Abstract theory of planning

Abstract Theory of Planning

by

A.E. Eiben

88/12

June, 1988

COMPUTING SCIENCE NOTES

This is a series of notes of the Computing Science Section of the Department of Mathematics and Computing Science Eindhoven University of Technology.
Since many of these notes are preliminary versions or may be published elsewhere, they have a limited distribution only and are not for review.
Copies of these notes are available from the author or the editor.

Eindhoven University of Technology
Department of Mathematics and Computing Science
P.O. Box 513
5600 MB EINDHOVEN
The Netherlands

editors: prof.dr.M.Rem
prof.dr.K.M.van Hee.

ABSTRACT THEORY OF PLANNING

A. E. Eiben

Eindhoven University of Technology
Department of Mathematics and Computing Science
P. O. Box 513
5600 MB Eindhoven, The Netherlands

June 1988

ABSTRACT

A mathematical theory of planning is worked out in this paper. The terms 'planning problem', 'plan', 'solution of a planning problem' etc. are often used in the fields of Artificial Intelligence, Decision Support Systems and Operation Research. The meaning of these terms, however, is rather intuitive. This article presents a general model of planning that covers a wide range of 'planning problems'. Examples given at the end demonstrate that this framework can also serve as a high level description language of different 'planning type' tasks.

CONTENTS

0. INTRODUCTION

The Eindhoven University of Technology is running a project that investigates interactive planning systems. The goal is to develop a domain independent shell that, filled in with knowledge, becomes a decision support system (DSS) on a certain field. One of our first questions was "What kind of problems should the DSS solve ?", and we realized that 'planning problem' was not a well defined notion. Much of the literature on the subject is restricted to a specific class of problems. Another difficulty is that conceptual and representational issues are seldom kept apart. Constraints originating from the properties of the considered problem class and constraints arisen from the applied representation method are not explicitly distinguished. These shortcomings initiated a study to work out a general concept of describing and solving planning problems.

Investigating 'planning problems' one soon realizes that a large number of tasks can be classified as such. These tasks lie in different fields, are described by different formalism, solved by different methods, but they have something in common. The term 'planning problem' is mostly associated with situations where certain circumstances are given and one needs to reach a desired state. The word 'plan' is used for a collection of activities. The solution of such a problem is a plan that leads from the initial–, to the goal–configuration. Well–known examples of this type are for instance the blocks–world, the job–shop, and the routing problem. The general model that captures these problems is expected to have multiple advantages :

1. In its structure the similarities between different problem classes are more visible, likewise the essential differences are more distinguishable.
2. It provides a formalism for high level task specification. A planning problem can be described in such a way that :

   – "from above" it is clear enough, it is easy to see whether a specification matches the intuitive interpretation of the problem;

   – "from below" it is precise enough, it is possible to justify that an actual representation (matrices, Horn–clauses etc.) fits the specification.
3. Since it is a uniform framework to describe planning problems, it can be the kernel of the DSS shell mentioned above.

The definitions of the first section establish our basic notions concerning planning problems. Algorithmic aspects, representation and solution methods are not considered here. Roughly speaking we model the world, that is we set up a framework to formulate knowledge about ***what*** is to solve.

In § 1.2 there are general concepts discussing ***how*** to solve it. Filling in the frame presented in § 1.1 we obtain a world description and a problem setting. Specifying the entities introduced by § 1.2 leads to a solution method.

In § 1.3 we give a short overview about some questions related to the foregoing. We unfold properties and assumptions which are not always valid in the real world, therefore are not enclosed in the general theory. As we shall see, they might be of decisive importance and formulizing them provides the ability to test them if needed.

In § 2 we describe three planning problems with the formalism developed in the first chapter. This part illustrates the use of the theory and the easiness of representing different types of problems in this uniform framework.

1. THEORY

Let us first display our intuitive view on planning problems. This illuminates our preliminary assumptions, explains the motivations behind the definitions, and it also determines the domain covered by our theory.

We see the world as a process that is, state transitions embedded in the time. Human influence on the world can be carried out by actions that also take place in time. Applying an action to a process yields a new process. Specifying the application conditions and the effects of the possible actions fully determines the planning environment. Plans are simply sets of actions without any further structure, their effect on processes is computed from the effect of their constituents. Within a planning situation a task is given by an initial process and a set of goal processes. A plan is a solution of such a task, if applied to the initial process it leads to the goal set.

Pay attention to the definition of the planning graph; regardless to algorithmic details it indicates our intended methodology : generating a solution by graph search.

1.1 THE PLANNING PROBLEM, OBJECT LEVEL

DEFINITION 1.1.1 A planning universe is an ordered triple : $\langle S,A,T \rangle$, where

S is a set called the set of world states

A is a set called the set of actions

T is an ordered set called the set of time instances

Denoting time segments we shall use the following notational convention :

$tT := \{\tau \in T \mid t \leq \tau\}$

$Tt := \{\tau \in T \mid \tau < t\}$

$T\underline{t} := Tt \cup \{t\}$ for any $t,\tau \in T$.

DEFINITION 1.1.2 A process of a planning universe is a function $f : T' \rightarrow S$, where $T' \subseteq T$. The set of all processes of a planning universe U is denoted by F(U).

Intuitively a process stands for "The Flow of Nature". In our approach the basic entities to represent the world are processes rather then states. In this way time is naturally incorporated, it is easy to handle cases where the state of the world changes without outside actions (see § 2.2). A process f can be seen as a film about the world, a certain state f(t) in the range of f is the snapshot taken at the moment t. From mathematical point of view a process is a parameterization of S by T'. If time does not play a role then constant parameterization (where |rng(f)| = 1) can be used. Assuming that processes with the same range need not to be distinguished, all the processes having the range {s} can be identified with the same state s. It is easy to see that the obtained structure is the equivalent of the state based approach.

A process can be changed by actions that take place in time. The effect of an action is described by a function e, the precise definition of which is presented later. The interpretation of the formula e(f,(a,t)) = g is that applying the action a at the time t the former process f changes and the new process is g.

EXAMPLE 1 FALLING

S = {held, falling, broken}

A = {drop}

$T = \mathbb{R}_0^+$

Let us take a process f, and an effect–function e as follows :

f(t) = held for every $t > 0$ and

$$e(f,(drop,t))(\tau) = \begin{cases} \text{falling} & \text{if } t \le \tau < t + \text{falltime} \\ \text{broken} & \text{if } t + \text{falltime} \le \tau \end{cases} \qquad \text{for any } t > 0,\ \tau \ge t.$$

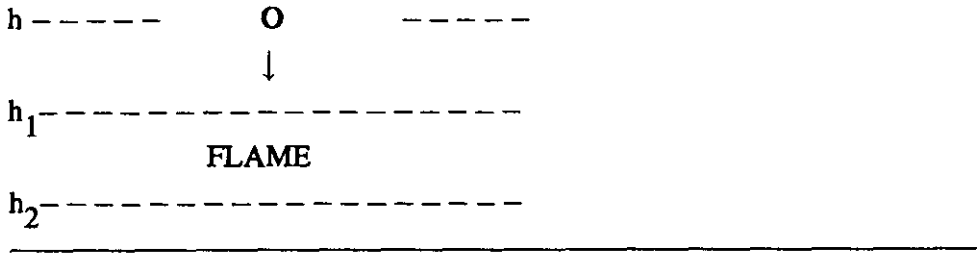
Another effect–function can be :

$$e(f,(drop,t))(\tau) = \begin{cases} \text{held} & \text{if } 0 \le < \tau < t \\ \text{falling} & \text{if } t \le \tau < t + \text{falltime} \\ \text{broken} & \text{if } t + \text{falltime} \le \tau \end{cases}$$

This effect–function, unlike the previous one preserves the information about the "past".

