In: Beiträge zum 11. Workshop *Planen und Konfigurieren (PuK-97)*

Advantages of Types in Partial-Order Planning

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

Abstract

This paper shows an approach to profit from type information about planning objects in a partial-order planner. The approach turns out to combine representational and computational advantages. On the one hand, *type hierarchies* allow better structuring of domain specifications. On the other hand, operators contain *type constraints* which reduce the search space of the planner as they partially achieve the functionality of filter conditions.

1 Motivation

Planning with STRIPS-like [5] domain representation in most cases ignores the aspect of type information that becomes more interesting if we deal with domains containing a richer variety of elements than blocksworld or transportation domain. One remarkable feature of such domains is the high number of different kinds of goals and operators. Naturally, the question of how to find right operator to be applied to a goal becomes more crucial.

Although most planners are not able to represent it explicitly, objects involved in a planning problem always have a certain type and there are relations between the different types. However, for domains like blocksworld it seems at least unclear what advantages an explicit representation of type information could bring – only a few types can be identified and there are no or only few relations between the types in these domains.

In domains containing a richer variety of elements we have a different situation. An example is the domain of process planning for rotary symmetrical workpieces to be machined on a lathe [9]. Although this domain concentrates on a certain restricted spectrum of workpieces, typical problem specifications deal with many objects (often more than 30) of different types and a type-subtype (is-a) relation can be identified between the object types.

For a workpiece like the one in figure 1 the symbolic representation must describe all areas that have to be removed (cutted) when starting with a block of raw material. There are different types of so-called *processing areas* such as outlines, sides, undercuts, or features like drilled holes or threads. Additionally, there are different kinds of cutting, drilling and other machining tools that have to be represented, e.g., LeftRotaryTool, DrillTool, ..., and they build a hierarchy of tools. The planners task is to choose the tools from the available ones and to find a sequence of tool insertions, tool changing, clamping, and cutting operations that manufacture the workpiece. While it is obvious that structuring types in hierarchies

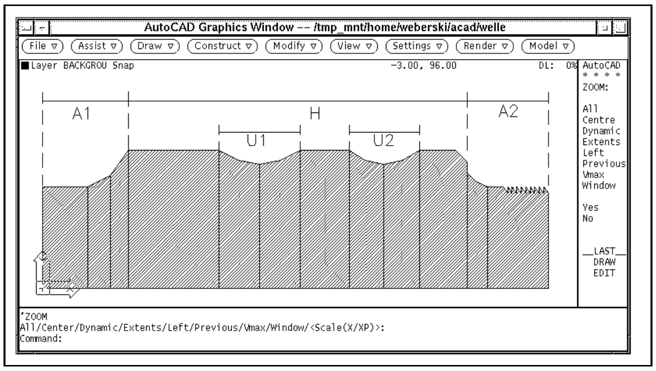

Figure 1: Example for a workpiece

```
ProcessingArea
  Outline
    Center
    ...
  WPSide
  ...
  Feature
    Hole
    Thread
    Slope
    ...

Tool
  LeftRotaryTool
  RightRotaryTool
  ...
  FeatureTool
    DrillTool
    TappingTool
    ...
```

Figure 2: Type hierarchies

(figure 2) has representational advantages, the question is, whether we can profit from the information about types and type hierachies in a partial-order planner based on SNLP.

The following pages describe how the partial-order planner CAPlan extends SNLP to represent and use type information (section 2). In section 3 we will show and discuss that type constraints are an elegant way of solving (at least partially) the problem of having filter conditions in SNLP (also addressed in [3]), i.e., we will show that the type constraints of CAPlan can be understood as filter (applicability) conditions of operators. Section 4 summarizes some empirical results and section 5 concludes with a discussion of the approach.

