KfK 3982
Oktober 1985

A Theory of Partial Systems

F. J. Polster
Institut für Datenverarbeitung in der Technik

Kernforschungszentrum Karlsruhe

KERNFORSCHUNGSZENTRUM KARLSRUHE

Institut für Datenverarbeitung in der Technik

KfK 3982

A Theory of Partial Systems

Franz J. Polster

Kernforschungszentrum Karlsruhe GmbH, Karlsruhe

Kernforschungszentrum Karlsruhe GmbH
ISSN 0303-4003

Abstract:

One approach to improving software productivity is reuse of general software to avoid development of code. Frequently, for a particular application a "partial system" of a given general program system is sufficient, where a partial system consists only of those code "fragments" of the system, that implement the capabilities required by that application.

Interdependencies among fragments must be observed for the construction of partial systems. They describe the structure each partial system must adhere to and form an implicit characterization of the set of partial systems. The notion of "fragment system" is introduced as a formalization of the properties of such interconnections among fragments; it is a model for system families, where the members of a system family (the partial systems) are composed of a subset of some collection of shared system components (the fragments).

An algorithm for the construction of a "characteristic set" CF is presented: CF is a minimal subset of fragments with the property, that it is sufficient to indicate for the fragments of CF only, whether or not they are relevant for a partial system.

An explicit representation of the set of partial systems is developed as a subset of $\{0,1\}^{|CF|}$, the elements of which satisfy certain Boolean expressions, called "restrictions". Restrictions are inherent to the fragment system and can be algorithmically derived. Restrictions may also be employed to model semantics of the system interface.

This representation is of importance for the computer-aided specification and construction of partial systems: it determines the minimal amount of information to be entered for the specification of a partial system; based on such a representation a specification system can check, whether or not a specification entered describes a correct partial system.

Eine Theorie der Teilsysteme

Zusammenfassung:

Ein Ansatz zur Erhöhung der Software-Produktivität besteht in der Wiederverwendung allgemeiner Software. Für eine gegebene Anwendung genügt häufig ein Teilsystem eines allgemeinen Programmsystems, wobei ein Teilsystem nur aus den Code-Fragmenten besteht, die die erforderlichen Fähigkeiten des Systems realisieren.

Zur Erzeugung eines Teilsystems sind Querbeziehungen zwischen Fragmenten zu beachten. Sie beschreiben eine Struktur, der jedes Teilsystem zu genügen hat, und bilden eine implizite Charakterisierung der Menge der Teilsysteme. Der Begriff "Fragmentsystem" wird eingeführt als Formalisierung der Eigenschaften solcher Querbeziehungen; zugleich ist dies ein Modell für Systemfamilien, bei denen jedes System der Familie aus einer Teilmenge gemeinsamer Komponenten erzeugt wird.

Ein Algorithmus für die Konstruktion einer "charakteristischen Menge" CF wird angegeben: CF ist eine minimale Teilmenge der Fragmente, so daß es genügt, nur für diese Fragmente anzugeben, ob sie für ein Teilsystem relevant sind oder nicht.

Es wird eine explizite Darstellung der Menge der Teilsysteme eines Systems als Teilmenge von $\{0,1\}^{|CF|}$ angegeben, deren Elemente sogenannten Restriktionen genügen. Restriktionen werden aus dem Fragmentsystem algorithmisch hergeleitet. Mittels Restriktionen werden auch semantische Eigenschaften der Systemschnittstelle modelliert.

Diese explizite Darstellung ist für die rechnergestützte Spezifikation und Konstruktion von Teilsystemen von Bedeutung: sie bestimmt die für die Spezifikation von Teilsystemen erforderliche minimale Information; weiterhin bildet sie die Grundlage für Prüfungen, ob eine Spezifikation ein korrektes Teilsystem beschreibt.

CONTENTS

1. Introduction

Software reuse has become a keystone in many current efforts to improve productivity. Reusability can come in many forms (cf. e.g. the September 1984 issue of IEEE Transactions on Software Engineering), one of them is reuse of code.

Reuse of code entails the design and implementation of general software, i.e. systems, which perform frequently used, common, and repetitive data processing tasks ("reusable functional collections", "generic systems" [4]). Typical examples are operating systems, compilers, database management systems, mathematical subroutine packages.

By definition, a general software system P has to provide services for as wide a spectrum of applications of the respective application area as possible. For a particular application, however, usually an often small subset of the features provided by P suffices such that the immediate use of P is at least wasteful and uneconomical, if not impossible altogether, e.g. due to efficiency problems or limited resources. Thus, it may be desirable to employ instead of the complete system P "versions" of P that provide only a subset of the capabilities of P and consist only of the parts of the program of P necessary for their implementation:

- This is basically the motivation for "SYSGEN" options of operating systems and research into families of operating systems [7, 9, 16, 17].
- Mary Shaw discusses in [18] the usefulness of and the benefits to be gained from having available for a programming language a "language contraction", i.e. a family of programmming languages produced by successively factoring out groups of features of the language: it is shown that this is a technique for improving compilation efficiency; in

particular, the sizes for the compilers pertaining to the sublanguages of a contraction are smaller than the size of the complete compiler implementing the full language.

* The use of versions of a database management system that provide only subsets of the capabilities supported by the system is presented in [11, 15] as a way to benefit from general database software also in environments that do not allow the use of the complete database management system.

Versions in this sense of a program system P will be referred to in the following as "partial systems" of P. We say that the program of P consists of "algorithms", where we rely on the intuitive notion of an algorithm as a set of one or more pieces of code required for the execution of some function provided by P. A partial system, thus, implements only a subset of the optional algorithms of P.

The problem of generating partial systems, in particular the decomposition of the program of P into code fragments as building blocks for the programs of the partial systems, is dealt with in [10, 12, 13]: a fragment may be a program unit, a sequence of statements of a program unit or even a substring of a statement; also, rather than thinking of a fragment as a simple substring of the complete program it is essential to provide for nested fragments. Formally (cf. [12, 13]):

* a <u>fragment</u> f is a not empty list of substrings of the complete program and fragments $f_i \neq f$; the fragments f_i are called the subfragments of f
* a fragment g is called to be <u>nested</u> in fragment f if and only if g is a subfragment of f or g is nested in a subfragment of f.

The four-step method of [13] for the construction of a set of fragments is based on flow analysis (for details see [12, 13]):

In the first step for each program unit u of the program system a

```
1        PROCEDURE DBMS
1           . . . . . . . . . .
1           IF (OP<1 OR OP>6)
1              THEN return 'operation unknown'
1           CASE OP OF
1.1            1: OPEN
1.2            2: CLOSE
1.3            3: FIND
1.4            4: GET
1.5            5: INSERT
1.6            6: DELETE
1           END
1        END
2        PROCEDURE OPEN
2           OPEN_RF
2.1         OPEN_IF
2        END
3        PROCEDURE FIND
3           USE INDEXES
3           evaluate INDEX_TABLE
3           STRTGY
3           return qss
3        END
4        PROCEDURE STRTGY
4           determine access-strategy and
4            set ACCESS_TYPE
4           CASE ACCESS_TYPE OF
4              1:
4.1               build seq.search qss
4.             2: BEGIN
4.2               CASE FILE_TYPE OF
4.2.1               1: calculate tid
4.2.2               2: . . . . . . . .
4.2.2                  RETRIEVE_TID_LIST
4.2.2                  . . . . . . . .
4.2               END
4.2               build direct-access qss
4                 END
4           END
4        END
5        PROCEDURE GET
5           NEXT_TUPLE:
5           CASE ACCESS_TYPE OF
5              1: . . . . . . . .
5.1               NEXT_SEQ
5.1               . . . . . . . .
5              2: . . . . . . . .
5.2               NEXT_TID
5.2               . . . . . . . .
5           END
5.3         IF (qualifikation is not satisfied)
5.3            THEN GO TO NEXT_TUPLE
5        END
6        PROCEDURE NEXT_SEQ
6           CASE FILE_TYPE OF
6              1:
6.1               next_1
6              2:
6.2               next_2
6           END
6        END
7        PROCEDURE NEXT_TID
7          return next tid of tid-list
7        END
```

Fig. 1: Fragmentation of the example system

```
8       PROCEDURE INSERT
8          CASE FILE_TYPE OF
8             1:
8.1              INSERT_1
8             2:
8.2              INSERT_2
8          END
8.3        INSERT_TID
8       END
9       PROCEDURE CLOSE
9          CLOSE_RF
9.1        CLOSE_IF
9       END
10      PROCEDURE DELETE
10         CASE FILE_TYPE OF
10            1:
10.1             DELETE_1
10            2:
10.2             DELETE_2
10         END
10.3       DELETE_TID
10      END
11      PROCEDURE OPEN_RF
11        . . . . . . . .
11        GET
11        . . . . . . . .
11      END
12      PROCEDURE CLOSE_RF
12        . . . . . . . .
12      END
13      PROCEDURE OPEN_IF
13        USE INDEXES
13        . . . . . . . . .
13        GET
13        . . . . . . . . .
13      END
14      PROCEDURE CLOSE_IF
14        USE INDEXES
14        . . . . . . . . .
14      END
15      PROCEDURE INSERT_1
15        . . . . . . . . .
15      END
16      PROCEDURE INSERT_2
16        . . . . . . . . .
16      END
17      PROCEDURE DELETE_1
17        . . . . . . . . .
17      END
18      PROCEDURE DELETE_2
18        . . . . . . . . .
18      END
19      PROCEDURE INSERT_TID
19        USE INDEXES
19        . . . . . . . . .
19      END
20      PROCEDURE DELETE_TID
20        USE INDEXES
20        . . . . . . . . .
20      END
21      PROCEDURE RETRIEVE_TID_LIST
21        . . . . . . . . .
21      END
22      PACKAGE INDEXES
22.1       INDEX_TABLE: ARRAY OF INTEGER
22      END
```

Fig. 1: Fragmentation of the example system (continued)

fragment comprising u is defined.
Applied to the example system we obtain the fragments 1 through 22 as shown in fig. 1.[1]

In the second step for each fragment with statements, which (1) implement an optional algorithm of the program system or (2) the execution of which leads to the execution of statements implementing an optional algorithm, subfragments comprising these statements are defined.
In-depth knowledge of the internal design of the system and the "meaning" of program statements are indispensable for this step [12, 13]. It makes available as building blocks parts of program units that either implement or (directly or indirectly) invoke an optional algorithm.
The set of subfragments of a fragment f introduced in this step may contain subsets X(f), such that with the execution of f exactly one fragment of X(f) is executed. In the example system we have:

X(1) = { 1.1 , 1.2 , 1.3 , 1.4 , 1.5 , 1.6 } ,

X(4) = { 4.1 , 4.2 } , X(4.2) = { 4.2.1 , 4.2.2 } ,

X(5) = { 5.1 , 5.2 } , X(6) = { 6.1 , 6.2 } ,

X(8) = { 8.1 , 8.2 } , X(10) = {10.1 ,10.2 } .

The remaining subfragments of f, denoted O(f), introduced in this step are "really" optional: they can be omitted or included irrespective of

---

[1] the program system of fig. 1 is used for illustration purposes throughout this paper. Program lines belonging to a fragment are marked with the name of that fragment at the left margin. E.g. the lines of code of fig. 1 marked with "1" belong to fragment 1 (program unit DBMS). Dots in fragment names indicate the nesting of fragments.

the presence or absence of the other subfragments. In the example system these are: O(2)={2.1} , O(5)={5.3} , O(8)={8.3} , O(9)={9.1} , O(10)={10.3}

In general additional fragments must be introduced in order to obtain partial systems without superfluous code (cf. [12]). Step 3 completes the fragmentation of executable code: for each fragment f with statements, that can be executed only when subfragments of f are executed, fragments comprising these statements are defined. In step 4 fragmentation of definitional statements is done: for each fragment f with declarations of data objects, which are referenced only by statements of subfragments of f, fragments comprising these declarations are defined; for each global data object a fragment comprising its declaration is defined.
In fig. 1 step 4 leads to fragment 22.1 (declaration of the global data object INDEX_TABLE). Examples for the application of step 3 can be found in [12].

Let F denote the set of fragments of P constructed in this way. Not any arbitrary subset of F may be used for the construction of partial systems, rather, interdependencies among fragments exist, which must be observed as "composition rules". Examples: a version with fragment 1.1 must also contain fragment 1; inclusion of fragment 2 (program unit OPEN) implies inclusion of fragment 1.1 (the call to program unit OPEN) and vice versa; a prerequisite for fragment 5.3 is fragment 5, the reverse, however, does not hold.
Therefore, with each fragment information as to whether or not f is relevant for a partial system must be associated. The "relevance" of a fragment f can formally be thought of as a mapping ρ_f:

$$\rho_f(t) := \begin{cases} 0 : \text{f is not relevant for partial system t} \\ 1 : \text{f is relevant for partial system t} \end{cases}$$

Then the interdependencies from above can be written as implications that must hold for each partial system t of the example system:

$$\rho_{1.1}(t)=1 \implies \rho_1(t)=1$$

$$\rho_2(t)=1 \implies \rho_{1.1}(t)=1$$

$$\rho_{1.1}(t)=1 \implies \rho_2(t)=1$$

$$\rho_{5.3}(t)=1 \implies \rho_5(t)=1$$

These interdependencies can be viewed as attributes of a graph (F,R), where relation $R \subseteq F \times F$ is defined through the method for the construction of F: $(f,g) \in R$ ("fragment f references fragment g") if (1) g contains a program unit, which is directly referenced by f; or (2) g is a subfragment of f according to step 2; or (3) g is a fragment according to step 3 and the execution of f entails the execution of some statement of g; or (4) g is a fragment according to step 4 and f references a definition of g.

Examples: $(1.1, 2) \in R$, $(1, 1.1) \in R$, $(5, 5.3) \in R$, $(20, 22) \in R$.

(F,R) models the data and control flow (cf. e.g. [6]) among the fragments of the program system: $(f,g) \in R \iff$ either flow of control can transfer from code fragment f to code fragment g or a statement in f references a definitional statement of fragment g.

These interdependencies among relevances are statements about the structure each partial system adheres to and constitute an implicit characterization of the set of partial systems of P. The objective of this work is to develop an explicit representation of the set of partial systems. This is not only of theoretical interest, such a representation is also of practical importance, in particular for the computer-aided specification and construction of partial systems (cf. [14, 15]): (1) it determines the minimal amount of information to be entered for the specification of a partial system. Example: fragments 1.1 and 2 of the example system have the same relevance. Therefore, it is sufficient to indicate the relevance of just one of them. (2) Based on this representation a specification system can check, whether or not a specification entered describes a correct partial system.

Section 2 presents the concept of "fragment system" as a model for systems with partial systems.

Section 3 shows that the relevance of a fragment may be determined by the relevances of other fragments in form of "relevance expressions". This suggests to look for a minimal subset CF of the set of fragments such that for each fragment f the relevance of f can be expressed in terms of the relevances of CF. A subset of fragments with these properties is called a "characteristic set", section 4 gives the formal definition of this notion and presents an algorithm for the construction of characteristic sets.

In section 5 we prove properties of the algorithm. They are employed in section 6 where it is shown, how to find for a given fragment a relevance expression involving only relevances of characteristic fragments.

Section 7 shows that the set of partial systems can be viewed as a subset of $\{0,1\}^{|CF|}$.

The reader is referred to appendix I for the basic concepts and notations used in this paper.

Appendix IV presents FSA (Fragment System Analyser), a PROLOG-implementation of the algorithms and techniques of this paper.

2. Fragment systems: definition, rationale

This section introduces the notion of fragment system and explains the rationale behind this concept.

2.1. Definition, terminology

Throughout this paper we use the following notation:

- T refers to the set of partial systems of a system P, F denotes the set of fragments
- B denotes the set {0, 1} of truth values (0: FALSE; 1: TRUE).

DEFINITION 1:

Let $F \neq \emptyset$ be a finite set, $R \subseteq F \times F$ a relation on F, G the directed graph (F,R), E the set $\{f \mid f \varepsilon F, PRED(f)=\emptyset\}$. Furthermore, let there be mappings $X: F \dashrightarrow \P(F)$, $O: F \dashrightarrow \P(F)$ and $\rho: T \times F \dashrightarrow B$. We define:

$$\overleftarrow{X}(f) := \{ g \mid g \varepsilon F, f \varepsilon X(g) \} \qquad \overleftarrow{O}(f) := \{ g \mid g \varepsilon F, f \varepsilon O(g) \}$$

(F,R,X,O,E,ρ) is called a <u>fragment system</u>, if G is an acyclic graph and axioms FG1-FG5 are satisfied.

FG1: For each $e \varepsilon E$ there is at least one $t \varepsilon T$ with $\rho(t,e)=1$

FG2: For each $f \varepsilon F$ holds:

- $\overleftarrow{X}(f) \neq \emptyset$ or $\overleftarrow{O}(f) \neq \emptyset \implies |PRED(f)| = 1$
- $X(f) \subseteq SUCC(f)$, $O(f) \subseteq SUCC(f)$, $X(f) \ast O(f) = \emptyset$
- $|X(f)| \neq 1$

FG3: For each $f \varepsilon F$-E holds:

$\rho(t,f)=1 \implies$ there is a vertex $g \varepsilon PRED(f)$ with $\rho(t,g)=1$

FG4: For each $f \varepsilon F$ with $X(f) \neq \emptyset$ holds:

$\rho(t,f)=1 \implies$ there is a vertex $g \in X(f)$ with $\rho(t,g)=1$

FG5: For each $g \in PRED(f)$ with $f \in F$ and $\overleftarrow{X}(f)=\overleftarrow{O}(f)=\emptyset$ holds:

$\rho(t,g)=1 \implies \rho(t,f)=1$

Remark:

Since (F,R) is an acyclic graph and |F| finite, property FG0 holds:

FG0: $E \neq \emptyset$; each $f \in F$ is accessible from at least one $e \in E$.

Terminology, notations:

- (F,R,X,O,E) is called the <u>fragment graph</u> of the fragment system (F,R,X,O,E,ρ).
- The elements of X(f) are called the <u>X-fragments</u> of f, those of O(f) the <u>O-fragments</u> of f. The elements of E are the <u>entry-fragments</u> of the fragment system.
- For $f \in F$ the mapping ρ_f: T --> B is defined as follows:

  $$\rho_f(t) := \rho(t,f)$$

  ρ_f is called the <u>relevance</u> of f, $\rho_f(t)$ the <u>relevance value</u> of f for the partial system t. f and g are said to have the <u>same relevance</u> iff $\rho_f \equiv \rho_g$.

  A <u>relevance expression</u> is a relevance or a Boolean expression with relevances as operands. The Boolean operators are defined on relevances in the obvious way; e.g.: ρ_f OR ρ_g (t) := ρ_f(t) OR ρ_g(t)

2.2. Fragment system as a model for families of partial systems

The concept of fragment system has originally been designed as a model for families of partial systems of program systems as discussed in section 1 [10, 12]:

a) The set F represents the set of code fragments, which form the building blocks for the programs of the partial systems. Relation R represents the "references"-relationship among fragments of the introduction.