Such a description resembles the so called qualitative or common sense physics and indeed, the case of the ball falling through a flame [Forbus, 84] can be described easily.

EXAMPLE 2 FALLING & HEATING



Now the environment is built up as follows :

$S = \{ \text{heldat}(h),\ \text{fallingat}(h),\ \text{fallingat}(h) \wedge \text{heated},\ \text{broken} \mid h \in [0,100] \}$

$A = \{\text{drop}\}$

$T = \mathbb{R}_0^+$

Let $h \in (0,100]$ and $f_0(t) = \text{heldat}(h)$ for every $t \in T$ be the initial process. Our knowledge about this world can be represented by the following effect function :

$$e(f_0,(\text{drop},t))(\tau) = \begin{cases} \text{fallingat}(h') & \tau < t + \text{reachflame} \\ \text{fallingat}(h') \wedge \text{heated} & t + \text{reachflame} \le \tau < t + \text{leaveflame} \\ \text{fallingat}(h') & t + \text{leaveflame} \le \tau < t + \text{falltime} \\ \text{broken} & t + \text{falltime} \le \tau \end{cases}$$

where the exact value of h', falltime, reachflame, leaveflame can be calculated from h, h_1, h_2, t by the well-known Newtonian laws of mechanics.

With such a detailed description we can formulate precise conditions (commands for a robot arm) to rescue the falling object, eg. :

$$\text{rescued}_h \Leftrightarrow \text{catchat}(\text{time},\text{pos}) : \text{pos} \in (0,h) \wedge \text{time} = \sqrt{\frac{2(h - \text{pos})}{g}} .$$

DEFINITION 1.1.3 An operation (interpreted as the execution of the action a at the time t) is a pair $(a,t) \in A \times T$, and let time : $A \times T \longrightarrow T$ be the projection function, that is $\text{time}((a,t)) := t$ for any operation.

DEFINITION 1.1.4 The allowability relation of a planning universe U is a relation ALL $\subseteq$ F(U) $\times$ (A$\times$T). It defines two allowability functions :
α : F(U) $\longrightarrow$ $\mathcal{P}$(A$\times$T) , where $\alpha(f) := \{ o \in A\times T \mid (f,o) \in ALL \}$
α' : A$\times$T $\longrightarrow$ $\mathcal{P}$(F(U)), where $\alpha'(o) := \{ f \in F(U) \mid (f,o) \in ALL \}$
The elements of $\alpha(f)$ are called the allowable operations of f.

It is easy to see that any of the three of ALL, α, α' uniquely determines the other two. Throughout this article the function α will be used.
Observe that the implication $(a,t) \in \alpha(f) \Rightarrow t \in dom(f)$ does not necessarily hold. One could require it based on intuition, but it does not follow from the previous definitions, and is not needed for the later properties.

DEFINITION 1.1.5 Let U = <S,A,T> be a planning universe. First we define nil as a distinguished element of F(U), or say that we extend F(U) with nil. Then an effect–function is a function e : F(U) $\times$ (A$\times$T) $\longrightarrow$ F(U) such that :

– nil is an universal absorber (a, b,)

– the past of a process can not be extended by an operation (c,)

– the past of a process can not be changed by an operation (d,)

that is for any $f,g \in F(U)$ and $o = (a,t), (a,t) \in A\times T$

a, $e((nil,o)) = nil$

b, $e((f,o)) = nil$ if $o \notin \alpha(f)$

c, $e(f,o) = g \Rightarrow dom(g) \cap T_{\downarrow t} \subseteq dom(f) \cap T_{\downarrow t}$

d, $e(f,o) = g \Rightarrow f = g$ on $dom(g) \cap T_{\downarrow t}$

Notice that in Example 1 the value of the effect–function was specified only for a subset of its domain. In order to give an exact meaning to such an incomplete definition we introduce a convention. We assume that in the undefined cases the value of the effect function is nil. As a consequence, it is enough to determine the value of e for some $(f,o) \in F(U) \times (A\times T)$. Due to the above convention it can be regarded as the definition of the complete effect–function e.

DEFINITION 1.1.6 A plan is a set of operations. A plan P is called a section iff $\forall\, o_1,o_2 \in P$ $time(o_1) = time(o_2)$. The function time can be extended for non empty sections by time(P) := time(o) , where $o \in P$ is arbitrary.

DEFINITION 1.1.7 $P_1,...,P_n$ are called the slices of the plan P iff

a, $P = \bigcup_{i=1}^{n} P_i$,

b, $\forall i \in \{1,...,n\}$ P_i is a section,

c, $\forall i,j \in \{1,...,n\}$ [$time(P_i) = time(P_j) \Rightarrow i = j$].

LEMMA 1.1.1 The slices of a plan are uniquely determined that is, for any plan P there is one and only one set $\{P_1,...,P_n\}$ satisfying a, b, c of definition 1.1.7.

PROOF It is straightforward, observe that the slices of a plan P are the maximal sections contained in P. ∎

If P has n slices then a numbering of them is a bijection from the set $\{1,...,n\}$ onto the set of slices of P. Obviously, the slices of a plan P can be numbered in many ways. We distinguish one numbering.

DEFINITION 1.1.8 The natural numbering of a plan P is that numbering which satisfies

$\forall i,j \in \{1,..,n\}$ [$i < j \Leftrightarrow time(P_i) < time(P_j)$]

The notation $P = [P_1,..,P_n]$ stands for a plan P with the slices $P_1,..,P_n$ numbered by the natural numbering.

PROPOSITION 1.1.1 For any plan P there is one and only one natural numbering.

PROOF By the lemma 1.1.1 we have that

$\exists !$ set of slices of P, say $\{P_1,...,P_n\}$. This implies that

$\exists ! \{t_1,...,t_n\} \subseteq T : \forall i \in \{1,..,n\}$ [$t_i = time(P_i)$]. Since T is ordered we also have that

$\exists !$ permutation π of $\{1,..,n\}$: $t_{\pi(1)} < ... < t_{\pi(n)}$.

It is easy to see that the bijection $\forall i \in \{1,..,n\}$ $\pi(i) \mapsto P_i$ satisfies the definition of natural numbering. ∎

Next we extend the effect–function from operations to plans. The major problem is to determine the result of equitemporal operations. The effect of n operations can be computed in n! different orders, which may lead to n! different results. If the operations occur at the same time, then intuition does not help us to choose among the different outcomes. Therefore we take each of them, that is a subset of F(U) instead of a single process.

