Therapy Plan Generation in Complex Dynamic Environments *

Oksana Arnold† and Klaus P. Jantke‡

TR-94-054

October 1994

Abstract

There has been developed a methodology for the automatic synthesis of therapy plans for complex dynamic systems. An algorithm has been implemented and testet. This is the core of some control synthesis module which is embedded in a larger knowledge-based system for control, diagnosis and therapy. There are several applications.

The approach is based on certain concepts of structured graphs. The overall search space is a family of hierarchically structured plans. Together with some goal specification it is forming a so-called rooted family. Simple concepts of graph substitution and rewriting are introduced. The output of the planner is a hierarchically structured plan. This has a uniquely determined normal form taken for execution.

Plan generation is interpreted as inductive program synthesis. Indeed, the planner developed and implemented works as an inductive inference machine.

It turns out that consistency and executability are two fundamental, but distinguished concepts. When describing the program synthesis algorithm, we focus on constraint monitoring. This is taken as a basis for generating programs being consistent with the underlying technology representation.

---

*The work has been partially supported by the German Federal Ministry for Research and Technology (BMFT) within the Joint Project (BMFT-Verbundprojekt) **Wiscon** on **Development of Methods for Intelligent Monitoring and Control** under contract no. 413-4001-01 IW 204 B.

†Hochschule für Technik, Wirtschaft und Kultur Leipzig (FH), Fachbereich Informatik, Mathematik & Naturwissenschaften, P.O.Box 66, 04251 Leipzig, Germany. Part of this work was done while the author was visiting the "International Computer Science Institute", Berkeley, CA

‡Hochschule für Technik, Wirtschaft und Kultur Leipzig (FH), Fachbereich Informatik, Mathematik & Naturwissenschaften, P.O.Box 66, 04251 Leipzig, Germany

Contents

1 Motivation and Introduction **2**

2 Towards Knowledge-Based Supervision and Control **3**
2.1 Integrated Architectures 4
2.2 Relating Therapy Plan Generation 5
2.3 Issues of Knowledge Representation 6

3 Complex Dynamic Systems **7**
3.1 Characteristics of Dynamic Processes 7
3.2 Incomplete and Dynamic Knowledge Representation 8
3.3 Pecularities of Therapy Plan Generation 9

4 Planning in Artificial Intelligence **11**
4.1 Planning Approaches 11
4.2 Plan Concepts 12

5 Therapy Plans as Hierarchically Structured Graphs **13**
5.1 Action Scripts 13
5.2 Therapy Plans 14

6 Knowledge–Based Plan Synthesis **16**
6.1 System Architecture 16
6.2 Therapy Plan Synthesis 17
6.3 Implementation 23
6.4 Applications 24

7 Conclusions **25**

References **27**

1 Motivation and Introduction

The main intention of the present paper is to develop the authors' overall approach to the automatic synthesis of therapy plans for complex dynamic systems. Due to a couple of pecularities of the intended target domain, this approach is considerably distinguished from most other planning approaches. The authors' survey [AJ94a] is devoted to some comparison in this respect.

In current process automation, there is an obvious development towards more complex, more efficient and less controllable dynamic processes. In cases of emergency, when disturbances occur and timely responses are urgently required, human beings have usually serious difficulties to process the enormous amount of information to come up with the right diagnosis as well as the right control decisions. Several catastrophic crashes bear abundant evidence of the need of a remarkable computer support ultimately focussed on therapy plan generation. The authors advocate a knowledge-based approach.

Motivated by a comprehensive approach towards knowledge-based supervision and control undertaken in the joint project WISCON supported by the German Federal Ministry for Research and Technology (BMFT), there has been developed an approach to therapy plan synthesis within an integrated knowledge processing architecture (cf. [AMM92]). The general approach is outlined in some detail in [Arn92a]. Chapter 2 of the present paper is dedicated to a brief up-to-date motivation relating therapy plan generation to the overall approach.

There is a particular class of target processes focussed on. Chapter 3 is aimed at briefly characterizing the pecularities of those processes. These pecularities have an essential impact on all further decisions.

The chapters 2 and 3 are setting the stage for characterizing the approach to therapy plan generation under development. Chapter 4 is briefly relating the authors' approach to the activities of the relevant community. This will complete the preliminaries.

A careful inspection of some representative sample of current work in planning (cf. [AJ94a]) exhibits some lack of precision. The deficiency of precise concepts[1] has some implications. As long as certain formalisms are unclear, a couple of detailed questions cannot be answered. For instance, it seems a crucial problem in plan generation and plan execution to model properly simultaneuous actions. This has to be distinguished carefully from concurrent activities which may be performed sequentially in any order. Motivated by the class of intended target domains where simultaneuous actions are sometimes inevitably necessary, the authors feel an urgent need of mathematically well-based underlying concepts.

Chapter 5 is presenting our formalisms to describe plans based on concepts of hierarchically structured graphs. These concepts are adopted from computational complexity investigations and tailored to the needs of therapy plan generation. The fundamentals, which have been outlined in [AJ94c], [AJ94b] and [AJ94d] first, are developed in some detail.

Based on this formal background, chapter 6 contains the therapy plan generation approach. Interestingly, the generation of therapy plans as developed turns out to be inductive program synthesis, in some sense. [AJ94b] is focussed on this particular insight.

[1] Surprisingly, 6 out of the most recent (and best) 10 German Ph.D. theses which deal with planning ([Gün92], [Wöh92], [Hör92], [Fis93], [Sau93], [Köh93], [Wer93], [Win94], [Köh94], [Thi94]) are suffering from the lack of a somehow formal and explicit plan concept (cf. [AJ94a] for an analysis).

2 Towards Knowledge-Based Supervision and Control

The present chapter is intended to briefly sketch the knowledge processing problems surrounding our present approach. We do not aim at a comprehensive introduction to knowledge-based process supervision and control.

Let us start with an illustrative example. As a typical representative of industial chemical processes we consider a part of a technological equipment for the production of chemical fibres. The esterification part consists, for simplification, of two reactors and one distillation column. In both reactors, there some raw materials produce PTA by reaction. A secondary reaction product is withdrawing from the reactors and is separated by the distillation column in order to regain one of the raw materials.

Now, a short disturbance description is given, before we sketch an appropriate therapy control. If an excessive foam formation is occurred in one of the reactors, some PTA may reach the destillation column. There the product solidifies on grounds of low temperature and obstructs column trays. The difference of pressure decreases and the vacuum breaks down. The distillation fails. There are more consequences of the sketched disturbance we do not describe in detail. But in the end, a reduced conversion rate and the disturbed composition effect on quality of chemical fibre in negative sense.

In order to minimize the amount of less quality material, the whole equipment is shut down step by step. In the same time, the distillation column should be repaired. After finishing the repair actions the production should be recovered. To perform the start-up and shut-down processes as well as to activate and monitor repair measures, a variety of temporally staggered control actions has to be initialized and executed. Initially, the foam regulating valve should be closed to avoid a further escape of reaction mixture into the distillation column. Then, both the distillation column could be repaired and a suitable process regime could be established for the esterification reactors, in parallel. In order to repair the distillation column two different approaches are applicable. First, it is possible to remove the solidified PTA from the trays by scalding. For that, the bottom temperature is increasing. But it is usually unknown how much PTA is in the column and in which kind the trays are obscured. These parameters are unmeasurable. Therefore, the success of scalding is monitored by the registration of the pressure difference over the column. Here, another problem becomes explicit. In general, it is impossible to say in advance, how long some action needs to reach it's goal. Only if there is no success after a maximal duration which is determined by experience, the corresponding action should be aborted and replaced by another action available for the necessary goal. In our example the distillation column should be cleaned mechanically.

For human operators, it is usually extremely difficult to process the huge amount of low level information provided by a process control system when some disturbance happened. Opposed to the low level information provided, human reasoning requires some high level insights to identify certain primary faults and to decide about appropriate therapy actions. This becomes even more complex if timely responses are required. The intention of the BMFT joint project WISCON already mentioned is to invoke knowledge processing principles and techniques for attacking the type of problems explained.

It is essential to understand that a particular task in knowledge-based supervision and control

is inevitably depending on other surrounding knowledge processing tasks. It is embedded in an ensemble of tasks related to each other. A careful conceptual design of one component must be based not only on a specification of its intended functionality, but also on a careful inspection of its interaction with other components and the role it is playing in the ensemble.

2.1 Integrated Architectures

Before developing any knowledge base, one should carefully examine the knowledge to be represented. The authors' approach has been investigated and described in [Arn94a]. If possible, insights gained from such an investigation should be utilized for structuring the knowledge base under development. First, the knowledge about possible faults and control measures depends on the constructive performance of technological components. Second, technological equipments usually have a remarkable number of repeatedly occurring units. This results in similar or even identical knowledge structures for diagnosis as well as therapy. Furthermore, the knowledge base should maintain all relevant process data, which are available about the past, the present and the future of the process and which are needed for any derivations, as illustrated above. That means that the knowledge base has to perform the communication between the expert system and the upstream supervision and control computer system.

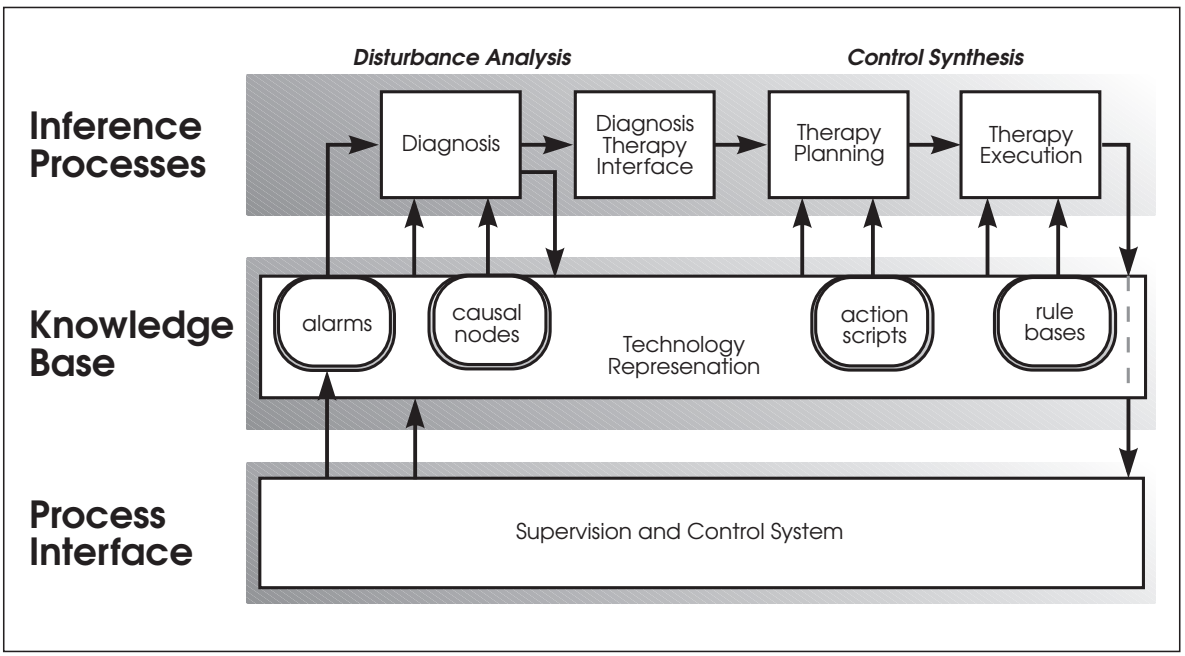

Figure 1: System Architecture

In order to reflect these insights and to meet these requirements, it seems to be efficient to structure the knowledge base by modelling the technological equipment. We will go into a little more detail, below.

On this knowledge base, a couple of knowledge processing problems have to be solved. In the problem area under consideration, the key problems are diagnosis and therapy. For these tasks,

there are several dedicated software modules, which should have access to particular knowledge units contained in the structured knowledge base. Additionally, it is highly desirable to integrate further specialized modules for particular service. For instance, a simulation[2] module able to forecast values of process parameters according to a therapy plan generator's request may be of great importance for synthesizing more reliable therapy plans. The figure above is illustrating an appropriate architecture. Additional modules like the one for simulation are not depicted.