Remark:

(F,R) being acyclic does not necessarily imply that partial systems of program systems with recursive procedures cannot be modeled as fragment systems:

Let procedure U_0 be recursive, i.e. there are procedures U_1, U_2, ... such that U_i calls U_{i+1} with $U_n=U_0$ for some n≥0. Each procedure forms a fragment, this leads to a graph (F,R) with a cycle $<f_0,...,f_n,f_0>$. Since we need information only on which fragments are required (and not on flow of control as such), edge (f_n,f_0) is redundant and can be omitted. This eliminates the cycle.

b) Mapping ρ: T×F --> B represents the family of relevance mappings ρ_f

c) Entry-fragments represent those code fragments of the program system, that must be executed in order to invoke the system to perform some operation. Typically, for systems running as separate tasks entry-fragments are main programs, for program systems in form of a subroutine package these are usually subprograms. In the latter case there are in general several entry-fragments: |E|>1.

Fragment 1 is the only entry-fragment of the example system: E={1}. A fragment system with six entry-fragments would result, if the six program units called in the CASE-statement would implement the system interface immediately (instead of the one program unit DBMS).

d) The mappings X and O model the sets X(f) and O(f), respectively.

X and O of the example system:

```
         +-
         | { 1.1 , 1.2 , 1.3 ,                               +-
         |   1.4 , 1.5 , 1.6 } : f= 1                        | { 2.1 } : f= 2
         | { 4.1 , 4.2 }       : f= 4                        | { 5.3 } : f= 5
         | { 4.2.1 , 4.2.2 }   : f= 4.2                      | { 8.3 } : f= 8
X(f) = <   { 5.1 , 5.2 }       : f= 5            O(f) = <
         | { 6.1 , 6.2 }       : f= 6                        | { 9.1 } : f= 9
         | { 8.1 , 8.2 }       : f= 8                        | { 10.3} : f= 10
         | {10.1 ,10.2 }       : f= 10                       |    ∅    : else
         |       ∅             : else                        +-
         +-
```

e) FG1, FG0 are necessary conditions for partial systems without superfluous ("dead") code.

f) FG2 formally describes the facts that

- exactly one fragment references an O- or X-fragment
- a fragment cannot be both a X-fragment and an O-fragment
- a fragment has either no or at least two X-fragments.

g) Axioms FG3, FG4 and FG5 are the formalizations of the interdependencies among fragments:

FG3 states that, if a fragment is relevant for partial system t, at least one of the fragments referencing it must be relevant for t; the reverse holds for fragments that are neither O- nor X-fragments, this is axiom FG5; FG4 is the definition of the sets X(f).

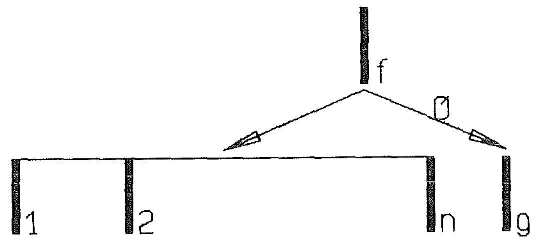
(Note that due to $X(f) \subseteq SUCC(f)$ is FG4 weaker than FG5.)

Figure 2 depicts the fragment graph of the example system. Fragment graphs are visualized in this paper as follows:

- vertical bars represent the fragments, i.e. the vertices of (F,R); with each bar is associated the name of the respective fragment.
- an edge $(f,g) \in R$ is diagrammed as an arrow from bar f to bar g.

  In order to avoid crossing arrows an edge (f,g) may be drawn as an arrow from bar f to the name of g and a second arrow from the name of f at some other position to bar g (cf. edge (13,22)).

- if g is an O-fragment of f, then the arrow from f to g is labelled with with "O". If f has n>0 X-fragments, then the n bars representing the vertices of X(f) are linked with a horizontal line and one arrow is drawn from f to that connecting line:



|<------ X-fragments ------>|
of f

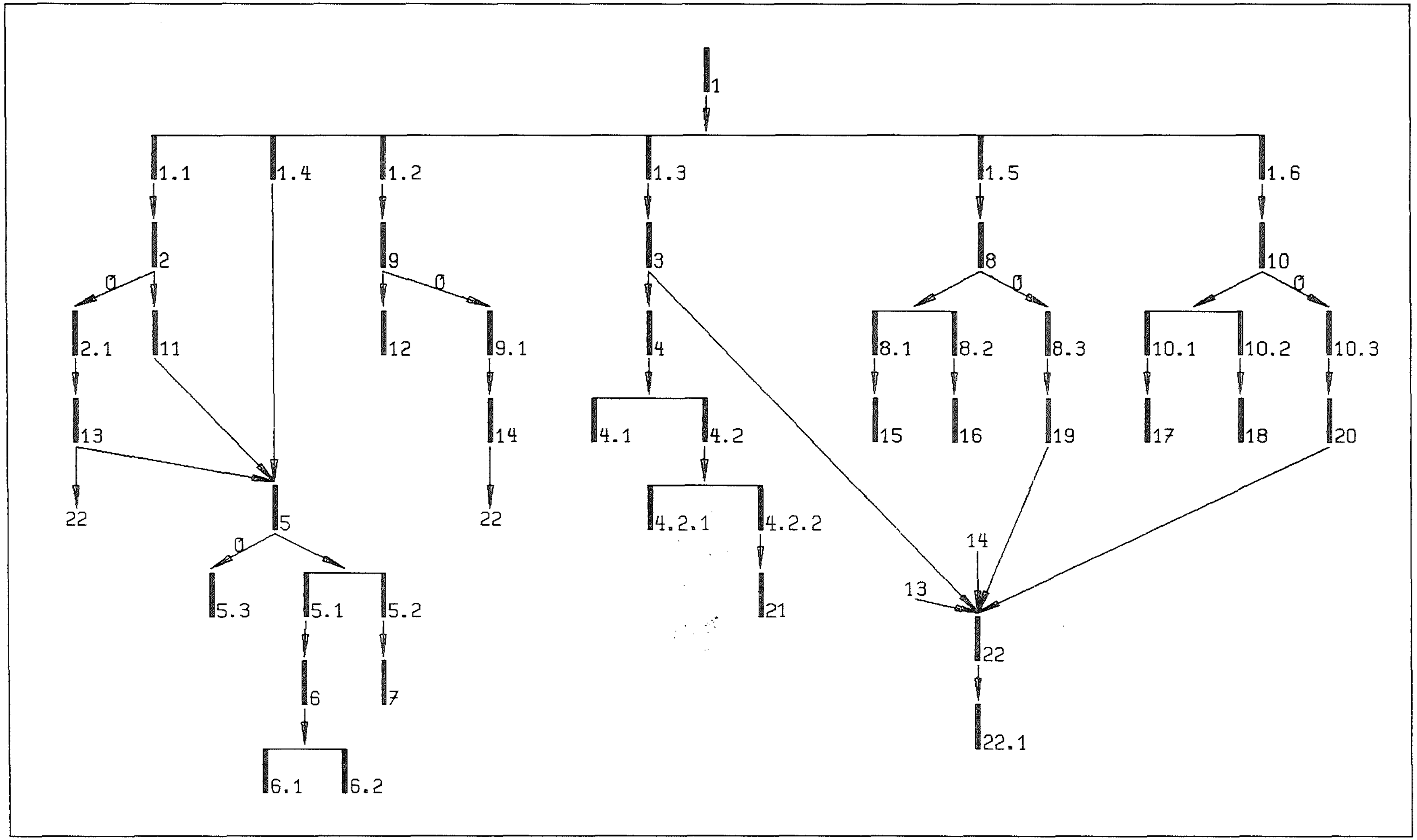


Fig. 2: The fragment graph of the example system

2.3. Fragment system as a model for system families.

Fragment systems also model system families, where a system family consists of "version groups" and "configurations", cf. e.g. [20]:

- The components of a version group are considered equivalent according to some criterion: the components, program modules or subsystems, share the same interface and abstract specifications, but may be implemented differently or tailored to different operating systems or user groups; a module may exist as a sequence of revisions.

  A version group implies a choice - one may choose one or several of its constituent versions.
- The components of a configuration must be combined, a configuration, thus, implies an integration process (e.g. a link-edit process).

In [20] Tichy models system families as AND/OR graphs. An AND/OR graph [8] is a directed acyclic graph, in which each vertex is either a leaf, an AND vertex or an OR vertex:

- leaves are the primitive objects and present program modules, documentation fragments, test data, etc.
- OR vertices represent version groups: one may choose one (or several) of its successors.
- AND vertices represent configurations: all successors of an AND vertex must be combined to form a configuration.

A system family in this sense can be viewed as a fragment system:

- the fragments F are the vertices of the AND/OR graph
- relation R models the successor relationship of the AND/OR graph
- the successsors of an OR vertex in the AND/OR graph form (depending on the number of successors) O- or X-fragments of the corresponding

fragment system. AND vertices correspond to fragments, the successors of which in the fragment graph are neither O- nor X-fragments.

- the set T represents the set of members of the family; as above mapping ρ indicates whether or not a fragment, i.e. a configuration or version or leaf, is a component of a member of the family.

Example (adapted from [20]):

Let an I/O-subsystem have two versions, one for the line printer (LPT), and one for the terminal (TERMINAL). The LPT version be a configuration consisting of three components: OPEN, CLOSE and PUT. The modules OPEN and CLOSE exist as a sequence of revisions, the module PUT have two machine specific versions, one for the VAX and one for the PDP11, each of those again with several revisions.

Figure 3 shows this system family modeled as a fragment system, the vertices are labeled with names of the versions, configurations and revisions, respectively.

Remark:

The novel idea with the concept of fragment system is the notion of relevance, i.e. mapping ρ. With the AND/OR model in order to specify the "proper" members of the system family it is necessary to add "selection mechanisms" [20]. ρ is a formalization and generalization of these selection mechanisms.

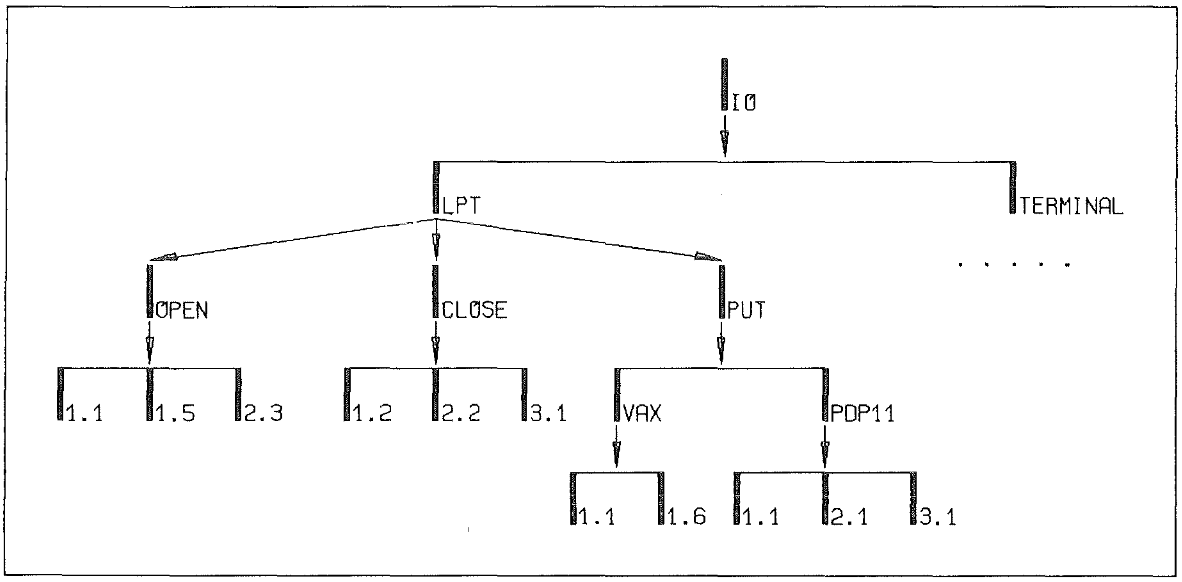


Fig. 3: A system family as fragment graph

3. Relevance expressions

Axioms FG3 through FG5 are statements about the relevances of a fragment system. This section shows that the relevance of a fragment can be equal to relevance expressions with relevances of other fragments, in other words a relevance may be determined by other relevances.

3.1. The relevances of f and PRED(f)

Let $f \varepsilon F$ have relevance ρ_f and $n>0$ predecessors $PRED(f)=\{f_i | 1 \le i \le n\}$, ρ_i be the relevance of f_i, $1 \le i \le n$.

f is no entry-fragment, thus, due to FG3 we have the implication

(I1) $$\rho(t,f)=1 \implies OR_{i=1}^{n} \rho(t,f_i)=1$$

from which we infer

(I2) $$OR_{i=1}^{n} \rho(t,f_i)=0 \implies \rho(t,f)=0$$

If in addition f is neither an 0- nor X-fragment, then follows from FG5

$$OR_{i=1}^{n} \rho(t,f_i)=1 \implies \rho(t,f)=1$$

Thus, for fragments $f \varepsilon F\text{-}E$, which are neither 0- nor X-fragments we have the equation:

(G1) $$\rho_f \equiv OR_{i=1}^{n} \rho_i$$

Remark:

If f is neither an 0- nor an X-fragment and PRED(f)={g}, then f and g have the same relevance: $\rho_f \equiv \rho_g$

DEFINITION 2:

A <u>R1-path</u> from f to g of a fragment graph (F,R,X,O,E) is a path P of graph (F,R) from f to g, such that no vertex $x \in P$ with $x \neq f$ is an O- or X-fragment.

Example: The path < 1.1 , 2 , 11 , 5 > of figure 2 is a R1-path.

Later we will need the following statement:

THEOREM 1:

$f \in F$ be neither an O- nor X-fragment. Let $E' = \{e_i \mid 1 \le i \le m\} \subseteq E$ be the set of all entry-fragments, from which there is at least one path to f: for $1 \le i \le m$ there be p_i paths $P_{i,j}$, $1 \le j \le p_i$, from $e_i \in E'$ to f. Then

$$\rho_f \equiv OR_{g \in C}\ \rho_g$$

where $C := \{g_{i,j} \mid 1 \le j \le p_i,\ 1 \le i \le m\}$ and

$$g_{i,j} := \begin{cases} e_i & : P_{i,j} \text{ is a R1-path} \\ x & : x \in P_{i,j},\ x \text{ is an O- or X-fragment and the subpath of } P_{i,j} \text{ from } x \text{ to } f \text{ is a R1-path} \end{cases}$$

Proof:

We prove the statement of this theorem through repetitive application of equation G1 (utilizing the commutativity of the OR-operator):

ρ_f is identical to the OR-ing of the relevances of the predecessors of f. If the relevance expression for ρ_f contains relevances of fragments that are neither O- nor X-fragments, then each of these relevances is replaced with relevance expressions according to G1. Since |F| is finite and each path in (F,R) is acyclic, this substitution process yields in a finite number of steps a relevance expression for ρ_f with relevances of O-, X- or entry-fragments only.

Since this replacement process visits the vertices of all paths of (F,R) that lead to f, starting at f until the first O-, X- or entry-fragment of the respective path is encountered, these fragments are exactly those of the set C of the statement.

□

Examples:

- In figure 2 there are three paths from E'=E={1} to fragment 5:

$$P_{1,1} = < 1 , 1.1 , 2 , 2.1 , 13 , 5 > \qquad g_{1,1} = 2.1$$
$$P_{1,2} = < 1 , 1.1 , 2 , 11 , 5 > \qquad g_{1,2} = 1.1$$
$$P_{1,3} = < 1 , 1.4 , 5 > \qquad g_{1,3} = 1.4$$

Thus: $\rho_5 \equiv \rho_{2.1} \text{ OR } \rho_{1.1} \text{ OR } \rho_{1.4}$

- For fragment 22 we obtain: $\rho_{22} \equiv \rho_{2.1} \text{ OR } \rho_{1.3} \text{ OR } \rho_{9.1} \text{ OR } \rho_{8.3} \text{ OR } \rho_{10.3}$

3.2. The relevances of f and SUCC(f)

Let $f \in F$ have relevance ρ_f and n>0 successors $SUCC(f)=\{f_i | 1 \le i \le n\}$, ρ_i be the relevance of f_i, $1 \le i \le n$.

In order to be able to deduce results similar to those above, we must make additional assumptions as to the sets $PRED(f_i)$. Due to $X(f)*O(f)=\emptyset$ (axiom FG2) the set SUCC(f) can be written as the union of four pairwise disjoint, not necessarily nonempty sets:

$$SUCC(f) := X(f) + O(f) + SUC1(f) + SUCM(f) \qquad \text{with}$$

$$SUC1(f) := \{ x \mid x \in SUCC(f), |PRED(x)|=1, x \neg\in X(f), x \neg\in O(f) \}$$
$$SUCM(f) := \{ x \mid x \in SUCC(f), |PRED(x)|>1 \}$$

Without loss of generality let

$$X(f) := \{ f_i \mid 1 \le i < nx\} \qquad O(f) := \{ f_i \mid nx \le i < no\}$$

$$SUC1(f) := \{ f_i \mid no \le i < n1\} \qquad SUCM(f) := \{ f_i \mid n1 \le i \le n \}$$

with integers nx, no, n1 satisfying $1 \le nx \le no \le n1$.

Examples:

SUCC(8) = { 8.1 , 8.2 } + { 8.3 } + ∅ + ∅ (i.e.: nx=3, no=4, n1=4, n=3)

SUCC(2) = ∅ + { 2.1 } + { 11 } + ∅ (i.e.: nx=1, no=2, n1=3, n=2)

Fragment 5 is one of the fragments of figure 2 with more than one predecessor: SUCM(13)=SUCM(11)=SUCM(1.4)={5}

DEFINITION 3:

A <u>S-fragment</u> is a fragment $f \in F$ with $|PRED(f)| > 1$.

3.2.1. f and its X-fragments

Let be $X(f) \ne \emptyset$, i.e. $nx>1$. $f_i \in X(f)$ is no entry-fragment, from FG2 follows $PRED(f_i)=\{f\}$; therefore, FG3 yields:

$$\rho(t,f_i)=1 \text{ for an index } i \text{ with } 1 \le i < nx \implies \rho(t,f)=1 \quad \text{and thus:}$$

$$OR_{i=1}^{nx-1}\ \rho(t,f_i)=1 \implies \rho(t,f)=1$$

Because of FG4 also the reverse is true:

$$\rho(t,f)=1 \implies OR_{i=1}^{nx-1}\ \rho(t,f_i)=1$$

such that holds:

$$\rho_f \equiv OR_{i=1}^{nx-1}\ \rho_i$$

3.2.2. f and its O-fragments

Let be $O(f)\neq\emptyset$, i.e. no>nx. $f_i \in O(f)$ is no entry-fragment, from FG2 follows $PRED(f_i)=\{f\}$; therefore, FG3 yields:

$$\rho(t,f_i)=1 \text{ for an index } i \text{ with } nx\leq i<no \implies \rho(t,f)=1 \quad \text{and thus:}$$

$$OR_{i=nx}^{no-1}\ \rho_i(t)=1 \implies \rho_f(t)=1$$

Since neither FG4 nor FG5 apply here, the reverse, and thus an equation analogous to the one of section 3.2.1 does not hold here.