DEFINITION 1.1.9 For a given U = <S,A,T>, α and e the extended effect–function $e' : \mathcal{P}(F(U)) \times \mathcal{P}(A \times T) \longrightarrow \mathcal{P}(F(U))$ is defined inductively :

a, $e'(F,\emptyset) = F$ for any $F \subseteq F(U)$

b, if $f \in F(U)$, $P = \{o_1,..,o_n\}$ is a section then

$$e'(\{f\},P) = \bigcup_{\{i_1,..,i_n\}} \{e(..e(f, o_{i_1})..., o_{i_n})\},$$

where $\{i_1,..,i_n\}$ ranges over the set of all permutations of $\{1,..,n\}$.

c, if $F \subseteq F(U)$, P is a section then $e'(F,P) = \bigcup_{f \in F} e'(\{f\}, P)$

d, if $F \subseteq F(U)$, $P = [P_1,..,P_n]$ then $e'(F,P) = e'(...e'(F, P_1)...,P_n)$

For singletons we shall write e'(f,P) instead of e'({f},P), likewise e'(F,P) = g stands for e'(F,P) = {g}.

Notice that the intuitive interpretation of time is hidden right here in using strictly the natural numbering in point d. This establishes that the effect of the operations with a smaller (earlier) assigned time instance precedes the effect of the ones with a larger (later) time instance.

Due to definition 1.1.5 the past of a process -- related to a time instance t – can not be changed by an operation which takes place at the moment t. The next statement extends the past invariance property for plans in general.

PROPOSITION 1.1.2 (Past Invariance Lemma)

Let $f,g \in F(U)$, $P = [P_1,...,P_n]$ a plan, $t_i = time(P_i)$ for every $i \in \{1,...,n\}$.

If $e'(f,P) = g$ and $g \neq nil$ then $f = g$ on $dom(g) \cap Tt_1$

PROOF

(i) $n = 1$ (P is a section).

Let $P = \{o_1,..,o_k\}$. By point d of definition 1.1.5 we have that

$\forall\, 1 < m \leq k$ $[e(..e(f,o_1)...,o_m) = g \Rightarrow e(..e(f,o_1)...,o_{m-1}) = g$ on $dom(g) \cap Tt_1]$.

Hence $e(f,o_1) = g$ on $dom(g) \cap Tt_1$, thus again by definition 1.1.5 $f = g$ $dom(g) \cap Tt_1$.

(ii) $n > 1$

Applying the assertion of (i) we obtain that

$\forall\, 1 < m \leq n$ $[e(..e(f,P_1)...,P_m) = g \Rightarrow e(..e(f,P_1)...,P_{m-1}) = g$ on $dom(g) \cap Tt_m]$.

This results $e(f,P_1) = g$ on $dom(g) \cap Tt_2$, which implies $f = g$ $dom(g) \cap Tt_1$. ∎

DEFINITION 1.1.10 The triple $<U,\alpha,e>$ of a planning universe, an effect–function and an allowability–function is given a special name, it is called a planning situation.
A planning problem is defined by a planning situation $<U,\alpha,e>$ and a pair $<f,G>$, where $f \in F(U)$ is the initial process, $G \subseteq F(U)$ is the goal set.
The set G is often specified by a predicate over the processes, called the goal predicate. A plan P is a solution of the planning problem iff $e'(f,P) \subseteq G$, (the result of applying the plan P to f satisfies the goal predicate).

Defining a (partial) ordering on the set G the solutions can be compared, thus searching for the (an) optimal solution is reasonable.

1.2 THE PLANNING GRAPH, REASONING LEVEL

As it was formerly said, in § 1.1 we discuss what is the problem to solve, and in § 1.2 we investigate the question of how to solve it. We shall follow the search space paradigm, i.e. that a solution is reached by traversing a so called search space consisting of objects being of the same type as the expected solution. The search space will be defined as general as possible and we will point out what kind of restrictions are needed to specify a certain solution algorithm.

Usually a method to solve a given planning problem proceeds by modifying the plan(s) generated so far. Manipulations as entities changing plans are strictly distinguished from operations acting on processes. In our approach manipulating a plan P (a set of operations) is carried out by computing the symmetric difference of P and another set of operations R. This explains the following definition.

DEFINITION 1.2.1 A manipulation in the planning universe $<S,A,T>$ is a set $R \subseteq A \times T$. The result of applying the manipulation R to the plan P is the plan $P \Delta R = (P \cup R) \setminus (P \cap R)$.

It is easy to see that the usual "add", "delete", "replace", etc. can be obtained by choosing an appropriate set R.

DEFINITION 1.2.2 In the planning graph of a planning universe the nodes belong to plans the edges belong to plan manipulations. The edge labeled by R connects the nodes labeled by P and Q iff $P \Delta R = Q$.

PROPOSITION 1.2.1 For any planning universe the corresponding planning graph is a complete connected graph without multiple joins.

PROOF We need to show that there is exactly one edge between any two nodes that is,
$\forall P,R \subseteq A \times T \ \exists ! \ Q \subseteq A \times T \ : \ P \Delta Q = R$.
The existence is obvious, notice that $Q = P \Delta R$ will do.
To verify that there are no multiple joins we prove that no other Q is good. Let us suppose that $Q \subseteq A \times T$ and $P \Delta Q = R$.
a, First observe that $Q \subseteq P \cup R$.
If $Q \setminus (P \cup R) \neq \emptyset$ then $(P \Delta Q) \setminus (P \cup R) \neq \emptyset$, thus $(P \Delta Q) \setminus R \neq \emptyset$, which implies that $P \Delta Q \neq R$.

b, $\exists x : x \in Q \wedge x \notin P \Delta R$

From $Q \subseteq P \cup R$ and $x \in Q \wedge x \notin P \Delta R$ we can conclude $x \in P \cap Q \cap R$. This implies $x \notin (P \cup Q) \setminus (P \cap Q)$, hence $x \notin R$, that contradicts $x \in P \cap Q \cap R$.

c, $\exists x : x \notin Q \wedge x \in P \Delta R$

c_1 $x \notin Q \wedge x \in R \setminus P \Rightarrow x \notin P \cup Q \Rightarrow x \notin (P \cup Q) \setminus (P \cap Q) \Rightarrow x \notin R$, that contradicts $x \in R \setminus P$

c_2 $x \notin Q \wedge x \in P \setminus R \Rightarrow x \notin P \cap Q \Rightarrow x \in (P \cup Q) \setminus (P \cap Q) \Rightarrow x \in R$, that contradicts $x \in P \setminus R$

From b and c it follows that $Q \setminus (P \Delta R) = \emptyset \wedge (P \Delta R) \setminus Q = \emptyset$, that is $Q = P \Delta R$. ∎

The planning graph embodies the entire search space. The search for a solution can be considered as traversing this graph. It may start at an arbitrary (not necessarily the empty) node and should reach a node which belongs to a solution (if there exists any).
Dealing with such a successive node transition requires the usage of paths, that is composed manipulations.

DEFINITION 1.2.3 Let Q and R be manipulations in a planning universe. The composition of Q and R (designated by $Q \circ R$) is the manipulation $Q \Delta R$.

