Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

Research
Report
RR-90-09

Plan Generation Using a Method of Deductive Program Synthesis

Susanne Biundo

July 1990

Deutsches Forschungszentrum für Künstliche Intelligenz
GmbH

Postfach 20 80
D-6750 Kaiserslautern, FRG
Tel.: (+49 631) 205-3211/13
Fax: (+49 631) 205-3210

Stuhlsatzenhausweg 3
D-6600 Saarbrücken 11, FRG
Tel.: (+49 681) 302-5252
Fax: (+49 681) 302-5341

Plan Generation Using a Method of Deductive Program Synthesis

Susanne Biundo

DFKI-RR-90-09

This report is the extended version of a paper presented at the 4th Workshop "Planen und Konfigurieren" held at the FAW Ulm in April 1990.

PLAN GENERATION USING A METHOD OF DEDUCTIVE PROGRAM SYNTHESIS

Susanne Biundo
Deutsches Forschungszentrum für Künstliche Intelligenz
Stuhlsatzenhausweg 3
D-6600 Saarbrücken 11

ABSTRACT

In this paper we introduce a planning approach based on a method of deductive program synthesis.

The program synthesis system we rely upon takes first order specifications and from these derives recursive programs automatically. It uses a set of transformation rules whose applications are guided by an overall strategy. Additionally several heuristics are involved which considerably reduce the search space.

We show by means of an example taken from the blocks world how even recursive *plans* can be obtained with this method. Some modifications of the synthesis strategy and heuristics are discussed, which are necessary to obtain a powerful and automatic planning system. Finally it is shown how subplans can be introduced and generated separately.

TABLE OF CONTENTS

1. INTRODUCTION

In the field of *deductive program synthesis* several methods have been developed to derive algorithmic definitions of functions from formal specifications (cf. e.g. [Manna/ Waldinger 80], [Bibel 80], [Goad 80], [Hsiang 83], and [Franova 88]).
Specifications are formal descriptions of functions given in a specific high level programming or specification language. They specify the input and output variables and a relation the output is intended to satisfy. Additionally an input condition expresses the class of legal inputs to which the program is expected to apply. A specification may be completely *non-constructive* as, for example:

input	x,y		
find	z	*such that*	R(x,y,z)
		where	P(x,y) .

It expresses the purpose of the desired program, without giving any hint towards an "implementation".

Transformations rules (cf. [Manna/Waldinger 79] and [Manna/Waldinger 80]) are used to derive a program from such a specification. The program computes the specified function and is written in a simple target programming language. The transformation rules represent knowledge about the program's subject domain and their application is guided by several strategies and heuristics [Manna/Waldinger 79].

A specification however, like the one given above, can be read as a *formal plan specification* as well: P(x,y) and R(x,y,z) describe the initial state and goal state respectively. *z* is the plan that is searched for and which transforms the initial state into the desired goal state.

A program or plan specification like the one above can be expressed in terms of a first order formula: $\forall x,y\ [P(x,y) \rightarrow \exists z\ R(x,y,z)]$.

Thus the program synthesis system DEPSY (cf. [Biundo 90]) which we will introduce as a plan generation tool in this paper uses first-order predicate logic as a specification language. Specifications are existence formulas of the form $\psi = \forall x^*\ \exists y\ \varphi[x^*\ y]$ (where x^* is a list of variables and φ is a first-order formula without any quantifier). Yet not only the specification language but the target programming language as well is first-order predicate logic: The program (plan) which is derived from such a specification consists of a set of conditional equations, called *definition formulas.*

Starting from a specification formula several transformation rules are applied in turn. These rule applications are driven by an overall strategy which has been developed in order to keep the search space small and to obtain goal directed transformations. Additionally several heuristics are involved selecting the appropriate rules.

The close connection between deductive program synthesis and deductive planning has been posed by other authors as well. Wolfgang Bibel, for example, states in [Bibel 86]:

> *"...the plan generation problem appears to be very much of the same nature except that the actions to be carried out by the plan (resp. program) are actions in the environment rather than in the computer."*

He therefore proposes a deductive solution to planning which works in a way similar to deductive program synthesis (cf. [Bibel 86] and [Bibel et al. 89]). States are characterized by logical formulas. They describe the properties holding in a specific state and include a special state literal. As in the case of program synthesis a specification theorem is proved. As soon as the proof is complete the desired plan occurs as an argument of the state literal in the goal state's description and can be extracted from it.

Starting from their work on deductive program synthesis, where program derivation is regarded as a theorem proving task (cf. [Manna/Waldinger 80]), Manna and Waldinger have adapted their approach to the solution of planning problems as well (cf. [Manna/Waldinger 86]). They introduce a new version of situational logic based on the works of McCarthy and Green (cf. [McCarthy 68] and [Green 69]). The deductive mechanism used to generate plans by proving specification theorems is the *deductive tableau system* initially being developed to solve the program synthesis problem based on resolution. The deduction rules working on that tableau produce both *conditional* and *recursive* plans. The latter are obtained applying an appropriate *induction rule*.