2.2 Relating Therapy Plan Generation

A therapy control in engineering is some ordering of control actions (a therapy plan[3]) in order to influence the technical process in timely response. It's aim is to remove the causes of a detected disturbance, to perform a new control regime to minimize the losses of production, and to adopt the situation recognition which monitors alarm boundaries. These tasks should be done by both sending new setpoint values, control values, and alarm boundaries to a conventional supervision and control system as well as informing the operator about some mechanical measures (cf. [BMHC90], [BKM92], [Fri93]). In dependence on the discovered fault, the current reserves of the system, and the state of the process, the control synthesis should derive and execute an appropriate therapy plan. Thus, therapy plans are formal objects having some operational semantics in the process under consideration. They are programs (cf. [AJ94b]).

The module for therapy plan generation is interacting with a large number of modules in its knowledge processing environment. Roughly speaking, there are the following mates in communication:

- The module for diagnosis, which is triggering therapy plan generation via some interface translating detected primary faults into goal specifications,
- the knowledge base module containing action scripts (attached to technological classes and technological objects) used as building blocks for therapy plans,
- the technology representation containing static (i.e. structural) knowledge about the underlying process as well as dynamic knowledge (i.e. values of process parameters related to time points or time intervals) used for further action specification and constraint checking,
- the module of operational interpretations of elementary actions (rule basis, in our present approach) accessed through the plan execution module, and
- the plan execution module which, on the one hand, accepts plans for execution and, on the other hand, triggers plan revision in case that plan execution fails.

Generated therapy plans are hypotheses about possibilities to control the underlying complex dynamic process. Due to the inevitable incompleteness of the process information available (cf. chapter 3), the executability of plans is not formally derivable. Instead, plans generated for execution are carefully monitored during plan execution for starting plan revision whenever necessary. Thus, the plan generation process is a highly interactive one.

[2] Within the WISCON architecture, a simulation module implemented on a dedicated transputer architecture is integrated.

[3] The formalisms will be developed in chapter 5 of the present paper.

2.3 Issues of Knowledge Representation

As developed in [Arn94a], the structure of the knowledge base should reflect essential structural properties of the target process. Therefore, we provide a frame-oriented language called the technology representation, which permits the definition of technological classes and instances. In general, technological classes are abstract objects, which summarize all common properties and parameters of equal components and devices, i.e. they generalize regularities of the knowledge about constructive relations. Moreover, technological classes should be defined in advance and represent prototypes of technological units. Actually existing technological units are instances of created classes and inherit all defined knowledge from the corresponding classes and superclasses. The described hierarchy of technological objects is called classification taxonomy.[4]

Proceeding from the mentioned points, the dependency of knowledge on the construction of technological components and subequipments and the possibility of generalization, we assign complex and elementary action scripts to those technological object classes or instances they have relevance for. This allows for the generalization of knowledge about therapy and reduces the amount of action scripts. In this context, a unary function is defined which is able to return technological objects referenced by action scripts.

Actions scripts have names which may occur repeatedly. The name of an action script determines the goal the execution of the action script should reach.

With each action script, a preference declaration is connected which provides a natural number that indicates the costs of the defined action. Larger numbers indicate higher preference. Action scripts of a higher preference are prefered by the therapy planner to those of a lower preference while generating an appropriate therapy control.

Criteria for the inheritance of action scripts are established by the following commitments:

- Action scripts with the same name and different preferences collected along a classification path form alternatives[5] for the considered object class or instance.
- The definition of two action scripts which are related to the same technological object and have the same name and preference is not permitted.
- Action scripts with the same name and the same preference collected along a classification path form *specializations*. Hence, the most special action script which can be found starting with the considered technological object and going towards the classification root is chosen for that technological object.

This structure of knowledge base has, on the one hand, the advantage that it is understandable for process engineers and system operators. On the other hand, it permits the composition of system independent component-libraries to provide reuseable modules of diagnostic and therapy knowledge.

[4] In addition to the classification taxonomy, further taxonomies are available in the technology representation, but they play a minor role in the presented context.

[5] Alternatives for performing the same control goal correspond to different control measures which differ in the necessary system reserves, energy, and material streams. In that way, it is possible to continue aspiring the desired control goal besides additional disturbances and falling out of further resources by means of alternative control measures.

3 Complex Dynamic Systems

3.1 Characteristics of Dynamic Processes

The intention of the present chapter is to introduce the main pecularities of our underlying class of problem domains. It is distinguished from the areas investigated in most of the approaches found in the recent literature about planning. One of the essentials is that executability is not a formal concept. Therefore, we need another concept for verifying generated plans. This concept is the concistency of plans with respect to knowledge represented in the technology representation. Consequently, the domain under consideration requires rather an inductive planning algorithm than a deductive approach.

There are a lot of properties charaterizing the problem domain and having impact on modelling the planning knowledge as well as on planning and execution. Thus, they are fundamental for our program synthesis approach. Among the most important aspects are the following:

I Essential Characteristics of the Domain

* Values of process parameters may be set only by indirect influence. Consequently, the effects of actions (postconditions) are not always definite.
* Sometimes, parameters which are necessary to describe the complete state of some device are not accessible.
* The success of actions depends, in general, on the availability of technological resources (preconditions), like energy, conveyance pipes and storage capacity.
* There may be some need for simultaneous actions resulting from the structure of the equipment and the running processes.

Further properties are relevant in other domains, too. In this context, they complicate the planning problem additionally.

II Further Characteristics of the Domain

* The resource dependencies on actions are not limited to those which should be fulfilled at the beginning of actions. There are resource constraints which should be met during the whole execution time of actions.
* Actions require a certain amount of time to be completed. This time has to be taken into account but may unknown in advance.
* Setpoints of process parameters may be unstable. Thus, the whole process is subject to permanent changes.
* In general, there exist alternative actions which differ in resource constraints and cost.

Consequently, the therapy control is contingent on the duration of the primary fault and the current reserves of the system. Moreover, its duration is determined by residence time of units, capacity of stores, etc. During therapy execution the conditions may change substantially, both due to unforeseeable process conditions and due to unexpected side effects of the therapy itself

including interferences of individual therapy actions. Roughly speaking, there is no hope for complete information.

In order to take all pecularities of the domain into account within one approach, it is necessary to develop an appropriate language for expressing action operators.[6] Based on this language generated therapy plans should be executable to meet their goal specification, i.e. to be semantically correct like in program synthesis. But among others, the indirect setting of values exhibits that executability is determined by the process dynamics. Thus, the executability of a generated plan is not a formal concept in our domain. Correctness and completeness of any planner cannot be completely formalized with respect to the underlying process.

The ultimate goal of our present approach is a formalization of basic concepts to manifest the distinction from both conventional planning approaches and conventional program synthesis approaches. Based on this approach, there is developed and implemented some algorithm for therapy plan generation.

3.2 Incomplete and Dynamic Knowledge Representation

The knowledge base assumed contains, roughly speaking, a static and a dynamic part. Basic constituents are depicted in figure 1 above. Certain questions of representing static knowledge have been discussed in chapter 2.3. Here, we focus on the dynamic aspects.

In process supervision and control, a discrete time scale is assumed. One may consider time points as natural numbers. Furthermore, time points are assigned to measured values of process parameters in order to specify the validity of those values. Time intervals will get relevance, if, especially, simulation for forecasting particular process parameters is invoked. Measured values and forecasted data are written into the knowledge base. The technology representation is changed by these newly incoming data. Current values of process parameters become historical on expiration. Thus, histories of process values for each parameter are performed. Formally, we introduce the notation $\langle x \in_{[t_1,t_2]} [v_1, v_2]\rangle \in \mathcal{TR}$ in order to express that some process parameter x ranges between v_1 and v_2 during the time interval $[t_1, t_2]$. The whole notions and notations will be carefully introduced below, but here we want to point out, that the introduced representation is appropriate for both historical and forecasted values of parameters. This is due to the uncertainty of measurements and modelling of physics. However, a standard entry may particularly mean $\langle x \in_{[t,t]} [v, v]\rangle \in \mathcal{TR}$.

Besides this, there is a second level of time considerations. Let us focus again on technology representation which permanently change by incomming data. Nevertheless, at any time point n, there is a certain content of the technology representation. The knowledge base at time n is briefly denoted by $\mathcal{TR}_n$. $\mathcal{TR}_n$ may contain data about the past, the present (with respect to n), and the future. Consequently, with growing time there is a permanently changing sequence of dynamic knowledge bases $\ldots, \mathcal{TR}_n, \mathcal{TR}_{n+1}, \ldots, \mathcal{TR}_m, \ldots$ If only the process supervision and control system is writing process information into the dynamic part of the technology representation, this knowledge base developes essentially monotonously. In formal terms, $n < m$

[6] The causes for a new approach are different. First, it seems quite impossible to use STRIPS-like action definitions. They are useless if there is no way to define precisely postconditions. Furthermore, they are only applicable in the face of interest in sequential plans. Second, we want to use as much information available about the process, disturbance states and therapy measures (i.e. manuals, experience, physical models) as possible.

implies $\mathcal{TR}_n \subseteq \mathcal{TR}_m$. Furthermore, reasoning at time n is based on $\mathcal{TR}_n$. But when a certain reasoning step has been successfully completed, the knowledge base has been changed to some $\mathcal{TR}_m$ $(m > n)$. This is unavoidable and, trivially, there is no reset.

If there has been built some therapy plan based on some technology representation state $\mathcal{TR}_n$, is has to be executed during some time interval $[t_1, t_2]$ with $n < t_1$. Consequently, due to the process dynamics which is essential for the considered class of target processes (cf. chapter 2), the therapy plan may fail within $[t_1, t_2]$, although it has been correct with respect to the knowledge presented in $\mathcal{TR}_n$. There is no general deductive way to check this problem in advance. We will come back to problems of this type, below. In particular, we will complement the desired, but unprovable property of executability by the theoretical concept of consistency.

3.3 Pecularities of Therapy Plan Generation

Therapy plan generation for complex dynamic processes as described above is characterized by two basic pecularities. These may be very briefly described as follows. First, therapy plan generation is necessarily based on incomplete information. Second, it is generating programs for execution. Thus, in its right perspective, therapy plan generation turns out to be inductive program synthesis. This has been made explicit in [AJ94b], for the first time.

3.3.1 Program Synthesis

For automatic program synthesis, [BB93] is an excellent recent reference: *The design and implementation of correct software meeting given requirements continues to be most relevant, practical, and scientifically challenging problem. There are many lines of research directed towards solving this problem. Automatic Programming is one among those. It is the study of techniques for generating executable code from information which may be fragmentary and may only indirectly specify the target behavior.*
The field ist based on the idea that ultimately we need to engage the machine itself in the process of programming machines, since only machines offer the important property needed for this task which is the ability to work without making mistakes. The "Automatic" in the name does not necessarily refer to a full automation of programming, but rather to a considerably higher degree of automation than in other lines of software production pursued by the literature. We will adopt this view in the sequel.

Among the approaches to program synthesis which are relevant to our problems, inductive program synthesis seems to be most appropriate. Generating therapy plans for process control in dynamic environments needs to be based on incomplete information, only. Processing incomplete information for coming up with complete solutions is the crux of problems attacked by inductive inference research (cf. [AS83], [KW80], [Ang92], [Wie92]), in general, and inductive program synthesis (cf. [Sum75], [BBP75], [BK76], [Sum77], [Bau79], [Sha81], [Sha83], [JK83], [BK86], [DP90], [FD93], [DR93], e.g.), in particular. The approaches referred to vary from purely recursion-theoretic to those synthesizing LISP expressions or Prolog programs ([Sum75], [Sum77], [Sha81], [Sha83] e.g.). This is the type of work we want to relate to therapy plan generation.

However, the type of control problems to be attacked below require a more explicit handling of multiple knowledge (cf. [Arn93], [Arn94b]) than considered in most of the papers mentioned

above. In this regard, approaches like [Bar79], [MW74], [MW83], and [Neu92] seem more appropriate, although the latter three are deductively oriented.