PROPOSITION 1.2.2
For any plan P and manipulations M_1 and M_2, $P(M_1 \circ M_2) = (P M_1) M_2$ holds.
The manipulations with the composition form an Abelian group, where all the elements are idempotent.
PROOF It is straightforward, it needs only elementary set theoretical properties. ∎

With these definitions the necessary notions of path, descendant nodes, ancestor nodes etc. are rather obvious.

After making a satisfactory description of the planning problem the planner has to decide about the solution method. While the problem formulation is quite determined by the considered circumstances, there is a large flexibility in defining a reasonable search algorithm. Nevertheless, there are some general steps to be performed in every case. In order to specify a search algorithm one needs to tell which part of the graph will be traversed and how will it be traversed. This means :

to prune the search space by

- making a restriction on the nodes of the graph,
- making a restriction on the edges of the graph,

to direct the search by

- specifying a search–strategy (how to move within the graph).

There is much known about search strategies, (see eg. Pearl,84) thus let us have a closer look on the first issue. First of all let us make a trivial restriction about the edges. From now on we consider only those edges that belong to a singleton $\{o\} \subseteq A \times T$. Proposition 1.2.1 guarantees that we loose no power with this.
The restrictions on the considered nodes are often carried out by means of "feasible" plans. How feasibility is defined is strongly task–dependent. It is mostly intended to express some soft constraints, that is requirements set up by the user and not by the features of the problem (hard constraints). Having seen many cases we have experienced that very often the property defining feasible plans is descending, that is subplans of feasible plans are also feasible. Accepting it as a general concept of feasibility we obtain the following definition :

DEFINITION 1.2.4 Let U be a planning universe, $\mathbb{P}$ the set of all plans in U, φ a predicate over $\mathbb{P}$. We say that φ is a <u>feasibility</u> iff φ is not universally false and φ is descending, that is

$\exists P \in \mathbb{P} [\varphi(P)] \wedge \forall P,R \in \mathbb{P} [\varphi(P) \wedge R \subseteq P \Longrightarrow \varphi(R)]$.

A plan P is called <u>feasible</u> (φ–feasible) iff $\varphi(P)$ holds.

The properties discussed in 2.2 illustrate this matter particularly for the job shop problem.
The next proposition shows that restricting the search space to the feasible plans is safe in general.

PROPOSITION 1.2.3 In any planning universe, for any feasibility

- the empty plan is feasible
- any feasible plan (thus also the feasible solutions) can be reached from the empty plan by a feasible path (where all the nodes are feasible).

PROOF It is straightforward from the definition of feasibility. ∎

A general method to restrict the edges taken into account can be based on the idea of executability. We intend to call an edge $\{o\} \subseteq A \times T$ executable w.r.t. a process f, if $e(f,\{o\}) \neq nil$, that is $o \in \alpha(f)$ holds for the operation $o \in A \times T$. Extending a plan P by an operation o and stepping further from the node P by the edge $\{o\}$ is the same thing. The process that can be assigned to a node P is $e'(f_0,P)$, the result of applying P to the initial process. During the search we want to consider only such extensions/steps that are executable w.r.t. the actual $e'(f_0,P)$. This explains the following definition. Observe that feasibility can be defined within the terms of a planning universe <S,A,T>. To have executability we need <S,A,T,α,e> and at least the initial process from the pair $<f_0,G>$.

DEFINITION 1.2.5 Let <S,A,T,α,e> be a planning situation, $f_0 \in F(U)$, and let Γ denote the planning graph of <S,A,T,α,e>. The set E of **executable plans** is defined inductively.

Let $E^{(1)} := \{ \{a\} \mid a \in \alpha(f_0) \}$

(the executable plans containing one operation)

$E^{(n)} := \{ P \cup \{a\} \mid P \in E^{(n-1)}, a \in \alpha(e'(f_0,P)) \}$ $(1 < n)$

(the executable plans containing n operations)

$E := \{\emptyset\} \cup \bigcup_{i=1}^{\infty} E^{(i)}$ (the set of executable plans).

The edges belonging to the elements of the set $\alpha(e'(f_0,P))$ are called the **executable edges w.r.t. the node P in Γ**. The executable edges of the executable nodes make up the set of all **executable edges in Γ**.

Roughly speaking, we can characterize executable plans as those ones which do not transfer f_0 to nil.* Executable edges preserve executability in the sense, that stepping away from an executable node by an executable edge, leads to an executable node. This is why one could require that the search considers only executable edges (manipulations).

A predicate expressing that a plan is executable, ($\varphi(P)$ is true iff $P \in E$) is not descending in general. Still, we can state the analogue of Proposition 1.2.3.

---

* Under the Good Allowability Assumption (see 1.3.2) $P \in E$ and $e'(f_0,P) \neq nil$ are equivalent.

PROPOSITION 1.2.4 Let $<S,A,T,\alpha,e>$, $<f_0,G>$ be a planning problem, the set E as defined in Definition 1.2.5. Then

- the empty plan is executable
- any executable plan can be reached from the empty plan by an executable path (where all the edges are executable).

PROOF It is obvious from the definition of E. ∎

Hereby we have sketched two general principles to reduce the search space. The most likely way of applying them is a combined usage. To prune the search space one defines a feasibility and requires that the search should proceed by always taking executable edges that lead to feasible nodes.

Viewing plans simply as objects which we can modify raises the straightforward question : is the reasoning level of a planning situation a planning situation itself ? The answer is yes. Without the formal assertion one can figure a universe where :

- the states are the plans of the original universe,
- the actions are the plan manipulations,
- the effect function is defined to be conform with the effect of the manipulations.

This observation indicates that the structure $<S, A, T, \alpha, e>$ can be widely recognized / applied.

1.3 ADDITIONAL RESULTS

Section 1.1 supplies the basic definitions to capture planning problems. Further study has led to extensions and sophistications which are not discussed there. This section gives an overview about two of the considered topics. A more extensive survey on the matter is available at the author.

1.3.1 CONCURRENCY

The terms 'concurrency' or 'parallelism' are mostly applied to cases where more activities, events occur at the same time. In our model a process assigns one state to a time instance, all events are enclosed in the state. Looking at the inner side of a state may display that the state constitutes of several pixels which represent parallel activities (see § 2.2). On the general level we do not bother about such details, as far as the world is concerned, there is always one object (a state) present at any time instance. For this reason, we use the word 'concurrency' when there are more actions to apply at the same time.
According to point d of definition 1.1.9 the order to compute the effect of different sections is determined by their time precedence relations. For equitemporal operations we found no general principle to determine the computing order. Due to point b the set of all possible results was taken, hence $|e'(f,P)| > 1$ can occur. This yields ambiguity about the result of applying a plan to a process. The uncertainty can not be excluded in general, but there are (a lot of) examples where we have a much better case. This motivated the following definitions.