The paper is organized in the following way: In Section 2 we give a short overview of the synthesis method underlying DEPSY. Section 3 outlines the overall synthesis strategy and some of the heuristics involved. By means of an example taken from the blocks world it is demonstrated in Section 4 how the synthesis system can be used to generate plans and which modifications have to be made to obtain an automatic planning system. One of these modifications refers to the introduction of subplans and is pointed out in Section 5. Finally we conclude with some remarks in Section 6.

2. A METHOD OF DEDUCTIVE PROGRAM SYNTHESIS

The logical basis of our system is built by a many-sorted first-order language together with a set of transformation rules. The system maintains a data base Ax consisting of first-order axioms which describe the domain currently under consideration. Starting from a specification formula transformation rules are applied in turn which take a formula (*goal*) and, using the axioms of the data base, produce a set of formulas (*subgoals*). This process continues until a definition of the specified function has been obtained.

In the case of *program synthesis* the data base consists of:

- Axioms characterizing the various datastructures under consideration (as far as it is possible within first-order logic), e.g. natural numbers, lists of natural numbers, trees, etc.

- Axioms representing the programs which operate on these datastructures, e.g. addition of natural numbers, concatenation of lists, flatten of trees, etc. These axioms are conditional equations, i.e. definition formulas, which satisfy certain conditions. The programs they represent are defined by means of complete case analyses, recursion, and functional composition.
- Axioms describing properties of these programs, e.g. commutativity, associativity, idempotence, etc.

To generate *plans* with our synthesis method axioms analogously are required to describe the planning domain. In this paper we choose the well known blocks world as our planning domain. The first-order representation we use is similar to the one proposed by Kowalski in [Kowalski 79].
There the two-place predicate "HOLDS" is introduced which takes a *property* as its first argument. The second argument denotes a *state* where this property is demanded to hold. In contrast to the approach of Kowalski, where only Horn formulas are admitted, we will use full predicate logic instead.
At first we need
- axioms to describe states.
  They may look like, for example:

  $\forall$x,y:block $\forall$s:state [H(on(x y) s) $\rightarrow \neg$H(clear(y) s)]

  $\forall$x:block $\forall$s:state [$\neg$H(on(x x) s)] .

- Axioms characterizing basic operators

describe the preconditions as well as the consequences of an operator.
One of the axioms characterizing the *put*-operator, for example, reads:

$\forall$x,y:block $\forall$s:state [H(clear(x) s) $\wedge$ H(clear(y) s) $\wedge$ $\neg$x=y $\rightarrow$ H(on(x y) s!put(x y))] .

Finally
- axioms describing *properties* of basic operators are required.

The *frame axioms* play a main role in this context. Fortunately the formal framework we rely upon provides an elegant solution to the formulation of frame axioms (cf. [Kowalski 79]): Introducing the HOLDS-predicate entails the possibility of quantifying over "properties". Thus for each operator only one single frame axiom is required. For the above *put*-operator we have the following frame axiom:

$\forall$x,y:block $\forall$s:state $\forall$m:property [H(m s) $\wedge$ $\neg$m=clear(y) $\rightarrow$ H(m s!put(x y))] .

Starting from a specification of the form $\psi = \forall x^* \exists y\ \varphi[x^*\ y]$ various transformation rules are applied in order to derive the set of definition formulas representing the specified program or plan.
Applying a transformation rule R to a formula φ produces a set of formulas Φ and is denoted $\varphi \Rightarrow_R \Phi$.

For a set of formulas Ψ we define

$\Psi \cup \{\varphi\} \equiv>_R \Psi \cup \Phi$ iff $\varphi \Rightarrow_R \Phi$.

All of the transformation rules we use are *sound*. That means, for each φ, Φ and R we have that: $\varphi \Rightarrow_R \Phi$ implies $Ax \models_s \bigwedge\Phi \rightarrow \varphi$

(where $\bigwedge\Phi$ denotes the conjunction $\bigwedge_{\delta\in\Phi}\delta$ and $Ax \models_s \psi$ denotes $M \models \psi$ for each standard modell M of Ax).

Among others the following sound transformation rules are provided.

Evaluation Rule

(Replacement of a term by symbolic evaluation)

Let π denote a position and

$D = [D' \rightarrow q\equiv r]$, $D \in Ax$, $\sigma q = t$ and $\sigma D' \subset C$.

$[\, C \rightarrow L[\pi,t]\,] \Rightarrow_{EV,\, D} \{ [\, C \rightarrow L[\pi,\sigma r]\,] \}$.

Substitution Rule

(Replacement of a term at a certain position in a clause by an equal term)

$[\, C \wedge q\equiv r \rightarrow L[\pi,q]\,] \Rightarrow_{SUB} \{[\, C \wedge q\equiv r \rightarrow L[\pi,r]\,]\}$

Implication Rule

(Replacement of a literal σL by literals σK_i which are sufficient for σL)

Let $D = [K_1 \wedge ... \wedge K_n \rightarrow L]$, $D \in Ax$.