2 The Use of Types in CAPlan

CAPlan [13] is an SNLP-like [7, 1] domain-independent, partial-order planner. Plans are represented as 4-tupes (S, O, B, L), where S is set of plan steps, O is a set of ordering constraints that define a partial order over S, B contains binding constraints over variables occurring in the plan steps, and L contains causal links that capture causal dependencies between plan steps. Planning proceeds by establishing open preconditions of plan steps and by solving conflicts (called threats).

CAPlan extends SNLP [7] in some ways. In the context of this paper we focus on the extended domain specification mechanisms.[1] CAPlan allows an explicit representation of object types and the type-subtype relation in form of *type hierarchies*. Figure 2 shows an example for parts of type hierarchies from our specification of the process planning domain. Types are referred to in two places:

- **Problem specifications** include a type declaration for all objects involved in a planning problem, e.g., `Hole(h)` declares `h` to be of type `Hole`. This type information is invariant during a problem solving episode.
- **Operator specifications** can also contain type information in form of type constraints. Besides preconditions and effects, the STRIPS-like operator representation in CAPlan not only allows codesignation/noncodesignation constraints [2] but also type constraints of the form `IsOfType(<TYPE>,<var>)` or `IsNotOfType(<TYPE>, <var>)`.[2] They force the binding of a variable `<var>` to be (or not to be) of a

[1]Other aspects, e.g., the extended problem specifications, are described in [13, 8].

[2]STRIPS [5] did not have constraints at all, but most planners with STRIPS-like operator representation added variables and simple codesignation/noncodesignation constraints [2] to the operators.

```
drill-hole(h,dt)
   Constraints:  IsOfType(Hole,h)
                 IsOfType(DrillTool,dt)
       Purpose:  +processed(h)
  Side Effects:  −unprocessed(h)
 Preconditions:  +available(dt)
                 +toolHeld(dt)
```

Figure 3: Operator definition

```
Problem:
 Objects:  Hole(h)
           DrillTool(dt)
 Initial:  available(dt)
           unprocessed(h)
   Goals:  +processed(h)
```

Figure 4: Problem definition

certain type <TYPE> or a specialization of <TYPE>.

CAPlan works like SNLP: if it establishes an open goal using a domain operator *op*, i.e., adding a new plan step to the plan (S, O, B, L), the constraints of *op* are added to B. But *op* is chosen only if its contraints are consistent with B. So, in CAPlan it can happen that *op* is not applied because of inconsistent type constraints. The fact that CAPlan may determine the non-applicability of an operator with type constraints is exactly the point where CAPlan has an advantage over other SNLP-like planners.[3]

Example: Figure 3 and figure 4 show an operator from the domain of process planning and a part of a problem specification. The goal of the problem is to process a hole and there is a certain drilling tool available. The domain contains several operators that achieve `+processed(h)` for a hole `h`, e.g., `drill-hole`, `tap-hole`, ... The operator `drill-hole` can be applied as its type constraints are consistent, i.e., there exists a consistent possibility to bind the variable *dt* of this operator. For all other operators this consistency check fails because there is no possible binding consistent with the type constraints of the operator. So, all other operators are considered to be inapplicable here.

3 Advantages of Types

After the last section explained how CAPlan extends SNLP to allow having type information in domain and problem specifications, we now will explain the advantage of this representation. If types are not available to represent domain knowledge, additional predicates can be used to simulate type information. Let us compare these two representations for a partial-order planner with STRIPS-like operator representation:

(A) Representation with type constraints: As explained in section 2, CAPlan extends the kinds of constraints by allowing types constraints (e.g., figure 3). Type constraints restrict the application of the operator by restricting the possible bindings of the variables to objects of certain types.

(B) Representation with preconditions: Information about object types can also be encoded in predicates that are added to operators as preconditions. These predicates should be unachievable[4] as they represent static information about objects that cannot be changed by operators, but they can only be true in the initial situation. For representing the operator in figure 3 without type constraints we must replace the two type constraints with two new preconditions `IsHole(h)` and `IsDrillTool(dt)`. An appropriate problem

[3] As type information of objects is invariant during a planning episode, inconsistent type constaints of an operator can never become consistent later. So, type constraints do not affect the completeness.