- A plan P is called commutative, if the set e'(f,P) in Definition 1.1.9 / b is a singleton. This means that the result of the simultaneous operations is independent from the computation order, that is uniquely determined.
- In certain cases there is no need to know the exact result of a plan. It can be enough that the set e'(f,P) surely satisfies some conditions, i.e. it is surely within a set $H \subseteq F(U)$. In this case we say that P is safe w.r.t. H.

Observe that recognizing these features in a concrete planning situation can make the computation much easier.

1.3.2 TRANSITION BETWEEN PROCESSES

A process can be considered as an elementary object that is transformed into another process by a plan. The definition of the transition relation is self–evident.

DEFINITION 1.3.1 Let $\langle U,\alpha,e\rangle$ be a planning situation. The transition relation $\rightarrow$ on F(U) is defined as follows :
for any $f,g \in F(U)$ $f \rightarrow g$ iff $\exists P \subseteq A \times T : e'(f,P) = g$.

The relation $\rightarrow$ is clearly reflexive since $\forall f \in F(U)\ [e'(f,\emptyset) = f]$ by definition. With some surprise we realized that transitivity did not hold for $\rightarrow$ in general. Investigating the cause of this fact we found quite natural conditions to imply transitivity.

The following properties do not have universal validity in practice and they are independent from the former definitions. Therefore we formulate them as assumptions which need to be tested in each planning situation.

GOOD ALLOWABILITY ASSUMPTION (GAA)
For all $f \in F(U)$ and $o \in A \times T$ $e(f,o) = nil \Leftrightarrow o \notin \alpha(f)$.

The next two assumptions have a common underlying idea. Namely, that being in a world, our knowledge about it is strongly based on the past. From within we can not distinguish two processes which have the same history up to now. With other words, it is only the past and the present that determine a situation, regardless to the future which would have come without our interference. The two assumptions below formally express that
– the set of our possible actions, and
– the effect of the actions
is independent from the future.

DETERMINATIVE PAST ASSUMPTION 1 (DPA 1)
For any $f,g \in F(U)$ and $t \in T$ $f \restriction T_t = g \restriction T_t \Rightarrow \alpha(f) \restriction t = \alpha(g) \restriction t$.

The notation $\alpha(f) \restriction t$ stands for $\{ (a,\tau) \in \alpha(f) \mid \tau = t \}$.

DETERMINATIVE PAST ASSUMPTION 2 (DPA 2)

For any $f,g \in F(U)$ and section $P \neq \emptyset$: time(P) = t $\quad f \restriction T_{\downarrow t} = g \restriction T_{\downarrow t} \implies e'(f,P) = e'(g,P)$.

These assumptions are not fully independent since **GAA & DPA 2** $\implies$ **DPA 1** can be proved. The most interesting statement, however, is the following.

PROPOSITION 1.3.1 (Transitivity)

GAA and DPA 2 imply the transitivity of $\rightarrow$, that is, if GAA and DPA 2 hold then

$\forall f,g,h \in F(U)\ \forall P,Q \subseteq A \times T\ [\ e'(f,P) = g\ \wedge\ e'(g,Q) = h \implies \exists R \subseteq A \times T : e'(f,R) = h\ ]$.

PROOF

Let us take $f,g,h \in F(U)$ and $P,Q \subseteq A \times T$ such that $e'(f,P) = g$ and $e'(g,Q) = h$.

Let $P = [P_1,...,P_n]$, $Q = [Q_1,...,Q_m]$ sliced and numbered by the natural numbering. We define $t_i := time(P_i)$, i = 1,..,n and $\tau_i := time(Q_i)$, i = 1,..,m.

(i) $\quad \tau_1 \leq t_1$

$e'(f,P) = g \implies f = g$ on $dom(g) \cap T_{\downarrow t_1}$ by the past invariance lemma, so $f = g$ on $dom(g) \cap T_{\downarrow \tau_1}$ holds too. Then from DPA 1 we have that $Q \subseteq \alpha(f)$ and DPA 2 implies that $e'(f,Q) = e'(g,Q) = h$.

(ii) $\quad \tau_1 > t_1$

Let $f_0 := f$, $f_i := e'(f_{i-1},P_i)$ for every i = 1,...,n. Due to the past invariance lemma $f_i = g$ on $T_{\downarrow t_{i+1}}$ for all i = 1,...,n−1. Let k be such that $k = \max \{\ i \mid t_i < \tau_1\ \}$.

Then $t_k < \tau_1 \leq t_{k+1}$ and $f_k = g$ on $T_{\downarrow t_{k+1}}$ imply that $f_k = g$ on $T_{\downarrow \tau_1}$. Applying DPA 1 and DPA 2 we obtain that $Q \subseteq \alpha(f_k)$ and $e'(f_k,Q) = h$. Hence

$e'(f,P_1 \cup ... \cup P_k \cup Q) = e'(e'..(e'(f,P_1)...,P_k),Q) = e'(f_k,Q) = h$, that is for $R = P_1 \cup ... \cup P_k \cup Q$ $e'(f,R) = h$ holds. ∎

The essential role of Determinative Past Assumptions can be demonstrated with an example where we have no DPA 2 and the transitivity of $\rightarrow$ does not hold. Since it is quite a formal construction, we omit it now.

Mesarovic and Takahara [Mesarovic and Takahara, 75] discuss so called causality, and within that 'past–determinacy' for general response systems.

2. APPLICATIONS

In this part we describe three examples in the terms of the model worked out in the first chapter. The emphasis is put on how to use the formalism as a high level description language to specify planning problems. As for any description language there are two important features :

a, "From above" : it is easy to read, simple entities are used to describe a task. One can easily see whether the specification matches the intuitive interpretation of the problem.

b, "From below" : it is strict enough. This makes it possible to justify that an actual implementation (using matrices, Horn–clauses or whatsoever) fits the specification.

In the following examples the abstract entities like states, actions are built up from more elementary objects. We use sets and formulas for this construction mainly to achieve the goal stated in a, above. The richer structure on the skeleton $\langle S,A,T,\alpha,e\rangle$ enables us to formulate special requirements and properties typical for the given task.

Due to definition 1.1.2 there is no restriction on the domain of processes in general. In this chapter, however, all the processes have connected domains which are infinite in the direction of the 'future'. Observe what such a condition means : from a certain time instance there are no blind spots, nothing happens without us knowing about it. If the task allows it, we always take this assumption.

There are examples where the time does not play a role; it seems that time aspects can be omitted from the problem description. The underlying assumption in such a case is that no state transition happens without our actions. Expressing it explicitly : we only have constant processes (or step–functions in a more elaborated case) which are transformed to constant processes by the operations. This permits to identify processes with states and to use time instances only to determine the order of the operations.

2.1. THE BLOCKS–WORLD AS PLANNING PROBLEM

We consider the well–known blocks–world example (see eg. [Nilsson,82]) with one moving arm.

Formal description

After analyzing the blocks–world we decide to use
- constant symbols to denote the objects present,
- a binary predicate ON to describe relative positions of the objects,
- a binary predicate MOVEONTO to describe the actions of the robot arm.

The ***world states*** are built up in the following way :

$OBJ := \{a, b, c, table\}$

$S_0 := \{ ON(x,y) \mid x \in OBJ \setminus \{table\}, y \in OBJ, x \neq y \}$.

