of IJCAI-89*, pages 931–936.

Currie, K. and Tate, A. (1991). O-plan: The open planning architecture. *Artificial Intelligence*, 52(1):49–86.

Doyle, J. (1979). A truth maintenance system. *Artificial Intelligence*, 12(3):231–272.

Drummond, M. and Currie, K. (1989). Goal ordering in partially ordered plans. In *Proceedings of IJCAI-89*, pages 960–965.

Erol, K. (1995). *Hierarchical Task Network Planning: Formalization, Analysis, and Implementation.* PhD thesis, University of Maryland.

Erol, K., Hendler, J., and Nau, D. (1994). Htn planning: Complexity and expressivity. In *Proceedings of AAAI-94*, pages 1123–1128.

Erol, K., Hendler, J., and Nau, D. (1996). Complexity results for hierarchical task-network planning. *Annals of Mathematics and Artificial Intelligence*, 18:69–93.

Etzioni, O. (1993a). Acquiring search-control knowledge via static analysis. *Artificial Intelligence*, 62:255–301.

Etzioni, O. (1993b). A structural theory of explanation-based learning. *Artificial Intelligence*, 60:93–139.

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

Gerevini, A. and Schubert, L. (1996). Accelerating partial-order planners: Some techniques for effective search control and pruning. *Journal of Artificial Intelligence Research*, 5:95–137.

Irani, K. and Cheng, J. (1987). Subgoal ordering and goal augmentation for heuristic problem solving. In *Proceedings of IJCAI-87*, pages 1018–1024.

Joslin, D. and Pollack, M. (1994). Least-cost flaw repair: A plan refinement strategy for partial-order planning. In *Proceedings of AAAI-94*, pages 1004–1009.

Kambhampati, S. (1994). Multi-contributor causal structures for planning: A formalization and evaluation. *Artificial Intelligence*, 69:235–278.

Kambhampati, S. (1995). Admissible pruning strategies based on plan minimality for plan-space planning. In *Proceedings of IJCAI-95*.

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 Intern. Conf. on AI Planning Systems (AIPS-96)*, pages 125–133.

Kambhampati, S., Knoblock, C., and Yang, Q. (1995). Planning as refinement search: A unified framework for evaluating design tradeoffs in partial order planning. *Artificial Intelligence*, 76:167–238.

Kambhampati, S., Mali, A., and Srivastava, B. (1998). Hybrid planning for partially hierarchical domains. In *Proceedings of AAAI-98*.

Korf, R. (1987). Planning as search: A quantitative approach. *Artificial Intelligence*, 33:65–88.

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

Minton, S. (1988). *Learning Search Control Knowledge: An Explanation-Based Approach*. Kluwer Academic Publishers, Boston.

Muñoz-Avila, H. (1998). *Integrating Twofold Case Retrieval and Complete Decision Replay in CAPlan/CbC*. PhD thesis, University of Kaiserslautern.

Muñoz-Avila, H. and Hüllen, J. (1995). Retrieving relevant cases by using goal dependencies. In Veloso, M. and Aamodt, A., editors, *Case-Based Reasoning Research and Development, Proc. of the 1st Intern. Conference (ICCBR-95)*, number 1010 in LNAI. Springer.

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

Muñoz-Avila, H. and Weberskirch, F. (1996b). A specification of the domain of process planning: Properties, problems and solutions. Technical Report LSA-96-10E, Centre for Learning Systems and Applications, University of Kaiserslautern, Germany.

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

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

Pollack, M., Joslin, D., and Paolucci, M. (1997). Flaw selection strategies for partial-order planning. *Journal of Artificial Intelligence Research*, 6:223–262.

Russell, S. (1992). Efficient memory-bounded search methods. In Neumann, B., editor, *Proceedings of the 10th Europ. Conf. on Artificial Intelligence (ECAI-92)*, pages 1–5.

Ryu, K. and Irani, K. (1992). Learning from goal interactions in planning: Goal stack analysis and generalization. In *Proceedings of AAAI-92*, pages 401–407.

Smith, D. and Peot, M. (1993). Postponing threats in partial-order planning. In *Proceedings of AAAI-93*, pages 500–506.