3.3.2 Inductive Hypothesizing

As explained in some detail above, in dependence on the complexity and dynamics of the target processes as well as in dependence on the available knowledge representation, therapy plan generation may be essentially inductive. This motivates the following brief excursion to inductive inference research.

Inductive inference is a well-studied research area of theoretical computer science which focusses on the problem of learning from incomplete information. This research area has its origins in the early papers [Sol64] and [Gol67]. We consider [Gol67] the seminal publication of inductive inference. [AS83] and [KW80] are very good surveys. Additionally, there is a large number of easy introductions, like [Jan89], for instance. Therefore, we won't give a new introduction into inductive inference. Instead, we stress some selected aspects which have guided our investigations below. The first aspect is the one of disproving instead of proving formulae. The second one is the problem of consistent vs. inconsistent inductive inference. There is a huge amount of further exciting questions not considered here.

Because of the essential incompleteness of information presented to an inductive inference device, such a mechanism is "guessing" its hypotheses in a sense, although these guesses may be algorithmically constructed in a sophisticated manner. Usually, there is no way to prove a guess correct. Instead, if a former guess becomes provably incompatible with recent information, it has to be changed. In other words, the key deductive task to be performed by an inductive inference device is disproving, not proving formulae. In [Jan84], this phenomenon has been made explicit.

Interestingly, this exactly meets the needs of therapy plan generation over some sequence of changing technology representations $\ldots, \mathcal{TR}_n, \mathcal{TR}_{n+1}, \ldots, \mathcal{TR}_m, \ldots$ There is usually no way to prove that the choice of some action script fit into some therapy plan will result in an executable plan. However, one may avoid some non-executable plans by checking whether or not the available knowledge is already sufficient to refute some particular action script. If this works, it may reduce the search space underlying plan generation enormously. This insight will be used below.

Recall that executability is the ultimately desirable property of therapy plans to be generated. However, it is not formally provable, in general. Thus, one tries to generate therapy plans which are at least "probably" executable in the future. From another perspective, one tries *not* to generate those plans which are "probably" *not* executable. This leads to the notion of consistent plans. In inductive inference, a hypothesis is called consistent, exactly if it is not contradicting the incomplete information presented. First, most inductive inference algorithms developed work consistently. Second, – and this is one of the deeper insights provided by inductive inference research – it may be more efficient to use an inference algorithm which has the freedom to generate, at least sometimes, inconsistent guesses. [Wie92] is a recent reference, in this respect. There are results of two different types. First, there are inductive inference problems which are solvable, but only if the inference algorithm used is not required to work consistently (cf. [JB81] contains both a couple of results and references to further publications). Second, there are inductive inference problems for which inconsistent algorithms are remarkably more efficient than consistent algorithms (cf. [Wie92] and references therein).

Our basic planning algorithm developed below (cf. [Arn92a], [AJ94c], [AJ94b]) works consistently. In fact, refuting inconsistent action scripts during plan generation is the main deductive task involved. But we can imagine particular circumstances where inconsistent planning may gain an advantage over consistent algorithms. This remains an exciting area of future research.

4 Planning in Artificial Intelligence

Planning is a central area of artificial intelligence research and applications. Papers and systems are mushrooming. It exceeds the authors' competence considerably to survey the area. But there is some methodological need to relate the own approach.

4.1 Planning Approaches

Planning has its origins in the situation calculus (cf. [MH69]) usually ascribed to McCarthy, although many researchers contributed to its development. From a quite general and somehow vague perspective, planning is searching for certain actions to transfer some given intial state into some desired goal state. The result generated during planning is called a plan. In the situation calculus, the key concepts are situations and operators. Situations may be understood as snapshots of the domain under consideration. They are described by sets of formulae. Operators are representing actions in the domain. They formally describe the way in which one situation may be transformed into another one. Classical approaches to planning like STRIPS ([FN71], [Lif87]), NONLIN ([Tat77]), NOAH ([Sac77]), and TWEAK ([Cha87]) adopted this view. These approaches have some common characteristics. One leading principle is that operators are characterized by preconditions and postconditions. Time is implicitly given by the (partial) ordering of situations.

There are only a few efforts to classify approaches to planning. [Her93] and [Her94] present one of the first attempts.

According to [Her94] and [BGH$^+$93], there are several categories of non-classical approaches to planning. Those approaches are distinguished from the ground version by abandoning one or the other assumption. The following not mutually exclusive categorization reflects the state of the art by stressing typical issues:

- Another time model ([Ver83], [All83], [All84], [Pel91]),
- incompleteness of information ([MW86], [DW91]),
- computation time restrictions incl. *anytime planning* ([DB88], [RW91]),
- dynamics and unforeseeable effects in plan execution ([Ped87], [GL87], [Dru89], [Fir92]),
- abstraction ([Sac74], [Sac77], [Ten89]),
- distributed reasoning ([BG88], [LB89], [Mar93]),
- refinement by learning ([Ham89], [GD92], [DeJ94]), and
- case-based reasoning ([KH92]).

The pecularities of our target domains introduced above exhibit serious difficulties in specifying postconditions assigned to operators. This key difficulty is rarely considered in the recent literature. There is one relaxation of the standard approach worth to be mentioned: [Her94] takes

non-deterministic effects of actions into account. Although this is a first step towards a higher degree of flexibility, it is still far from the needs of the process class circumscribed in chapter 2 above. The crucial point is that finitely many possibilities of some non-deterministic action effect can easily be decomposed into alternatives being deterministic, again.

A key parameter for classifying planning approaches is the plan concept in use. A careful inspection exhibits that this is a crucial point, indeed.

4.2 Plan Concepts

To the authors' big surprise, as already mentioned initially, about half of the recent publications of the DISKI[7] series (cf. [Gün92], [Wöh92], [Hör92], [Fis93], [Sau93], [Köh93], [Wer93], [Win94], [Köh94], [Thi94],) which deal with planning do *not* contain an explicit plan concept.

In [Her94], there is a quite general formulation[8] characterizing a plan as a structure containing descriptions of actions and goals for reasoning about the effects of future actions and for controlling goal-oriented actions.

On the one hand, there must be some syntax of plans, as these plans have to be generated by some system. On the other hand, those syntactic objects must have some semantics carrying substantial properties important both for reasoning about plans and for plan execution.

In some well-formalized approaches (cf. [Köh94], e.g.), plans are particular formulae. In many approaches which come closer to the needs of complex dynamics processes, plans are somehow formalized as partially ordered structures. This provides, at least potentially, some background to describe concurrent or even simultaneous actions.

Unfortunately, most authors tend to avoid the problems occurring with properly concurring and, perhaps, conflicting actions. In [Pel91], for instance, the problem is eliminated as expressed by the statement: *...if two actions can be executed individually and they do not interfere, then they can be executed together.*. In [Hör92], the author is investigating planning problems in robotics for two-arm systems where properly simultaneous actions may occur. This comes very close to the type of problems we are interested in. However, in [Hör92] on page 116 there is the explicit requirement that, whenever any effective concurrency may occur, one of the actors is paralyzed by definition.

To sum up, the consideration of planning approaches and plan concepts gave rise to some important insights. First of all, we should point out that none of common planning approaches fits all these pecularities. Therefore, we persue two goals. On the one hand, we are interested in developing an appropriate planning approach which meets the considered requirements. On the other hand, we want to exhibit the essentials of planning in domains of the type we are faced to, to clarify the distinction from other more conventional approaches to planning, and to find out pointers to appropriate solutions. Furthermore, the investigation made have shown that we need clear and precise notions and notations on the planning approach and especially on plan concept itself. Only in this way, it would be possible to compare approaches and to answer deeper questions.

[7] DISKI is a publication series of distinguished German Ph.D. theses in artificial intelligence.

[8] *... eine Struktur, welche Repräsentationen von Handlungen und Zielen enthält und dazu dient, über die Wirkung zukünftiger Handlungen zu räsonieren und die zielgerichtete Ausführung von Handlungen zu beeinflussen.*

5 Therapy Plans as Hierarchically Structured Graphs

5.1 Action Scripts

The syntactic constituents of therapy plans are called *action scripts*. Their corresponding semantics are control tasks in the underlying technological equipment. Thus, this semantics is not exclusively formal. As a side effect, the concept of executability has to be complemented by the formal property of consistency.

There are elementary as well as compound action scripts. Only elementary action scripts have a direct operational semantics in the technological equipment. Both elementary and compound action scripts are attached to technological objects of the technology representation (see above). They are distinguished by *preferences* and hierarchically ordered according to the structure of the technology representation. Every action script has some name indicating the goal to be met. It contains certain constraints. The semantics is that, assumed the contraints can be respected, the control task associated to the script will resolve the given problem by meeting the goal. Recall the inheritance principles of action scripts already introduced above:

- Action scripts with the same name and different preferences collected allong a classification path form alternatives for the corresponding object class or instance.
- Actions scripts of the same name attached to a common technological object must be distinguished by preference.
- Action scripts of the same name and the same preference collected along a classification path through the technology representation form specializations.

Elementary action scripts are of the following form

(CREATE-ACTION	
:ACTION	⟨ActionName⟩ ::= ⟨SYMBOL⟩
:OBJECT	⟨ObjectName⟩ ::= ⟨SYMBOL⟩
:PREFERENCE	⟨INTEGER⟩
:START-CONDITIONS	([⟨Constraint⟩]*)
:INTERVAL-CONDITIONS	([⟨Constraint⟩]*)
:PROCEDURE	(⟨ProcedureCall⟩⟨ObjectName⟩⟨ProcedureName⟩⟨ProcedureArgs⟩))

where the semantics of "PROCEDURE" is currently implemented by activating some rule base. Further operational interpretations may be incorporated.

As action scripts may be attached to classes of technological objects in the technology representation, the corresponding operational semantics may be only partially defined here. As a consequence, there may be no explicit execution time assigned to the elementary action script. Instead, during plan synthesis, when action scripts attached to classes are instantiated, the related procedures have a determined maximal duration time which can be propagated and used for constraint monitoring, for instance.

Compound action scripts are used to recursively define nets of elementary actions. They do not have any immediate operational semantics.

(CREATE-ACTION

:ACTION	⟨ActionName⟩ ::= ⟨SYMBOL⟩
:OBJECT	⟨ObjectName⟩ ::= ⟨SYMBOL⟩
:PREFERENCE	⟨INTEGER⟩
:SUBACTIONS	([(⟨SubActionName⟩⟨ObjectReference⟩⟨IntervalIdentifier⟩)]$^+$)
:TIME-RELATIONS	([(⟨TimeRel⟩⟨IntervalIdentifier1⟩⟨IntervalIdentifier2⟩)]*)
:START-CONDITIONS	([⟨Constraint⟩]*)
:INTERVAL-CONDITIONS	([⟨Constraint⟩]*))

where the collection of pairwise orderings introduced by the slot "TIME-RELATIONS" defines a partial ordering on the set of interval identifiers listed in the slot "SUBACTIONS", i.e. interval identifiers to be (partially) ordered need to be introduced before.

There is no need to introduce further interpretations. For understanding the program synthesis approach below, which allows to generate therapy plans in combining action scripts, both the syntax above and the knowledge that only elementary action scripts are carrying operational semantics will do.

5.2 Therapy Plans

Plans are introduced as particular graphs (cf. [AJ94c]). Hence, therapy plan synthesis resp. program synthesis is synthesis of certain hierarchically structured graphs. There has been introduced a hierarchy of formal concepts denoted

- a hierarchically structured family of plans,
- a rooted family,
- a hierarchically structured plan,
- a plan.

These are the key concepts:

Definition 1
A *hierarchically structured family of plans* is a finite collection $\mathcal{F} = \{\mathcal{G}_1, \ldots, \mathcal{G}_k\}$ of pin graphs $\mathcal{G}_i = [V_i, E_i, P_i^{in}, P_i^{out}, C_i, sub_i]$ $(i = 1 \ldots k)$ such that for every $i \in \{1, \ldots, k\}$ we have:

1. $\mathcal{G}_i' = [V_i, E_i]$ is a finite, directed, acyclic graph with the set of vertices V_i and the set of edges E_i.