$V \subseteq S_0$ is termed sound iff
- every block is standing on exactly one object and
- the table is standing on no object and
- only the table may carry more than one block and
- no impossible configuration occurs.

Formally speaking it means that :

$\forall x \in OBJ \setminus \{table\}\ \exists!\ y \in OBJ\ (\ ON(x,y) \in V\ ) \wedge$

$\neg\exists\ x \in OBJ\ (\ ON(table,x) \in V\ ) \wedge$

$\forall x,y,z \in OBJ\ (ON(x,y) \in V \wedge ON(z,y) \in V\ \&\ x \neq z \implies y = table) \wedge$

the transitive closure of the relation ON is not reflexive.

Observe that sound sets belong to possible configurations.

Then the ***planning universe*** U consists of :

$S := \{ V \subseteq S_0 \mid V \text{ is sound } \}$

$A := \{ MOVEONTO(x,y) \mid x \in OBJ \setminus \{table\}, y \in OBJ \}$

$T := \mathbb{R}_0^+$.

Concerning the time we take the two assumptions mentioned before. Formally we restrict ourselves to the set $F := \{ f \in F(U) \mid dom(f)$ is a ray and $rng(f)$ is a singleton $\}$.
($H \subseteq \mathbb{R}_0^+$ is a ray iff $\forall x \in H \, \forall y \in \mathbb{R}_0^+ \, (x \leq y \Rightarrow y \in H)$
Henceforth we make no distinction between processes with the same range, we identify them all with the same state. Therefore we refer to processes as states.

The conditions of putting a block x on the object y are natural :
- x is not already on y and
- no object is standing on x and
- no object is standing on y or y is the table.

The *allowable operations* of a certain $f \in F$ are :

$$(MOVEONTO(x,y),t) \in \alpha(f) \Leftrightarrow t \in dom(f) \wedge ON(x,y) \notin f \wedge \neg\exists z \in OBJ \, (ON(z,x) \in f) \wedge (\neg\exists z \in OBJ \, (ON(z,y) \in f)) \vee (y = table).$$

The *effect* of an operation $(MOVEONTO(x,y),t) \in \alpha(f)$ on the process $f \in F$ is as follows :

$$e(f,(MOVEONTO(x,y),t)) := (f \setminus \{ON(x,z)\}) \cup \{ON(x,y)\} \quad \text{on } tT,$$

where z is the (only) element of OBJ such that $ON(x,z) \in f$.

PROPOSITION 2.1.1
If $V \subseteq S_0$ is a sound set (a state) and $a \in \alpha(V)$, then $V' = e(V,a)$ is also a sound subset of S_0,that is allowable operations transform states to states. ∎

To express that there is one arm to move blocks we do not allow plans with more actions at the same time, that is we define a *feasibility* :

$\varphi(P)$ iff $P \subseteq A \times T$ and all the sections of P are singletons

and consider only the feasible plans. It is easy to see that
- φ is descending (subplans of a feasible plan are also feasible),
- every feasible plan is commutative (the result of the plan is unique on any process).

A ***planning problem*** can be specified by the previously defined U,α,e and a pair <f,G>, eg. :

f := {ON(c,a), ON(a,table), ON(b,table)}

G := { f ∈ F | ON(c,table) ∈ f }.

The general definition of a ***solution*** can also be restricted by demanding that a solution is feasible. Notice that this does not require that all the plans generated during the search are also feasible.

It can be shown that and GAA and DPA2 holds with respect to F and H, hence we have the transitivity of the $\rightarrow$ relation by Proposition 1.3.1.

2.2 THE JOBSHOP AS PLANNING PROBLEM

We take a simple model as an example. Giving it further structure more sophisticated cases can be described as well. Suppose there is a finite number of all different machines (set M) and all different jobs (set J). Furthermore we have :

- a precedence relation on the jobs, that is a partial ordering : pre $\subseteq J \times J$,
- a relation able $\subseteq M \times J$ that shows which jobs can be performed by which machines,
- a function $d : M \times J \longrightarrow \mathbb{R}_0^+$ that indicates the duration of the performance of a job on a machine.

The task is to complete all the jobs before a certain deadline.

Formal description

The symbols we will use are :

- different constant symbols to denote the machines and the jobs,
- PRE, a binary predicate to denote the relation pre,
- ABLE, a binary predicate to denote the relation able,
- dur, a function symbol for the function d,
- BUSY, a binary predicate to tell that a machine is working on a job,
- READY, a unary predicate to indicate that a job is completed,
- BEGIN, a binary predicate denoting the action of beginning a job on a machine.

Let $M = \{m_1,...,m_k\}$ and $J = \{j_1,...,j_n\}$ be disjoint finite sets standing for the machines and the jobs respectively.

The ***world states*** are constructed as follows :

$S_0 := \{ BUSY(x,y) \mid x \in M, y \in J \} \cup \{READY(y) \mid y \in J \}$.

$V \subseteq S_0$ is called sound iff

- a machine is doing at most one job at a time and
- a job is being done at most on one machine at a time and
- ready jobs are not being performed,

that is

$\forall x \in M\ \forall y,z \in J\ [\ BUSY(x,y) \in V \wedge BUSY(x,z) \in V \Rightarrow y = z\ ]\ \wedge$
$\forall x,y \in M\ \forall z \in J\ [\ BUSY(x,z) \in V \wedge BUSY(y,z) \in V \Rightarrow x = y\ ]\ \wedge$
$\forall y \in J\ [\ READY(y) \in V \Rightarrow \neg\exists\, x \in M\ (\ BUSY(x,y) \in V\ )]$.

The intention is clear, world states are to be the possible snapshots during the job performing process. The situation at a time instance, however, is not fully determined by the ongoing activities. Jobs completed earlier are influencing the situation as well. Hence, a choice needs to be made whether checking the history before decisions, or defining a representative of the relevant aspects of the history and completing the snapshots with it. We have chosen the second possibility, this explains the role of 'READY'.

The ***planning universe*** U consists of :

$S := \{\ V \subseteq S_0 \mid V \text{ is sound}\ \}$
$A := \{\ BEGIN(x,y) \mid x \in M,\ y \in J\ \}$
$T := \mathbb{R}_0^+$.

We assume that we always see what is happening and that ready jobs remain ready forever, that is, we restrict ourselves to

$F := \{f \in F(U) \mid dom(f) \text{ is a ray and } \forall y \in J\ [\{\ t \in T \mid READY(y) \in f(t)\ \} \text{ is a ray or } \emptyset]\ \}$
$(H \subseteq \mathbb{R}_0^+ \text{ is a ray iff } \forall x \in H\ \forall y \in \mathbb{R}_0^+\ (x \leq y \Rightarrow y \in H)$

The conditions to begin a job j on a machine m are :

- m has the ability to perform j,
- j is not ready yet,
- all the predecessors of j are ready,
- no other job requires m while performing j,
- no other machine wants to begin j while being performed on m.