Srivastava, B. and Kambhampati, S. (1998). Synthesizing customized planners from specifications. *Journal of Artificial Intelligence Research*, 8:98–128.

Veloso, M. (1994). *Planning and learning by analogical reasoning*. Number 886 in Lecture Notes in Artificial Intelligence. Springer Verlag.

Veloso, M. and Blythe, J. (1994). Linkability: Examining causal link commitments in partial-order planning. In *Proceedings of the 2nd Intern. Conf. on AI Planning Systems (AIPS-94)*, pages 13–19.

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.

Weberskirch, F. and Muñoz-Avila, H. (1997). Advantages of types in partial-order planning. In Müller, M., Schumann, O., and Schumann, S., editors, *Beiträge zum 11. Workshop 'Planen und Konfigurieren' (PuK-97)*, number FR-1997-001 in FORWISS-REPORT. FORWISS.

Weld, D. (1994). An introduction to least commitment planning. *AI Magazine*, 15(4):27–61.

Wilkins, D. (1988). *Practical Planning - Extending the classical AI Planning Paradigm*. Morgan Kaufmann.

A Domain Specifications

This section gives details of the definition of several artificial domain used throughout this report. Most of the domains can be found in literature but it is advantageous to give an overview of the definitions as some of them are needed in detail.

When we describe a domain, we will list the possible predicates that occur in a domain only in case this is not straightforward. The most interesting part is the definition of the operators. We use the notation

$$\{\text{preconditions}\} \text{ operator name } \{\text{effects}\}$$

for the description of an operator. Positive and negative effects (ADD/DELETE lists) are distinguished by adding a sign: $+P$ or simply P is a positive literal, $-P$ is a negative literal. The same is true for preconditions as CAPLAN also allows positive and negative preconditions. If necessary, we will also define a typical problem, the possible solutions, and the maximal set of necessary goal orderings.

A.1 D^mS^1, D^mS^2, D^mS^{2*}

These domains where originally presented in (Barrett and Weld, 1994). A typical problem (I, G) of size n is given by $I = \{I_1, \ldots, I_n\}$ and $G = \{G_1, \ldots, G_n\}$. The domains differ only in the operators that are available.

D^mS^1: All operators are of the form $\{I_i\}\ A_i\ \{+G_i; -I_j|_{j<i}\}$. There exists one solution to such a problem; it consists of a totally ordered sequence of $A_1, A_2, \ldots, A_n$.

D^mS^2: This domains consists of two levels of operators for each goal G_i which are needed to achieve the goal: $\{I_i\}\ A_i^1\ \{+P_i; -I_j|_{j<i}\}$, $\{P_i\}\ A_i^2\ \{+G_i; -I_j|_{\forall j}, -P_j|_{j<i}\}$. There exists one solution to such a problem; it consists of a totally ordered sequence of $A_1^1, A_2^1, \ldots, A_n^1, A_1^2, A_2^2, \ldots, A_n^2$.

D^mS^{2*}: Here we have slight modification w.r.t. D^mS^2, which makes the domain significant more difficult. There is an additional goal G_* and one action that achieves it. We have the following operators: $\{I_i\}\ A_i^1\ \{+P_i; -P_j|_{j<i}\}$, $\{P_i\}\ A_i^2\ \{+G_i; -P_j|_{j<i}\}$, $\{I_*\}\ A_*\ \{+G_*; -I_i|_{\forall i}, -P_i|_{\forall i}\}$.[21] A typical problems of size n here has also G_* as a goal, i.e., $G = \{G_1, \ldots, G_n, G_*\}$, and shows the typical difficulty of this domain. Again, there exists one solution consisting of a totally ordered sequence of actions $A_n^1, \ldots, A_2^1, A_n^1, A_*, A_n^2, \ldots, A_2^2, A_1^2$.

In these domains we therefore have necessary goal orderings of the form $G_{i-1} <_G G_i$ for $i = 2, \ldots, n$. In D^mS^{2*} we additionally have $G_* <_G g_i$ for $i = 1, \ldots, n$.