2. $P_i^{in} \cup P_i^{out}$ are called the pins of $\mathcal{G}_i$ with $P_i^{in}, P_i^{out} \subseteq V_i$ and are defined by

   (a) $P_i^{in} = \{\, v \mid v \in V_i \wedge \neg\exists u \in V_i\,(\,(u, v) \in E_i\,)\,\}$

   (b) $P_i^{out} = \{\, v \mid v \in V_i \wedge \neg\exists u \in V_i\,(\,(v, u) \in E_i\,)\,\}$

3. The vertices in $C_i \subseteq V_i$ are understood compound to be substituted later.

4. $sub_i : C_i \to 2^{\{1,\ldots,k\}} \setminus \emptyset$ is a mapping indicating which Graphs $\mathcal{G}_j$ may be substituted for the compound nodes in C_i.

For every hierarchically structured family of plans $\mathcal{F}$, there is a substitution ordering $\preceq_{\mathcal{F}}$ on $\{\mathcal{G}_1, \ldots, \mathcal{G}_k\}$ indicating which graphs can be substituted into each other according to $\{sub_i\}_{i=1\ldots k}$. This ordering is simply defined such that $\mathcal{G}_i \preceq_{\mathcal{F}} \mathcal{G}_j$ holds, if and only if $\exists c \in C_i\,(\, j \in sub_i(c)\,)$. The transitive closure of $\preceq_{\mathcal{F}}$ is denoted by $\preceq^+_{\mathcal{F}}$.

Definition 2
Assume any hierarchically structured family of plans $\mathcal{F} = \{\mathcal{G}_1, \ldots, \mathcal{G}_k\}$, any $\mathcal{G}_i \in \mathcal{F}$, any $c \in C_i$, and any $j \in sub_i(c)$. The *substitution of* $\mathcal{G}_j$ *in* $\mathcal{G}_i$ at $c \in C_i$ yields another pin graph denoted by $\mathcal{G}_i[c \hookleftarrow \mathcal{G}_j] = [V, E, P^{in}, P^{out}, C, sub]$ and defined as follows.

1. $V = (\, V_i \setminus \{c\}\,) \cup V_j$ (This is always assumed to be a disjoint union.)
2. $E = (\, (E_i \cup E_j) \setminus (V_i \times \{c\} \cup \{c\} \times V_i)\,) \cup ((\{v \mid (v,c) \in E_i\} \times P^{in}_j) \cup (P^{out}_j \times \{v \mid (c,v) \in E_i\}))$
3. $P^{in} = P^{in}_i$
4. $P^{out} = P^{out}_i$
5. $C = (\, C_i \setminus \{c\}\,) \cup C_j$
6. $sub = sub_{i/C_i \setminus \{c\}} \cup sub_j$

This definition easily generalizes to host graphs (called $\mathcal{G}_i$ above) which do not belong to $\mathcal{F}$.

Substitution as above determines some rewrite relation $\Rightarrow_{\mathcal{F}}$ among pin graphs. For any two pin graphs $\mathcal{G}$ and $\mathcal{G}'$, one writes $\mathcal{G} \Rightarrow_{\mathcal{F}} \mathcal{G}'$ if and only if there exists some $\mathcal{G}_j \in \mathcal{F}$ with $\mathcal{G}' = \mathcal{G}[c \hookleftarrow \mathcal{G}_j]$. By $\Rightarrow^+_{\mathcal{F}}$ we denote the transitive closure of $\Rightarrow_{\mathcal{F}}$.

Definition 3
Assume any hierarchically structured family of plans $\mathcal{F} = \{\mathcal{G}_1, \ldots, \mathcal{G}_k\}$ and any $\mathcal{G}_i \in \mathcal{F}$.

1. All normal forms of $\mathcal{G}_i$ w.r.t. $\Rightarrow^+_{\mathcal{F}}$ are called the (flat) *plans* specified by $\mathcal{G}_i$ via $\mathcal{F}$.
2. If any $\mathcal{G}$ is in normal form w.r.t. $\mathcal{F}$, this is denoted by $\mathcal{G}\downarrow_{\mathcal{F}}$.
3. $\mathcal{R} = [\mathcal{F}, \mathcal{G}_i]$ is called a *rooted family*, if and only if $\forall \mathcal{G}_j \in \mathcal{F}\,(\, \mathcal{G}_i \neq \mathcal{G}_j \;\Rightarrow\; \mathcal{G}_i \preceq^+_{\mathcal{F}} \mathcal{G}_j\,)$.
4. Any rooted family $\mathcal{P} = [\mathcal{F}, \mathcal{G}_i]$ is called a *hierarchically structured plan*, if and only if $\mathcal{G}_i$ has a uniquely defined normal form $\mathcal{G}^*$ w.r.t. $\Rightarrow^+_{\mathcal{F}}$.
5. $\mathcal{L}(\mathcal{F}) \;=\; \{\, \mathcal{G} \mid \exists \mathcal{G}_i \in \mathcal{F}(\mathcal{G}_i \Rightarrow^+_{\mathcal{F}} \mathcal{G}) \wedge \mathcal{G}\downarrow_{\mathcal{F}}\,\}$

Obviously, if and only if the relation $\preceq^+_{\mathcal{F}}$ is reflexive, the resulting language of plans $\mathcal{L}(\mathcal{F})$ is infinite. There are certain application domains where even infinite spaces of potential plans are of interest. In those cases, theoretic problems related to the classical theory of formal string languages are becoming especially important.

The concepts introduced here are essential for a formal approach to plan synthesis, to plan execution, and to plan revision, if necessary.

6 Knowledge–Based Plan Synthesis

Obviously, the therapy plan generator based on the graph-theoretic concepts introduced needs to interact with all the knowledge sources available. Its embedding into the overall structure of the knowledge-based system has already been displayed in chapter 2. This is repeated here to compare it to the subsequently following refinements.

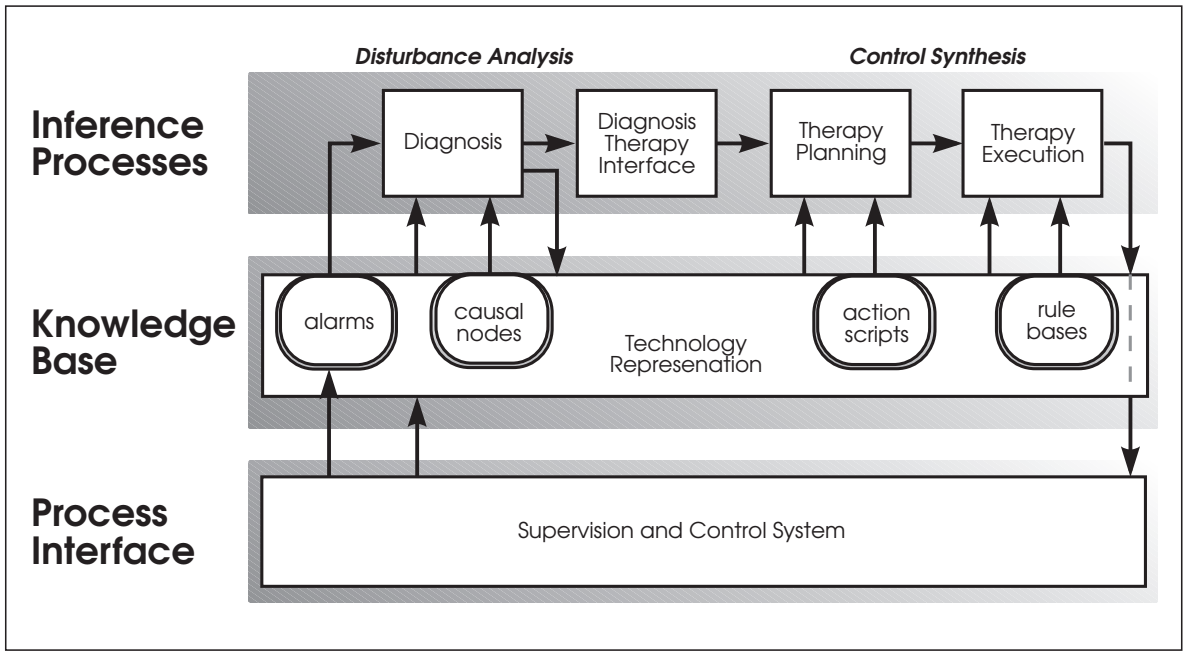

Figure 2: System Architecture

After repeating a few remarks on the overall architecture, we will focus on therapy plan synthesis.

6.1 System Architecture

Therapy plan synthesis is triggered by disturbance analysis. Diagnosis is yielding a classification of some malfunction discovered. On the one hand, this knowledge is added to the technology representation to reflect the current state of the system. On the other hand, the output of diagnosis is interpreted by the diagnosis therapy interface as some therapy control goal related to a particular object of the technology representation. Since every possible goal have to name some action which is able to resolve the specified problem, the therapy planner selects an (compound) action script on the basis of both some knowledge base briefly denoted by $\mathcal{TR}_n$ and the pair of the current action name and the current object name. Then, planning proceedes recursively as outlined in the sequel. The result is either a therapy plan consistent with $\mathcal{TR}_n$ to be executed in the technological equipment given or a statement about the unsolvability of the problem on hand. As consistency of plans does not imply executability, a failing of plan execution may trigger planning again. Based on the general architecture depicted, the following discussion will focus on planning, exclusively.

6.2 Therapy Plan Synthesis

The module of action scripts, which are all related to particular technological objects of the underlying technology representation, specifies the search space of all potentially available therapy plans. It is forming a structured family of plans $\mathcal{F}$. In the case of any disturbance, the diagnosis yields some result interpreted by the diagnosis therapy interface as some therapy control goal. A given goal specification may be understood as a pin graph $\mathcal{G}_i$ which has only one compound node $\mid V_i \mid = \mid C_i \mid = 1$. The substitution mapping sub_i indicates all action scripts $\mathcal{G}_{\mid}$ are available to reach that goal. $\mathcal{F}$ and $\mathcal{G}_i$ are implicitly forming the rooted family $\mathcal{R} = [\mathcal{F}', \mathcal{G}_i]$, where $\mathcal{F}'$ results from $\mathcal{F}$ by removing unnecessary action scripts. $\mathcal{R} = [\mathcal{F}', \mathcal{G}_i]$ is taken as the recent search space for planning.

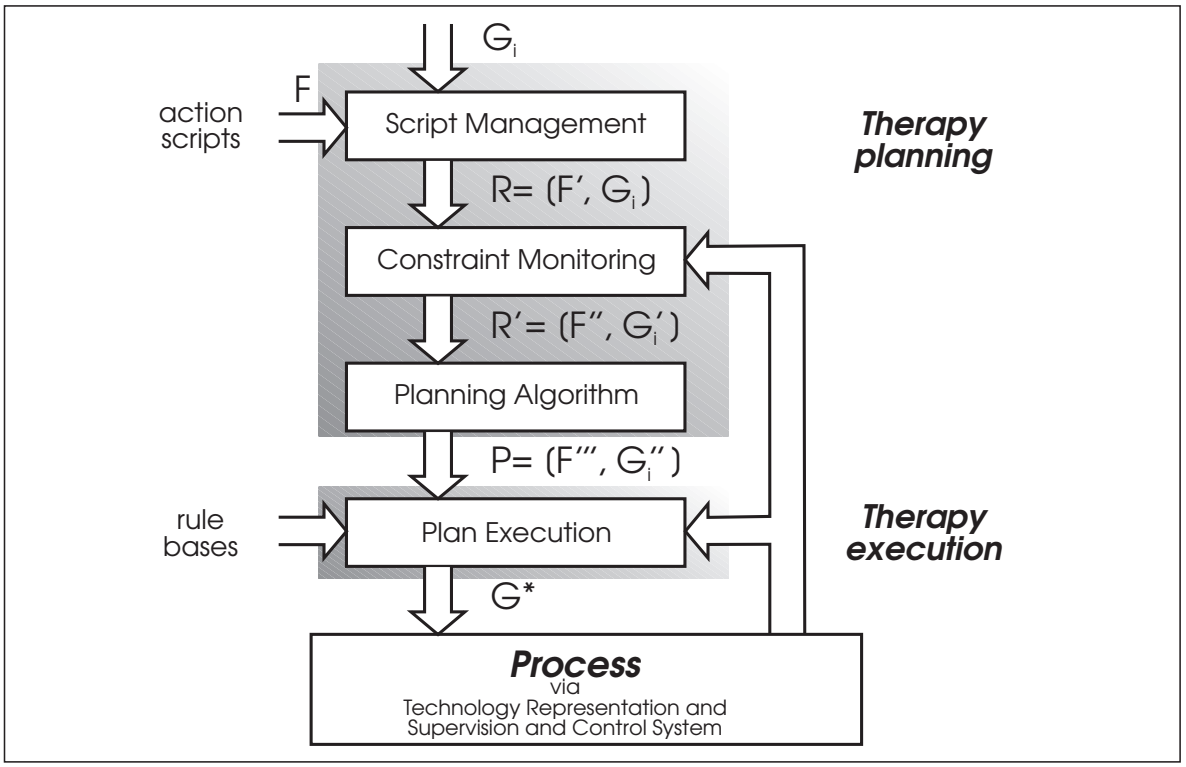

Figure 3: Therapy Planning

Here, we are going to describe the planning algorithm, i.e. the program synthesis algorithm, in some detail. Basic procedures are discussed in [Arn92b] much more intensively. These basic constituents of the therapy control synthesis software, i.e. the program synthesis mechanism, under consideration are

- a procedure called EXPAND being the main program,
- a procedure for predictive constraint monitoring,
- a procedure for action script refinement.