Formally we define the ***allowability function*** $\alpha : F \longrightarrow P(A \times T)$ as follows :

$\forall x \in M\ \forall y \in J\ \forall t \in dom(f)$
$(BEGIN(x,y),t) \in \alpha(f) \Leftrightarrow ABLE(x,y)\ \wedge$
$READY(y) \notin f(t)\ \wedge$
$\forall z \in J\ [\ PRE(z,y) \Rightarrow READY(z) \in f(t)]\ \wedge$
$\neg\exists\, z \in M\ \neg\exists\, \tau \in [t,t+dur(x,y)) : BUSY(z,y) \in f(\tau)\ \wedge$
$\neg\exists\, z \in J\ \neg\exists\, \tau \in [t,t+dur(x,y)) : BUSY(x,z) \in f(\tau)$.

Notice that the last two conditions refer to the future of the time instance t. This causes that the first Determinative Past Assumption does not hold in this planning situation.

The ***effect-function*** $e: F \times (A \times T) \rightarrow F$ is :

$$e(f,(BEGIN(x,y),t))(\tau) := \begin{cases} f(t) & \tau < t \\ f(t) \cup \{BUSY(x,y)\} & t \leq \tau < t+dur(x,y) \\ [f(t) \cup \{READY(y)\}] \setminus \{BUSY(z_\tau,y)\} & t+dur(x,y) \leq \tau \end{cases}$$

where $(BEGIN(x,y),t) \in \alpha(f)$ and z_τ is the (only) machine from M that is (might be) busy with y at the time τ.

PROPOSITION 2.2.1

Let $V \subseteq S_0$ a sound set (a state), $a \in \alpha(V)$, then every $V' \in rng(e(V,a))$ is sound too. Furthermore, if $V \in F$ then $V' \in F$, that is allowable operations preserve the validity of our assumption. ∎

A typical ***planning problem*** can be determined by :

$f_0(t) := \emptyset$ for all $t \in T$

$G := \{ f \in F \mid \forall y \in J \ (READY(y) \in f(t)) \text{ if } t > deadline \}$, deadline $\in T$.

Reasonable properties to require about plans generated during the search, are for instance :

a, if the job j is scheduled in the plan P then every predecessor of j ($\forall i \in J : PRE(i,j)$) is also scheduled and is already finished when j begins.

b, if the job j is scheduled in the plan P then every predecessor of j which is also scheduled must be already finished when j begins.

It is easy to see that the first one is not descending. The second property inherits to subplans due to the following statement.

PROPOSITION 2.2.2

Let $\varphi(P) \Leftrightarrow \quad P \subseteq A \times T \ \wedge \ \forall x,y \in M \ \forall u,v \in J \ \forall \tau,t \in T$

$[BEGIN(x,u),\tau) \in P \wedge PRE(u,v) \wedge (BEGIN(y,v),t) \in P \Longrightarrow \tau+dur(x,u) < t]$.

Then $\varphi(P) \wedge P' \subseteq P \implies \varphi(P')$.

PROOF

It is enough to show that deleting one element of P does not falsify φ. ∎

2.3 ROUTING AS PLANNING PROBLEM

The task in consideration is a basic version of several routing problems. There are n locations to visit in such a way, that no location is visited twice and the end position is the same as the beginning position.

Formal description

Constant symbols will denote the locations,
CONN, a binary predicate indicates whether two locations are connected or not,
AT, a unary predicate points out our position, ie. the location where we are,
SEEN, a unary predicate is to mark locations that have already been visited,
TO, a binary predicate is to represent the action of changing position.

The ***world states*** are constructed from the following entities :

$OBJ := \{c_1,...,c_n\}$

$S_0 := \{ AT(x) \mid x \in OBJ \} \cup \{ SEEN(x) \mid x \in OBJ \}$

A set $V \subseteq S_0$ is called sound iff we are at most at one location at a time, that is :

$\forall x,y \in OBJ (AT(x) \in V \wedge AT(x) \in V \Rightarrow x = y)$

The constituents of the ***planning universe*** U are :

$S := \{ V \subseteq S_0 \mid V \text{ is sound} \}$

$A := \{ TO(x,y) \mid x,y \in OBJ \}$

$T := \mathbb{R}_0^+$.

As before, we consider only the processes that belong to the set

$F := \{ f \in F(U) \mid dom(f) \text{ is a ray } \& \text{ rng}(f) \text{ is a singleton } \}$.

Without loosing expressive power we can identify the processes with their range, and refer to them as states.

The ***allowability function*** is : for every $f \in F$, $t \in dom(f)$

$(TO(x,y),t) \in \alpha(f) \Leftrightarrow AT(x) \in f \wedge SEEN(y) \notin f \wedge CONN(x,y)$.

The ***effect-function*** is defined as follows : if $(TO(x,y),t) \in \alpha(f)$ then

$e(f,(TO(x,y),t)) := (f \setminus \{AT(x)\}) \cup \{AT(y), SEEN(y)\}$ on tT.

Let us have a look on the role of 'SEEN'. If we use constant processes only, then the information about the past is lost after each transition. Hence, – unlike 'READY' in 2.2 – the predicate 'SEEN' is not only a labour–saving invention, but a crucial information carrier. Notice that a design decision was taken that says, that a location becomes seen when arriving to it. We gain an easy way to express the requirement of returning to the beginning position.

PROPOSITION 2.3.1

Let $V \subseteq S_0$ a sound set (a state), $a \in \alpha(V)$, then $V' = e(V,a)$ is sound too.
Furthermore, if $V \in F$ then $V' \in F$. ∎

A typical ***planning problem*** for this task is specified by :

$f_0 := \{AT(c_1)\}$ and

$G := \{ f \in F \mid \forall x \in OBJ\ (SEEN(x) \in f) \}$

Notice that the defining formula of G says nothing about visiting the locations only once and about returning to the beginning position. Proposition 2.3.2 states that no plan can lead to G without satisfying these requirements.

PROPOSITION 2.3.2

Let E be the set of executable plans as in Definition 1.2.5.

$\forall f \in G\ [\ AT(c_1) \in f\ ]$ and

$\forall P \in E\ [e'(f_0,P) \in G \Longrightarrow ((\forall x \in OBJ\ \exists!\ y \in OBJ\ \exists!\ t \in T : (TO(x,y),t) \in P) \wedge$
$(\forall x \in OBJ\ \exists!\ y \in OBJ\ \exists!\ t \in T : (TO(y,x),t) \in P))]$. ∎

3. SUMMARY

A model of planning problems has been presented in this paper. Due to this model a planning situation is determined by the five–tuple $<S,A,T,\alpha,e>$. Within a planning situation the pair of f_0 and G specifies a planning problem. A fundamental choice in the model construction is the use of processes. In this way time is naturally incorporated, it is easy to describe cases where the state of the world changes without outside actions. As to interpret human acts we use operations, i.e. an action coupled with a time instance. The presence of the time component causes that plans need no structure, they are sets of operations. As a consequence, modifying a plan is simple and unconstrained; this leaves us rather free when choosing search strategy.