A.2 Θ_n-Variants of D^mS^1 Domains: $\Theta_2 D^mS^1$, $\Theta_2 D^0S^1$

These domains where also originally presented in (Barrett and Weld, 1994). The Θ_n transformation of a domain adds the difficulty of an operator selection problem (see Section 4.3.2) to the base domain, because only one alternative of n is correct in an overall solution. All problems with the additional goal G_α are significant more difficulty. A typical (difficult) problem (I, G) of size n is given by $I = \{I_1, \ldots, I_n, P_\alpha, P_\beta\}$ and $G = \{G_1, \ldots, G_n, G_\alpha\}$.

[21]Notice, the main difference to D^mS^2 is that the effect $-P_j|_{j<i}$ occurs in actions of both levels, A_i^1 and A_i^2!

$\Theta_2 D^m S^1$**:** The definition of the operators is as follows:

$$\{I_i, P_\alpha\}\ A_i^\alpha\ \{+G_i; -I_j|_{j<i}\}, \quad \{I_i, P_\beta\}\ A_i^\beta\ \{+G_i; -I_j|_{j<i}\} \quad \text{and}$$
$$\{\}\ A_\alpha\ \{+G_\alpha; -P_\beta, -G_i|_{\forall i}\}$$

There exists exactly one solution to such a typical problem and it consists of a totally ordered sequence of actions $A_\alpha, A_1^\alpha, \cdots, A_n^\alpha$.

$\Theta_2 D^0 S^1$**:** We also used a variant which differs only by the fact that there are not interactions between the actions A_i^α or A_i^β:

$$\{I_i, P_\alpha\}\ A_i^\alpha\ \{+G_i\}, \quad \{I_i, P_\beta\}\ A_i^\beta\ \{+G_i\} \quad \text{and}$$
$$\{\}\ A_\alpha\ \{+G_\alpha; -P_\beta, -G_i|_{\forall i}\}$$

Here, there a many solutions, all start with A_α which is followed by all the A_i^α steps which do not need to be ordered with respect to each other.

In $\Theta_2 D^m S^1$ we have $maxNecessary((I,G)) = \{G_\alpha <_G G_i|_{i=1..n}, G_i <_G G_{i+1}|_{i=1..n-1}\}$, in $\Theta_2 D^0 S^1$ it is a subset because of the missing establisher ordering problem, $\{G_\alpha <_G G_i|_{i=1..n}\}$.

A.3 Θ_n^2-Variants of $D^m S^1$ Domains

Here we will define a new variant of the $D^m S^1$ domains which is very closely related to the domains create by the so-called Θ_n transformation (see Appendix A.2). The domain named $\Theta_2^2 D^m S^1$ is a variant of $\Theta_2 D^m S^1$. There is one more operator which, however, does not cause an additional problem for an SNLP planner. But this domain has a very different property with respect to the computation and usage of goal orderings because the optimal goal ordering w.r.t. a $<_G$-consistent goal selection strategy is different from the optimal ordering w.r.t. the ordered establisher condition.

The definition of the operators A_i^α and A_i^β is the same as in $\Theta_2 D^m S^1$. But instead of a single A_α there are two actions, A_α^1 and A_α^2, that are necessary for achieving the goal G_α. The definition of these two operators is as follows:

$$\{\}\ A_\alpha^1\ \{+X_\alpha; -P_\beta, -G_i|_{\forall i}\}, \quad \{+X_\alpha\}\ A_\alpha^2\ \{+G_\alpha; -P_\alpha, -P_\beta, -I_i|_{\forall i}\}$$

This domain uses the same typical problems as $\Theta_2 D^m S^1$ and there also exists exactly one solution to a problem. It consists of a totally ordered sequence of actions $A_\alpha^1, A_1^\alpha, \ldots, A_n^\alpha, A_\alpha^2$.