We confine our explanation here to the key ideas. The following steps are describing the essentials of the therapy plan synthesis process itself. Implementation details are suppressed.

The refinement of action scripts means substitutions formally expressed by $\mathcal{G}' = \mathcal{G}[c \hookleftarrow \mathcal{G}_j]$ above. In [DW91], this is called *task reduction*.

In the sequel, we will put some emphasis on the use of constraints and the difficulties of constraint monitoring. These aspects are basic to understand the differences between the central concepts of *consistency* and *executability*. In our opinion, this is an interesting contribution of application-oriented program synthesis as developed here to theoretical investigations in inductive learning.

Action scripts as formalized above contain constraints of different type. The distinction into *start conditions* and *interval conditions* is essential for constraint monitoring during plan execution. The validity of start conditions has to be checked initially, whereas the validity of interval conditions is monitored permanently during execution. For synthesizing plans, this difference does not matter. Also, for our general approach to the synthesis of therapy plans, the particular formal language for formalizing constraints is less important.

Exclusively elementary action scripts have an operational semantics. Thus, constraints are originally assigned only to elementary action scripts. During plan synthesis, it is highly important to check constraints as early as possible. Therefore, constraints which have to be met by certain elementary action scripts simultaneously are lifted to compound action scripts in which they occur together. The same applies to compound action scripts. Practically, this offers the opportunity to free lower order action scripts from constraints assigned to higher order action scripts by assuming inheritance. Theoretically, we assume that the sets of constraints assigned to vertices grow monotonically during task reduction, i.e. graph substitution.

Definition 4

Assume any hierarchically structured family of plans $\mathcal{F} = \{\mathcal{G}_1, \ldots, \mathcal{G}_k\}$.

1. There is assumed some language $\mathcal{WFF}$ of well-formed formulae.
2. SC and IC are two mappings $SC, IC : \mathcal{F} \rightarrow 2^{\mathcal{WFF}}$ assigning to each graph $\mathcal{G}_i$ both a finite set of *start conditions* $SC(\mathcal{G}_i)$ and a finite set of *interval conditions* $IC(\mathcal{G}_i)$.
3. Constraints are inherited as follows.
   (a) $\forall i, j \in \{1, \ldots, k\}\, \forall c \in C_i\ (\ j \in sub_i(c) \Rightarrow\ IC_i \subseteq IC_j\ )$
   (b) $\forall i, j \in \{1, \ldots, k\}\, \forall c \in C_i \cap P_i^{in}\ (\ j \in sub_i(c) \Rightarrow\ SC_i \subseteq SC_j\ )$
4. $SC(\mathcal{G}_i)$ and $IC(\mathcal{G}_i)$ are briefly denoted by SC_i and IC_i, respectively.

Constraints occuring in action scripts (see above) are formulae where the occurring free variables have particular instantiations in the technology representation $\mathcal{TR}$. More precisely, variables are representing certain process parameters. Values of process parameters are stored in $\mathcal{TR}$ together with certain time information specifying its validity as already discussed in some detail above. Thus, one may have values of parameters referring to the past, the present, and the future. For readability, we provide a particularly simple formalization. Among others, variants taking open and half-open intervals into account are just ignored here. It remains an area of open problems

to investigate the impact of the underlying logic of constraints on the planning approach.

Definition 5
Assume any formula φ containing the free variables $x_1, \ldots, x_n$. Assume t_1 and t_2 denote any points in time satisfying $t_1 \leq t_2$.

1. If x is any process parameter and v_1, v_2 are two of its possible values, the technology representation $\mathcal{TR}$ may contain some information indicating that during $[t_1, t_2]$ the value of x ranges between v_1 and v_2. This is written as $\langle x \in_{[t_1,t_2]} [v_1, v_2] \rangle \in \mathcal{TR}$, for short. (Note that in some cases, t_1 and t_2 or v_1 and v_2 (or even both pairs) coincide.)

2. If $v_1, \ldots, v_n$ are particular values of the corresponding variables $x_1, \ldots, x_n$, the notation $\models \varphi(v_1, \ldots, v_n)$ means that this formula is logically valid within the underlying arithmetics. $\models \neg\varphi(v_1, \ldots, v_n)$[9] is understood accordingly.

3. φ is refutable w.r.t. the technology representation during the time period $[t_1, t_2]$, if and only if there are entries $\langle x_1 \in_{[t_{11},t_{12}]} [v_{11}, v_{12}] \rangle, \ldots, \langle x_n \in_{[t_{n1},t_{n2}]} [v_{n1}, v_{n2}] \rangle \in \mathcal{TR}$ such that

   (a) $\forall k \in \{1, \ldots, n\}$ ($t_{k1} \leq t_1 \leq t_2 \leq t_{k2}$)

   (b) $\forall k \in \{1, \ldots, n\}\, \forall v_k \in [v_{k1}, v_{k2}]$ ($\models \neg\varphi(v_1, \ldots, v_n)$)

4. The refutability of φ during $[t_1, t_2]$ w.r.t. $\mathcal{TR}$ is abbreviated by $\mathcal{TR} \models_{[t_1,t_2]} \neg\varphi$.

Note, that we have defined only some of the basic concepts. *Validity* and *satisfiability* are less important here. For the type of constraint monitoring invoked, *refutability* is the key concept. Some constraint is refutable within some time interval as defined above, exactly if there is evidence in the technology representation that for all parameters there must be values violating this constraint throughout the whole time interval considered (see below). In other words, there is no hope to satisfy this constraint.

The reader should note that missing information is increasing the probability *not* to refute constraints.

Constraint checking is quite difficult, as only procedures do admit an *execution time*. The maximal time duration assigned to some procedure for execution is called its *time out*. If during planning there has been substituted some elementary action script into a certain plan under development, this elementary action script is calling a uniquely determined rule base. To this rule base, there is attached some time out.

Definition 6
Assume any hierarchically structured family of plans $\mathcal{F} = \{\mathcal{G}_1, \ldots, \mathcal{G}_k\}$ and any elementary vertex $v \in \bigcup_{i=1,\ldots,k} (V_i \setminus C_i)$.
$to(v) \in \mathbb{N}$ denotes the time out of the procedural semantics of v.

When planning starts, there is estimated the expected time used by the planner. This is taken as a basis for estimating the expected start time t_0 of the therapy plan to be generated. One requires that constraints of action scripts must not be violated within the time interval ranging

[9] Naturally, we assume constraints to be decidable on given data.

from t_0 to the latest start time of the considered action script. This time interval is called the *integral predecessor time out* $\tau(v)$, and it is defined recursively. In other words, when planning uses $\mathcal{TR}_n$ as its basis, it needs statements about values of process parameters during $[t_0, t_0 + \tau(v)]$, for all vertices v inserted during plan generation.

Definition 7
Assume any pin graph $\mathcal{G} = [V, E, P^{in}, P^{out}, C, sub]$.

1. Any $v \in V$ is called *active*, if and only if it holds
   $v \in C \ \wedge \ \neg\exists u \in C \ ((u, v) \in E)$
2. $\tau(v) \ = \ \begin{cases} \max_{(u,v)\in E}\{ \tau(u) + to(u) \} & : \ \exists u \in V \ ((u, v) \in E) \\ 0 & : \ otherwise \end{cases}$

This is providing a sufficient basis for restricting alternative substitutions to only those being *consistent* with respect to the underlying technology representation $\mathcal{TR}_n$. If there is no danger of confusion, we omit the lower index n and write $\mathcal{TR}$ instead of $\mathcal{TR}_n$ to simplify the presentation of formulae.

Definition 8
Assume any pin graph $\mathcal{G} = [V, E, P^{in}, P^{out}, C, sub]$ and any start time $t_0 \in \mathbb{N}$.

1. $sub^{cons}_{t_0}(v) \ = \ \begin{cases} \{ i \mid \ i \in sub(v) \wedge \forall \varphi \in SC_i \cup IC_i \, (\mathcal{TR} \not\models_{[t_0, t_0+\tau(v)]} \neg\varphi) \} & : \ v \ is \ active \\ sub(v) & : \ otherwise \end{cases}$
2. Changing every $sub(v)$ to $sub^{cons}_{t_0}(v)$ within $\mathcal{G}$ transforms the pin graph $\mathcal{G}$ into some pin graph called $\mathcal{G}^{cons}_{t_0}$.

There has to be introduced a rewriting relation ${}_{t_0}\Rightarrow_{\mathcal{F},\mathcal{TR}}$ resulting from $\Rightarrow_{\mathcal{F}}$ by restriction. A rewrite step according to ${}_{t_0}\Rightarrow_{\mathcal{F},\mathcal{TR}}$ is permitted, if and only if the compound vertex c to be replaced is active. In this case, the alternatives for substitution are restricted to $sub^{cons}_{t_0}(c)$ compared to $sub(c)$ before.

Definition 9
Assume any hierarchically structured family of plans $\mathcal{F} = \{\mathcal{G}_1, \ldots, \mathcal{G}_k\}$. Moreover, assume any pin graph $\mathcal{G} = [V, E, P^{in}, P^{out}, C, sub]$ and any start time $t_0 \in \mathbb{N}$.

1. If $\mathcal{G} \ \Rightarrow_{\mathcal{F}} \ \mathcal{G}'$ holds by some substitution of a certain pin graph $\mathcal{G}_i$ for some compound vertex c of $\mathcal{G}$, i.e. $\mathcal{G}' = \mathcal{G}[c \hookleftarrow \mathcal{G}_i]$,
   then $\mathcal{G} \ {}_{t_0}\Rightarrow_{\mathcal{F},\mathcal{TR}} \ \mathcal{G}'$ holds, if and only if
   (a) c is active and
   (b) $i \in sub^{cons}_{t_0}(c)$.
2. As usual, ${}_{t_0}\Rightarrow^{+}_{\mathcal{F},\mathcal{TR}}$ denotes the transitive closure of ${}_{t_0}\Rightarrow_{\mathcal{F},\mathcal{TR}}$.

Note that the efforts to construct only consistent plans may sometimes result in empty substitution mappings, i.e. $sub_{t_0}^{cons}(v) = \emptyset$ which are held for certain compound vertices v of particular pin graphs. Thus, there may exist normal forms with respect to ${}_{t_0}\Rightarrow^{+}_{\mathcal{F},\mathcal{TR}}$ which can not reasonably be interpreted as executable plans. Thus, the target language is defined less smoothly. If a particular graph $\mathcal{G}$ is in normal form w.r.t. ${}_{t_0}\Rightarrow^{+}_{\mathcal{F},\mathcal{TR}}$, this is briefly indicated by $\mathcal{G}_i \downarrow_{\mathcal{TR}}$, where the parameters t_0 and $\mathcal{F}$ are omitted, for readability.