[4] A predicate is unachievable if it does not appear as an effect of any operator.

description needs these two predicates in its initial situation.

Comparision and consequences. In both cases, what we actually have done is to add some kind of *filter condition* to an operator. Filter conditions are known from HTN-planners like NONLIN/O-Plan [12, 4] or SIPE [14] and are originally thought to help the planner in finding applicable operators during the planning process (see also [6]). Operators with filter conditions that are not satisfied are not considered to be applicable to a goal, so the operator selection criterion is more restrictive than a simple one that only check for a matching effect. Of course, it might be desirable to have filter conditions in a partial-order planner like SNLP. However, [3] pointed out the problems of filter conditions in an SNLP planner and concluded that it is impossible to add those filter conditions (as preconditions of operators) without affecting the completeness of the algorithm.

The two representations described above basically have the effect, that the planner will discard the operator if a filter condition is not satisfied (if represented as a constraint) or cannot be established (if represented as a precondition). But there is an important difference: In representation (A) the failure is detected *before* the operator is actually applied because the operators constraints are inconsistent, whereas in (B) the operator is applied but later the planner backtracks as it cannot establish the precondition. Unfortunately, the application of an operator (i.e., the addition of a new step) might cause new threats and these threats might be selected first by the planner (with SNLP standard strategy, or others, e.g., DSep [11]). So, in (B) the search space will be bigger because only in the best case the planner will select the precondition representing the filter condition immediately after the precondition was added and fail to establish it, thus backtrack over the application of this wrong operator. In all other cases, the planner selects another goal or threat first and adds superfluous backtracking points.

From this considerations we can expect that a planner using type constraints will have to do less backtracking and expand a smaller part of the possible search space. On the other hand, type constraints need to be checked and this will cause some overhead.

4 Empirical Results

We conducted two experiments to investigate the effects of a representation of filter conditions as types constraints and compared them with the representation as preconditions.

Artificial domain: We defined two artificial domains that are oriented at the D^1S^2 and D^mS^2 domains [1].[5] We modified these domains in the following way: we replaced the 0-ary goals G_i by one unary goal $G(x)$. In a *typed version* of the domains we defined types T_i that appear in type constraints of operators (figure 5). A typical problem specifies objects $o_1, o_2, ...$ of the types $T_1, T_2, ...$ and goals $G(o_1), G(o_2), ...$ A *flat version* of each domain (without types) represents the filter conditions as preconditions (figure 6). In both versions a goal $G(o_i)$ must be achieved using the sequence of operators A_i^1 followed by A_i^2. The crutial point for the planner is the selection of the right A_i^2 for a goal $G(o_i)$.

In the experiment we defined five types and 20 problems with one to five goals for objects of different types. To complicate problem solving, we let the planner select goals and operators randomly and we made 10 iterations with each problem. Table 1 shows the average results for all iterations of all problems. It shows that the average problem solving time and the number of inference and backtracking steps for the typed domains are significantly better

[5] The domains originally show advantages of plan-space planners over state-space planners.

Domain: *Typed-*D^mS^2
Types: $T_{1\cdots n}$; Predicates: I, P_i, $G(x)$
Operators:
- A_i^1 constraints: —
  precond: I, effects: $+P_i$
- A_i^2 constraints: $IsOfType(T_i, x)$,
  precond: P_i, effects: $+G(x), -P_{j|j<i}$

Figure 5: Artifical Domain *Typed-*D^mS^2

Domain: *Flat-*D^mS^2
Types: —; Predicates: I, P_i, $G(x)$, $IsT_i(x)$
Operators:
- A_i^1 (like in figure 5)
- A_i^2 constraints: —
  precond: $IsT_i(x)$, P_i, effects: $+G(x), -P_{j|j<i}$

Figure 6: Artifical Domain *Flat-*D^mS^2

than for the flat domains. Also, for typed domains we have a much higher percentage of correct inference steps (inference steps that are not revised later) and of correctly expanded nodes in the search space. We also see that the planner needs more time per inference step in the typed domains. This is due to the overhead caused by checking type constraints.