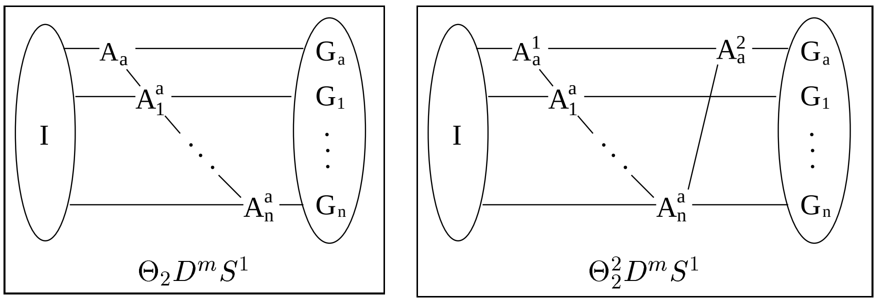


Figure 22: Solution plan for problem of size n in $\Theta_2 D^m S^1$ and $\Theta_2^2 D^m S^1$

The interesting point is that in each plan the goal G_α effectively is achieved after all goals G_i are achieved. Figure 22 shows the different solution plans for a problem of size n in $\Theta_2 D^m S^1$ and $\Theta_2^2 D^m S^1$. It follows that the ordering $G_i \prec_O G_\alpha$ is a necessary ordering in $\Theta_2^2 D^m S^1$ but an impossible ordering in $\Theta_2 D^m S^1$. On the other hand, $G_\alpha \prec_O G_i$ is an impossible ordering in $\Theta_2^2 D^m S^1$ but a necessary ordering in $\Theta_2 D^m S^1$. Nevertheless, the best goal selection strategy must select G_α first, in particular, the maximal necessary ordering of a $\Theta_2 D^m S^1$ problem is the optimal ordering for a goal selection strategy in both domains.

A.4 ART-1D-RD and the Variant ART-0D-RD

These domains ART-1D-RD and ART-MD-RD variants of Barrett and Weld's D^1S^1 and D^mS^1 domains. The main difference is the presence of a resource allocation problem which is encoded implicitly in the actions that achieve the typical goals. This encoding makes uses of two high-frequency conditions h_f, h_e (Kambhampati et al., 1995) which are needed and deleted alternately by the different actions. This can be interpreted as an resource allocation process: actions with an odd index allocate the resource, action with even index release the resource. This makes the domain significant more difficult. A typical problem (I, G) of size n is given by $I = \{I_1, \ldots, I_n, h_f\}$ and $G = \{G_1, \ldots, G_n\}$.

The operator are defined as follows:

ART-1D-RD: There are two classes of operators, the one needs h_f and produces h_e, the other does it in the opposite way.

$\{I_i, h_f\}\ A_i\ \{+G_i, +h_e; -I_{i-1}, -h_f\}$ for $i = 1, 3, 5, \cdots$
$\{I_i, h_e\}\ A_i\ \{+G_i, +h_f; -I_{i-1}, -h_e\}$ for $i = 2, 4, 6, \cdots$

The typical solution for a problem of size n is the totally ordered sequence of actions $A_1, \cdots, A_n$.

ART-0D-RD: The only difference to ART-1D-RD here we have only interactions because of the resource allocation problem encoded by h_f, h_e.

$\{I_i, h_f\}\ A_i\ \{+G_i, +h_e; -h_f\}$ for $i = 1, 3, 5, \cdots$
$\{I_i, h_e\}\ A_i\ \{+G_i, +h_f; -h_e\}$ for $i = 2, 4, 6, \cdots$

A typical solution for a problem of size n is a totally ordered sequence of actions with the restriction that an action with odd index must always be followed and preceded by an action with even index.

A.5 Link Repeat, Link Chain

These both domains have explicitly designed to show advantages of Means-Ends-Analysis planners like PRODIGY over causal-link planners like SNLP (Veloso and Blythe, 1994). In both cases SNLP has more difficulties mainly because it normally prefers simple establishment to step addition, but in both domains the ideal strategy is to work first on those goals for which simple establishment is not possible (with is the default strategy of a MEA planner).

Link Repeat: A typical problem (I, G) of size n is given by $I = \{g_*\}$ and $G = \{g_1, \ldots, g_n, g_*\}$. The actions are $\{g_*, g_{i-1}\}\ A_i\ \{+g_i, -g_*\}$ and $\{\}\ A_*\ \{+g_*\}$.