Definition 10
Assume any hierarchically structured family of plans $\mathcal{F} = \{\mathcal{G}_1, \ldots, \mathcal{G}_k\}$. $\mathcal{TR}$ denotes the underlying technology representation, and $t_0 \in I\!N$ is any point in time.

1. $\mathcal{NF}_{t_0\Rightarrow^{+}_{\mathcal{F},\mathcal{TR}}}$ denotes the set of all normal forms of graphs $\mathcal{G}_i \in \mathcal{F}$ with respect to ${}_{t_0}\Rightarrow^{+}_{\mathcal{F},\mathcal{TR}}$, i.e. $\mathcal{NF}_{t_0\Rightarrow^{+}_{\mathcal{F},\mathcal{TR}}} = \{\, \mathcal{G} \mid \exists \mathcal{G}_i \in \mathcal{F}\,(\, \mathcal{G}_i \,{}_{t_0}\Rightarrow^{+}_{\mathcal{F},\mathcal{TR}}\, \mathcal{G}\,) \wedge \mathcal{G} \downarrow_{\mathcal{TR}} \,\}$.
2. $\mathcal{L}_{t_0}^{cons}(\mathcal{F},\mathcal{TR}) \;=\; \mathcal{L}(\mathcal{F}) \cap \mathcal{NF}_{t_0\Rightarrow^{+}_{\mathcal{F},\mathcal{TR}}}$

The rewrite relation ${}_{t_0}\Rightarrow^{+}_{\mathcal{F},\mathcal{TR}}$ is inheriting termination from $\Rightarrow_{\mathcal{F}}$, but it may terminate earlier. This implies immediately $\mathcal{L}_{t_0}^{cons}(\mathcal{F},\mathcal{TR}) \subseteq \mathcal{L}(\mathcal{F})$. The language $\mathcal{L}_{t_0}^{cons}(\mathcal{F},\mathcal{TR})$ may be structurally more complex than $\mathcal{L}(\mathcal{F})$, in general.

Therefore, it is highly desirable to characterize classes of constraints which reduce the structural complexity of $\mathcal{L}_{t_0}^{cons}(\mathcal{F},\mathcal{TR})$, but which are sufficiently expressive for the intended domain of applications. For this investigation, we are interested in all potential sequential orderings of terminal vertices. This is expressed as some sublanguage of $(\bigcup_{l=1\ldots k} V_l \setminus C_l)^*$. For technical reasons, we will need a slightly modified concept of integral predecessor time out.

Definition 11
Assume any hierarchically structured family of plans $\mathcal{F} = \{\mathcal{G}_1, \ldots, \mathcal{G}_k\}$. $\mathcal{TR}$ denotes the underlying technology representation. For any time $t_0 \in I\!N$, there is the language of non-refutable sequences $\mathcal{L}_{t_0}^{nrs}(\mathcal{F},\mathcal{TR})$ defined as follows.
$v_1 \ldots v_r \in \mathcal{L}_{t_0}^{nrs}(\mathcal{F},\mathcal{TR})$ if and only if

1. $\forall \rho \in \{1,\ldots,r\}\,(\, v_\rho \in (\bigcup_{l=1\ldots k} V_l \setminus C_l)\,)$
2. $\tau'(v_\rho) \;=\; \begin{cases} 0 & : \quad \rho = 1 \\ \tau'(v_{\rho-1}) + to(v_{\rho-1}) & : \quad otherwise \end{cases}$
3. $\forall \rho \in \{1,\ldots,r\}\, \forall i \in \{1,\ldots,k\}\,(\, v_\rho \in V_i \;\Rightarrow\; \forall \varphi \in SC_i \cup IC_i\,(\, \mathcal{TR} \not\models_{[t_0, t_0+\tau'(v_\rho)]} \neg\varphi\,)\,)$

In our application domain, we have used as constraints only quite elementary formulae, so far. These formulae permit to describe only comparison of rational process parameters w.r.t. $<$, $\leq$, and $=$, where delay in time may be taken into account.

If some hierarchically structured plan has been generated, it defines its unique normal form $\mathcal{G}^*$. Each node of this normal form has a well-defined operational semantics given as some rule base. For every rule base, there is specified its maximal time required for execution. Based on an a priori estimated time for planning, there is a hypothetical start time t_0 for plan execution. By computing the maximal execution time of all its predecessors, every node gets its latest start

time t. An individual elementary action script $\mathcal{G}_i$ is *acceptable* w.r.t. $\mathcal{TR}$, if and only if for all its start and interval constraints $\varphi \in SC_i \cup IC_i$ (see the structure of action scripts above) $\mathcal{TR} \models_{[t_0,t]} \neg\varphi$ *does not hold.* A plan $\mathcal{G}^*$ is said to be *consistent* w.r.t. $\mathcal{TR}$, if and only if all its vertices are acceptable w.r.t. $\mathcal{TR}$.

This is our basic approach. The approach really chosen is slightly more complex. For every graph substitution (in a leftmost fashion), acceptability w.r.t. $\mathcal{TR}$ is checked. Thus, the concept of consistency is lifted to hierarchically structured plans. This will be made explicit by presenting the algorithm in its basic version.

algorithm: $planner1$
input: $\mathcal{G}_i$, t_0 , $\mathcal{R} = [\mathcal{F}', \mathcal{G}_i]$
output: $\mathcal{P} = [\mathcal{F}''', \mathcal{G}_i]$

1. $\mathcal{G} := \mathcal{G}_i$
2. $\mathcal{P} := [\mathcal{F}''', \mathcal{G}_i]$ with $\mathcal{F}''' := \{\mathcal{G}_i\}$
3. **while** $C \neq \emptyset$ **do**
   - 3.1 $A := \{v \mid v \; active \; in \; \mathcal{G}\}$
   - 3.2 **forall** $v \in A$ **do**
     $sub_{curr}(v) := sub^{cons}_{t_0}(v)$
   - 3.3 **while** $A \neq \emptyset$ **do**
     - 3.3.1 **find** $v \in A$
     - 3.3.2 **repeat**
       - i $j := \max sub_{curr}(v)$
       - ii $sub_{curr}(v) := sub_{curr}(v) \setminus \{j\}$
       - iii $\mathcal{F}'_{part} := \{\mathcal{G} \mid \mathcal{G}_i \preceq^+_{\mathcal{F}'} \mathcal{G}\}$
       - iv $\mathcal{P}_{part} = [\mathcal{F}'''_{part}, \mathcal{G}_i] := planner1(\, \mathcal{G}_j, \tau(v), [\mathcal{F}'_{part}, \mathcal{G}_i]\,)$

       until $sub_{curr}(v) = \emptyset \vee \mathcal{P}'''_{part} \neq \emptyset$
     - 3.3.3 **if** $\mathcal{P}_{part} \neq \emptyset$
       then
       - i **find** $\mathcal{G}_{part}$ with $\mathcal{G}_i \Rightarrow^+_{\mathcal{F}'''} \mathcal{G}_{part} \wedge \mathcal{G}_{part}\downarrow$
       - ii $\mathcal{G} := \mathcal{G}[v \hookleftarrow \mathcal{G}_{part}]$
       - iii $\mathcal{F}''' := \mathcal{F}''' \cup \mathcal{F}'''_{part}$
       - iv $\mathcal{P} := [\mathcal{F}''', \mathcal{G}_i]$

       else
       - i $\mathcal{P} := [\mathcal{F}''', \mathcal{G}_i]$ with $\mathcal{F}''' = \emptyset$
4. **return** $\mathcal{P}$

Figure 4: The Basic Planning Algorithm

The plan $\mathcal{G}$ is a program operating on the underlying dynamic process for therapy.

From the viewpoint of the unavoidable incompleteness of information about the underlying process, we are faced to a problem of inductively generating or learning plans.

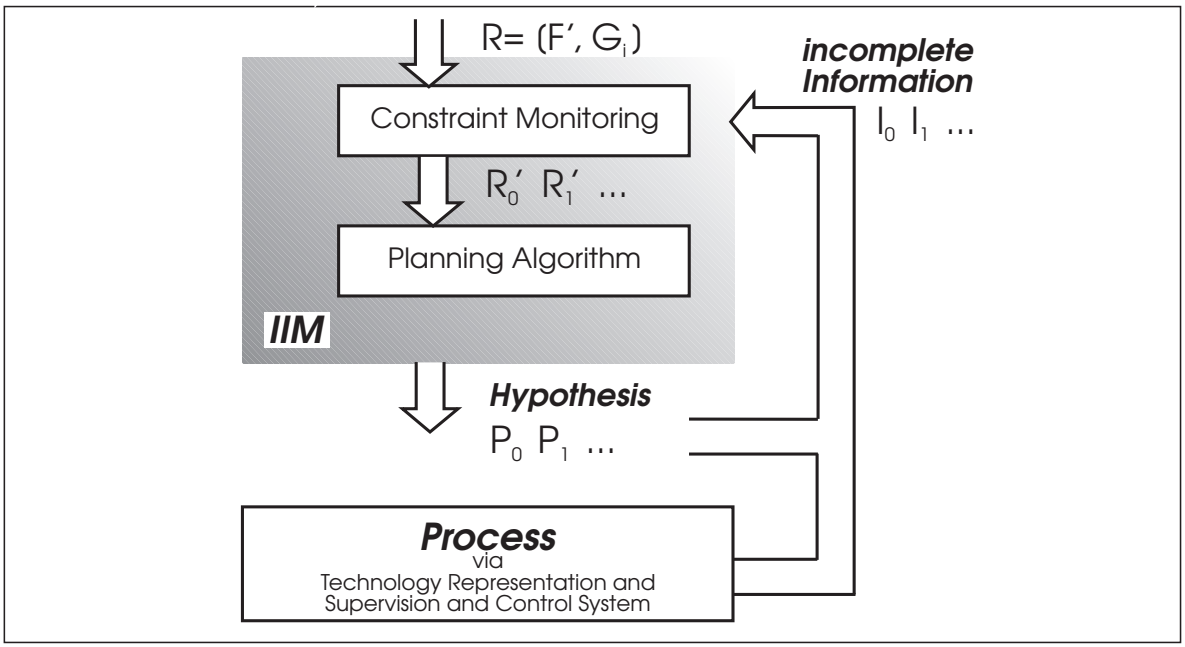

Figure 5: Inductive Inference

The figure 5 is extracting the inductive inference task from our therapy plan synthesis approach. In fact, the core part of our control synthesis system is acting as an inductive inference machine (cf. [AS83], [KW80], for instance) getting fed in some information about the underlying process and generating plans resp. programs as hypotheses. Replanning is working iteratively by stepwise modification of plans within the given rooted family.

6.3 Implementation

The sketched planning algorithm is implemented in LISP and running under SUN/OS on Sparc Stations. Technological object classes, instances, and action scripts are represented as CLOS-objects. Thus, all information necessary for planning is available by accessing slots of those objects.

On its top level, the planning algorithm is starting by a function "make-plan" taking some given goal specification $\mathcal{G}_i$ as argument. This function is using the procedures described. It returns a graph object as the hierarchically structured plan which satisfies all constraints for which there have been data available in the technology representation. In other words, it is returning a consistent hierarchically structured plan.