3.2.3. f and the elements of SUC1(f)

Let be $SUC1(f)\neq\emptyset$, i.e. n1>no. Because of FG3 and FG5 f and the fragments of SUC1(f) have the same relevance (cf. remark in section 3.1):

$$\rho_f \equiv \rho_i \qquad no\leq i<n1$$

3.2.4. f and its S-fragments

Let be $SUCM(f)\neq\emptyset$, i.e. $n1\leq n$. Because of FG5 for each element f_i of SUCM(f), i.e. for $n1\leq i\leq n$, holds

$$\rho_f(t)=1 \implies \rho_i(t)=1$$

These implications is all we can prove, FG3 is not sufficient to show the reverse.

Example: The following implications hold for fragment 5:

$$\rho_{13}(t)=1 \implies \rho_5(t)=1 \qquad \rho_{11}(t)=1 \implies \rho_5(t)=1 \qquad \rho_{1.4}(t)=1 \implies \rho_5(t)=1$$

3.3. Entry-fragments and $f \neg\varepsilon E$

THEOREM 2:

If $\rho_f(t)=1$ for $f \varepsilon F\text{-}E$, then there is at least one $e \varepsilon E$ and a path P from e to f, such that $\rho_x(t)=1$ holds for each $x \varepsilon P$; in particular: $\rho_e(t)=1$.

Proof:

Because of $f \varepsilon F\text{-}E$ FG3 (cf. also I1 of section 3.1) implies the existence of a vertex $u \varepsilon PRED(f)$ with $\rho_u(t)=1$; if $u \varepsilon F\text{-}E$, then by the same token there is also a predecessor v of u with $\rho_v(t)=1$, and so forth: since F is finite and (F,R) is acyclic repeated application of FG3 yields a path P from some entry-fragment to f, such that $\rho_x(t)=1$ holds for each $x \varepsilon P$.

□

THEOREM 3:

Let be $F' \subseteq F$, $F' \neq \emptyset$. If each path from E to $f \varepsilon F$ contains at least one vertex of F', then

$$OR_{g \varepsilon F'}\ \rho_g(t)=0 \quad ==> \quad \rho_f(t)=0$$

Proof:

For $f \varepsilon F'$ or $F'=F$ there is nothing to been shown. Let be $f \varepsilon F\text{-}F'$, $F' \neq F$, i.e. $f \neg\varepsilon E$, and $OR_{g \varepsilon F'}\ \rho_g(t)=0$.

Due to theorem 2 $\rho_f(t)=1$ implies the existence of a path P from some entry-fragment to f, with $\rho_x(t)=1$ for each $x \varepsilon P$. Because of $P * F' \neq \emptyset$ this leads to $OR_{g \varepsilon F'}\ \rho_g(t)=1$, a contradiction.

□

COROLLARY 1:

Let be $E' \subseteq E$ the set of entry-fragments, from which there is a path to $f \in F$. If $E' \neq \emptyset$, then the following implication holds:

$$OR_{e \in E'}\ \rho_e(t)=0 \quad \Longrightarrow \quad \rho_f(t)=0$$

Proof:

With $F' := E'$ this is an immediate consequence of theorem 3.

□

COROLLARY 2:

Let be $E' \subseteq E$ the set of entry-fragments, from which there is a path to $f \in F$. If $E' \neq \emptyset$ and for each $e \in E'$ there is at least one R1-path from e to f, then

$$OR_{e \in E'}\ \rho_e \equiv \rho_f.$$

Proof:

From corollary 1 follows: $OR_{e \in E'}\ \rho_e(t)=0 \Longrightarrow \rho_f(t)=0$

Since there is for each $e \in E'$ a R1-path to f, f is neither an O- nor a X-fragment (cf. definition 2). According to theorem 1 there exists a subset $F' \subseteq F$ with $E' \subseteq F'$ such that $\rho_f(t) = OR_{x \in F'}\ \rho_x(t)$. Therefore:

$$OR_{e \in E'}\ \rho_e(t)=1 \Longrightarrow OR_{x \in F'}\ \rho_x(t)=\rho_f(t)=1.$$

□

4. Characteristic fragments

4.1. Definition

The preceding section demonstrated that the relevance of a fragment may be given with the relevances of other fragments of the fragment system. This suggests to look for a subset of fragments with the property that their relevances determine those of the remaining fragments; furthermore, such a subset should be as small as possible. The following definition is a formal statement of these properties:

DEFINITION 4:

A set $CF \subseteq F$ is called a <u>characteristic set</u> of fragment system (F,R,X,O,E,ρ), if it satisfies CF1 and CF2:

CF1: For each $f \in F$ there is a set $C(f) \subseteq CF$, $C(f) \neq \emptyset$, such that holds:

$$\rho_f \equiv OR_{g \in C(f)}\ \rho_g$$

CF2: For $f \in CF$ there is no set $C \subseteq CF-\{f\}$, $C \neq \emptyset$, with: $\rho_f \equiv OR_{g \in C}\ \rho_g$

Terminology:

- The elements of a characteristic set are called <u>characteristic fragments</u>
- A set $C(f)$ with property CF1 is called a <u>CF-representation</u> of $f \in F$ and $OR_{g \in C(f)}\ \rho_g$ a <u>CF-expression</u> for ρ_f.

The interest in characteristic sets stems from the fact that in order to specify the set of fragments relevant for a partial system t it is sufficient to indicate the relevance values $\rho_f(t)$ of the fragments $f \in CF$

only. CF2 says that there is at least one $t \in T$ such that it is necessary to indicate for each $f \in CF$, whether or not f is relevant for t. As we shall see in section 7 it may be the case, however, that there are partial systems such that the relevance values of a subset of the characteristic set are sufficient.

4.2. Construction of a characteristic set

4.2.1. R1-sets

DEFINITION 5:

- The R1-set of $f \in F$, denoted by R1(f), is the set

$$\{f\} + \{\ g \mid g \in F, \text{ there is a R1-path P from f to g}, \text{ P-}\{f\} \text{ contains no S-fragment } \}$$

- f is called the root-fragment of R=R1(f), it is denoted by ROOT(R).

Remark:

In algorithm 1 the definition of ROOT(M) will be extended to non-R1-sets $M \subseteq F$.

From definition 5 follows immediately:

a) $g \in R1(f) \Longrightarrow \rho_g \equiv \rho_f$ (cf. section 3.1 or 3.2.3)

b) If f is an O-, X- or S-fragment, then there is no $x \in F$ such that $x \neq f$ and $f \in R1(x)$.

c) Each vertex of R1(f)-{f} has exactly one predecessor. Therefore:
   - any subgraph of a fragment graph consisting of the vertices of a R1-set is a tree. This is the justification for the term "root"-fragment.
   - each path from $x \in F\text{-}R1(f)$ to $g \in R1(f)$ contains f.

For a certain class of R1-sets we can show a maximality property.

THEOREM 4:

Let f_1 be an O-, X-, S- or entry-fragment, $f_2 \varepsilon F$. Then

$$R1(f_2) \subseteq R1(f_1) \quad \text{or} \quad R1(f_2)*R1(f_1)=\emptyset$$

Proof:

Case 1: $f_2 \varepsilon R1(f_1)$

From definition 5 follows immediately: $R1(f_2) \subseteq R1(f_1)$.

Case 2: $f_2 \neg\varepsilon R1(f_1)$

$\Longrightarrow R1(f_2) \neg\subseteq R1(f_1)$ (due to $f_2 \neq f_1$)

Suppose $R1(f_2)*R1(f_1) \neq \emptyset$, i.e. there is some $x \varepsilon F$ with $x \varepsilon R1(f_1)$ and $x \varepsilon R1(f_2)$: then there exists a R1-path from f_1 to x, which must contain f_2 (statement c, second part), a contradiction to $f_2 \neg\varepsilon R1(f_1)$. Therefore, $R1(f_2)*R1(f_1)=\emptyset$ must hold.

□

4.2.2. The algorithm

For a fragment system (F,R,X,O,E,ρ) algorithm 1 yields in step 3 a set $CF \subseteq F$, which will be proved a characteristic set. At first, F is partitioned into subsets, so-called "Ω-sets" with the property that the fragments of a Ω-set have the same relevance. Since by definition a characteristic set can contain at most one element of each Ω-set, a directed graph is constructed, the vertices of which represent the Ω-sets. The set CF is described in terms of vertices of this graph.

ALGORITHM 1: Construction of a characteristic set

Input : fragment system (F,R,X,O,E,ρ)

Output: a characteristic set $CF \subseteq F$

Algorithm:

Step 1:

$$i = 0$$

$$\Omega^{(0)} = \{ R1(f) \mid f \varepsilon F,\ f \text{ is a X-, O- or S-fragment or } f \varepsilon E \}$$

Step 2:

<u>WHILE</u> (there are $\omega_1, \omega_2 \ \varepsilon\ \Omega^{(i)}$ such that $f = ROOT(\omega_2)$ is a S-fragment and $PRED(f) \subseteq \omega_1$)

<u>DO</u>

$$i = i + 1$$

$$\omega^{(i)} = \omega_1 + \omega_2$$

$$ROOT(\omega^{(i)}) = ROOT(\omega_1)$$

$$\Omega^{(i)} = \Omega^{(i-1)} + \{ \omega^{(i)} \} - \{ \omega_1, \omega_2 \}$$

<u>END</u>

Step 3:

$$\Omega = \Omega^{(i)}$$

$G\Omega$ be the directed graph $(F\Omega, R\Omega)$, where

$F\Omega = \{ f \mid f \ \varepsilon\ F$, there is $\omega \ \varepsilon\ \Omega$ with $ROOT(\omega)=f \}$

$R\Omega = \{ (f,g) \mid f,g \ \varepsilon\ F\Omega,\ PRED(g)*\omega \neq \emptyset$ for $\omega \ \varepsilon\ \Omega$ with $ROOT(\omega)=f \}$

Mapping $X\Omega$: $F\Omega \dashrightarrow \P(F\Omega)$ is defined as follows:

$X\Omega(f) = \{ g \mid g \ \varepsilon\ F\Omega,\ \overleftarrow{X}(g) \neq \emptyset$,
each path in $G\Omega$ from E to g contains f,
$(f,g) \ \varepsilon\ R\Omega$ or there is in $G\Omega$ a path P from f to g,
such that $P-\{f,g\}$ contains neither an X- nor an O-fragment$\}$

$CF = \{ f \mid f \ \varepsilon\ F\Omega,\ |PRED(f)| \leq 1,\ X\Omega(f) = \emptyset \}$

Note:

Algorithm 1 does not specify the pair to be merged if in step 2 several pairs (ω_1,ω_2) satisfy the condition for the construction of $\omega^{(i)}$. As will be shown below (theorem 11) this choice is inessential in that step 2 always produces the same set Ω for a given fragment system.

Terminology: The elements of Ω are called Ω-sets.

From the definition of $G\Omega=(F\Omega,R\Omega)$ in step 3 follows immediately:

a) $|F\Omega| = |\Omega|$, $E \subseteq F\Omega$, $|\omega * F\Omega|=1$ for each $\omega \in \Omega$

b) Let be $\omega_1,\omega_2 \in \Omega$, $f_1=ROOT(\omega_1)$, $f_2=ROOT(\omega_2)$ and P a path in $G=(F,R)$ from f_1 to f_2.

   Then, $P\Omega := P*F\Omega$ is a path from f_1 to f_2 in $G\Omega$ and $\{f_1,f_2\} \subseteq P\Omega$

c) $(f_1,f_2) \in R\Omega$ ==> there is a path P in G from f_1 to f_2 and $P-\{f_1,f_2\}$ contains no X- or 0-fragments.

   More general: if $P\Omega$ is a path in $G\Omega$ from f_1 to f_2, then there is a path P in G from f_1 to f_2 and $P\Omega \subseteq P$

Example:

The encircled sets of fragments in figure 4 are the elements of $\Omega^{(0)}$ (the R1-sets) of the example system (cf. figure 2); no pair of elements of $\Omega^{(0)}$ satisfies the condition of step 2, thus $\Omega=\Omega^{(0)}$ and

$F\Omega$ = { 1 , 1.1 , 1.4 , 1.2 , 1.3 , 1.5 , 1.6 ,
2.1 , 9.1 , 4.1 , 4.2 , 4.2.1 , 4.2.2 ,
8.1 , 8.2 , 8.3 , 10.1 , 10.2 , 10.3 ,
5 , 5.1 , 5.2 , 5.3 , 6.1 , 6.2 , 22 }

Figure 5 depicts the directed graph $(F\Omega,R\Omega)$.

The mapping $X\Omega$:

$$X\Omega(f) = \begin{cases} \{ 1.1 , 1.2 , 1.4 , 1.3 , 1.5 , 1.6 \} & : f=1 \\ \{ 4.1 , 4.2 \} & : f=1.3 \\ \{ 8.1 , 8.2 \} & : f=1.5 \\ \{10.1 ,10.2 \} & : f=1.6 \\ \{ 4.2.1 , 4.2.2 \} & : f=4.2 \\ \{ 5.1 , 5.2 \} & : f=5 \\ \{ 6.1 , 6.2 \} & : f=5.1 \\ \emptyset & : \text{else} \end{cases}$$

Since fragments 5 and 22 are S-fragments algorithm 1 yields for the example system:

$$CF = F\Omega - \{ 5 , 22 \} - \{ 1 , 1.3 , 1.5 , 1.6 , 4.2 , 5.1 \}$$

Asterisks mark these fragments in figure 4 and figure 5.

We show that the set CF of algorithm 1 is a characteristic set:

- in section 4.3 we prove the minimality property CF2
- as to property CF1 we show in section 5 that Ω is a disjoint decomposition of F and that the elements of a Ω-set have the same relevance. Therefore, the problem is reduced to the determination of CF-representations for the root-fragments of the Ω-sets. This is dealt with in section 6.

In section 4.3 we need a description of CF as a subset of F:

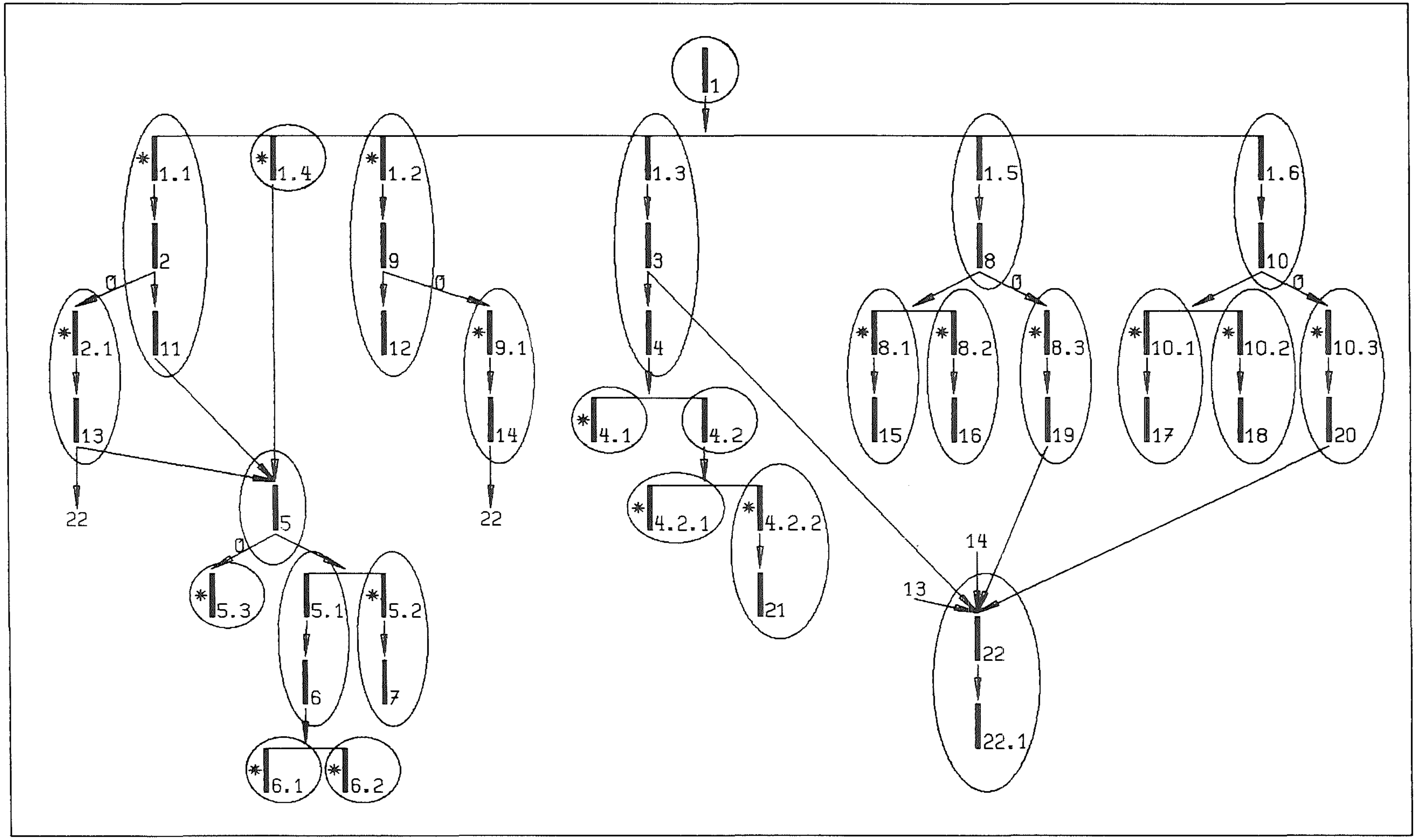


Fig. 4: The Ω-sets and set CF of the example system

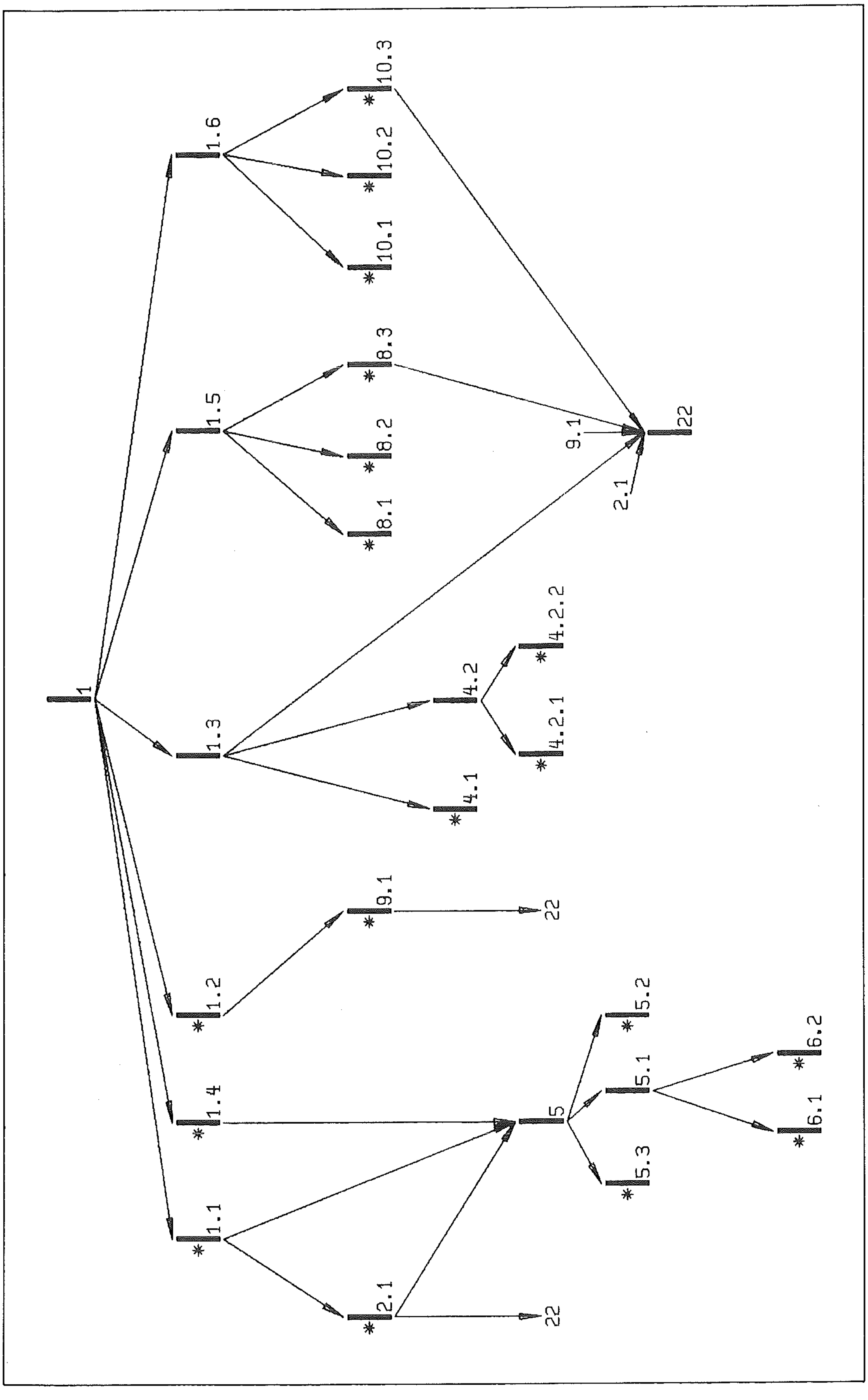


Fig. 5: The graph GΩ and the set CF of the example system

THEOREM 5:

The set CF of algorithm 1 is the set

$\{ f \mid f \in F$, f is a X-, O- or entry-fragment,

there is no X-fragment g such that holds:

(*) each path in G from E to g contains f and

(**) ($f \in \overleftarrow{X}(g)$ or there is in G a R1-path from f to $x \in \overleftarrow{X}(g)$)$\}$

Proof:

According to step 3 of algorithm 1 CF contains all fragments $f \in F$, which are X-, O- or entry-fragments and for which there is no X-fragment $g \in F$, that satisfies both (1) and (2):

(1) each path in $G\Omega$ from E to g contains f

(2) $(f,g) \in R\Omega$ or there is in $G\Omega$ a path P from f to g, such that P-{f,g} contains neither an X- nor an O-fragment.

Thus we have to show: (1) and (2) <==> (*) and (**)

The equivalence of (1) and (*), i.e. (1) <==> (*), follows immediately from the above statements b) and c).

Statement (2) implies (**):

- $(f,g) \in R\Omega$ ==> there is in G a R1-path from f to $x \in \overleftarrow{X}(g)$

  or $g \in X(f)$ ==> (**)
- there is in $G\Omega$ a path P from f to g, such that P-{f,g} contains neither an X- nor an O-fragment ==> there is in G a R1-path from f to $x \in \overleftarrow{X}(g)$ ==> (**)

Statement (**) implies (2):

- $f \in \overleftarrow{X}(g)$ ==> $(f,g) \in R\Omega$ ==> (2)
- there is in G a R1-path from f to $x \in \overleftarrow{X}(g)$ ==> (2)

□

Because of $\overleftarrow{X}(x)$=PRED(x) for X-fragments x follows immediately

COROLLARY 3:

CF = { f | f ε F, f is a X-, O- or entry-fragment,
there is no g ε F with X(g)≠∅, such that holds:
each path in G from E to g contains f and
(f=g or there is in G a R1-path from f to g) }

Convention:

Unless otherwise indicated in the remainder a "<u>path from f to g</u>" is a <u>path in G=(F,R)</u>, and not in GΩ!

4.3. Proof of minimality

CF has property CF2 if and only if for each $f \in CF$ holds:

$$\rho_f \neg\equiv OR_{g\varepsilon C}\ \rho_g \text{ for each subset } C \subseteq CF-\{f\},\ C\neq\emptyset$$

which is equivalent to property CF2':

CF2': for each $C \subseteq CF-\{f\}$ there is at least one $t_C \in T$ with

$$\rho(t_C,f) \neq OR_{g\varepsilon C}\ \rho(t_C,g)$$

We will prove CF2' by showing how to construct for a given fragment $f \in CF$ two partial systems $t_1,t_2 \in T$ with

$$\rho(t_1,f) \neq \rho(t_2,f) \quad \text{and}$$
$$\rho(t_1,g) = \rho(t_2,g) \quad \text{for } g \in CF-\{f\}$$

Since, if two such partial systems exist, we can define for $C \subseteq CF-\{f\}$, $C\neq\emptyset$:

$$t_C := \begin{cases} t_1 : & \rho(t_1,f) \neq OR_{g\varepsilon C}\ \rho(t_1,g) \\ t_2 : & \rho(t_1,f) = OR_{g\varepsilon C}\ \rho(t_1,g) \end{cases}$$

and $\rho(t_C,f) \neq OR_{g\varepsilon C}\ \rho(t_C,g)$ holds:

- $t_C=t_1 \implies \rho(t_C,f) = \rho(t_1,f) \neq OR_{g\varepsilon C}\ \rho(t_1,g) = OR_{g\varepsilon C}\ \rho(t_C,g)$
- $t_C=t_2 \implies \rho(t_C,f) = \rho(t_2,f) \neq \rho(t_1,f) = OR_{g\varepsilon C}\ \rho(t_1,g)$
  $= OR_{g\varepsilon C}\ \rho(t_2,g) = OR_{g\varepsilon C}\ \rho(t_C,g)$

It is interesting to note that it is sufficient to consider for any $f \in CF$ only two partial systems, which furthermore depend only on f and not on the set C: C determines just which one of both is t_C. (See also the example at the end of this section.)

The following result is instrumental in constructing t_1, t_2:

THEOREM 6:

Let be $D:=\{f_i \mid 1 \le i \le n\} \subseteq F$ a nonempty set of 0-, X- or entry-fragments, such that $X(g) \neg\subseteq D$ for each $g \in F$ with $X(g) \neq \emptyset$.

$F' := \{ g \mid g \in F, P*D \neq \emptyset$ for each path P from E to $g \}$.

Then the elements of $F-F'$ are the fragments of a partial system, i.e. there is $t_D \in T$ with

$$\rho(t_D,g) := \begin{cases} 0 : & g \in F' \\ 1 : & g \in F-F' \end{cases}$$

Proof:

It must be shown that t_D satisfies FG3, FG4 and FG5.

- Let be $f \in F-E$:

  $\rho(t_D,f)=1 \Longrightarrow f \in F-F'$

  $\Longrightarrow$ there is a path P from E to f with $P*D=\emptyset$

  $PRED(f)\neq\emptyset \Longrightarrow$ there is a fragment $g \in PRED(f)$ with $g \in P$

  $\Longrightarrow P':=P-\{f\}$ is a path from E to $g \in PRED(f)$ with $P'*D=\emptyset$, and therefore $g \in F-F'$

  $\Longrightarrow$ there is a vertex $g \in PRED(f)$ with $\rho(t_D,g)=1$.

  Thus, property FG3 holds.

- Let be $f \in F$ with $X(f)\neq\emptyset$.

  $\rho(t_D,f)=1 \Longrightarrow f \in F-F'$

  $\Longrightarrow$ there exists a path P from E to f with $P*D=\emptyset$

  From $X(f)-D\neq\emptyset$ and $PRED(g)=\{f\}$ for each $g \in X(f)$ follows:

  there is a vertex $g \in X(f)$, such that $P':=P+\{g\}$ is a path from E to g with $P'*D=\emptyset$, i.e. $g \in F-F'$

  $\Longrightarrow$ there is a vertex $g \in X(f)$ with $\rho(t_D,g)=1$

  Thus, property FG4 holds.

- Let be $f \varepsilon F$, $\overleftarrow{X}(f)=\overleftarrow{O}(f)=\emptyset$, $g \varepsilon PRED(f)$:

  $\rho(t_D,g)=1 \Longrightarrow g \varepsilon F-F'$

  $\Longrightarrow$ there exists a path P from E to g with $P*D=\emptyset$

  $f \neg\varepsilon D \Longrightarrow P':=P+\{f\}$ is a path from E to f with $P'*D=\emptyset$, and therefore

  $f \varepsilon F-F'$

  $\Longrightarrow \rho(t_D,f)=1$.

  Thus, property FG5 holds.

□

Let be $f \varepsilon CF$. The partial systems t_1, t_2 to be constructed consist of the fragments of the sets $F-F_1$ and $F-F_2$, respectively, where

$F_1 := \{ g \mid g \varepsilon F, P*\{f\}\neq\emptyset$ for each path P from E to g $\}$

$FO := \{ g \mid g \varepsilon F_1, g\neq f$, g is an O-fragment $\}$,

$F_2 := \{ g \mid g \varepsilon F, P*FO\neq\emptyset$ for each path P from E to g $\}$

i.e.:

$$\rho(t_1,g) := \begin{cases} 0 : g \varepsilon F_1 \\ 1 : g \varepsilon F-F_1 \end{cases} \qquad \rho(t_2,g) := \begin{cases} 0 : g \varepsilon F_2 \\ 1 : g \varepsilon F-F_2 \end{cases}$$

t_1 and t_2 are partial systems:

- $t_1 \varepsilon T$ follows immediately from theorem 6 (with $D=\{f\}$)
- if $FO\neq\emptyset$, then $t_2 \varepsilon T$ due to theorem 6 (with D=FO); if $FO=\emptyset$, then $F_2=\emptyset$ and, thus, $\rho(t_2,f)=1$ for each $f \varepsilon F$, i.e. t_2 is the complete system: $t_2 \varepsilon T$

t_1, t_2 have the postulated properties:

From the definition of FO follows $F_2 \subseteq F_1$; F_2 is even a proper subset of F_1 because of $f \neg\varepsilon F_2$ and $f \varepsilon F_1$. Thus, for $g \varepsilon F$ holds:

$$\rho(t_1,g)=1 \implies \rho(t_2,g)=1$$

$$\implies \qquad g \in CF-\{f\},\ \rho(t_1,g)=1 \implies \rho(t_2,g)=1$$

It remains to be shown:

$$g \in CF-\{f\},\ \rho(t_1,g)=0 \implies \rho(t_2,g)=0$$

Proof:

Let be $g \in CF-\{f\}$. $\rho(t_1,g)=0$ implies $g \in F_1$.

- g is no entry-fragment because of $g \neq f$ (cf. definition of set F1).
- If g is an O-fragment, then $g \in FO$ (due to $g \neq f$), and thus $\rho(t_2,g)=0$ (since $FO \subseteq F_2$).
- Let g be a X-fragment.

  Each path from f to g contains at least one O-fragment besides f (f may be an O-fragment).

  In order to prove this let us assume that there is a path P' from f to g, such that P'-{f} contains no O-fragment, however at least one X-fragment, namely vertex g.

  P' be the list k_i, $1 \leq i \leq j$, with $k_1=f$ and $k_j=g$, m be the smallest index such that $X(k_m) \neq \emptyset$, i.e. k_{m+1} is a X-fragment of k_m. Then $m<j$ (since $k_j=g$ is a X-fragment) and each path from E to k_{m+1} contains f.
  If $m>1$, then there is a R1-path from f to $k_m \in \overleftarrow{X}(k_{m+1})$, otherwise $k_m=f \in \overleftarrow{X}(k_{m+1})$. Due to theorem 5 this is a contradiction to $f \in CF$ and disproves the assumption.

  Therefore, since each path from E to g contains f, each path from E to g contains at least one element of FO, i.e. $g \in F_2$ and thus $\rho(t_2,g)=0$.

□

We have shown

$$\rho(t_1,g) = \rho(t_2,g) \quad \text{for } g \in CF-\{f\}$$

$$\text{and} \qquad 0=\rho(t_1,f) \neq \rho(t_2,f)=1 \quad (\text{because of } f \in F_1,\ f \neg\in F_2)$$

i.e. t_1 and t_2 are the partial systems to be constructed for f.

Consequently, for each f ε CF there are partial systems t_1, t_2, such that f is the only element in CF with different relevance values for t_1 and t_2; in other words, through assigning relevance values to the fragments of CF-{f} only one cannot obtain all possible partial systems. This is the minimality property CF2 (or CF2').

Example:

For fragment f=1.1 of the example system

$$F_1=\{\ 1.1\ ,\ 2\ ,\ 11\ ,\ 2.1\ ,\ 13\ \} \quad FO=\{2.1\} \quad F_2=\{\ 2.1\ ,\ 13\ \}$$

Thus, t_1 consists of the fragments F-{1.1,2,11,2.1,13}, t_2 of F-{2.1,13}. If C={2.1}, then $t_C=t_2$; else, even if 2.1 ε C, $t_C=t_1$ (all characteristic fragments of CF-{1.1} except for 2.1 have relevance value 1 for t_1).

5. Properties of Ω-sets

In this section we examine properties of Ω-sets. These results will be used in the remainder show that algorithm 1 yields a characteristic set CF. Also, Ω of step 2 is shown to be unique for a fragment system.

THEOREM 7:

Each fragment $f \in F$ is element of exactly one R1-set of $\Omega^{(0)}$

Proof:

- $\omega_1, \omega_2 \in \Omega^{(0)}$ are R1-sets, the respective root fragments are X-, O-, S- or entry-fragments. Due to theorem 4

  $$\omega_2 \subseteq \omega_1 \quad \text{or} \quad \omega_2 * \omega_1 = \emptyset$$

  and

  $$\omega_1 \subseteq \omega_2 \quad \text{or} \quad \omega_1 * \omega_2 = \emptyset$$

  $\Longrightarrow$ $\omega_1 = \omega_2$ or $\omega_1 * \omega_2 = \emptyset$, i.e. $f \in F$ is element of at most one R1-set of $\Omega^{(0)}$.

- Suppose $f \in F$ is not element of any R1-set.
  Then, f is no entry- or S-fragment, i.e. $|PRED(f)|=1$. In addition, f is neither a X- nor an O-fragment: thus the predecessor f' of f, too, cannot be element of a R1-set, since this set would contain f. These arguments also apply to f', thus the predecessor of f' cannot be element of a R1-set, and so forth. Since F is a finite set this implies the existence of a R1-path from some $e \in E$ to f, the vertices of which are not contained in any R1-set. It follows in particular that e is not element of a R1-set, a contradiction to step 1 of algorithm 4.1.

□

The statements on Ω in the remainder of this section follow from properties of the sets $\Omega^{(i)}$, the proofs are through induction on index i.

COROLLARY 4:

a) Ω is a disjoint decomposition of F

b) Each $\omega \varepsilon \Omega$ is the union of R1-sets of $\Omega^{(0)}$

Proof:

a) $\Omega^{(0)}$ is a disjoint decomposition of F according to theorem 7.

In step 2 of algorithm 1 $\Omega^{(i+1)}$ is derived from $\Omega^{(i)}$ through replacing two elements of $\Omega^{(i)}$ with their union

==> if $\Omega^{(i)}$ is a disjoint decomposition of F, then this holds also for $\Omega^{(i+1)}$.

I.e. for each $i \geq 0$ $\Omega^{(i)}$ is a disjoint decomposition of F. Since F is finite, this is true also for Ω.

b) Similarly, straightforward induction on i shows that for each index $i \geq 0$ each element of $\Omega^{(i)}$ is the union of elements of $\Omega^{(0)}$. Again, since F is finite this implies statement b.

□

THEOREM 8:

$$f,g \;\varepsilon\; \omega,\; \omega \;\varepsilon\; \Omega^{(i)},\; i \geq 0 \implies \rho_f \equiv \rho_g$$

Proof:

The theorem holds for i=0 (statement a on definition 5).

Inductive hypothesis: the theorem is true for the elements of $\Omega^{(i)}$, $i \geq 0$.

Inductive step: let be $\omega \varepsilon \Omega^{(i+1)}$.

- For $\omega \varepsilon \Omega^{(i)} * \Omega^{(i+1)}$ nothing is to be shown, the inductive hypothesis applies immediately.
- For $\omega \varepsilon \Omega^{(i+1)} - \Omega^{(i)}$ holds:

$\omega=\omega_1+\omega_2$, where $\omega_1,\omega_2 \in \Omega^{(i)}$ and (without loss of generality) $PRED(ROOT(\omega_2)) \subseteq \omega_1$.

With $u:=ROOT(\omega_1)$, $v:=ROOT(\omega_2)$ follows from the inductive hypothesis:

$$\rho_u \equiv \rho_x \text{ for } x \in \omega_1, \qquad \rho_v \equiv \rho_x \text{ for } x \in \omega_2.$$

Because of $PRED(v) \subseteq \omega_1$ equation G1 of section 3.1 (v is a S-fragment) yields: $\rho_v \equiv OR_{x \in PRED(v)}\, \rho_x \equiv \rho_u$

$\Longrightarrow$ $\rho_x \equiv \rho_u \equiv \rho_v$ for $x \in \omega_1+\omega_2$.

I.e. the statement of the theorem is true also for the elements of $\Omega^{(i+1)}$ and, thus, holds for each $i \geq 0$.

□

THEOREM 9:

Let be $\omega \in \Omega^{(i)}$, $i \geq 0$. Each path from $x \in F-\omega$ to $g \in \omega$ contains $ROOT(\omega)$.

Proof:

The theorem holds for i=0 (second part of statement c on definition 5).

Inductive hypothesis: the theorem is true for the elements of $\Omega^{(i)}$, $i \geq 0$.

Inductive step: let be $\omega \in \Omega^{(i+1)}$.

- For $\omega \in \Omega^{(i)} * \Omega^{(i+1)}$ nothing is to be shown, the inductive hypothesis applies immediately.
- For $\omega \in \Omega^{(i+1)}-\Omega^{(i)}$ holds:

  $\omega=\omega_1+\omega_2$, where $\omega_1,\omega_2 \in \Omega^{(i)}$ and (without loss of generality) $PRED(ROOT(\omega_2)) \subseteq \omega_1$.