A typical solution plan is a totally ordered sequence of actions like $A_1, A_*, A_2, A_*, \ldots$.

Link Chain: A typical problem (I, G) of size n is given by $I = \{g_1, \ldots, g_{n-1}\}$ and $G = \{g_1, \ldots, g_n\}$. The actions are $\{g_j|_{\forall j<i}\}\ A_i\ \{+g_i, +g_j|_{\forall j<i-1}, -g_{i-1}\}$.

A typical solution plan is a totally ordered sequence of actions like $A_n, \ldots, A_1$, but there are many possible other solution plans with more than n steps. Depending on the control strategy the minimal plan can not be guaranteed to be found.

LinkRepeat has the property that we have an extreme form of goal subsumption, $g_i \triangleright g_j|_{\forall j<i}$, but nevertheless, this domain is quite easy as all goals are easy to achieve and at least all problems with goal that are a combination of the goals g_i are trivially serializable. The additional goal g_* destroys this property but the problems don't get really more complicated as g_* is very easy to achieve with an action that interacts with no other action.

The typical problems of LinkChain are difficult for SNLP because each operator can achieve nearly every goal and all goals but one are already true in the initial situation. For n goals there are $n-1$ for which SNLP first tries simple establishment, but none of them is correct in an overall solution.

B Proofs

B.1 Properties of Goal Orderings

Lemma 1: Properties of and maximal possible orderings (see page 14)

Proof:

1. Definition 14 says that the maximal possible orderings are inferred from the plan step orderings between the establishers of planning goals. SNLP uses the concept of causal links to record which step serves as the establisher of an open condition and it allows exactly one establisher for any goal (unlike other approaches, e.g., TWEAK (Chapman, 1987) or multi-contributor approaches (Kambhampati, 1994)). As we also have transitivity for $\prec_O$ there exists one unique set of orderings between the establishers $\{establisher(g_i) \prec_O establisher(g_j) | g_i, g_j \in G\}$ which maps to exactly one set of maximal possible goal orderings (Definition 14). (q.e.d.)

2. There cannot exist possible orderings that are a superset of $maxPossible((I,G),P)$ for some $P \in solutions((I,G))$. To proof this, let $<'$ be a possible ordering w.r.t. an arbitrary P and let us first assume $<' \supset <= maxPossible((I,G),P)$. Then there must exist some $(g_i <' g_j) \notin maxPossible((I,G),P)$ (*). But because $<'$ is a possible goal ordering we have $g_i <' g_j \Rightarrow establisher(g_i) \prec_O establisher(g_j)$ (Definition 8/10/12) and also $establisher(g_i) \prec_O establisher(g_j) \Rightarrow g_i < g_j$ (**), which means $(g_i <' g_j) \in maxPossible((I,G),P)$. This is a contradiction with (*), and we can conclude that we must have $<' = maxPossible((I,G),P)$. $<' \subseteq <$ is true for some plan P because we have that P is consistent modulo $<'$ because of $possible((I,G),<' P)$ and P is consistent modulo $<$ because of (**). (q.e.d.)

3. This follows from (2.) as we have the following: if $necessary((I,G),<_G)$ is true, then this implies that $\forall P \in solutions((I,G)) : possible((I,G),<_G,P)$ (Definition 11/12). (q.e.d.)

Lemma 2: Properties of sets of goal orderings (see page 15)

Proof:

1.(a) We have to show that we cannot have necessary orderings which are supersets of the $maxNecessary((I,G))$. Let $<'$ be a necessary ordering for problem (I,G) and let us further assume that $<' \supset <= maxNecessary((I,G))$ holds.
Then there must exist a $(g_i <' g_j) \notin maxNecessary((I,G))$ (*). Because $<'$ is a necessary goal ordering we have $g_i <' g_j \Rightarrow establisher(g_i) \prec_O establisher(g_j)$ for all solution plans of (I,G) (Definition 8/10/11) and also $establisher(g_i) \prec_O establisher(g_j) \Rightarrow g_i < g_j$ (**), which means $(g_i <' g_j) \in maxNecessary((I,G))$. This is a contradiction with (*), and we can conclude that we must have $<' = maxNecessary((I,G))$. (q.e.d.)
1.(b) Let $<$ be an impossible ordering for problem (I,G). By Definition 13 this means $solutions_{CAPlan}((I,G),<) = \emptyset$, so we have $\forall P \in solutions((I,G)) : P$ is not consistent modulo $<$. With Definition 8 we can conclude that $\forall P : \exists establisher(g_j) \prec_O establisher(g_i)$ such that $(g_j < g_i) \notin maxPossible((I,G),P)$. This, means that $maxPossible((I,G),P) \subset <$ or that $g_i < g_j \in maxPossible((I,G),P)$. If we look back at Figure 5 on page 14 we find an example: $\{g_1 <_G g_2, g_3 <_G g_1\}$ is an impossible orderings because these orderings are not true in any solution, P or P'. The orderings are a superset of $maxPossible((I,G),P)$ and inconsistent with $maxPossible((I,G),P')$. (q.e.d.)

2.(a) If we imagine a domain and a problem with three goals g_1, g_2, g_3 which have to be solved in a linear order such that no two establishers are parallel, then $\{g_1 <_G g2\}$ or $\{g_2 <_G g3\}$

both define possible orderings.
2.(b) The same problem could have two different solutions with the same goal orderings if there is more that one possibility for establishing one of the goals without causing interactions with other steps. Figure 5 also shows an example for that: in both solutions P, P' we have the goal ordering $g_1 <_G g_2$.

3.(a) This is a special case subsumed by Definition 15. $maxNecessary((I,G))$ is the intersection of all maximal set of possible goal orderings, which always is a subset of the intersection of two such maximal sets of possible goal orderings.
3.(b) If we imagine a domain where a problem with goals g_1, g_2 has two solutions, one solves the g_1 first, the other solves g_2 first. In that case we have two totally different sets of maximal possible orderings for each solution and $maxNecessary((I,G)) = \emptyset$. Figure 5 is another example for that.

Lemma 3: Maximal sets of necessary of possible orderings are not enough to make a problem become $<_G$-constrained trivially serializable (see page 22).

Proof: We give an example that serves as a proof. Imagine we have a problem with two goals, $(I,G) = (\{+p_1, +p_2\}, \{+g_1, +g_2\})$, and the domain consists of the following operators: $\{+p_1\}\ Op_{11}\ \{+g_1, -g_2\}$, $\{+p_1\}\ Op_{12}\ \{+g_1, -g_2, -p_2\}$, $\{+p_2\}\ Op_2\ \{+g_2\}$.[22] There are two possible subplans that solve goal g_1: P_{11} consists of Op_{11}, P_{12} consists of Op_{12}. g_2 can only be achieved by Op_2.
Now we have the following situation: P_{11} can be serially extended to a plan $P = P_{11} + P_{delta}$ that solves both, g_1 and g_2. For the other plan, P_{12}, such a serial extension does not exist because Op_{12} produces two threats that cannot be resolved. So, the plan consisting of Op_{11} and Op_2 is the only solution to the problem and we have $maxNecessary((I,G)) = \{g_1 <_G g_2\} = maxPossible((I,G),P)$ if P is the solution plan. But the problem is not $<_G$-constrained trivially serializable w.r.t. this ordering because not all partial plans achieving g_1 are serially extensible to a solution plan for both goals. (q.e.d.)

Lemma 4: Even subsets of maximal sets of necessary orderings can be enough to make a problem become $<_G$-constrained trivially serializable (see page 22)

Proof: The artificial domain $\Theta_n D^m S^1$ (Barrett and Weld, 1994) is an example that proofs this lemma (see Appendix A.2 for details of the domain specification): here we have for a problem (I,G) of size n

$$maxNecessary((I,G)) = \{G_\alpha <_G G_i|_{i=1..n}, G_i <_G G_{i+1}|_{i=1..n-1}\}.$$