The structure represented by $<S,A,T,\alpha,e>$ can be considered as a framework. We believe that this frame carries all the relevant aspects of a planning situation in general. Instantiating these five variables we obtain a full domain description, specifying f_0 and G determines a task to solve. The second chapter demonstrated that indeed, the model can be used as a high level description language. The examples were not too sophisticated, therefore the descriptions were not too complex. Despite of the fact that 'real–life' problems yield more difficult specifications we are convinced that systematic work pays off.

The formalism used in these examples is that of usual mathematics, requiring only elementary set theoretical and logical background. This language must be familiar to all those who work as knowledge engineers. The clarity mentioned in the introduction is with respect to this people.

To place the results of this paper in a wider context let us have a look on further work. To achieve the interactive planning system shell, the following steps need to be performed :

1, Making a model of the problems in consideration.

2, Choosing a representation form that is suitable to describe the abstract entities of the model.

3, Choosing solution method(s) appropriate to the representation language and the problem class covered by the model.

4, Implementation, that is design of the system, functional, technical specification and the programming.

A major aim along our project is to keep modeling, representational, algorithmic and implementational aspects apart. This helps to have a clear view about the features of the system in every phase of the development. Keeping a track on the decisions taken we can trace back to the cause of a certain feature, and change it if needed.

This article considers the first step of the four mentioned above. Obviously the second and the third phase are strongly connected, they will be investigated in parallel. It is remarkable that point 4 covers more than one fourth of the whole work. It will be better articulated as the investigation proceeds and its details become more visible.

Further results will be reported in the same series : Computing Science Notes, Eindhoven University of Technology, the Netherlands.

4. INDEX

5. REFERENCES

Forbus, K. D., Qualitative Process Theory, Artificial Intelligence 24 (1984) 85–168.

Mesarovic, M. D. and Takahara, Y., General Systems Theory : Mathematical Foundations, Academic Press, 1975.

Nilsson, N. J., Principles of Artificial Intelligence, Springer–Verlag, 1982.

Pearl, J., Heuristics, Intelligent Search Strategies for Computer Problem Solving, Addison–Wesley, 1984.

In this series appeared :

No.	Author(s)	Title
85/01	R.H. Mak	The formal specification and derivation of CMOS-circuits
85/02	W.M.C.J. van Overveld	On arithmetic operations with M-out-of-N-codes
85/03	W.J.M. Lemmens	Use of a computer for evaluation of flow films
85/04	T. Verhoeff H.M.J.L. Schols	Delay insensitive directed trace structures satisfy the foam rubber wrapper postulate
86/01	R. Koymans	Specifying message passing and real-time systems
86/02	G.A. Bussing K.M. van Hee M. Voorhoeve	ELISA, A language for formal specifications of information systems
86/03	Rob Hoogerwoord	Some reflections on the implementation of trace structures
86/04	G.J. Houben J. Paredaens K.M. van Hee	The partition of an information system in several parallel systems
86/05	Jan L.G. Dietz Kees M. van Hee	A framework for the conceptual modeling of discrete dynamic systems
86/06	Tom Verhoeff	Nondeterminism and divergence created by concealment in CSP
86/07	R. Gerth L. Shira	On proving communication closedness of distributed layers
86/08	R. Koymans R.K. Shyamasundar W.P. de Roever R. Gerth S. Arun Kumar	Compositional semantics for real-time distributed computing (Inf.&Control 1987)
86/09	C. Huizing R. Gerth W.P. de Roever	Full abstraction of a real-time denotational semantics for an OCCAM-like language
86/10	J. Hooman	A compositional proof theory for real-time distributed message passing
86/11	W.P. de Roever	Questions to Robin Milner - A responder's commentary (IFIP86)
86/12	A. Boucher R. Gerth	A timed failures model for extended communicating processes

86/13	R. Gerth W.P. de Roever	Proving monitors revisited: a first step towards verifying object oriented systems (Fund. Informatica IX-4)
86/14	R. Koymans	Specifying passing systems requires extending temporal logic
87/01	R. Gerth	On the existence of sound and complete axiomatizations of the monitor concept
87/02	Simon J. Klaver Chris F.M. Verberne	Federatieve Databases
87/03	G.J. Houben J.Paredaens	A formal approach to distributed information systems
87/04	T.Verhoeff	Delay-insensitive codes - An overview
87/05	R.Kuiper	Enforcing non-determinism via linear time temporal logic specification.
87/06	R.Koymans	Temporele logica specificatie van message passing en real-time systemen (in Dutch).
87/07	R.Koymans	Specifying message passing and real-time systems with real-time temporal logic.
87/08	H.M.J.L. Schols	The maximum number of states after projection.
87/09	J. Kalisvaart L.R.A. Kessener W.J.M. Lemmens M.L.P. van Lierop F.J. Peters H.M.M. van de Wetering	Language extensions to study structures for raster graphics.
87/10	T.Verhoeff	Three families of maximally nondeterministic automata.
87/11	P.Lemmens	Eldorado ins and outs. Specifications of a data base management toolkit according to the functional model.
87/12	K.M. van Hee and A.Lapinski	OR and AI approaches to decision support systems.
87/13	J.C.S.P. van der Woude	Playing with patterns, searching for strings.
87/14	J. Hooman	A compositional proof system for an occam-like real-time language

87/15	C. Huizing R. Gerth W.P. de Roever	A compositional semantics for statecharts
87/16	H.M.M. ten Eikelder J.C.F. Wilmont	Normal forms for a class of formulas
87/17	K.M. van Hee G.-J.Houben J.L.G. Dietz	Modelling of discrete dynamic systems framework and examples
87/18	C.W.A.M. van Overveld	An integer algorithm for rendering curved surfaces
87/19	A.J.Seebregts	Optimalisering van file allocatie in gedistribueerde database systemen
87/20	G.J. Houben J. Paredaens	The R^2 -Algebra: An extension of an algebra for nested relations
87/21	R. Gerth M. Codish Y. Lichtenstein E. Shapiro	Fully abstract denotational semantics for concurrent PROLOG
88/01	T. Verhoeff	A Parallel Program That Generates the Möbius Sequence
88/02	K.M. van Hee G.J. Houben L.J. Somers M. Voorhoeve	Executable Specification for Information Systems
88/03	T. Verhoeff	Settling a Question about Pythagorean Triples
88/04	G.J. Houben J.Paredaens D.Tahon	The Nested Relational Algebra: A Tool to handle Structured Information
88/05	K.M. van Hee G.J. Houben L.J. Somers M. Voorhoeve	Executable Specifications for Information Systems
88/06	H.M.J.L. Schols	Notes on Delay-Insensitive Communication
88/07	C. Huizing R. Gerth W.P. de Roever	Modelling Statecharts behaviour in a fully abstract way
88/08	K.M. van Hee G.J. Houben L.J. Somers M. Voorhoeve	A Formal model for System Specification
88/09	A.T.M. Aerts K.M. van Hee	A Tutorial for Data Modelling

88/10	J.C. Ebergen	A Formal Approach to Designing Delay Insensitive Circuits
88/11	G.J. Houben J.Paredaens	A graphical interface formalism: specifying nested relational databases
88/12	A.E. Eiben	Abstract theory of planning
88/13	A. Bijlsma	A unified approach to sequences, bags, and trees