  With $u:=ROOT(\omega_1)$, $v:=ROOT(\omega_2)$ the inductive hypothesis yields:

  (1) $g \in \omega_1 \Longrightarrow$ each path from $x \in F-(\omega_1+\omega_2)$ to g contains u

  (2) $g \in \omega_2 \Longrightarrow$ each path from $x \in F-(\omega_1+\omega_2)$ to g contains v

  $PRED(v) \subseteq \omega_1 \Longrightarrow$ each path from $x \in F-(\omega_1+\omega_2)$ to $g \in \omega_2$ contains an element of ω_1 and because of (1) also u

  $\Longrightarrow$ each path from $x \in F-(\omega_1+\omega_2)$ to $g \in \omega_1+\omega_2$ contains $u=ROOT(\omega)$.

I.e. the statement of the theorem is true also for the elements of $\Omega^{(i+1)}$ and, thus, holds for each $i \geq 0$.

□

Since $|F|$ is finite and because of $\Omega = \Omega^{(i)}$ (step 3 of algorithm 1), the statements of both theorem 8 and 9 hold also for Ω:

COROLLARY 5:

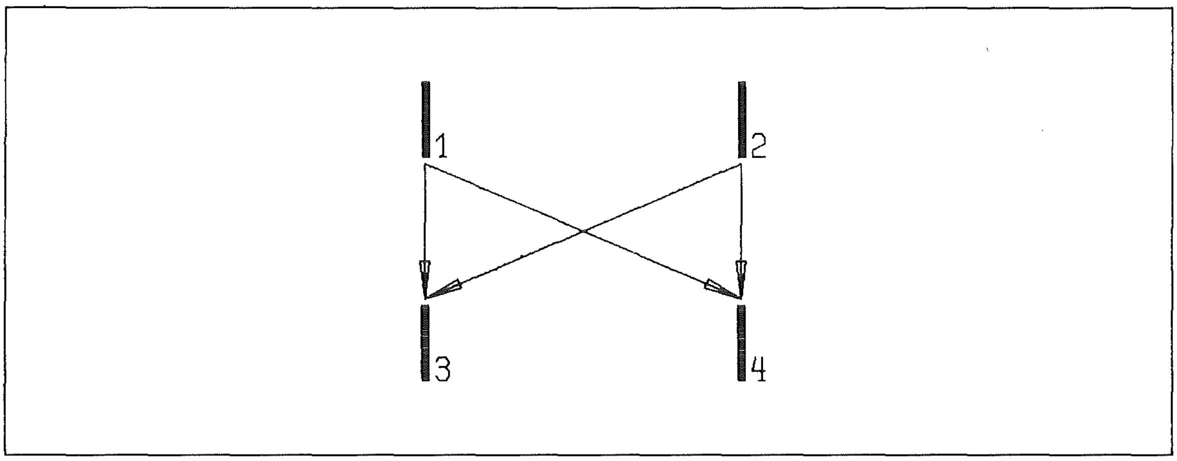
Let be $\omega \in \Omega$:

a) $f,g \in \omega \implies \rho_f \equiv \rho_g$

b) Each path from $x \in F-\omega$ to $g \in \omega$ contains ROOT(ω).

Remark:

In general the reverse of corollary 5a does not hold. Consider the fragment graph consisting of entry-fragments 1 and 2 and S-fragments 3 and 4:



Here $\rho_3 \equiv \rho_4 \equiv \rho_1$ OR ρ_2, the fragments 3 and 4, however, are elements of different Ω-sets: Ω = { {1},{2},{3},{4} }.

THEOREM 10:

f be an O-, X-, S- or entry-fragment, $g \varepsilon F$, $\omega \varepsilon \Omega$. If $f \varepsilon P$ for each path P from an entry-fragment to g and each path from f to g is a R1-path, then holds: $f \varepsilon \omega \implies g \varepsilon \omega$

Proof:

Let $L(y)$ denote the maximum of the path-lengths of all paths from f to $y \varepsilon F$, i.e. $L(y) := \max \{ |P|-1 \mid y \varepsilon F, P \text{ is a path from } f \text{ to } y \}$.

Since F is finite and (F,R) is acyclic $|F|$ is an upper bound for the length of a path from f to g: $L(g) \leq |F| < \infty$. Therefore, we show by induction on $L(g)$ that the statement of the theorem holds for all finite values of $L(g)$.

$L(g)=1 \implies g$ is no S-fragment (each path from E to g contains f)

$\implies g \varepsilon R1(f)$. Therefore:

$f \varepsilon \omega \implies R1(f) \subseteq \omega \implies g \varepsilon \omega$ (cf. corollary 4b)

Inductive hypothesis: the statement holds for $g \varepsilon F$ with $L(g) \leq k$, $k>1$.

Inductive step:

Let g be a fragment with $L(g)=k+1$, $PRED(g)=\{g_i \mid 1 \leq i \leq m\}$, $f \varepsilon \omega$. Since each path from $e \varepsilon E$ to g contains vertex f, this is true also for each path from e to $x \varepsilon PRED(g)$. Because of $L(x) \leq k$ for $x \varepsilon PRED(g)$ the inductive hypothesis yields for $1 \leq i \leq m$: $f \varepsilon \omega \implies g_i \varepsilon \omega$.

- If $m>1$, i.e. if g is a S-fragment, then $R1(g) \subseteq \omega$ due to step 2 of algorithm 1 and corollary 4b;
- else (i.e. if $PRED(g)=\{g_1\}$) the path from g_1 to g is a R1-path. Thus: $g_1, g \varepsilon R1(x)$ for some $x \varepsilon F$ and $R1(x) \subseteq \omega$ (because of $g_1 \varepsilon \omega$ and corollary 4b).

In both cases follows $g \varepsilon \omega$, such that the statement holds for each finite value of $L(g) \geq 1$.

□

As has been pointed out above if in step 2 there is a choice of pairs for the construction of $\omega^{(i)}$ algorithm 1 does not specify, which pair to merge. We now show that irrespective of the order of merging algorithm 1 always produces the same disjoint decomposition of F, i.e. Ω is uniquely determined:

THEOREM 11:

For a given fragment system algorithm 1 produces exactly one Ω-set.

Proof:

Let Ω' and Ω'' be two Ω-set constructed with algorithm 1 for a fragment system (F,R,X,O,E,ρ). Let be $\omega' \in \Omega'$ and $ROOT(\omega')=r'$.

- It is $R1(r') \subseteq \omega'$. According to corollary 4 there is some $\omega'' \in \Omega''$ such that $R1(r') \subseteq \omega''$.
- For other R1-sets in ω', i.e. for each R1-set $R=R1(f) \in \Omega^{(0)}$ with $R \subseteq \omega'$ and $R \neq R1(r')$ holds:

  each path from E to f contains r' (corollary 5b) and each path from r' to f is a R1-path (FGO guarantees the existence of such a path). From $r' \in \omega''$ follows $f \in \omega''$ (theorem 10) and $R \subseteq \omega''$ (corollary 4b).

I.e. for each $\omega' \in \Omega'$ there is $\omega'' \in \Omega''$, such that ω'' contains all R1-sets of ω', and thus $\omega' \subseteq \omega''$. Both Ω' and Ω'' being disjoint decompositions of F implies $\Omega'=\Omega''$.

□

6. The relevances of a fragment system

This section shows how to obtain for the fragments $f \varepsilon F$ of a fragment system CF-representations $C(f) \subseteq CF$ and, thus, relevance expressions $\rho_f \equiv OR_{g \varepsilon C(f)}\ \rho_g$. In this way we prove that the set CF of algorithm 1 has property CF1.

Since the elements of a Ω-set have the same relevance (corollary 5a) and Ω is a disjoint decomposition of F (corollary 4a), it suffices to construct CF-representations of the root-fragments of the Ω-sets, i.e. for the elements of $F\Omega$.

DEFINITION 6:

SUCX(f) := { g | g ε F, X(g)≠∅, each path in G from E to g contains f,

(*) f=g or there is in G a R1-path from f to g }

(Examples follow in section 6.4)

Remarks:

- According to theorem 5 and corollary 3 holds for $f \varepsilon F\Omega$ with $|PRED(f)| \leq 1$: $SUCX(f)=\emptyset \iff X\Omega(f)=\emptyset \iff f \varepsilon CF$
- If f is an entry-fragment, then f is the only entry-fragment, from which there is a path to the fragments of SUCX(f).

6.1. The relevances of X- and entry-fragments

Let be $f \in F\Omega$ a X- or entry-fragment.

6.1.1. CF-representations

- $f \in CF$: a CF-representation of f is {f}, because of the minimality property CF2 this is the only one.
- $f \neg\in CF$:

  ==> $SUCX(f) \neq \emptyset$ (cf. remark on definition 6).

  Let be $n:=|SUCX(f)|\geq 1$, $SUCX(f):=\{f_i|1\leq i\leq n\}$ and $X(f_i):=\{f_{i,j}|1\leq j\leq m(i)\}$, $1\leq i\leq n$.

  For each i with $1\leq i\leq|SUCX(f)|$ and $t \in T$ follows

  $\rho(t,f)=0 \implies \rho(t,f_i)=0$ (theorem 3 with F'={f}) and

  $\rho(t,f)=1 \implies \rho(t,f_i)=1$ (sections 3.2.3, 3.2.4 and *)

  ==> $\rho_f \equiv \rho_i$

  ==> there are $n=|SUCX(f)|$ equations for ρ_f (cf. section 3.2.1):

  $$\rho_f \equiv OR_{j=1}^{m(i)} \rho_{i,j} \qquad 1\leq i\leq|SUCX(f)|$$

  and n equations for C(f):

  $$C(f) = C(f_i) = +_{j=1}^{m(i)} C(f_{i,j}) \qquad 1\leq i\leq|SUCX(f)|$$

  If there is an index i such that $f_{i,j} \in CF$ for $1\leq j\leq m(i)$, then $+_{j=1}^{m(i)} C(f_{i,j}) = +_{j=1}^{m(i)}\{f_{i,j}\}$ is a CF-representation of f.

  If $f_{i,j} \neg\in CF$, i.e. $SUCX(f_{i,j})\neq\emptyset$, then $C(f_{i,j})$ itself is the union of CF-representations of the X-fragments of an element of $SUCX(f_{i,j})$, etc. Since F is finite and (F,R) acyclic, this substitution process yields after a finite number of steps a set $C(f) \subseteq CF$ with $\rho_f \equiv OR_{g\in C(f)}\, \rho_g$.

These considerations lead to the following recursive algorithm for the construction of CF-representations:

ALGORITHM 2: Construction of a CF-representation C(f)

Input : fragment graph (F,R,X,O,E), X- or entry-fragment $f \in F$

Output: a CF-representation $C \subseteq CF$ of f

Algorithm:

$$C = C(f)$$

with:

```
FUNCTION C(f)
   IF (SUCX(f)=∅)
       THEN C = {f}
       ELSE DO
            select a fragment d ε SUCX(f)
            let be X(d)={ d_i | 1≤i≤|X(d)| } and n=|X(d)|
            C = +_{i=1}^{n} C(d_i)
            END
END
```

6.1.2. Constraints

If there are $n=|SUCX(f)|>1$ equations for ρ_f, then the n relevance expressions must be identical, i.e. the following n-1 "relevance constraints" must hold:

(RC1) $$OR_{j=1}^{m(1)} \rho_{1,j} \equiv OR_{j=1}^{m(i)} \rho_{i,j} \qquad 2 \le i \le |SUCX(f)|$$

6.2. The relevances of O-fragments

Let be $f \varepsilon F\Omega$ an O-fragment.

6.2.1. CF-representations

- $f \varepsilon CF$: a CF-representation of f is {f}, because of the minimality property CF2 this is the only one.
- $f \neg\varepsilon CF$:

  $\Longrightarrow SUCX(f) \neq \emptyset$

  Let be $n:=|SUCX(f)|\geq 1$, $SUCX(f):=\{f_i | 1\leq i\leq n\}$ and $X(f_i):=\{f_{i,j} | 1\leq j\leq m(i)\}$, $1\leq i\leq n$.

  As in section 6.1.1 holds $\rho_f \equiv \rho_i$ such that there are $|SUCX(f)|$ equations for ρ_f: $\rho_f \equiv OR_{j=1}^{m(i)} \rho_{i,j}$ $\quad 1\leq i\leq |SUCX(f)|$

  For each X-fragment $f_{i,j}$ a CF-representation $C(f_{i,j})$ can be constructed with algorithm 2, which leads to n CF-representations of f as follows:

  $$C(f) = +_{j=1}^{m(i)} C(f_{i,j}) \qquad 1\leq i\leq |SUCX(f)|$$

6.2.2. Constraints

At least one constraint in form of an implication RC2 must hold:

- Let be $g \varepsilon F\Omega$ the predecessor of f in $G\Omega$, i.e. $(g,f) \varepsilon R\Omega$. Then according to section 3.2.2 (and with corollaries 4 and 5) must hold for $t \varepsilon T$:

(RC2) $$\rho_f(t)=1 \Longrightarrow \rho_g(t)=1$$

- If $|SUCX(f)|>1$ then in analogy to section 6.1.1 there are additional $|SUCX(f)|-1$ constraints of the form RC1.

6.3. The relevances of S-fragments

Let be $f \varepsilon F\Omega$ a S-fragment.

6.3.1. CF-representations

According to theorem 1 ρ_f is equal to a relevance expression with the relevances of O-, X- or entry-fragments $f_{0,j}$, say, $1 \leq j \leq m(0)$:

$$\rho_f \equiv OR_{j=1}^{m(0)} \rho_{0,j}$$

Since a CF-representation $C(f_{0,j})$ can be constructed for each of the fragments $f_{0,j}$ according to the preceding sections, a CF-representation of f can immediately be derived from this equation:

$$C(f) = +_{j=1}^{m(0)} C(f_{0,j})$$

6.3.2. Constraints

If $|SUCX(f)| \geq 0$, then in addition

$$\rho_f \equiv OR_{j=1}^{m(i)} \rho_{i,j} \qquad 1 \leq i \leq |SUXC(f)|$$

which leads to $|SUCX(f)|$ constraints of the type RC1:

$$OR_{j=1}^{m(0)} \rho_{0,j} \equiv OR_{j=1}^{m(i)} \rho_{i,j} \qquad 1 \leq i \leq |SUCX(f)|$$

6.4. The relevances of the example system

We apply the results of the preceding sections to the example system and determine its relevances.

a) The decomposition Ω of F (figure 4) leads to the following equations (corollary 5a):

* $\rho_{1.1} \equiv \rho_2 \equiv \rho_{11}$	* $\rho_{2.1} \equiv \rho_{13}$	* $\rho_{10.1} \equiv \rho_{17}$
* $\rho_{1.2} \equiv \rho_9 \equiv \rho_{12}$	* $\rho_{9.1} \equiv \rho_{14}$	* $\rho_{10.2} \equiv \rho_{18}$
$\rho_{1.3} \equiv \rho_3 \equiv \rho_4$	* $\rho_{8.1} \equiv \rho_{15}$	* $\rho_{10.3} \equiv \rho_{20}$
$\rho_{1.5} \equiv \rho_8$	* $\rho_{8.2} \equiv \rho_{16}$	$\rho_{5.1} \equiv \rho_6$
$\rho_{1.6} \equiv \rho_{10}$	* $\rho_{8.3} \equiv \rho_{19}$	* $\rho_{5.2} \equiv \rho_7$
	$\rho_{22} \equiv \rho_{22.1}$	* $\rho_{4.2.2} \equiv \rho_{21}$

Starred equations involve the relevance of a characteristic fragment, therefore in these cases a CF-expression and a CF-representation of the pertaining fragments is already given.

b) CF-representations of the X- and entry-fragments:

CF does not contain the X-fragments 1.3, 1.5, 1.6, 4.2, 5.1 and the entry-fragment 1 (cf. section 4.2.2). CF-representations C(f) of these fragments according to section 6.1:

f	SUCX(f)	C(f)
4.2	{4.2}	{ 4.2.1 , 4.2.2 }
1.3	{4}	{ 4.1 } + C(4.2) = { 4.1 , 4.2.1 , 4.2.2 }
1.5	{8}	{ 8.1 , 8.2 }
1.6	{10}	{ 10.1,10.2}
1	{1}	{ 1.1 , 1.2 , 1.4 } + C(1.3) + C(1.5) + C(1.6) =
		= { 1.1 , 1.2 , 1.4 , 4.1 , 4.2.1 , 4.2.2 ,
		8.1 , 8.2 , 10.1 , 10.2 }
5.1	{6}	{ 6.1 , 6.2 }

c) CF-representations of the O-fragments:

Since here the O-fragments are characteristic fragments, nothing needs to be done: C(f)={f}.

d) CF-representations of the S-fragments:

According to section 6.3 there are two CF-representations of fragment 5: { 2.1 , 1.1 , 1.4 } (theorem 1) and { 6.1 , 6.2 , 5.2 } because of SUCX(5)={5}$\neq\emptyset$.

For the remainder we set: C(5) := { 6.1 , 6.2 , 5.2 }

Due to SUCX(22)=$\emptyset$ a CF-representation of fragment 22 can be determined only by means of theorem 1. Replacing in the relevance expression for ρ_{22} according to theorem 1 (cf. section 3.1) the relevance $\rho_{1.3}$ with $OR_{f\varepsilon C(1.3)}\rho_f$ (the CF-expression for $\rho_{1.3}$) leads to:

C(22) = { 2.1 , 4.1 , 4.2.1 , 4.2.2 , 9.1 , 8.3 , 10.3 }.

7. The set of partial systems

In this section we shall develop an explicit specification for the mapping ρ and the set of partial systems T.
To this end we assume that a given set of fragments can be the building blocks for at most one partial system; in particular is not possible to construct with a given set of fragments simply through rearranging fragments two different partial systems. This one-to-one correspondence is no stringent restriction: e.g. the textual order of subprograms of a program system in general does not affect the behavior of the program system. (Note: we do not postulate that the order, in which fragments are integrated to form partial systems, does not matter; cf. [13].)

$CF=\{g_i | 1 \le i \le n\}$ be a characteristic set and ρ_i the relevance of g_i for $1 \le i \le n$. Let F_t denote the set of fragments relevant for partial system t.

$$F_t = \{ f \mid f \varepsilon F, \rho_f(t)=1 \}$$
$$= \{ f \mid f \varepsilon F, OR_{g \varepsilon C(f)} \rho_g(t) = 1 \}$$

i.e. F_t is determined by the $|CF|$ relevance values $\rho_i(t)$, $1 \le i \le |CF|$.
Because of the one-to-one correspondence between F_t and t with each t ε T can be associated exactly one element of $B^{|CF|}$, denoted by $\tau(t)$, as follows:

$$\tau(t) := <\rho_1(t), \dots , \rho_{|CF|}(t)>$$

$\tau(t)$ will be referred to as the <u>representation</u> of t ε T.
Note: The representation of the complete system is the element of $B^{|CF|}$ with the relevance value 1 for all components.

With the representations of the partial systems it is possible to explicitly specify mapping ρ:

$$\rho(t,f) = OR_{i \varepsilon I(f)}\ \tau(t)[i] \quad \text{for } t \ \varepsilon\ T,\ f\ \varepsilon\ F$$

where $I(f) := \{ i \mid g_i \ \varepsilon\ CF*C(f) \}$

(i.e. I(f) is the set of the indices of the characteristic fragments in the CF-representation C(f) of f ε F).