$[C \rightarrow \sigma L] \Rightarrow_{IMPL,D} \{ [C \rightarrow \sigma K_1] , ... , [C \rightarrow \sigma K_n] \}$.

Extraction Rule for Equalities

(Merging two equalities)

$[\, C \wedge q\equiv r \rightarrow t\equiv r\,] \Rightarrow_{EX.E} \{[\, C \rightarrow q\equiv t\,]\}$

Function Extraction Rule

(Isolating subterms)

Let $t_i = q_i$ for $i \in \{1,...,n\}\backslash\{k\}$.

$[C \rightarrow f(t_1,...,t_n) \equiv f(q_1,...,q_n)] \Rightarrow_{EX.F} \{ [C \rightarrow t_k \equiv q_k] \}$.

Case Analysis Rule

(Introducing additional assumptions)

$[D \rightarrow C] \Rightarrow_{CA} \{ [D \wedge L \rightarrow C] , [D \wedge \neg L \rightarrow C] \}$.

Elimination Rule

(Eliminating a literal from a clause)

$[D \wedge L \rightarrow C] \Rightarrow_{EL} \{ [D \rightarrow C] \}$.

The system proceeds in the following way:
Starting from an existence formula $\psi = \forall x^* \exists y\ \varphi[x^*\ y]$ first of all a skolemization step is carried out. The existentially quantified variable y in φ is replaced by a term $f(x^*)$, where f is a "new" function symbol. The resulting formula $\psi_0 = \forall x^*\ \varphi[x^*\ f(x^*)]$ is viewed as a specification of the unknown skolem function f. Various transformation rules are applied in turn until the process halts with a set Ψ of formulas:
$\{\psi_0\} \Rrightarrow_{R1} \dots \Rrightarrow_{Rn} \Psi$.
$\Psi = DEF_f \cup REM_f$ is syntactically divided into a set DEF_f of definition formulas for the skolem function f (they constitute the synthesized program resp. plan) and a set of so-called *remainder formulas* REM_f.

The remainder formulas can be viewed as verification conditions that, if holding, guarantee that the program DEF_f meets its specification ψ. In the case of plan synthesis these formulas are understood as additional assertions which have to be proved in order to assure that the generated plan (which is represented by the definition formulas) indeed will lead to the desired goal state.

3. STRATEGIES AND HEURISTICS

When synthesizing programs by our method the data base provides a set of axioms $Ax = Ax_{DS} \cup Ax_{Prog}$, characterizing datastructures as well as programs which operate on these datastructures. If the axiom set, for example, contains formulas representing the domain of natural numbers *nat*, and formulas describing a program *diff* computing the difference function on natural numbers a specification (of the addition function) may be the following:
$\forall x,y{:}nat\ \exists z{:}nat\ \ diff(z\ y)\equiv x$.
After skolemizing this specification we obtain: $\psi_0 = \forall x,y{:}nat\ \ diff(f(x\ y)\ y)\equiv x$.
From this formula a set of conditional equations has to be derived which constitutes a constructive definition of f. This new definition, like the user-defined ones already stored in the data base, has to be formulated by means of a complete case analysis, recursion, and functional composition.

Thus given the specification formula ψ_0 the system has to search for a complete case analysis as well as for an appropriate recursion scheme. Finally a set of conditional equations has to be derived which look like $C \rightarrow f(x\ y)\equiv t$. To achieve all these goals *automatically* the system makes use of a strong overall strategy which causes the transformation process to be carried out in a goal directed way. This strategy has turned out to be very successful and has been evaluated on a lot of examples (cf. [Biundo 90]). It is supported by several heuristics which in addition considerably reduce the search space. The strategy consists of four distinct phases. They read:

- Induction and Normalization
- Evaluation
- Extraction and
- Elimination.

In each of these phases a special task has to be performed and therefore only an appropriate subset of the transformation rules has to be applied. The transformation rules actually used in each phase are selected by heuristics corresponding to the task which has to be solved.

Since the program resp. plan to be generated has to be defined by a complete case analysis and by recursion the first phase of the transformation process aims at finding these. The close relation between induction and recursion (cf. [Boyer/Moore 79] and [Manna/Waldinger 80]) suggests to generate a recursion scheme using an appropriate *induction argument*. Consequently the first step in a synthesis process is to find an induction axiom which is adequate for the specification formula. This is done by the *induction rule*. Applied to the specification formula it produces a set of *induction formulas* (consisting of the so-called *base case* as well as the *induction case* formulas) and thus provides a complete case analysis for the definition to be synthesized. Simultaneously a recursion scheme is found for the skolem function, since one of the skolem terms occurring in the *induction hypothesis* will be used in the sequel as the recursive call for the new program resp. plan.