As there exists exactly one solution plan for each problem in this domain this is also the maximal set of possible orderings because $maxNecessary((I,G))$ already forces all establishers of all goals to be totally ordered. But we don't need the orderings $\{G_i <_G G_{i+1}|_{i=1..n-1}\}$ to show that the domain is $<_G$-constrained trivially serializable.
If we only have the orderings $<_G = \{G_\alpha <_G G_i|_{i=1..n}\} \subseteq maxNecessary((I,G))$ we must show that any $<_G$-consistent permutation is a serialization order. This is easy to see because of one special property of the $<_G$-consistent permutations: each such permutation π represents a serializable order and will have to start with $G_{\pi(1)} = G_\alpha$ which means that the planner will solve G_α using operator A_α first (see also (Barrett and Weld, 1994)). If it selects some other goal G_i, there will always be some plan consisting of A_i^β for that goal that cannot be serially extended to a plan for G_i and G_α. Let us now pick one arbitrary goal

[22] See Appendix A for some conventions about the notation of operator specifications.

$G_{\pi(2)} = G_k, k \in \{1, \cdots, n\}$. There are two possible operators in the $\Theta_n D^m S^1$ domain, A_k^α and A_k^β, but because the plan already contains A_α and the operator A_k^β will not be successful even if selected. So the planner will end up with A_k^α for goal G_k and this step will be ordered after A_α when all threats are resolved. The important thing is, that the refinements for solving G_α are not revised. If we now select another $G_{\pi(3)} = G_l \neq G_k$, the arguments are the same why it is ordered after A_α. The interaction between G_l and G_k can also be solved without revising the plan for G_α, G_k because these interactions are exactly the same as in the $D^m S^1$ domain which is known to be trivially serializable (Barrett and Weld, 1994). Finally, this can also be done for all other goals in a similar way and we always can extend the plan that has been found for $G_{\pi(1)}, \cdots, G_{\pi(m)}$ to a plan for $G_{\pi(1)}, \cdots, G_{\pi(m+1)}$. (q.e.d.)

B.2 Relation of Ordering Establishers and Goal Selection Strategy

Theorem 1: It is a property of the domain whether the same goal ordering can be used for the ordered establisher condition and the $<_G$-consistent goal selection strategy (see page 23)

Proof: In Section 3.3.7 we already explained four points which follow from the fact that possibly there is no relation between goal orderings that can be used for the ordered establisher condition and the $<_G$-consistent goal selection strategy.

We have to show point three and four and give an example for that situation. The artificial domain LinkRepeat (Veloso and Blythe, 1994) shows both points (see also Appendix A.5 for the detailed definition). Here a typical problem with goals $g_1, \cdots, g_n, g^*$ has a solution with a linear sequence of the actions $A_1, A^*, A_2, \cdots, A^*, A_n, A^*$. So, a goal ordering like $<_G = \{(g_i, g_{i+1})|_{\forall i}, (g_n, g^*)\}$ is obvious w.r.t. the ordered establisher condition. Empirical results however show that the performance decreases when this goal ordering is used for a $<_G$-consistent goal selection strategy. This effect can be explained as follows:

The domain LinkRepeat has been designed to show advantages of the Means-Ends-Analysis state-space planner PRODIGY over causal-link planners like SNLP. It consists of goals g_i which have the property that the solution of a g_i requires that the goals $g_j|_{j<i}$ are already solved because of the precondition g_{i-1} of each action A_i, i.e., $g_i \triangleright g_j|_{j<i}$ for all goals except g^*. This is the reason why we only have one possible linear order between all actions. The characteristic problem, however, is to find the right plan step achieving the precondition g^*: there are typically several ways for simple establishment, none of which will work, and exactly one with step addition. The MEA planner always does the right thing by focusing on goals that are not already true in the current state. A causal-link planner typically will prefer an increasing number of simple establishments before it finally selects step addition. Even with a $<_G$ as defined above, for which a typical problem is $<_G$-constrained trivially serializable the number of necessary inference steps to solve a g_i increases. The reason for that is that the precondition $g^*@A_i$ can be achieved by $i-1$ possible simple establishments or by step addition, altogether i alternatives and preferring simple establishment is the worst operator selection strategy here. For n goals we have $n(n-1)/2$ alternatives to establish the goals $g^*@A_i$. The result is that the goal ordering $<_G$ as defined above makes a problem $<_G$-constrained trivially serializable but the number of inference steps to extend the subplan for the first k goal to a solution for the next goal constantly increases.
On the other hand, if we look at the inverse ordering $\{g_{i+1} <_G g_i\}$, which is an impossible goal ordering (i.e., it cannot be used for planning with the ordered establisher condition), we do not have an ordering that makes the problem $<_G$-constrained trivially serializable but the number of inference steps does not increase with each goal. We always have only one simple establishment and one step addition, i.e., for n goals $2*n$ alternatives. This leads to