Example:

The example system has a characteristic set of |CF|=18 fragments (see figures 4 and 5); these fragments be assigned indices as follows:

i	1	2	3	4	5	6	7	8	9	10
g_i	1.1	2.1	1.2	9.1	1.4	5.2	5.3	6.1	6.2	4.1

i	11	12	13	14	15	16	17	18
g_i	4.2.1	4.2.2	8.1	8.2	8.3	10.1	10.2	10.3

Then, the representation of partial system t_ins (appendix II) is

$$\tau(t\_ins)=(1,0,1,0,0,0,1,1,0,0,0,0,1,0,0,0,0,0):$$

t_ins is the partial system with the characteristic fragments 1.1, 1.2, 5.3, 6.1 and 8.1.

For fragment 22 we have (cf. section 6.4d) I(22)={2,4,10,11,12,15,18}, thus: $\rho(t\_ins,22) = OR_{i \varepsilon I(22)}\tau(t\_ins)[i] = 0$.

The complete list of the sets I(f) for the example system and the relevance values for t_ins are given in appendix III.

Note that there is no partial system t of the example system with a representation $(0,1,\ \dots\ ) \varepsilon B^{18}$: because of $\rho(t,2.1)=1 \implies \rho(t,1.1)=1$ for each t ε T (cf. section 6.2, RC2) $\tau(t)[1]$ must be 1 whenever $\tau(t)[2]=1$.

In general $\tau\ \varepsilon\ B^{|CF|}$ in order to be a representation of a partial system t ε T must satisfy <u>restrictions</u> of the form

$$OR_{i\varepsilon I1}\tau[i] \;=\; OR_{i\varepsilon I2}\tau[i]$$
$$OR_{i\varepsilon I1}\tau[i]=1 \implies OR_{i\varepsilon I2}\tau[i]=1$$

with $I1,I2 \subseteq \{1, \ldots, |CF|\}$.

A subset of these restrictions is implied by the fragment graph: relevance constraints RC1 and RC2, which are obtained as a "side product" with the determination of the relevances according to section 6, must be satisfied by each partial system.

From these constraints restrictions for $\tau \; \varepsilon \; B^{|CF|}$ are derived by

T1: replacing relevances with their CF-expressions and

T2: substituting in the resulting relevance expressions $\tau[i]$ for each ρ_i or $\rho_i(t)$, respectively.

Since in general there may be several CF-representations for a fragment a single constraint may give rise to several different restrictions for τ. In order to obtain all restrictions the sets of CF-representations of the fragments involved in the constraints must be computed. To this end we define:

DEFINITION 7:

M_1 and M_2 be two sets, $S_1 \subseteq \P(M_1)$, $S_2 \subseteq \P(M_2)$.

$$S_1 ¤ S_2 := \{\; s_1+s_2 \mid s_1 \varepsilon S_1,\; s_2 \varepsilon S_2 \;\}$$

Explanation:

$S_1 ¤ S_2$ is the set consisting of the unions of the pairs of $S_1 \times S_2$:

- ¤ is commutative
- $|S_1|=|S_2|=1 \implies S_1 ¤ S_2 = \{s_1+s_2\}$

Let C_ALL(f) denote the set of all CF-representations of $f \varepsilon F$. It is sufficient to consider $f \varepsilon F\Omega$ with $|PRED(f)| \leq 1$, since the constraints to be manipulated according to T1 involve relevances of O-, X- or entry-fragments only:

$$C\_ALL(f) := \begin{cases} \{f\} & : SUCX(f)=\emptyset \text{ (i.e. } f \varepsilon CF) \\ +_{i=1}^{n} C\_ALL(f_i) & : SUCX(f)=\{f_i \mid 1 \leq i \leq n\} \end{cases}$$

where $n=|SUCX(f)|>0$, $C\_ALL(f_i) = C\_ALL(f_{i,1})¤\ldots¤C\_ALL(f_{i,m(i)})$ and

$X(f_i)=\{f_{i,j} \mid 1 \leq j \leq m(i)\}$

This leads to the following recursive algorithm, basically an extension of algorithm 2:

ALGORITHM 3: Determination of all CF-representations

Input : fragment graph (F,R,X,O,E); O-, X- or entry-fragment $f \varepsilon F$

Output: set C of all CF-representations of f

Algorithm:

C = C_ALL(f)

with:

```
FUNCTION C_ALL(f)
   IF (SUCX(f)=Ø)
       THEN C_ALL = {f}
       ELSE DO
            let be SUCX(f)={f_i|1≤i≤n}, n=|SUCX(f)| and
                   X(f_i)={f_i,j|1≤j≤m(i)} for 1≤i≤n.
            C_ALL = +_i=1^n( C_ALL(f_i,1)¤...¤C_ALL(f_i,m(i)) )
            END
END
```

Since F is finite and each path of (F,R) is acyclic, algorithm 3 yields in a finite number of steps the set of all CF-representations.

Not all restrictions are inherent to the fragment system and, thus, mechanically derivable. Additional restrictions may be necessary for the characterization of the set of correct partial systems. Such restrictions may e.g. stem from properties of the system interface: due to the semantics of the operations provided it may be the case that

- a set of two or more operations will always be used together
- the execution of an operation O implies the execution of one of n operations O_i, $1 \le i \le n$ (as prerequisite or consequence).

Such properties are statements on the relevances of fragments, which take the form of relevance constraints RC1 and RC2, respectively, and thus can be transformed into restrictions.

As will be exemplified below restrictions may also correspond to properties of the system that are not modeled by fragment systems as e.g. the "semantics" of modules.

Example: The restrictions of the example system

1) Inherent restrictions

Inherent to the fragment graph (figure 4) are the following constraints:

- for Ω-sets with an O-fragment as root (section 6.2.2, RC2):

$$\rho(t,\ 2.1)=1 \implies \rho(t,1.1)=1$$

$$\rho(t,\ 9.1)=1 \implies \rho(t,1.2)=1$$

$$\rho(t,\ 5.3)=1 \implies \rho(t,\ 5\ )=1$$

$$\rho(t,\ 8.3)=1 \implies \rho(t,1.5)=1$$

$$\rho(t,10.3)=1 \implies \rho(t,1.6)=1$$

- for the Ω-set with S-fragment 5 as root: <u>one</u> RC1-constraint because of $|SUCX(5)|=1$ (cf. section 6.3.2)

$$\rho_{2.1} \text{ OR } \rho_{1.1} \text{ OR } \rho_{1.4} \equiv \rho_{5.1} \text{ OR } \rho_{5.2}$$

No additional RC1-constraints can be inferred from the fragment system, since $|SUCX(f)| \leq 1$ for the root-fragments f of the other Ω-sets.

With the indices of CF from above and utilizing the CF-expressions of section 6.4 (cf. also the sets C(f) and I(f) of appendix III) these constraints correspond to the following restrictions:

$\tau[\ 2]=1 \implies \tau[\ 1]=1$

$\tau[\ 4]=1 \implies \tau[\ 3]=1$

because there are 2 CF-representations of fragment 5 (cf. section 6.4d):

$\tau[\ 7]=1 \implies \tau[\ 8]$ OR $\tau[\ 9]$ OR $\tau[\ 6] = 1$

$\tau[\ 7]=1 \implies \tau[\ 2]$ OR $\tau[\ 1]$ OR $\tau[\ 5] = 1$

$\tau[15]=1 \implies \tau[13]$ OR $\tau[14] = 1$

$\tau[18]=1 \implies \tau[16]$ OR $\tau[17] = 1$

$\tau[2]$ OR $\tau[1]$ OR $\tau[5] = \tau[8]$ OR $\tau[9]$ OR $\tau[6]$

2) Additional restrictions

As to the operations provided by DBMS description in appendix II states ("$\rightarrow$" stands for "requires"):

- OPEN$\rightarrow$CLOSE, CLOSE$\rightarrow$OPEN (operations OPEN, CLOSE must be used together):

  $\rho_{2.1} \equiv \rho_{9.1}$, $\rho_{1.1} \equiv \rho_{1.2}$
- INSERT $\rightarrow$ OPEN: $\rho_{8.1}(t)$ OR $\rho_{8.2}(t) = 1 \implies \rho_{1.1}(t)=1$
- FIND $\rightarrow$ OPEN: $\rho_{4.1}(t)$ OR $\rho_{4.2.1}(t)$ OR $\rho_{4.2.2}(t) = 1 \implies \rho_{1.1}(t)=1$
- GET $\rightarrow$ FIND: $\rho_{6.1}(t)$ OR $\rho_{6.2}(t)$ OR $\rho_{5.2}(t) = 1 \implies$

  $\implies \rho_{4.1}(t)$ OR $\rho_{4.2.1}(t)$ OR $\rho_{4.2.2}(t) = 1$

These five relevance constraints yield five restrictions:

$\tau[2]=\tau[4] \quad \tau[1]=\tau[3]$

$\tau[13]$ OR $\tau[14] = 1 \implies \tau[1]=1$

$\tau[10]$ OR $\tau[11]$ OR $\tau[12] = 1 \implies \tau[1]=1$

$\tau[8]$ OR $\tau[9]$ OR $\tau[6] = 1 \implies \tau[10]$ OR $\tau[11]$ OR $\tau[12] = 1$

In order to perform operation OPEN a partial system must retrieve information from system catalogues (see appendix II). To this end the same module (fragment 5) is invoked as the "ordinary" user operation GET does, with the effect, however, that only fragment 6.1, i.e. algorithm A8, is executed, but never fragments 6.2 or 5.2; i.e.:

OPEN $\rightarrow$ algorithm A8: $\rho_{1.1}(t)=1 \implies \rho_{6.1}(t)=1$

This constraint cannot be inferred from the fragment system! It yields the additional restriction $\tau[1]=1 \implies \tau[8]=1$

The set of partial systems of a fragment system can be viewed as a subset of $B^{|CF|}$. If NC is the number of characteristic fragments not involved in any constraint, then for the number $|T|$ of partial systems holds:

$$2^{NC} \leq |T| \leq 2^{|CF|}$$

The following result may be used to obtain better upper bounds for $|T|$:

THEOREM 12:

Let be $x=<x_1, \ldots , x_m> \varepsilon B^m$, $y=<y_1, \ldots , y_n> \varepsilon B^n$.

1) there are $(2^m-1)*(2^n-1)+1$ different elements $(x,y) \varepsilon B^{m+n}$ such that

$$OR_{i=1}^m x_i = OR_{i=1}^n y_i$$

2) there are $(2^m-1)*(2^n-1)+2^n$ different elements $(x,y) \varepsilon B^{m+n}$ such that

$$OR_{i=1}^m x_i =1 \quad ==> \quad OR_{i=1}^n y_i =1$$

Proof:

There are 2^i-1 elements of B^i with at least one component equal to 1, $1 \le i$.

For each $x \varepsilon B^m$ with $OR_{i=1}^m x_i = 1$ there are 2^n-1 elements $y \varepsilon B^n$ such that $OR_{i=1}^n y_i =1$, thus there are $(2^m-1)*(2^n-1)$ elements $(x,y) \varepsilon B^{m+n}$ such that both sides of the first equation evaluate to 1.

This concludes the proof of the first statement, since both sides of the equation evaluate to 0 if and only if $x=<0,\ldots,0>$ and $y=<0,\ldots,0>$.

The second statement is a consequence of the fact that, if $OR_{i=1}^m x_i =0$ is true, the implication holds for each $y \varepsilon B^n$ (and $|B^n|=2^n$).

□

This theorem says that there are at most $(2^m-1)*(2^n-1)+1$ partial systems satisfying $OR_{i\varepsilon I1}\tau[i] = OR_{i\varepsilon I2}\tau[i]$ and at most $(2^m-1)*(2^n-1)+2^n$ partial systems satisfying $OR_{i\varepsilon I1}\tau[i]=1 ==> OR_{i\varepsilon I2}\tau[i]=1$, where $m=|I1|$, $n=|I2|$ and $I1*I2=\emptyset$.

8. Conclusions

The notion of fragment system has been presented. Originally, this concept has been designed specifically as a model for program systems with partial systems; it is, however, generally applicable to families of software systems including their job control programs, documentation, test data as well as hardware systems. It models the interdependencies among the building blocks, out of which the members of a system family are constructed. For software systems these interdependencies may represent the data and control flow of the program system [12, 13] or information on interconnections among configurations, versions, revisions of the system family [20].
These interdependencies describe the structure each partial system adheres to and, thus, indirectly the set T of possible partial systems. An explicit representation of T as a subset of $\{0,1\}^n$ has been constructed, where n is minimal and its elements satisfy equivalences and implications of Boolean expressions with OR-operators only. Restrictions of this type are inherent to fragment systems and can be algorithmically inferred from them. Additional restrictions of this form representing e.g. semantics of the system interface may be necessary for the characterization of the set of partial systems.

The concepts and results of this paper have been employed for the generation of partial systems of an operational database management system (details are given in [11, 14, 15]): A specification system collects, among other things, the relevance values of the 32 characteristic fragments, constructs the representation τ of the desired partial system and passes τ to a program generator. For program generation at first job control programs for source program generation, compile and link steps are generated. These JCL-programs form partial systems of the JCL-programs of the given database management system, i.e. in this case besides source code fragments there are also fragments with JCL-statements. Their relevances form relevance expressions with relevances of code fragments.

Restrictions involving OR-operators only are sufficient for the characterization of partial systems of a program system in the sense of [12, 13]. For the use in general customizing systems, as e.g. MACS [5], or version control systems [3, 19, 20] two extensions are necessary:

First, general logical expressions must be allowed as restrictions:
In system families there are typically groups of components, e.g. revisions of a module, that exclude each other in that for the construction of a member of that family at most one element of a group of components may be used. In terms of relevances this means that the relevance of a fragment may be the "negation" of some relevance expression. This type of structural information cannot be modeled with fragment systems, rather it must be added (by the software engineer, say) in form of logical expressions involving the NOT-operator to the restrictions inferable mechanically.

A second extension refers to the fragment concept as such. The relevance of a fragment may depend also on the "environment" of the system: e.g. a module of a family of program systems may be required, only if the system is intended to run under a particular operating and/or hardware system. In order to be able to express options of this kind uniformly in terms of relevances, the set of fragments representing "real" system components can be supplemented with "virtual" fragments that represent such aspects of the environment. In the application mentioned above e.g. such a fragment was defined in order to be able to specify the compiler (FORTRAN IV or FORTRAN 77) to be used for the compilation step.

APPENDIX I: Notations, definitions

This appendix gives the basic definitions and notations of set and graph theory used in this paper (cf. e.g. [1], [2]).

Let M and N be sets:

A: The cardinality of M, denoted: $|M|$, is the number of elements in M.
$\emptyset$ denotes the empty set, i.e. $|\emptyset|=0$.

B: $M*N$ denotes the intersection $M+N$ the union of M and N. The union of n sets M_i, $1\le i\le n$, is denoted by: $+_{j=1}^{n} M_j$.
The Cartesian product of M and N is the set

$$M\times N := \{ (m,n) \mid m \in M, n \in N \}$$

$\P(M)$ denotes the power set of M, i.e. the set of all subsets of M.
A set of subsets M_i, $1\le i\le n$, is a disjoint decomposition of M if $M_i*M_j=\emptyset$ for $i\ne j$ and for each $m \in M$ there is some k, $1\le k\le n$, with $m \in M_k$.

C: A set $R \subseteq M\times N$ is called a (binary) relation from M to N.
A partial order on M is a relation $R \subseteq M\times M$ such that

- $(x,x) \in R$ for each $x \in M$ (R is reflexive)
- $(x,y) \in R$, $(y,x) \in R \implies x=y$ (R is antisymmetric)
- $(x,y) \in R$, $(y,z) \in R \implies (x,z) \in R$ (R is transitive)

D: With R a partial order on M a set $L \subseteq M$ is called a list, if for each pair $(x,y) \in L \times L$ either $(x,y) \in R$ or $(y,x) \in R$. L[i] denotes the i-th element of list L, L is written using angular brackets:

$$L = < L[1], L[2], \ldots, L[i], \ldots >$$

E: A mapping $f: M \dashrightarrow N$ is a relation $f \subseteq M \times N$ such that

$$(x,y) \in f,\ (x,z) \in f \implies y=z$$

Two mappings $f: M \dashrightarrow N$, $g: M \dashrightarrow N$ are said to be equal, denoted: $f \equiv g$, if $f(x)=g(x)$ holds for each $x \in M$.

F: A directed graph is a pair $G=(M,R)$, where M is a set and R a binary relation $R \subseteq M \times M$. The elements of M are called the vertices, the elements of R the edges of G.

Let k, k_1, k_2 be vertices of a directed graph $G=(M,R)$:

- The predecessors of k in G are the vertices of the set

$$PRED(k) := \{ x \mid x \in M, (x,k) \in R \}$$

- The successors of k in G are the vertices of the set

$$SUCC(k) := \{ x \mid x \in M, (k,x) \in R \}$$

- A path P from x to y is a list of $n \geq 2$ vertices k_i, $1 \leq i \leq n$, with $(k_i, k_{i+1}) \in R$ for $1 \leq i \leq n-1$ and $k_1=x$, $k_n=y$. P is a cycle if $x=y$. A path from a set K of vertices to y is a path from some element $x \in K$ to y.

- k_2 is said to be accessible from k_1, if there is a path from k_1 to k_2.

G: A <u>tree</u> is a directed graph $G=(M,R)$ such that:

1. G has no cycles
2. there is exactly one vertex $r \in M$ with $PRED(r)=\emptyset$; r is the <u>root</u> of G
3. $k \in M$, $k \neq r$ $\Longrightarrow$ $|PRED(k)|=1$
4. for each vertex $k \in M$, $k \neq r$, there exists a path from r to k.

A vertex $k \in M$ without successors, i.e. $|SUCC(k)|=0$, is called a <u>leaf</u> of G.

APPENDIX II: The example system DBMS

The program system of fig. A-1, called DBMS, is used throughout this paper for demonstration purposes (cf. [11]). It sketches the implementation of a database management system with a simple, one-tuple database interface consisting of the six operations of table A-1.

operation	semantics
OPEN	acquire a lock on a relation; in order to access the tuples of a relation the relation must be locked by the application program
CLOSE	release a lock; at the end of a transaction all locks acquired (with OPEN) must be released by the application program
FIND	select a set of tuples of a relation satisfying a qualification, make them available in a QSS
GET	retrieve a tuple of a QSS
INSERT	insert a tuple into a relation
DELETE	delete a tuple from a relation

Table A-1: The operations supported DBMS

For the implementation of relations DBMS supports two storage structures (cf. variables FILE_TYPE of program units INSERT and DELETE of fig. A-1), access paths can be supported through "sequential search", hashing or an inverted files.