The heuristics playing a main role in this context are those concerning the selection of induction variables. Like in the induction theorem proving system of Boyer and Moore (cf. [Boyer/Moore 79]) or in the INKA system (cf. [Biundo et al. 86] and [SFB 90]) this is done by inspecting the definitions of functions which occur in the specification formula.

The second phase is dedicated to symbolic evaluation. In each induction formula some of the terms occurring in the induction conclusion have to be evaluated. In the case of program synthesis this is done using the evaluation rule together with appropriate axioms of the data base.

The heuristics involved, for example, suggest additional case analyses to enable new promising evaluation steps. As far as base case formulas are concerned definition formulas often are obtained as soon as the evaluation process has finished. For the induction case, symbolic evaluation is an essential tool to achieve a match between the induction hypothesis and the induction conclusion. Such a match has to be completed during the following extraction phase.

The extraction process serves to produce the defining equations. Thus the goal is to reach a situation where both the induction hypothesis and the induction conclusion differ only in positions where skolem terms occur. Applying the extraction rule then yields a defining equation where the skolem term is expressed in terms of one of the skolem terms initially contained in the induction hypothesis.

The heuristics developed for this phase drive the selection of rule applications suitable to enable applications of the extraction rules. If an extraction rule fails a so-called *conflict* is generated representing the reason of failure. Subsequently all applications of transformation rules are computed which can solve the conflict. The transformations are ranked and the best solution is tried first.

Based on the results of the extraction loop the elimination process finally provides a definition formula for each of the recursion cases. Special conditions which would have been introduced during previous phases and the induction hypotheses are removed from the formulas in order to obtain a final set of definition formulas.

4. GENERATING PLANS

As has been pointed out in Section 1 there is a close connection between the deductive synthesis of programs and the deductive generation of plans.

Subsequently we therefore will demonstrate by means of an example borrowed from [Manna/Waldinger 86] that the program synthesis system DEPSY we introduced above in principle can be used as a deductive plan formation system. It seems to be an important point to stress that even recursive plans can be derived with it quite naturally.

Additionally it has turned out that our solution to the example problem avoids the problems occurring in the solution Manna and Waldinger give in [Manna/Waldinger 86]. It therefore seems to be quite promising to obtain an automatic planning system by suitably modifying the DEPSY synthesis strategy. We will return to this point at the end of this section.

The example is chosen from the blocks world domain and shows the generation of a plan for clearing a block.
The data base consists of a set of axioms $Ax = Ax_S \cup Ax_{Op} \cup Ax_F$ which contains descriptions of states together with operator and frame axioms. Among these axioms we assume having a description of the *puttable*-operator (it puts a block onto the table) as well as a frame axiom characterizing the empty operation:

Ax1: $\forall$u,v:block $\forall$t:state [H(on(u v) t) $\wedge$ H(clear(u) t) $\rightarrow$ H(clear(v) t!puttable(u))]
Ax2: $\forall$t:state $\forall$m:property [H(m t) $\rightarrow$ H(m t!emptyop)]
Ax3: $\forall$u,v:block $\forall$t:state $\forall$m:property [H(m t) $\wedge$ $\neg$m=on(u v) $\rightarrow$ H(m t!puttable(u))] .

We start from the following specification of the goal state:

$$\psi = \forall x{:}block\ \ \forall s{:}state\ \ \exists z{:}plan\ \ H(clear(x)\ s!z)$$

It demands that there exists a plan *z*, which, when applied in state *s* results in a state *s!z* where the block *x* is clear.
Skolemization of the existence formula ψ yields:

$$\psi_0 = \forall x{:}\text{block}\ \ \forall s{:}\text{state}\ \ H(\text{clear}(x)\ s!p(x\ s))\ .$$

From this modified specification formula ψ_0 a plan *p* has to be derived which is *sound* in the sense that it meets its original specification. To achieve such a plan we will for the present follow the DEPSY program synthesis strategy. Within the four transformation phases rules are applied starting from the initial goal ψ_0. The subgoals produced again are transformed until they can be proved valid or they contribute to the definition of the specified plan in terms of defining equations.

At first the induction rule is applied to ψ_0. It provides a complete case analysis for the definition of *p* and produces two subgoals: $\psi_0 \Rightarrow_{\text{IND}} \{\psi_1, \psi_2\}$, where

$\psi_1 = [H(\text{clear}(x)\ s) \rightarrow H(\text{clear}(x)\ s!p(x\ s))]$ and

$\psi_2 = [\neg H(\text{clear}(x)\ s) \wedge H(\text{clear}(\text{hat}(x))\ s!p(\text{hat}(x)s)) \rightarrow H(\text{clear}(x)\ s!p(x\ s))]$.

(Quantifiers are omitted, since all variables are assumed to be universally quantified before the implication.)

ψ_1 represents the base case where *x* is already assumed to be clear in *s*. ψ_2 , embodying the induction case, introduces an induction hypothesis which finally will provide a recursive call to the plan *p*.