Additionally, there has been implemented an interface (a plan tracer, in a sense) to the therapy planner which allows to display the generated hierarchically structured plan in some modes. For example, one may list the set $\mathcal{F}'''$ (see above). One may also display all performed substitutions $\mathcal{G}' = \mathcal{G}[c \hookleftarrow \mathcal{G}_j]$ starting at the goal specification $\mathcal{G}_i$. Double

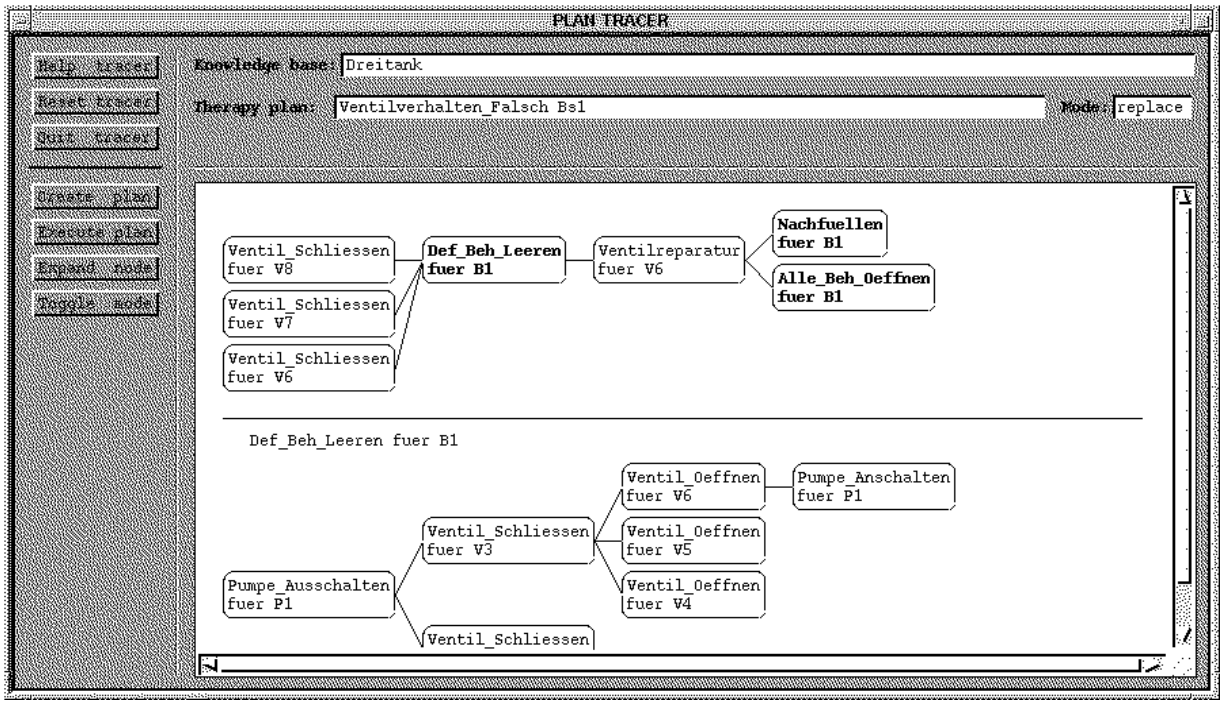

Figure 6: Some Therapy Plan within the Plan Tracer

clicking any compound vertex results in automatically expanding this node according to the hypothetical substitution proposed by the planner.

The figure 6 is illustrating a snapshot taken from a plan tracer session.

Finally, we have de facto some interpreter of plans. It is interpreting the normal form $\mathcal{G}$ of the generated hypothesis $\mathcal{P} = [\mathcal{F}''', \mathcal{G}_i]$ by calling rule bases and interacting with the supervision and control system.

6.4 Applications

The research presented is related to the joint project WISCON on the *Development of Methods for Intelligent Monitoring and Control* supported by the German Federal Ministry for Research and Technology (BMFT). Our implementation builds the therapy control synthesis component of the WISCON system under development.

There are currently three basic applications. One application is *flood prevention* used at the river Werra. Another application is a *ballast tank system* for stabilizing off-shore plants. Third, there is a *chemical equipment* for the production of chemical fibres based on polyethylene terephthalate (a saturated polyester). These three application are distinguished from each other by a number of interesting pecularities. They have had a certain impact on the ideas developed above.

7 Conclusions

We have described both the theoretical foundations as well as the implementation of a particular therapy planning algorithm. Since the plans generated are used for execution in dynamic environments, the present approach may be understood as an approach to program synthesis. The novelty of the present paper is to bridge the gap from the considered area of building knowledge-based systems to learning theory and vice versa. We have made explicit that certain approaches to planning are doing inductive program synthesis, indeed. But this type of program synthesis is considerably different from most of the former approaches referred to, as it essentially depends on executability considerations. Executability and consistency do not imply each other.

The inductive inference view at therapy plan synthesis has exhibited some substantial problems, which should be considered in future. Here, we are going to discuss at least one issue in more detail: Our approach as outlined above works *iteratively*, as further plans are derived from the version before according to information about failing attempts of execution. So far, this seemed natural from the view point of planning. But inductive inference research bears abundant evidence of the restrictedness of iterative learning. ([JB81] and others contain a couple of formal results, in this respect.) Consequently, it seems worth to think about a more general and, perhaps, explicitly non-incremental approach to therapy plan synthesis. From learning theory, it is well-known that a couple of results depend essentially on both the problem area and the particular space of hypotheses taken into account. Hence, it may also happen that in the application domain under consideration, iterative learning is sufficiently powerful. Unfortunately, there is no explicit knowledge at all whether or not this is really the case.

Another area of questions relates to the efficiency resp. complexity of plan generation. Important variants of the basic algorithms result from narrowing nondeterminism. Statement 3.3.1 of figure 4 is a crucial candidate. There are lots of heuristics for appropriately choosing $v \in A$. The reader should recall that refuting plan hypotheses is the only deductive problem which can be successfully attacked, in general. Thus, if there is only a small number of alternative substitutions, this increases the probability of refuting parts of the search space as early as possible. We are finally presenting a corresponding version of our basic algorithm to illustrate further directions of research.

The knowledge-based systems approach to inductive learning, especially to inductive program synthesis, results in a closer look at properties deemed important in inductive inference. For example, the standard property of *consistency* needs to be complemented by the *executability* concept, at least in application areas like complex dynamic processes. Besides introducing new concepts, this suggests certain architectures of learning algorithms. Note that in [Jan92] we have found a similar idea of AI motivated architectures for inductive inference.

algorithm: $planner3$
input: $\mathcal{G}_i$, t_0 , $\mathcal{R} = [\mathcal{F}', \mathcal{G}_i]$
output: $\mathcal{P} = [\mathcal{F}''', \mathcal{G}_i]$

1. $\mathcal{G} := \mathcal{G}_i$
2. $\mathcal{P} := [\mathcal{F}''', \mathcal{G}_i]$ with $\mathcal{F}''' := \{\mathcal{G}_i\}$
3. **while** $C \neq \emptyset$ **do**
   - 3.1 $A := \{v \mid v \; active \; in \; \mathcal{G}\}$
   - 3.2 **forall** $v \in A$ **do**
     $sub_{curr}(v) := sub^{cons}_{t_0}(v)$
   - 3.3 **while** $A \neq \emptyset$ **do**
     - 3.3.1 **find** $v \in A$ such that $sub_{curr}(v)$ is minimal
     - 3.3.2 **repeat**
       - i $j := \max sub_{curr}(v)$
       - ii $sub_{curr}(v) := sub_{curr}(v) \setminus \{j\}$
       - iii $\mathcal{F}'_{part} := \{\mathcal{G} \mid \mathcal{G}_i \preceq^+_{\mathcal{F}'} \mathcal{G}\}$
       - iv $\mathcal{P}_{part} = [\mathcal{F}'''_{part}, \mathcal{G}_i] := planner3(\; \mathcal{G}_j, \tau(v), [\mathcal{F}'_{part}, \mathcal{G}_i]\;)$

       until $sub_{curr}(v) = \emptyset \vee \mathcal{P}'''_{part} \neq \emptyset$
     - 3.3.3 **if** $\mathcal{P}_{part} \neq \emptyset$
       then
       - i **find** $\mathcal{G}_{part}$ with $\mathcal{G}_i \Rightarrow^+_{\mathcal{F}'''} \mathcal{G}_{part} \wedge \mathcal{G}_{part}\downarrow$
       - ii $\mathcal{G} := \mathcal{G}[v \hookleftarrow \mathcal{G}_{part}]$
       - iii $\mathcal{F}''' := \mathcal{F}''' \cup \mathcal{F}'''_{part}$
       - iv $\mathcal{P} := [\mathcal{F}''', \mathcal{G}_i]$

       else
       - i $\mathcal{P} := [\mathcal{F}''', \mathcal{G}_i]$ with $\mathcal{F}''' = \emptyset$
4. **return** $\mathcal{P}$

Figure 7: The Planning Algorithm using a Simple Heuristics

References

[AJ94a] Oksana Arnold and Klaus P. Jantke. Grundbegriffe der Planung I. Studie der Forschungsgruppe Algorithmisches Lernen, HTWK Leipzig (FH), FB Informatik, Mathematik & Naturwissenschaften, August 1994. Studie #01/94, Version 1.0.

[AJ94b] Oksana Arnold and Klaus P. Jantke. Therapy plan generation as program synthesis. In Setsuo Arikawa and K.P. Jantke, editors, *Algorithmic Learning Theory, AII'94 and ALT'94*, volume 872 of *LNAI*, pages 40–55. Springer-Verlag, 1994.

[AJ94c] Oksana Arnold and Klaus P. Jantke. Therapy plans as hierarchically structured graphs. WISCON Report 02/94, HTWK Leipzig (FH), Fachbereich IMN, April 1994.

[AJ94d] Oksana Arnold and Klaus P. Jantke. Therapy plans as hierarchically structured graphs. In *Fifth International Workshop on Graph Grammars and their Application to Computer Science, Williamsburg, Virginia, USA*, November 1994.

[All83] James F. Allen. Maintaining knowledge about temporal intervals. *Comm. ACM*, 26(11):832–843, 1983.

[All84] James F. Allen. Towards a general theory of action and time. *Artificial Intelligence*, 23(2):123–154, 1984.

[AMM92] Oksana Arnold, Volker May, and Uwe Metzner. Wissensverarbeitung in dynamischen Prozeßumgebungen – Eine Anforderungsspezifikation. WISCON Report 01/92, HTWK Leipzig (FH), Fachbereich IMN, March 1992.

[Ang92] Dana Angluin. Computational learning theory: Survey and selected bibliography. In *ACM Symposium on Theory of Computing, STOC'92*, pages 351–368. ACM Press, 1992.

[Arn92a] Oksana Arnold. Reaktive Therapieplanung in dynamischen Prozeßumgebungen. WISCON Report 03/92, HTWK Leipzig (FH), Fachbereich IMN, October 1992.

[Arn92b] Oksana Arnold. Wissensverarbeitung in dynamischen Prozeßumgebungen: Reaktive Therapieplanung in dynamischen Prozeßumgebungen. WISCON Report 10/92, Technische Hochschule Leipzig, FB Mathematik & Informatik, October 1992.

[Arn93] Oksana Arnold. Using multiple models for therapy planning in dynamic environments. In *submitted*, 1993.

[Arn94a] Oksana Arnold. An expert system architecture to support process supervision and control. In Eberhard Köhler, editor, *39. Internationales Wissenschaftliches Kolloquium, Band 3*, pages 30–36. Technische Universität Ilmenau, 1994.

[Arn94b] Oksana Arnold. Towards structure and management of knowledge bases for controlling technological equipments. In *submitted*, 1994.

[AS83] Dana Angluin and Carl H. Smith. A survey of inductive inference: Theory and methods. *Computing Surveys*, 15:237–269, 1983.

[Bar79] David R. Barstow. An experiment in knowledge-based automatic programming. *Artificial Intelligence*, 12(1):73–119, 1979.

[Bau79] Michael A. Bauer. Programming by examples. *Artificial Intelligence*, 12(1):1–21, 1979.

[BB93] W. Bibel and A. W. Biermann. Special issue: Automatic programming - foreword of the guest editors. *Journal of Symbolic Computation*, 15(5 & 6):463–465, 1993.

[BBP75] Alan W. Biermann, Richard I. Baum, and Frederick E. Petry. Speeding up the synthesis of programs from traces. *IEEE Transactions on Computers*, 24(2):122–136, 1975.

[BG88] A.H. Bond and L. Gasser. *Readings in Distributed Artificial Intelligence*. Morgan Kaufmann, 1988.

[BGH+93] Susanne Biundo, Andreas Günter, Joachim Hertzberg, Josef Schneeberger, and Wolfgang Tank. Planen und Konfigurieren. In Günther Görz, editor, *Einführung in die künstliche Intelligenz*, pages 767–828. Addison–Wesley, 1993.