There are two access methods (variable ACCESS_TYPE of program unit GET): "sequential search" und "direct access" (employing lists of tuple identifiers TID).

```
PROCEDURE DBMS
   IF (OP<1 OR OP>6)
      THEN return 'operation unknown'
   CASE OP OF
     1: OPEN
     2: CLOSE
     3: FIND
     4: GET
     5: INSERT
     6: DELETE
   END
END
```

```
PROCEDURE OPEN
   OPEN_RF
   OPEN_IF
END
```

```
PROCEDURE CLOSE
   CLOSE_RF
   CLOSE_IF
END
```

```
PROCEDURE FIND
   USE INDEXES
   evaluate INDEX_TABLE
   STRTGY
   return qss
END
```

```
PROCEDURE STRTGY
   determine access-strategy and
    set ACCESS_TYPE
   CASE ACCESS_TYPE OF
     1: build seq.search qss
     2: BEGIN
        CASE FILE_TYPE OF
          1: calculate tid
          2: . . . . . . . .
             RETRIEVE_TID_LIST
             . . . . . . . .
        END
        build direct-access qss
        END
   END
END
```

```
PROCEDURE GET
   NEXT_TUPLE:
   CASE ACCESS_TYPE OF
     1: NEXT_SEQ
        . . . . . . . .
     2: NEXT_TID
        . . . . . . . .
   END
   IF (qualification is not satisfied)
      THEN GO TO NEXT_TUPLE
END
```

```
PROCEDURE NEXT_SEQ
   CASE FILE_TYPE OF
     1: next_1
     2: next_2
   END
END
```

```
PROCEDURE NEXT_TID
   return next tid of tid-list
END
```

```
PROCEDURE RETRIEVE_TID_LIST
   . . . . . . . . .
END
```

```
PACKAGE INDEXES
  INDEX_TABLE: ARRAY OF INTEGER
END
```

```
PROCEDURE OPEN_RF
  . . . . . . .

  GET
  . . . . . . .
END
```

```
PROCEDURE OPEN_IF
  USE INDEXES
  . . . . . . .
  GET
  . . . . . . .
END
```

```
PROCEDURE CLOSE_RF
  . . . . . . .

END
```

```
PROCEDURE CLOSE_IF
  USE INDEXES
  . . . . . . .
END
```

```
PROCEDURE INSERT
  CASE FILE_TYPE OF
    1: INSERT_1
    2: INSERT_2
  END
  INSERT_TID
END
```

```
PROCEDURE DELETE
  CASE FILE_TYPE OF
    1: DELETE_1
    2: DELETE_2
  END
  DELETE_TID
END
```

```
PROCEDURE INSERT_1
  . . . . . . .
END
```

```
PROCEDURE DELETE_1
  . . . . . . .
END
```

```
PROCEDURE INSERT_2
  . . . . . . .
END
```

```
PROCEDURE DELETE_2
  . . . . . . .
END
```

```
PROCEDURE INSERT_TID
  USE INDEXES
  . . . . . . .
END
```

```
PROCEDURE DELETE_TID
  USE INDEXES
  . . . . . . .
END
```

Fig. A-1: The example system DBMS

Table A-2 delineates the implementation of the operations of table A-1, with the pertaining program fragments in angular brackets (statements, subroutine calls); the right-most column contains the "names" of the algorithms of DBMS in form of the integers 1 through 17.

Partial system t_ins of DBMS (cf. [11]):

Let A be a database application, the only purpose of which is to collect and store data in (one or several relations of) a database.

We assume that storage structure 1 is used for the implementation of the relations to be operated on by A; since (i) there are no retrieval operations to be supported and (ii) the maintenance of inverted files slows down update operations, no inverted files will be employed for A.

For this type of application algorithm 12 (insertion according to storage structure 1) suffices for the implementation of operation INSERT. A has to lock and unlock the relations to be accessed (operations OPEN, CLOSE; see table A-1): for these purposes only algorithms 1 and 3, respectively, are necessary for A (and not algorithm 2 or 4, since here inverted files will not be encountered). It is assumed that the relations ("system catalogues") holding the database schema are implemented according to storage structure 1, too: access to system catalogues (the call to GET in OPEN_RF!) requires algorithms 8 and 11.

The partial system of DBMS providing the operations OPEN, CLOSE and INSERT with these five algorithms is referred to as t_ins.

operation	implementation	algorithm
OPEN	- lock relation <OPEN_RF> - if inverted files exist for the relation, acquire locks and update INDEX_TABLE <OPEN_IF>	1 2
CLOSE	- release lock for relation <CLOSE_RF> - if inverted files exist for the relation, release locks and update INDEX_TABLE <CLOSE_IF>	3 4
FIND	- determine in INDEX_TABLE the available inverted files <evaluate INDEX_TABLE> - determine access technique and create a subset (QSS) for sequential search <build seq.search qss> or direct access employing: hashing <calculate tid> TID-list via inverted file <RETRIEVE_TID_LIST>	 5 6 7
GET	- retrieve next tuple through: sequential search <NEXT_SEQ> according to storage structure 1 <next_1> or storage structure 2 <next_2> direct access with a TID-list <NEXT_TID> - check, whether qualification is satisfied	 8 9 10 11
INSERT	- insert a tuple according to storage structure 1 <INSERT_1> or 2 <INSERT_2> - determine in INDEX_TABLE the available inverted files and perform updates, where applicable <INSERT_TID>	 12 13 14
DELETE	- delete a tuple according to storage structure 1 <DELETE_1> or 2 <DELETE_2> - determine in INDEX_TABLE the available inverted files and perform updates, where applicable <DELETE_TID>	 15 16 17

Table A-2: The algorithms of DBMS

A four-step method yields a fragmentation of DBMS as shown in fig. A-2 (for details the reader is referred to [12]):

- Each program unit is defined a fragment. This leads to fragments 1 through 22.
- Each fragment with optional code is partitioned into subfragments that enclose these pieces of code: in this way we obtain e.g. the subfragments 1.1 through 1.6 of fragment 1 or the subfragments 4.2.1 and 4.2.2 of fragment 4.2, which itself is a subfragment. (Dots in the fragment names indicate the nesting of fragments).

  A fragment f may have subsets X(f), such that with the execution of f exactly one fragment of X(f) is executed. Fragments with this property are the X - f r a g m e n t s of f; the fragments that are optional without any restriction are called the O - f r a g m e n t s of f.

  The sets of X-fragments of this example:

  X(1) = { 1.1 , 1.2 , 1.3 , 1.4 , 1.5 , 1.6 } ,

  X(4) = { 4.1 , 4.2 } , X(4.2) = { 4.2.1 , 4.2.2 } ,

  X(5) = { 5.1 , 5.2 } , X(6) = { 6.1 , 6.2 } ,

  X(8) = { 8.1 , 8.2 } , X(10) = {10.1 ,10.2 } .

  The O-fragments: 2.1 , 5.3 , 8.3 , 9.1 , 10.3

- After the definition of X- and O-fragments additional fragments are introduced according to the following rules:

  a) For each fragment f with statements that can be executed only when subfragments of f are executed define fragments comprising these statements.

  b) For each fragment f with declarations of data objects that are referenced only by statements of subfragments of f define fragments

```
1        PROCEDURE DBMS
1           . . . . . . . . . .
1           IF (OP<1 OR OP>6)
1               THEN return 'operation unknown'
1           CASE OP OF
1.1           1: OPEN
1.2           2: CLOSE
1.3           3: FIND
1.4           4: GET
1.5           5: INSERT
1.6           6: DELETE
1           END
1        END
2        PROCEDURE OPEN
2           OPEN_RF
2.1         OPEN_IF
2        END
3        PROCEDURE FIND
3           USE INDEXES
3           evaluate INDEX_TABLE
3           STRTGY
3           return qss
3        END
4        PROCEDURE STRTGY
4           determine access-strategy and
4            set ACCESS_TYPE
4           CASE ACCESS_TYPE OF
4             1:
4.1              build seq.search qss
4.            2: BEGIN
4.2              CASE FILE_TYPE OF
4.2.1              1: calculate tid
4.2.2              2: . . . . . . . . .
4.2.2                 RETRIEVE_TID_LIST
4.2.2                 . . . . . . . . .
4.2              END
4.2              build direct-access qss
4                END
4           END
4        END
5        PROCEDURE GET
5           NEXT_TUPLE:
5           CASE ACCESS_TYPE OF
5             1: . . . . . . . .
5.1              NEXT_SEQ
5.1              . . . . . . . .
5             2: . . . . . . . .
5.2              NEXT_TID
5.2              . . . . . . . .
5           END
5.3         IF (qualifikation is not satisfied)
5.3            THEN GO TO NEXT_TUPLE
5        END
6        PROCEDURE NEXT_SEQ
6           CASE FILE_TYPE OF
6             1:
6.1              next_1
6             2:
6.2              next_2
6           END
6        END
7        PROCEDURE NEXT_TID
7          return next tid of tid-list
7        END
```

Fig. A-2: Fragmentation of the example system DBMS

```
8       PROCEDURE INSERT
8          CASE FILE_TYPE OF
8             1:
8.1              INSERT_1
8             2:
8.2              INSERT_2
8          END
8.3        INSERT_TID
8       END
9       PROCEDURE CLOSE
9          CLOSE_RF
9.1        CLOSE_IF
9       END
10      PROCEDURE DELETE
10         CASE FILE_TYPE OF
10            1:
10.1             DELETE_1
10            2:
10.2             DELETE_2
10         END
10.3       DELETE_TID
10      END
11      PROCEDURE OPEN_RF
11        . . . . . . . .
11        GET
11        . . . . . . . .
11      END
12      PROCEDURE CLOSE_RF
12        . . . . . . . .
12      END
13      PROCEDURE OPEN_IF
13        USE INDEXES
13        . . . . . . . . .
13        GET
13        . . . . . . . . .
13      END
14      PROCEDURE CLOSE_IF
14        USE INDEXES
14        . . . . . . . . .
14      END
15      PROCEDURE INSERT_1
15        . . . . . . . . .
15      END
16      PROCEDURE INSERT_2
16        . . . . . . . . .
16      END
17      PROCEDURE DELETE_1
17        . . . . . . . . .
17      END
18      PROCEDURE DELETE_2
18        . . . . . . . . .
18      END
19      PROCEDURE INSERT_TID
19        USE INDEXES
19        . . . . . . . . .
19      END
20      PROCEDURE DELETE_TID
20        USE INDEXES
20        . . . . . . . . .
20      END
21      PROCEDURE RETRIEVE_TID_LIST
21        . . . . . . . . .
21      END
22      PACKAGE INDEXES
22.1       INDEX_TABLE: ARRAY OF INTEGER
22      END
```

Fig. A-2: Fragmentation of the example system DBMS (continued) comprising these declarations.

c) For each global data object define a fragment comprising its declaration.

In this way we obtain fragment 22.1.

The program lines that form fragment f (and thus also the subfragments of f) are marked with the name of that fragment or one of its subfragment at the left of the program text. E.g. the lines of code of fig. A-2 with "1", "1.1", ..., "1.6" belong to fragment 1.

A fragment f can be considered a list of substrings of the source program and (sub)fragments $f_i \neq f$; with each fragment is associated "substitute" code (cf. [12]). The generation of the program of a partial system can informally be described as follows:
Starting with the first fragment the relevance value of each fragment is determined. In case a fragment is not relevant for t the substitute of that fragment is appended to the program text produced so far (the empty string is assumed as the initial value of the program to be generated); otherwise the fragment is "processed":

- if it is a substring of the source program, this string is appended to the program text generated so far
- if it is a list of substrings and fragments the substrings are appended to the program text generated so far, for each fragment as just described the relevance value is determined, ..., etc.

(A formal treatment of this process is given in [12].)

Figure A-3 shows the program of partial system t_ins. It is the result of applying this procedure to the fragmentation of figure A-2 with the relevance values ρ(t_ins,f) of appendix III.

```
1       PROCEDURE DBMS
1
1          IF (OP<1 OR OP>6)
1             THEN return 'operation unknown'
1          CASE OP OF
1.1          1: OPEN
1.2          2: CLOSE
**
1.5          5: INSERT
**
1          END
1       END
2       PROCEDURE OPEN
2           OPEN_RF
**
2       END
5       PROCEDURE GET
5          NEXT_TUPLE:
5          CASE ACCESS_TYPE OF
5            1: . . . . . . . .
5.1             NEXT_SEQ
5.1             . . . . . . . .
5            2: . . . . . . . .
**              return 'illegal access-type'
5          END
5.3        IF (qualifikation is not satisfied)
5.3           THEN GO TO NEXT_TUPLE
5       END
6       PROCEDURE NEXT_SEQ
6          CASE FILE_TYPE OF
6            1:
6.1             next_1
6            2:
**              return 'storage structure not accessible'
6          END
6       END
8       PROCEDURE INSERT
8          CASE FILE_TYPE OF
8             1:
8.1              INSERT_1
8             2:
**               return 'storage structure not accessible'
8          END
**
8       END
9       PROCEDURE CLOSE
9          CLOSE_RF
**
9       END
11      PROCEDURE OPEN_RF
11        . . . . . . . .
11        GET
11        . . . . . . . .
11      END
12      PROCEDURE CLOSE_RF
12        . . . . . . . .
12      END
15      PROCEDURE INSERT_1
15        . . . . . . . . .
15      END
```

Fig. A-3: The program of the partial system t_ins
(**: marks substitute code, within program units only!)

APPENDIX III: The fragments of DBMS and their relevances

With the characteristic set CF of section 4.2 the table below contains for each fragment of the example system a characteristic representation C(f) and the indices I(f) of these fragments according to section 7.

The rightmost column lists the relevance values ρ_f(t_ins): t_ins is the partial system with the characteristic fragments 1.1, 1.2, 5.3, 6.1 and 8.1. Thus: ρ(t_ins,f)=1 <==> {1.1,1.2,5.3,6.1,8.1}*C(f)$\neq\emptyset$

<==> { 1 , 3 , 7 , 8 , 13}*I(f)$\neq\emptyset$

f	C(f)	I(f)	ρ(t_ins,f)
1	{ 1.1 , 1.2 , 1.4 , 4.1 , 4.2.1 , 4.2.2 , 8.1 , 8.2 , 10.1 , 10.2 }	{ 1, 3, 5 , 10, 11, 12, 13, 14, 16, 17 }	1
1.1	{ 1.1 }	{ 1 }	1
1.2	{ 1.2 }	{ 3 }	1
1.3	{ 4.1 , 4.2.1 , 4.2.2 }	{ 10, 11, 12 }	0
1.4	{ 1.4 }	{ 5 }	0
1.5	{ 8.1 , 8.2 }	{ 13, 14 }	1
1.6	{ 10.1,10.2 }	{ 16, 17 }	0
2	{ 1.1 }	{ 1 }	1
2.1	{ 2.1 }	{ 2 }	0
3	{ 4.1 , 4.2.1 , 4.2.2 }	{ 10, 11, 12 }	0
4	{ 4.1 , 4.2.1 , 4.2.2 }	{ 10, 11, 12 }	0
4.1	{ 4.1 }	{ 10 }	0
4.2	{ 4.2.1 , 4.2.2 }	{ 11, 12 }	0
4.2.1	{ 4.2.1 }	{ 11 }	0
4.2.2	{ 4.2.2 }	{ 12 }	0
5	{ 6.1 , 6.2. , 5.2 }	{ 8, 9, 6 }	1
5.1	{ 6.1 , 6.2 }	{ 8, 9 }	1
5.2	{ 5.2 }	{ 6 }	0
5.3	{ 5.3 }	{ 7 }	1
6	{ 6.1 , 6.2 }	{ 8, 9 }	1
6.1	{ 6.1 }	{ 8 }	1
6.2	{ 6.2 }	{ 9 }	0
7	{ 5.2 }	{ 6 }	0
8	{ 8.1 , 8.2 }	{ 13, 14 }	1
8.1	{ 8.1 }	{ 13 }	1
8.2	{ 8.2 }	{ 14 }	0
8.3	{ 8.3 }	{ 15 }	0
9	{ 1.2 }	{ 3 }	1
9.1	{ 9.1 }	{ 4 }	0
10	{10.1,10.2 }	{ 16, 17 }	0
10.1	{10.1 }	{ 16 }	0
10.2	{10.2 }	{ 17 }	0
10.3	{10.3 }	{ 18 }	0
11	{ 1.1 }	{ 1 }	1
12	{ 1.2 }	{ 3 }	1
13	{ 2.1 }	{ 2 }	0
14	{ 9.1 }	{ 4 }	0
15	{ 8.1 }	{ 13 }	1
16	{ 8.2 }	{ 14 }	0
17	{10.1 }	{ 16 }	0
18	{10.2 }	{ 17 }	0
19	{ 8.3 }	{ 15 }	0
20	{10.3 }	{ 18 }	0
21	{4.2.2}	{ 12 }	0
22	{ 2.1 , 4.1 , 4.2.1 , 4.2.2 , 9.1 , 8.3 , 10.3 }	{ 2, 4, 10, 11, 12, 15, 18 }	0
22.1	{ 2.1 , 4.1 , 4.2.1 , 4.2.2 , 9.1 , 8.3 , 10.3 }	{ 2, 4, 10, 11, 12, 15, 18 }	0

APPENDIX IV: A Fragment System Analyser

This appendix describes FSA (Fragment System Analyser). FSA is a conversational system that makes available implementations of the algorithms of sections 4.2, 6 and 7.

FSA is written in PROLOG [21] using the IF/Prolog interpreter version 2.1 [22], it runs on a VAX 750 under VMS version 4.1.

The PROLOG programs implementing the various algorithms and the structure of FSA are explained. The reader is assumed to have at least basic knowledge of PROLOG (for an introduction to PROLOG see [21]). Excerpts from a FSA-session are given, where FSA is applied to the fragment system of the example system (appendix II or fig. 1).

1. PROLOG programs

1.1. Describing fragment systems in PROLOG

FSA must be presented the description of the fragment system to be analyzed in form of a PROLOG program in a separate file. FSA consults this file (predicate 'consult' [21]) in the course of initilization and adds the clauses specifying the fragment system to the actual FSA database.

A fragment system (F,R,X,O,E,ρ) is specified for FSA as follows:

- for each $f \varepsilon F$ there is exactly one fact 'fragment(f)'
- for each $(f,g) \varepsilon R$ there is exactly one fact 'edge(f,g)'
- mapping X is implemented as predicate 'x_fragments' with two arguments, where the first argument is a fragment $f \varepsilon F$ and the second a list with the elements of $X(f)$
- mapping O is implemented as predicate 'o_fragments' with two arguments, where the first argument is a fragment $f \varepsilon F$ and the second a list with the elements of $O(f)$.

Remark:

The set E of entry-fragments need not be specified explicitly, it is determined by FSA (cf. consult-file fg_general, below).

Example:

Fig. A-4 shows the PROLOG specification of the example fragment system of section 2.2.