With that the first transformation phase is complete.

The evaluation process how it proceeds in the case of program synthesis serves to modify the resulting induction formulas using appropriate operator axioms. The operator axioms however are by no means equations. Therefore in our planning environment symbolic evaluation plays no role at all. Instead it is aimed at modifying the induction formulas in a way such that they can be *subsumed* by suitable operator axioms. A transformation means, not provided among the DEPSY transformation rules, but indispensable in this context, is the so-called *equality matching* (EQM), applied to identify terms which cannot be matched by a substitution.

The base case formula ψ_1 suggests to use the frame axiom of the empty operation (Ax2) for "evaluation", because the property *clear(x)* holding in the initial state *s* should be preserved to hold in the goal state *s!p(x s)*. The substitution $\{m \leftarrow \text{clear}(x),\ t \leftarrow s\}$ instantiates the axiom Ax2 appropriately. Now the terms *p(x s)* and *emptyop* can be identified and with that a solution is obtained for the base case.

$\psi_1 \Rightarrow_{\text{EQM}} \{\psi_3\}$, where

$\psi_3 = [H(\text{clear}(x)\ s) \rightarrow p(x\ s){=}\text{emptyop}]$.

ψ_3 already represents the first definition formula for the new plan.

The induction case formula

ψ_2 = [¬H(clear(x) s) ∧ H(clear(hat(x)) s!p(hat(x)s)) → H(clear(x) s!p(x s))] will now be transformed in a similar way. Here the system must aim at a subsumption with the operator axiom

Ax1: ∀u,v:block ∀t:state [H(on(u v) t) ∧ H(clear(u) t) → H(clear(v) t!puttable(u))].

To reach this goal employing the substitution {u←hat(x), v←x, t←s!p(hat(x)s)} and applying the EQM-rule appropriately (p(x s) := p(hat(x)s)!puttable(u)) presupposes an additional case analysis step: a suitable instance of the subformula H(on(u v) t) of Ax1 has to be introduced into ψ_2 before the subsumption of ψ_2 comes into question.

We obtain $\psi_2 \Rightarrow_{CA} \{\psi_4, \psi_5\}$, where

ψ_4 = [¬H(clear(x) s) ∧ H(on(hat(x) x) s!p(hat(x)s)) ∧ H(clear(hat(x)) s!p(hat(x)s))
→ H(clear(x) s!p(x s))]

and

ψ_5 = [¬H(clear(x) s) ∧ ¬H(on(hat(x) x) s!p(hat(x)s)) ∧ H(clear(hat(x)) s!p(hat(x)s))
→ H(clear(x) s!p(x s))] .

$\psi_4 \Rightarrow_{EQM,Ax1} \{\psi_6\}$, where

ψ_6 = [¬H(clear(x) s) ∧ H(on(hat(x) x) s!p(hat(x)s)) ∧ H(clear(hat(x)) s!p(hat(x)s))
→ p(x s) = p(hat(x)s)!puttable(hat(x))] .

The extraction process is omitted now because a defining equation has already been obtained.

In a last transformation phase the elimination rule has to be applied twice to eliminate those conditions from ψ_6 which are irrelevant to the definition of *p*. The heuristics selecting appropriate rule applications in the elimination process are used to eliminate those induction hypotheses which already have contributed to the definition of the new plan. In our case this applies to the formula H(clear(hat(x)) s!p(hat(x)s)) . These heuristics are also applied to those formulas which have been introduced by additional case analysis steps (formula H(on(hat(x) x) s!p(hat(x)s))).

Thus we obtain $\psi_2 \Rightarrow_{EL} \{\psi_7\}$ and $\psi_7 \Rightarrow_{EL} \{\psi_8\}$, where

ψ_8 = [¬H(clear(x) s) → p(x s) = p(hat(x)s)!puttable(hat(x))].

This formula represents a second (recursive) definition formula for *p*.

With that starting from the specification formula

ψ_0 = ∀x:block ∀s:state H(clear(x) s!p(x s)) the following plan has been obtained:

$\mathrm{DEF}_p = \{\ \forall x{:}\mathrm{block}\ \forall s{:}\mathrm{state}\ [H(\mathrm{clear}(x)\ s) \rightarrow p(x\ s) = \mathrm{emptyop}]$

$\quad\quad \forall x{:}\mathrm{block}\ \forall s{:}\mathrm{state}\ [\neg H(\mathrm{clear}(x)\ s) \rightarrow p(x\ s) = p(\mathrm{hat}(x)s)!\mathrm{puttable}(\mathrm{hat}(x))]\ \}$.