[BK76] Alan W. Biermann and Ramachandran Krishnaswamy. Constructing programs from example computations. *IEEE Transactions on Software Engineering*, SE-2(3):141–153, 1976.

[BK86] Alvis Brazma and Efim B. Kinber. Generalized regular expressions - a language for synthesis of programs with branching in loops. *Theoretical Computer Science*, 46:175–195, 1986.

[BKM92] Dietrich Balzer, Volkmar Kirbach, and Volker May. Knowledge based process control. In Hartwig Steusloff and Martin Polke, editors, *Integration of Design, Implementation and Application on Measurement, Automation and Control, Intercama Congress 92*, pages 33–49, München, 1992. Oldenbourg.

[BMHC90] R. Bhatnagar, D. W. Miller, B.K. Hajek, and B. Chandasekaran. DPRL: A language for representation of operation and safety maintenance procedures of nuclear power plants. In *IEA/AIE–90*, pages 593–600. ACM, 1990.

[Cha87] David Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32:333–377, 1987.

[DB88] Thomas Dean and Mark Boddy. An analysis of time-dependent planning. In *Proceedings AAAI-National Conference on Artifical Intelligence, St. Paul, Minnesota, USA, 1988*, pages 49–54. Morgan Kaufmann, 1988.

[DeJ94] F. DeJongGerald. Learning to plan in continuous domains. *Artificial Intelligence*, 65:71–141, 1994.

[DP90] Nachum Dershowitz and Eli Pinchover. Inductive synthesis of equational programs. In *AAAI-90, Proceedings, Eighth National Conference on Artificial Intelligence*, pages 234–239. MIT Press, 1990.

[DR93] Nachum Dershowitz and Uday S. Reddy. Deductive and inductive synthesis of equational programs. *Journal of Symbolic Computation*, 15(5 & 6):467–494, 1993.

[Dru89] M. Drummond. Situated control rules. In *Proc. KR–89*, pages 103–113, 1989.

[DW91] T.L. Dean and M.P. Wellman. *Planning and Control*. Morgan Kaufmann, 1991.

[FD93] P. Flener and Y. Deville. Logic program synthesis from incomplete specifications. *Journal of Symbolic Computation*, 15(5 & 6):775–805, 1993.

[Fir92] R. James Firby. Building symbolic primitives with continuous control routines. In James Hendler, editor, *AIPS92*, pages 62–69. Morgan Kaufmann, 1992.

[Fis93] Klaus Fischer. *Verteiltes und kooperatives Planen in einer flexiblen Fertigungsumgebung*, volume 26 of *DISKI, Dissertationen zur Künstlichen Intelligenz*. infix, 1993.

[FN71] Richard E. Fikes and Nils J. Nilsson. STRIPS: A new approach to theorem proving in problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[Fri93] Gerhard Friedrich. Model-based diagnosis and repair. *AICOM*, 6(3/4):187–206, 1993.

[GD92] J. Gratch and Gerald F. DeJong. A framework of simplifications in learning to plan. In *AIPS92*, pages 78–87. Morgan Kaufmann, 1992.

[GL87] Michael P. Georgeff and Amy L. Lansky. Reactive reasoning and planning. In *Proc. AAAI–87*, pages 677–682, 1987.

[Gol67] E Mark Gold. Language identification in the limit. *Information and Control*, 14:447–474, 1967.

[Gün92] Andreas Günter. *Flexible Kontrolle in Expertensystemen zur Planung und Konfigurierung in technischen Domänen*, volume 3 of *DISKI, Dissertationen zur Künstlichen Intelligenz*. infix, 1992.

[Ham89] K. Hammond. *Case Based Planning. Viewing Planning as a Memory Task*. Academic Press, 1989.

[Her93] Joachim Hertzberg. KI-Handlungsplanung – Woran wir arbeiten, und woran wir arbeiten sollten. In Otthein Herzog, Thomas Christaller, and Dieter Schütt, editors, *Grundlagen und Anwendungen der Künstlichen Intelligenz. 17. Fachtagung für Künstliche Intelligenz (KI'93)*, pages 3–27. Springer-Verlag, 1993.

[Her94] Joachim Hertzberg. Planen von Aktionen und Reaktionen. Forschungsbericht des Lehrstuhls VIII (KI) LS-8 Report 7, Universität Dortmund, Fachbereich Informatik, 1994.

[Hör92] Andreas Hörmann. *Begleitende Montageablaufplanung für ein sensorgestütztes Zweiarm-Manipulatorsystem*, volume 11 of *DISKI, Dissertationen zur Künstlichen Intelligenz*. infix, 1992.

[Jan84] Klaus P. Jantke. The main proof-theoretic problems in inductive inference. In G. Wechsung, editor, *Frege Conference*, pages 321–330. Akademie-Verlag Berlin, 1984.

[Jan89] Klaus P. Jantke. Algorithmic learning from incomplete information: Principles and problems. In J. Dassow and J. Kelemen, editors, *Machines, Languages, and Complexity*, Lecture Notes in Computer Science, pages 188–207. Springer-Verlag, 1989.

[Jan92] Klaus P. Jantke. Case based learning in inductive inference. In *Proc. of the 5th ACM Workshop on Computational Learning Theory, COLT'92, July 27-29, 1992, Pittsburgh, PA, USA*, pages 218–223. ACM Press, 1992.

[JB81] Klaus P. Jantke and Hans-Rainer Beick. Combining postulates of naturalness in inductive inference. *EIK*, 17(8/9):465–484, 1981.

[JK83] Jean-Pierre Jouannaud and Yves Kodratoff. Program synthesis from examples of behavior. In Alan W. Biermann and Gérard Guiho, editors, *Computer Program Synthesis Methodologies*, pages 213–250. D. Reidel Publ. Co., 1983.

[KH92] Subbaro Kambhampati and James A. Hendler. A validation-structure-based theory of plan modification and reuse. *Artificial Intelligence*, 55:193–258, 1992.

[Köh93] Axel Köhne. *Integration von Aktionsplanung und Konfigurierung*, volume 38 of *DISKI, Dissertationen zur Künstlichen Intelligenz*. infix, 1993.

[Köh94] Jana Köhler. *Wiederverwendung von Plänen in deduktiven Planungssystemen*, volume 65 of *DISKI, Dissertationen zur Künstlichen Intelligenz*. infix, 1994.

[KW80] Reinhard Klette and Rolf Wiehagen. Research in the theory of inductive inference by GDR mathematicians - a survey. *Information Sciences*, 22:149–169, 1980.

[LB89] Gerd Lorscheid and Christian Bauer. Verteiltes Planen verteilter Problemlösungen. Arbeitspapiere der GMD 357, Gesellschaft für Mathematik und Datenverarbeitung, 1989.

[Lif87] V. Lifschitz. On the semantics of STRIPS. In Michael P. Georgeff and Amy L. Lansky, editors, *Proc. Workshop Reasoning about Actions and Plans, Timberline, OR, USA, 1986*, pages 1–9. Morgan Kaufmann, 1987.

[Mar93] Frank von Martial. Planen in Multi-Agenten Systemen. In J. Müller, editor, *Verteilte Künstliche Intelligenz: Methoden und Anwendungen*, Reihe Informatik, chapter 4. BI-Wissenschaftsverlag, 1993.

[MH69] John McCarthy and Patrick J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence*, 4:463–502, 1969.

[MW74] Zohar Manna and Richard Waldinger. Knowledge and reasoning in program synthesis. Technical Note 98, Stanford Research Institute, Menlo Park, CA, USA, November 1974.

[MW83] Zohar Manna and Richard Waldinger. Deductive synthesis of the unification algorithm. In Alan W. Biermann and Gérard Guiho, editors, *Computer Program Synthesis Methodologies*, pages 251–307. D. Reidel Publ. Co., 1983.

[MW86] Zohar Manna and Richard Waldinger. A theory of plans. In *Proceedings of the Workshop on Reasoning about Actions and Plans*, pages 11–45. Morgan Kaufmann, 1986.

[Neu92] Gerd Neugebauer. *Pragmatische Programmsynthese*, volume 18 of *DISKI, Dissertationen zur Künstlichen Intelligenz*. infix, 1992.

[Ped87] Edwin P.D. Pednault. Formulating multiagent, dynamic-world problems in the classical planning framework. In Michael P. Georgeff and Amy L. Lansky, editors, *Reasoning about Actions and Plans: Proceedings of the 1986 Workshop*, pages 47–82. Morgan Kaufmann Publishers, Inc., 1987.

[Pel91] Richard N. Pelavin. Planning with simultaneous actions and external events. In James F. Allen, Henry A. Kautz, Richard N. Pelavin, and Josh D. Tenenberg, editors, *Reasoning about Plans*, chapter 3, pages 127–211. Morgan Kaufmann, 1991.

[RW91] S. Russell and E. Wefald. *Do the Right Thing. Studies on Limited Rationality*. MIT Press, 1991.

[Sac74] E.D. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5(2):115–135, 1974.

[Sac77] E.D. Sacerdoti. *A Structure for Plans and Behavior*. Elsevier/North Holland, 1977.

[Sau93] Jürgen Sauer. *Wissensbasiertes Lösen von Ablaufplanungsproblemen durch explizite Heuristiken*, volume 37 of *DISKI, Dissertationen zur Künstlichen Intelligenz*. infix, 1993.

[Sha81] Ehud Y. Shapiro. An algorithm that infers theories from facts. In *Proc. 7th Intern. Joint Conference on Artificial Intelligence, Vancouver, Canada*, pages 446–451, 1981.

[Sha83] Ehud Y. Shapiro. *Algorithmic Program Debugging*. MIT Press, 1983.

[Sol64] R. Solomonoff. A formal theory of inductive inference, Part I and Part II. *Information and Control*, 7:1–22 and 234–254, 1964.

[Sum75] Philip D. Summers. *Program Construction from Examples*. PhD thesis, Yale University, Dept. Comp. Sci., 1975.

[Sum77] Philip D. Summers. A methodology for LISP program construction from examples. *Journal of the ACM*, 24(1):161–175, 1977.

[Tat77] Austin Tate. Generating project networks. In *Proc. International Joint Conference on Artificial Intelligence, IJCAI-77*, pages 888–893. Morgan Kaufmann, 1977.

[Ten89] Josh D. Tenenberg. Inheritance in automated planning. In Ronald J. Brachman, Hector J. Levesque, and Raymond Reiter, editors, *Proceedings of the 1st Intern. Conf. on Knowledge Representation and Reasoning, San Mateo, CA, USA, 1989*, pages 475–485. Morgan Kaufmann Publ. Co., 1989.

[Thi94] Markus A. Thies. *Planbasierte Hilfeverfahren für direkt-manipulative Systeme: Erkennung, Vervollständigung und Visualisierung von Interaktionsplänen*, volume 67 of *DISKI, Dissertationen zur Künstlichen Intelligenz*. infix, 1994.

[Ver83] S.A. Vere. Planning in time: Windows and durations for activities and goals. *IEEE Trans. Pattern Analysis and Machine Intelligence*, (PALMI–5):246–267, 1983.

[Wer93] Gerhard Werling. *Produktorientierte automatische Planung von Prüfoperationen bei der robotergestützten Montage*, volume 46 of *DISKI, Dissertationen zur Künstlichen Intelligenz*. infix, 1993.

[Wie92] Rolf Wiehagen. From inductive inference to algorithmic learning theory. In S. Doshita, K. Furukawa, K.P. Jantke, and T. Nishida, editors, *Proc. 3rd Workshop on Algorithmic Learning Theory, (ALT'92), October 20-22, 1992, Tokyo*, volume 743 of *Lecture Notes in Artificial Intelligence*, pages 13–24. Springer-Verlag, 1992.

[Win94] Andreas Winklhofer. *Zeitrepräsentation und merkmalsgesteuerte Suche zur Terminplanung*, volume 58 of *DISKI, Dissertationen zur Künstlichen Intelligenz*. infix, 1994.

[Wöh92] Günter Wöhlke. *Wissensbasierte Greifplanung für Mehrfinger-Roboterhände*, volume 6 of *DISKI, Dissertationen zur Künstlichen Intelligenz*. infix, 1992.