Domain of process planning: The domain of process planning specified for CAPlan also makes use of filter conditions as type constraints. We tested the original version (*PP-Typed*) against a version that represented some of the filter conditions as preconditions (*PP-Flat*) similar as explained for the first experiment.[6] The right side of table 1 shows the average results for a collection of 10 representative problems. Here we used the control strategy DSep [11] as for random strategies or SNLP strategy no solution can be expected within a reasonable amount of time. The results show again that the planner does better with the typed version of the domain.

Averages of	*Flat-*D^1S^2/D^mS^2	*Typed-*D^1S^2/D^mS^2	*PP-Flat*	*PP-Typed*
Problem Solving Time	2111/2556 ms	465/679 ms	12594 ms	10538 ms
Inference steps	112/132	21/33	147	115
Backtracking steps	45/54	4/9	37	23
Time per Inf. step	19.1/19.3 ms	25.2/24.5 ms	79.1 ms	83.7 ms
Correct Inf. steps	23.1/20.3 %	75.8/55.7 %	54.8 %	63.6 %
Correct search space	42.0/38.7 %	92.5/86.0 %	62.7 %	73.6 %

Table 1: Evaluation results for artificial domains and domain of process planning

5 Conclusions

We presented an approach for achieving the functionality of filter conditions in a partial-order planner using information about object types. In our approach, operators can have type constraints that force variable bindings to be of certain types or their subtypes. We explained that these type constraints can be understood as filter conditions of the operators as they are checked before the operator is actually applied.

[3] already investigated the problem of having filter conditions in SNLP. They presented an approach that is based on the concept of secondary preconditions [10] which results in operators with context-dependent effects. The difference of secondary preconditions to normal preconditions is that they do not determine the applicability of an operator but whether an operator has a certain effect of not. To establish both types of preconditions additional plan steps might be added. Filter conditions, however, are thought to judge

[6]The domain consists of 38 types. But we did not replace all of them by additional predicates. Nevertheless, the effect of this partial replacement is quite clear.

the applicability of an operator without ever adding plan steps for that purpose. In [3] a set of operators for which applicability would be decided by filter conditions is replaced by a single operator with conditional effects where these secondary preconditions are the filter conditions of the single operators. In this way, [3] shift the problem of determining the operator to be applied to the problem of determining the effects of this one operator in a certain situation. As they show this is an improvement, but still the secondary preconditions must be achieved by the planner in some way.

Although our approach using types and type constraints can only represent filter conditions that are equivalent to unachievable preconditions (but not arbitrary predicates), we predetermine the applicability of an operator. This is closer to the original idea of filter conditions. Additionally, it shows a way to combine *representational advantages* (type hierachies in domain specifications) and *computational advantages* as type constraints lead to an improvement of planning performance in CAPlan.

References

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

[2] D. Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32:333–377, 1987.

[3] G. Collins and L. Pryor. Achieving the functionality of filter conditions in a partial order planner. In *Proceedings of AAAI-92*, pages 375–380, 1992.

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

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

[6] S. Kambhampati. A comparative analysis of partial order planning and task reduction planning. *SIGART Bulletin*, 6(1), 1995.

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

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

[9] H. Muñoz-Avila and F. Weberskirch. 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, 1996.

[10] E. Pednault. Synthesizing plans that contain actions with context-dependent effects. *Computational Intelligence*, 4:356–372, 1988.

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

[12] A. Tate. Generating project networks. In *Proceedings of IJCAI-77*, pages 888–893, 1977.

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

[14] D.E. Wilkins. *Practical Planning - Extending the classical AI Planning Paradigm*. Morgan Kaufmann, 1988.