Finally the formula

$\psi_5 = [\neg H(\mathrm{clear}(x)\ s) \wedge \neg H(\mathrm{on}(\mathrm{hat}(x)\ x)\ s!p(\mathrm{hat}(x)s)) \wedge H(\mathrm{clear}(\mathrm{hat}(x))\ s!p(\mathrm{hat}(x)s))$

$\quad\quad \rightarrow H(\mathrm{clear}(x)\ s!p(x\ s))]$

resulting from the case analysis step above as a second one has to be considered. This formula cannot be transformed into a definition formula for *p* and therefore has to be proved as an assertion. Heuristics have been developed within DEPSY which simplify those assertions by providing adequate generalizations (cf. [Biundo 88]). Applying these heuristics to ψ_5 we finally obtain:

$[\neg H(\mathrm{clear}(x)\ s) \rightarrow H(\mathrm{on}(\mathrm{hat}(x)\ x)\ s!p(\mathrm{hat}(x)s))]$.

This assertion describes a property of the plan *p* and can be proved by induction using the definition formulas which have been synthesized together with the frame axiom Ax3 given for the *puttable*-operator. Such an induction proof can be performed by an automated induction theorem prover like the one of Boyer and Moore (cf. [Boyer/ Moore 79]) or the INKA-System (cf. [Biundo et al. 86]).

An automated induction proof of the above formula succeeds and thus the plan generation process for *p* is completed.

A deductive solution to the example planning problem of clearing a block has also been given in [Manna/Waldinger 86]. There, although the example seems to be quite simple, a problem occurs. Corresponding to formula ψ_5 above an assertion has to be proved claiming that *hat(x)* rests upon *x* even when *hat(x)* has been cleared. Since an induction proof of this formula cannot be obtained the formula has to be generalized. But "the strengthening required seems to be beyond the capabilities of the Boyer-Moore system or other current theorem provers" (cf. [Manna/Waldinger 86]). Thus an automatic proof of this formula lies not within reach.

In our approach the axiomatization of the planning domain is done using the HOLDS-predicate. This axiomatization seems to be more adequate than the formulation of situational logic choosen by Manna and Waldinger. Especially the formulation of frame axioms, which play an important role in this context, contributes greatly to the success of the planning process: They can be used by an automated induction theorem proving system to prove the resulting assertions which have been modified by the DEPSY generalization heuristics beforehand.

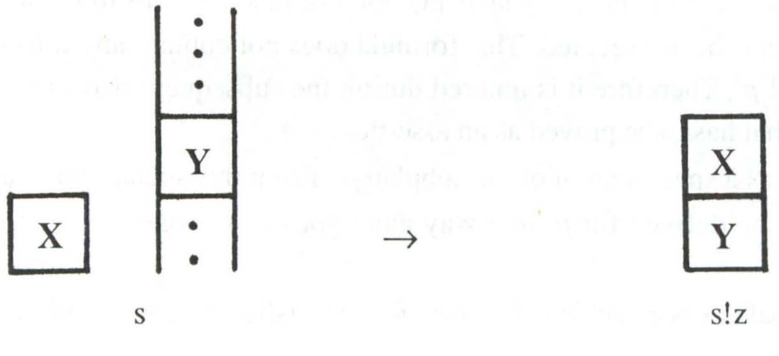
The example given above shows how a plan can be generated using the synthesis strategy DEPSY provides. Starting from a specification ψ_0 a recursive operator *p* (actually: "make-clear") has been synthesized which removes all blocks resting upon a given block *x*. Except for the evaluation phase where equality matching is used instead of symbolic evaluation the program synthesis strategy has been suitable for this case of plan formation.

This fact however may be accidental. As can even be seen from our example it might not be useful to maintain a strict distinction between extraction and evaluation phase. The reason is that in many cases the induction hypothesis should contribute to provide a match between the goal and the selected operator axiom during the "evaluation" phase, rather than being adapted to the induction conclusion afterwards. Thus a strategy should be developed where heuristics and strategies from both the extraction and the evaluation phases are combined in order to get the goal formula subsumed by an appropriate operator axiom.

Another modification of the synthesis strategy is required if subplans have to be introduced.

5. INTRODUCING SUBPLANS

Plans in general are *sequences* of (recursive) operations. Taking this fact into consideration sufficiently means to allow plans being divided into subplans which are generated independently. From the program synthesis point of view this compares to composing a specified function from particular auxiliary functions synthesized separately. Although the DEPSY strategy doesn't provide any means to introduce such functions we will demonstrate by the following example how the synthesis of subplans can still be performed using DEPSY transformation rules.
Suppose we have to deal with the following plan specification:



The formal specification reads:

$\psi = \forall x,y{:}block\ \ \forall s{:}state\ \ \exists z{:}plan\ \ [H(clear(x)\ s) \wedge \neg x{=}y \rightarrow H(on(x\ y)\ s!z)]$.

After skolemization we obtain:

$\psi_0 = \forall x,y{:}block\ \ \forall s{:}state\ \ [H(clear(x)\ s) \wedge \neg x{=}y \rightarrow H(on(x\ y)\ s!p(x\ y\ s)]$.