the situation that the impossible ordering allows to find the solution in less inference steps (and probably less time) than the necessary goal orderings.

B.3 Search Space

Theorem 2: Reduction of the size of the search space by goal orderings (see page 28)

Proof: We have to look in detail at the search spaces of SNLP and CAPLAN for a problem (I, G). The interesting points are the differences that are induced by the replacement of normal step addition and simple establishment by the $<_G$-consistent variants.

(a) The difference of the sizes of the search spaces can be proved by an examination of the consequences of a goal ordering. We will show that some nodes of the search space of SNLP cannot be part of the search space of CAPLAN.
Because $<_G \neq \emptyset$, there exist two ordered goals $g_i, g_j \in G$ with $g_i <_G g_j$. The ordered establisher condition implemented by the $<_G$-consistent plan refinements operators (Section 4.1.1) requires that the establishers of both goal are ordered. Because any solution must be complete we can identify $s_i \rightarrow g_i@s_\infty$ and $s_j \rightarrow g_j@s_\infty$ which declare s_i, s_j to be the establishers of g_i, g_j. The solution plan must further be consistent modulo $<_G$ in CAPLAN (Definition 10), so, $s_i \prec_O s_j$ must hold in every solution.
If we compare both search spaces we can identify nodes (plan states) in which we find the plan steps s_i, s_j and there are three possible ordering relations between both plan steps: (i) s_i and s_j are parallel, (ii) $s_i \prec_O s_j$, or (iii) $s_j \prec_O s_i$. The search space represents the worst case, so each of these three ordering might occur in at least one node although we know that $s_i \prec_O s_j$ must hold in all solutions. Let us look at these cases: s_i and s_j are parallel in a plan state if the final ordering is created by a threat resolution but that threat is not resolved in that plan state, or if s_i is added within the establishment of the preconditions of s_j and g_i is achieved by simple establishment later. $s_j \prec_O s_i$ cannot be refined into a solution because the ordering of the steps is incorrect.
In the search space of CAPLAN the $<_G$-consistent plan refinements definitely never will create a plan in which $s_i \prec_O s_j$ is not true if the refinement addresses the goals g_i, g_j. Refinements because of other goals or threats still may create other orderings like in the search space of SNLP. There must be at least one state which results from the application of a refinement addressing g_i or g_j. This is enough to conclude that the search space of CAPLAN consists of less nodes than the search space of SNLP. (q.e.d.)

(b) The path from the initial plan to a solution represents the sequence of plan refinement operators necessary to find the solution. We have to show that the shortest path to a certain solution P in the search space of SNLP can be mapped to a corresponding path in the search space of CAPLAN which has at most the same length.
The proof of (a) has already shown that there exist nodes in the search space of SNLP that do not exist in the search space of CAPLAN. Let us look at the possible differences between solution paths in both search spaces. We can identify three refinements that must be on every path to a solution for SNLP if goals g_i, g_j are to be achieved as in (a): the refinements adding s_i and s_j and either a threat resolution or a simple establishment adding $s_i \prec_O s_j$. In comparison, CAPLAN will never need an explicit third refinement adding the ordering between both steps like the threat resolution because the $<_G$-consistent refinement operator always will add this ordering if the other establisher exists. This makes clear that we have two or three plan refinements in SNLP for the goals but never more than two in CAPLAN. All other refinements must be the same as in SNLP because only ordered top-level goals are treated differently by CAPLAN. (q.e.d.)