This formula could be transformed using the implication rule together with the *put*-operator axiom

$\forall$u,v:block $\forall$t:state [H(clear(u) t) $\wedge$ H(clear(v) t) $\wedge$ $\neg$u=v $\rightarrow$ H(on(u v) t!put(u v))]

in the following way:

The implication rule

([C $\rightarrow$ σL] $\Rightarrow_{\text{IMPL,D}}$ { [C $\rightarrow$ σK_1] , ... , [C $\rightarrow$ σK_n] } iff
D = [$K_1 \wedge ... \wedge K_n$ $\rightarrow$ L] and D $\in$ Ax)

allows to replace a conclusion σL in a formula by formulas σK_i which are sufficient for it.

Thus using the substitution $\sigma = \{u \leftarrow x, v \leftarrow y\}$ the subformula H(on(x y) s!p(x y s)) in ψ_0 might be replaced by formulas H(clear(x) ...) and H(clear(y) ...) respectively. With that the original goal ψ_0 would have been transformed into two subgoals the solution of which seems to be quite simple.

However, to succeed in this way presupposes to divide the plan p(x y s) into two different subplans: p(x y s) := p'(x y s)!put(x y) .

Introducing a subplan *p'* and applying the implication rule then, together with the axiom of the *put*-operator (and $\sigma = \{u \leftarrow x, v \leftarrow y, t \leftarrow s!p'(x\ y\ s)\}$), yields three new formulas:

ψ_1 = [H(clear(x) s) $\wedge$ $\neg$x=y $\rightarrow$ p(x y s) = p'(x y s)!put(x y)] ,

ψ_2 = [H(clear(x) s) $\wedge$ $\neg$x=y $\rightarrow$ H(clear(x) s!p'(x y s))] and

ψ_3 = [H(clear(x) s) $\wedge$ $\neg$x=y $\rightarrow$ H(clear(y) s!p'(x y s))] .

ψ_1 represents a definition formula for *p* and isn't considered any longer.

Formula ψ_2 expresses that the property *clear(x)* holding in state *s* has to be preserved holding after the plan *p'* has been executed. This formula does not contain any information about the functionality of *p'*. Therefore it is ignored during the subsequent part of the generation process and after that has to be proved as an assertion.

ψ_3 finally is viewed as a specification of the subplan *p'*. From this specification a recursive plan definition will be derived for *p'* in a way analogous to the one demonstrated in Section 4.

Then even the original goal has been achieved: a plan which satisfies the specification ψ_0 .

6. CONCLUSION

By means of an example taken from the blocks world we have shown how plans can be generated using the deductive program synthesis method described in [Biundo 90].

The method in principle seems to be suitable for deductive planning (at least as far as the blocks world is concerned): Most of the transformation rules can be adopted to plan formation where they have to perform the same tasks as in the case of program synthesis. Even the overall synthesis strategy seems adequate to provide goal directed transformations in certain cases.

Finally, in addition to the means DEPSY provides for the synthesis of programs resp. plans, we have demonstrated how *subplans* can be introduced and generated separately.

However, to obtain a powerful and fully automatic planning system some strategic modifications have still to be made. The reason is that the deductive mechanism underlying DEPSY relies very much upon equality reasoning whereas our axiomatization of the planning domain requires dealing with the HOLDS predicate. Thus some of the DEPSY transformation rules have to be modified performing *equality* matching instead of normal matching alone.

Another modification refers to the overall synthesis strategy. This strategy aims at the derivation of *recursive* programs, whereas plans often are closely analogous to imperative programs. Thus a flexibilization of the strategy should be performed.

Nevertheless, in contrast to the approach proposed by Manna and Waldinger in [Manna/ Waldinger 86] it seems to be very promising to obtain a fully automatic planning system based on what we have shown in this paper. The reason for this is twofold:

Firstly with DEPSY a fully automatic synthesis system is available and most of its strategies and heuristics will be extremely helpful even if *plans* have to be synthesized. For the approach of Manna and Waldinger similar conditions do not hold, since an automatization of the underlying program synthesis system has not been obtained (cf. [Traugott 86], [Manna/ Waldinger 87]).

Secondly our axiomatization of the planning domain, where even frame axioms can be expressed conveniently, seems to be more adequate than the formulation of situational calculus Manna and Waldinger present. Consequently a proof of the assertions which in most cases has to follow the plan generation process can be performed by an automatic theorem proving system.

REFERENCES

[Bibel 80] **Bibel W.**
Syntax-Directed, Semantics-Supported Program Synthesis.
Artificial Intelligence 14 (1980)

[Bibel 86] **Bibel W.**
A Deductive Solution for Plan Generation.
New Generation Computing 4 (1986)

[Bibel et al. 89] **Bibel W., Farinas del Cerro L., Fronhöfer B. and Herzig A.**
Plan Generation by Linear Proofs: On Semantics.
Proc. of the 13th German Workshop on Artificial Intelligence,
Eringerfeld (1980)

[Biundo 88] **Biundo S.**
Automated Synthesis of Recursive Algorithms as a Theorem Proving
Tool. Proc. of the 8th European Conference on Artificial Intelligence
München (1988)

[Biundo 90] **Biundo S.**
Automatische Synthese rekursiver Algorithmen als Beweisverfahren.
Springer Informatik Fachberichte 1990 (to appear)

[Biundo et al. 86] **Biundo S., Hummel B., Hutter D., and Walther C.**
The Karlsruhe Induction Theorem Proving System.
Proc. of the 8th Intern. Conf. on Automated Deduction,Oxford (1986)

[Boyer/Moore 79] **Boyer R.S. and Moore J S.**
A Computational Logic. Academic Press (1979)

[Franova 88] **Franova M.**
Fundamentals of a New Methodology for Program Synthesis from
Formal Specifications: CM-Construction of Atomic Formulae.
These, Universite de Paris-Sud, Orsay (1988)

[Goad 80] **Goad C.A.**
Computational Uses of the Manipulation of Formal Proofs.
Ph.D. Thesis, Stanford University (1980)

[Green 69] **Green C.C.**
Application of Theorem Proving to Problem Solving.
Proc. of the 1st International Joint Conference on Artificial Intelligence,
Washington (1969)

[Hsiang 83] **Hsiang J.**
Topics in Automated Theorem Proving and Program Generation.
Ph.D. Thesis, University of Illinois, Urbana (1983)

[Kowalski 79] **Kowalski R.**
Logic for Problem Solving.
North Holland Publishing Company, Amsterdam (1979)

[Manna/Waldinger 79] **Manna Z. and Waldinger R.**
Synthesis: Dreams → Programs.
IEEE Transactions on Software Engineering Vol.SE-5 No 4 (1979)

[Manna/Waldinger 80] **Manna Z. and Waldinger R.**
A Deductive Approach to Program Synthesis.
ACM Transactions on Programming Languages and Systems
Vol.2 No 1 (1980)

[Manna/Waldinger 86] **Manna Z. and Waldinger R.**
How to Clear a Block: Plan Formation in Situational Logic.
Proc. of the 8th Intern. Conf. on Automated Deduction, Oxford (1986)

[Manna/Waldinger 87] **Manna Z. and Waldinger R.**
The Deductive Synthesis of Imperative LISP Programs.
Proc. of the 6th National Conf. on Artificial Intelligence, Seattle (1987)

[McCarthy 68] **McCarthy J.**
Situations, Actions, and Causal Laws.
In: Semantic Information Processing, M. Minsky (ed.), MIT Press,
Cambridge, Mass. (1968)

[SFB 90] Sonderforschungsbereich 314
Arbeits- und Ergebnisbericht für die Jahre 1988-1989-1990,
Universität Karlsruhe (1990)

[Traugott 86] **Traugott J.**
Deductive Synthesis of Sorting Programs.
Proc. of the 8th Intern. Conf. on Automated Deduction, Oxford (1986) .

Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

DFKI
-Bibliothek-
Stuhlsatzenhausweg 3
6600 Saarbrücken 11
FRG

DFKI Publikationen

Die folgenden DFKI Veröffentlichungen oder die aktuelle Liste von erhältlichen Publikationen können bezogen werden von der oben angegebenen Adresse.

DFKI Publications

The following DFKI publications or the list of currently available publications can be ordered from the above address.

DFKI Research Reports

RR-90-01
Franz Baader
Terminological Cycles in KL-ONE-based Knowledge Representation Languages
33 pages

RR-90-02
Hans-Jürgen Bürckert
A Resolution Principle for Clauses with Constraints
25 pages

RR-90-03
Andreas Dengel & Nelson M. Mattos
Integration of Document Representation, Processing and Management
18 pages

RR-90-04
Bernhard Hollunder & Werner Nutt
Subsumption Algorithms for Concept Languages
34 pages

RR-90-05
Franz Baader
A Formal Definition for the Expressive Power of Knowledge Representation Languages
22 pages

RR-90-06
Bernhard Hollunder
Hybrid Inferences in KL-ONE-based Knowledge Representation Systems
21 pages

RR-90-07
Elisabeth André, Thomas Rist
Wissensbasierte Informationspräsentation: Zwei Beiträge zum Fachgespräch Graphik und KI:

1. Ein planbasierter Ansatz zur Synthese illustrierter Dokumente
2. Wissensbasierte Perspektivenwahl für die automatische Erzeugung von 3D-Objektdarstellungen

24 pages

RR-90-08
Andreas Dengel
A Step Towards Understanding Paper Documents
25 pages

RR-90-09
Susanne Biundo
Plan Generation Using a Method of Deductive Program Synthesis
17 pages

DFKI Technical Memos

TM-89-01
Susan Holbach-Weber
Connectionist Models and Figurative Speech
27 pages

TM-90-01
Som Bandyopadhyay
Towards an Understanding of Coherence in Multimodal Discourse
18 pages

Plan Generation Using a Method of Deductive Program Synthesis

Susanne Biundo

RR-90-09
Research Report