Note that here r u l e s are employed to specify the fragments $f \varepsilon F$ with $X(f)=\emptyset$ and $O(f)=\emptyset$, respectively. In principle also facts 'x_fragments(f,[])' and 'o_fragments(f,[])' could have been used. The scheme of fig. A-4 may be advantageous, if the PROLOG description of the fragment system is to be generated automatically (by a program, which provides a user-friendly interface, say) and not by manually editing of a file.

```
fragment("1").
fragment("1.1").
fragment("1.2").
fragment("1.3").
fragment("1.4").
fragment("1.5").
fragment("1.6").
fragment("2").
fragment("2.1").
fragment("3").
fragment("4").
fragment("4.1").
fragment("4.2").
fragment("4.2.1").
fragment("4.2.2").
fragment("5").
fragment("5.1").
fragment("5.2").
fragment("5.3").
fragment("6").
fragment("6.1").
fragment("6.2").
fragment("7").
fragment("8").
fragment("8.1").
fragment("8.2").
fragment("8.3").
fragment("9").
fragment("9.1").
fragment("10").
fragment("10.1").
fragment("10.2").
fragment("10.3").
fragment("11").
fragment("12").
fragment("13").
fragment("14").
fragment("15").
fragment("16").
fragment("17").
fragment("18").
fragment("19").
fragment("20").
fragment("21").
fragment("22").
fragment("22.1").
```

```
edge("13","22").
edge("13","5").
edge("11","5").
edge("1.4","5").
%
edge("5","5.3").
edge("5","5.1").
edge("5","5.2").
%
edge("5.1","6").
edge("5.2","7").
%
edge("6","6.1").
edge("6","6.2").
%
edge("1.2","9").
edge("9","12").
edge("9","9.1").
edge("9.1","14").
edge("14","22").
%
edge("1.3","3").
edge("3","22").
edge("3","4").
edge("4","4.1").
edge("4","4.2").
edge("4.2","4.2.1").
edge("4.2","4.2.2").
edge("4.2.2","21").
%
edge("1.5","8").
edge("8","8.1").
edge("8","8.2").
edge("8","8.3").
edge("8.1","15").
edge("8.2","16").
edge("8.3","19").
edge("19","22").
%
edge("1","1.1").
edge("1","1.2").
edge("1","1.3").
edge("1","1.4").
edge("1","1.5").
edge("1","1.6").
%
edge("1.1","2").
%
edge("2","2.1").
edge("2","11").
%
edge("2.1","13").
```

Fig. A-4: The PROLOG description of the example fragment system

```
edge('1.6','10').
edge('10','10.1').
edge('10','10.2').
edge('10','10.3').
edge('10.1','17').
edge('10.2','18').
edge('10.3','20').
edge('20','22').
%
edge('22','22.1').
%
% mapping X:
x_fragments('1',    ['1.1','1.2','1.3','1.4','1.5','1.6']).
x_fragments('4',    ['4.1','4.2']).
x_fragments('4.2',['4.2.1','4.2.2']).
x_fragments('5',    ['5.1','5.2']).
x_fragments('6',    ['6.1','6.2']).
x_fragments('8',    ['8.1','8.2']).
x_fragments('10',   ['10.1','10.2']).
x_fragments(F,[]) :- F\='1',F\='4',F\='4.2',F\='5',F\='6',F\='8',F\='10'.
%
% mapping O:
o_fragments('2',    ['2.1']).
o_fragments('5',    ['5.3']).
o_fragments('8',    ['8.3']).

o_fragments('9',    ['9.1']).
o_fragments('10',   ['10.3']).
o_fragments(F,[]) :- F\='2',F\='5',F\='8',F\='9',F\='10'.
```

Fig. A-4: The PROLOG description of the example fragment system (cont.)

1.2. Construction of set Ω: steps 1 and 2 of algorithm 1 (fig. A-5)

Predicate 'omega' constructs the set Ω of the given fragment system, i.e. omega implements steps 1 and 2 of algorithm 1:
each $\omega \in \Omega$ with ROOT(ω)=r and thus finally each $\omega \in \Omega$ corresponds a fact 'omega_set(s,r)' of the database, where s is a list comprising the fragments of ω. The presence of the "dummy"-fact 'omega_set([],dummy)' in the FSA database indicates that Ω has been determined already, i.e. predicate 'omega_set' is available.

Predicate 'omega_0' constructs $\Omega^{(0)}$ (implementation of step 1):

- 'r1s_path(f,g,path)' succeeds if f and g are fragments and list path is a R1-path from f to g and path-{f} contains no S-fragment.
  'r1_set(f,s)' constructs in list s the R1-set of fragment f (cf. definition 5): by means of 'r1s_path' it collects all elements of R1(f) that are accessible from f and adds f.
- for each of these R1-sets a fact omega_set(s,r) is added to the database. They represent $\Omega^{(0)}$.

Predicate 'build_omega' implements step 2:

- 'sets_to_merge' implements the WHILE-condition, i.e. it succeeds, if there are sets to be merged, and returns a pair (S1,S2) of sets (with their respective roots R1 and R2) for merging.
- merging of sets implies removal of the corresponding omega_set-facts and addition of a (single) new one.
- the recursive definition of 'build_omega' is essential for the correct implementation of the WHILE-loop: 'sets_to_merge' can succeed at most once; it fails, when (1) there are no more sets to be merged or (2) an attempt is made to resatisfy it in the course of backtracking.

```
r1s_path(F,G,[F,G])  :- edge(F,G),
                        not(x_fragment(G)),
                        not(o_fragment(G)),
                        not(s_fragment(G)).
r1s_path(F,G,[F|T])  :- edge(F,X),
                        not(x_fragment(X)),
                        not(o_fragment(X)),
                        not(s_fragment(X)),
                        r1s_path(X,G,T).

r1_set(F,[F|T]) :- findall(X,r1s_path(F,X,P),T).

omega_0(F,S) :- x_fragment(F), r1_set(F,S).
omega_0(F,S) :- o_fragment(F), r1_set(F,S).
omega_0(F,S) :- s_fragment(F), r1_set(F,S).
omega_0(F,S) :- e_fragment(F), r1_set(F,S).

omega_0      :- asserta(omega_set([],dummy)),
                fragment(F),
                omega_0(F,S),
                asserta(omega_set(S,F)),
                nl, write(' R1-set constructed: '), write(S),
                write(' Root: '), write(F),
                fail.
omega_0.

sets_to_merge(S1,R1,S2,R2) :- omega_set(S1,R1), omega_set(S2,R2), S1\=S2,
                              predecessor_check(R2,S1).
predecessor_check(R,S)     :- predecessors(R,Pred), ! ,
                              Pred=[_|T], T\=[],
                              subset(Pred,S).

build_omega :- sets_to_merge(S1,R1,S2,R2),
               retract(omega_set(S1,R1)), retract(omega_set(S2,R2)),
               append(S1,S2,Snew), asserta(omega_set(Snew,R1)),
               nl, write(' merging '), write(S1), write(' with '), write(S2),
               build_omega.
build_omega :- nl, write(' End of step 2: OMEGA constructed').

 omega :- omega_0,
          build_omega.
```

Fig. A-5: PROLOG program for the construction of set Ω (fg_omega_sets)

1.3. Construction of set CF: step 3 of algorithm 1 (fig. A-6)

Predicate 'char_frags_alg' determines the set CF of characteristic fragments of a fragment system according to step 3 of algorithm 1. Predicate 'omega_set' must be defined (i.e. steps 1 and 2 of algorithm 1 must have been done; cf. predicate provide_omega_set of consult-file fg_general, below).

- 'omega_edge(f,g)' succeeds if (f,g) is an edge of graph GΩ.
  Note that since the vertices of GΩ are given with predicate 'omega_set' these two predicates constitute a PROLOG specification of graph GΩ.
- predicate 'xomega' implements the mapping XΩ.
  'xomega(f,s)' collects in list s all elements of the set XΩ(f) by means of predicate xomega_el, the specification of the properties to be satisfied by an element of XΩ(f):
  - no_omega_dominator(f,g)' succeeds, if there is a path in graph GΩ from some entry-fragment to g such that f is not element of this path; therefore: 'not no_omega_dominator(f,g)' succeeds, if each path in GΩ from E to g contains f.
  - 'omega_path_check(f,g)' succeeds if either (f,g) is an edge of GΩ or there is in GΩ a path P from f to g, such that P-{f,g} contains neither an X- nor an O-fragment
- 'cf_alg(f)' succeeds if f is a characteristic fragment, i.e. predicate 'cf_alg' implements the characterization of the elements of CF according to step 3 of algorithm 1.

```
cf_alg(F) :- omega_set(_,F), F\=dummy,
             predecessors(F,Pred),card(Pred,N_pred),N_pred=<1,
             xomega(F,[]).

xomega(F,S)     :- findall(X,xomega_el(F,X),S).
xomega_el(F,G)  :- omega_set(_,F),
                   x_fragment(G), omega_set(_,G),
                   omega_path_check(F,G),
                   not no_omega_dominator(F,G).

omega_path_check(F,G) :- omega_edge(F,G).
omega_path_check(F,G) :- omega_edge(X,G),
                         not x_fragment(X), not o_fragment(X),
                         omega_path_check(F,X).

no_omega_dominator(_,E)    :- e_fragment(E).
no_omega_dominator(F,G)    :- omega_edge(X,G), X\=F,
                              no_omega_dominator(F,X).

omega_edge(F,G) :- omega_set(S,F), omega_set(_,G),
                   edge(X,G), member(X,S).

char_frags_alg(Char_fragments) :-
                  provide_omega_set,
                  findall(X,cf_alg(X),Char_fragments).
```

Fig. A-6: Construction of set CF according to step 3 of algorithm 1
(consult-file: fg_char_frags)

1.4. Construction of set CF according to corollary 3 (fig. A-7)

Predicate 'char_frags_cor' determines the set CF of characteristic fragments of a fragment system as a subset of F according to corollary 3 (section 4.2.2). It collects (in list char_fragments) the characteristic elements of the fragment system by means of predicate 'cf_cor', which implements the characterization of the elements of CF according to corollary 3.

'check_x_frags(f,g)' (consult-file fg_general, below) succeeds if fragment g has X-fragments, i.e. $X(g) \neq \emptyset$, and each path G from E to g contains f and (f=g or there is in G a R1-path from f to g); therefore: 'not check_x_frags(f,g)' succeeds if there is no $g \in F$ with $X(g) \neq \emptyset$ such that holds: each path in G from E to g contains f and (f=g or there is in G a R1-path from f to g)

```
cf_cor(F) :- x_o_e_fragment(F), not check_x_frags(F,_).
char_frags_cor(Char_fragments) :- findall(X,cf_cor(X),Char_fragments).
```

Fig. A-7: Construction of set CF according to corollary 3

(consult-file: fg_c_set_cor)

1.5. Construction of CF-representations (fig. A-8)

'c(f,l)' constructs for fragment f in list l a CF-representation C(f) of f: l is the CF-representation of the root-fragment of the Ω-set, which contains f.

'cf_repr(f,l)' constructs for a fragment f, which must be the root of some Ω-set, in list l a CF-representation C(f) of f.

- 'sucx(f,l)' (consult-file fg_general, below) collects for fragment f in list l the elements of the set SUCX(f) of definition 6
- predicate 'cf_repr':
  - the first two 'cf_repr' rules implement the construction of CF-representations for X- ,O- and entry-fragments as described in sections 6.1 and 6.2, in particular they implement algorithm 2.
  - the third rule implements the construction of CF-representations for S-fragments as described in section 6.3:
    'cf_repr_s1' constructs CF-represents according to section 6.3.1;
    'cf_repr_s2' constructs CF-represents according to section 6.3.2, i.e. if |SUCX(f)|≥0.
- 'union_cf_reprs(frags,l)' constructs for frags, which must be instantiated to a list of X- O- or entry-fragments, in list l the concatenation of CF-representations of the fragments of list frags.

```
c(F,L) :-  fragment(F), provide_omega_set,
           omega_set(S,Root), member(F,S), ! ,
           cf_repr(Root,L).

cf_repr(F,[F]) :- x_o_e_fragment(F),
                  sucx(F,[]), ! .
cf_repr(F,L)   :- x_o_e_fragment(F),
                  sucx(F,X), X\=[], ! ,
                  member(D,X), x_fragments(D,Xd),
                  union_cf_reprs(Xd,L).
cf_repr(F,L)   :- cf_repr_s(F,L).
cf_repr_s(F,L) :- cf_repr_s1(F,L).
cf_repr_s(F,L) :- cf_repr_s2(F,L).
cf_repr_s1(F,L):- s_fragment(F),
                  findall(X, next_x_o_e_predecessor(F,X), L_x_o_e), ! ,
                  union_cf_reprs(L_x_o_e,L).
cf_repr_s2(F,L):- s_fragment(F),
                  sucx(F,X), X\=[], ! ,
                  member(D,X), x_fragments(D,Xd),
                  union_cf_reprs(Xd,L).

union_cf_reprs([Xfrag|Xtail],L) :- cf_repr(Xfrag,Cfrep),
                                   union_cf_reprs(Xtail,Ctail),
                                   append(Cfrep,Ctail,L).
union_cf_reprs([],[]).

next_x_o_e_predecessor(F,G) :- predecessors(F,Pred),
                               member(X,Pred), x_o_e_predecessor(X,G).
     x_o_e_predecessor(X,X) :- x_o_e_fragment(X), ! .
     x_o_e_predecessor(X,Y) :- next_x_o_e_predecessor(X,Y).
```

Fig. A-8: Construction of CF-representations (consult file: cf_repr)

1.6. Construction of constraints and restrictions (fig. A-9)

The null-ary predicate 'constraints' constructs the inherent constraints of the fragment system by adding predicate 'constraint' to the database.

The presence of the "dummy"-fact 'constraint(dummy,dummy,[dummy]))' in the FSA database indicates that the inherent constraints have been determined already, i.e. predicate 'constraint' is available.
Each constraint corresponds a fact 'constraint(t,f,l):

- a 'constraint(rc1,f,l)' represents a RC1-constraint:
  if f is an X- , O- or entry-fragment with |SUCX(f)|>1, then the list l is the set SUCX(f); if f is an S-fragment and |SUCX(f)|>0, then list l is the set SUCX(F).
- a 'constraint(rc2,f,l)' represents a RC2-constraint:
  f is an O-fragment and l=[f,g] with g the predecessor of f in GΩ.

The null-ary predicate 'restrictions' determines and diplays the inherent restrictions of the fragment system according to T1, T2 of section 7.
'restrictions' assumes that predicate 'constraint' is defined (i.e. that the inherent constraints have been constructed).

- display of restrictions:
  as a short form the Boolean expressions are output as lists of characteristic fragments: a list of characteristic fragments stands for the relevance expressions involving the relevances of that list; instead of indices (cf. section 7) FSA displays the fragment names.
- 'append_all_cf_reprs(l,lreps)' compiles in list lreps all CF-representations of the fragments of list l.

```
constraints       :- not provide_omega_set.
constraints       :- fragment(F), omega_set(_,F),
                     constraints(F),
                     fail.
constraints       :- asserta(constraint(dummy,dummy,[dummy])).
constraints(F)    :- x_e_fragment(F),
                     rc1_constraint(F).
constraints(F)    :- o_fragment(F),
                     omega_edge(X,F),
                     asserta(constraint(rc2,F,[F,X])),
                     rc1_constraint(F).
constraints(F)    :- s_fragment(F),
                     sucx(F,X), X\=[],
                     asserta(constraint(rc1,F,X)).
constraints(F)    :- x_o_e_fragment(F).
constraints(F)    :- s_fragment(F).

rc1_constraint(F) :- sucx(F,X),
                     card(X,X1), X1>1,
                     asserta(constraint(rc1,F,X)).

restrictions :- x_o_e_fragment(F), constraint(rc1,F,[H|T]),
                findall(Lh,c(H,Lh),H_reprs),
                append_all_cf_reprs(T,T_reprs),
                cartesian_product(H_reprs,T_reprs,P),
                write_boolean_exprs(equiv,P),
                fail.
restrictions :- s_fragment(F), constraint(rc1,F,Sucx),
                findall(L,cf_repr_s1(F,L),S1_reprs),
                append_all_cf_reprs(Sucx,S2_reprs_0),
                delete_list(S1_reprs,S2_reprs_0,S2_reprs),
                cartesian_product(S1_reprs,S2_reprs,P),
                write_boolean_exprs(equiv,P),
                fail.
restrictions :- o_fragment(F), constraint(rc2,F,[H|T]),
                findall(Lh,c(H,Lh),H_reprs),
                append_all_cf_reprs(T,T_reprs),
                cartesian_product(H_reprs,T_reprs,P),
                write_boolean_exprs(impl,P),
                fail.
restrictions :- nl, write(' end of restrictions').

append_all_cf_reprs([],[]).
append_all_cf_reprs([H|T],L) :- findall(Lh,c(H,Lh),H_reprs),
                                append_all_cf_reprs(T,T_reprs),
                                append(H_reprs,T_reprs,L).
```

Fig. A-9: Construction of constraints and restrictions
(consult file: fg_constraints)

1.7. The general predicates

Fig. A-10 and fig. A-11 contain FSA-predicates, which are either of general nature or are used in several of the programs of the preceding sections.

```
% tab(N): definition of tab predicate
% ------
  tab(0) :- ! .
  tab(N) :- N>0, put(32), M is N-1, tab(M).
%
% nl(N) performs predicate nl N-times
% -----
  nl(1) :- nl.
  nl(N) :- N>1, nl, M is N-1, nl(M).
%
% subset(X,Y) succeeds if: X is a subset of X (cf. /ClocMel/)
% -----------
  subset([A|X],Y) :- member(A,Y), subset(X,Y).
  subset([],Y).
%
% no_duplicates(L1,L2) constructs list L2 such that L2 contains the elements
% --------------------                        of list L1 without duplicates.
  no_duplicates([],[]).
  no_duplicates([H|T],L) :- no_duplicates(T,Lx),
                            insert_no_dupl(H,Lx,L).
  insert_no_dupl(E,L,Le) :- not member(E,L), append([E],L,Le).
  insert_no_dupl(E,L,Le) :- member(E,L), Le=L.
%
% delete_all(El,L1,L2) constructs list L2 from list L1 by removing from L1
% --------------------                       all ocurrences of element El.
  delete_all(_,[],[]).
  delete_all(El,[El|Tail],L2)  :- ! , delete_all(El,Tail,L2).
  delete_all(El,[H|T1],[H|T2]) :- delete_all(El,T1,T2).
%
% delete_list(L1,L2,L) constructs list L by removing from list L2 all
% --------------------                            elements of list L1.
  delete_list([],L,L).
  delete_list([H|T],L2,L) :- delete_all(H,L2,X),
                             delete_list(T,X,L).
%
```

Fig. A-10: consult-file p_programs

```
% card(L,Card) succeeds if: L is a list:
% ------------               Card is the cardinality (=length)  of L
  card([],0).
  card([H|T],Card) :- card(T,T_card), Card is T_card+1.
%
% cartesian_product(L1,L2,C) succeeds if: list C is the Cartesian Product
% -------------------------- of lists L1, L2, i.e. C is the list of pairs
%                            (l1,l2), where l1 is element of L1 and l2 is
%                            element of L2. (pairs are represented as lists)
  cartesian_product([],_,[]).
  cartesian_product([H|T],L,C) :- cartesian_product_1(H,L,X1),
                                  cartesian_product(T,L,X2),
                                  append(X1,X2,C).
  cartesian_product_1(Elem,[],[]).
  cartesian_product_1(Elem,[H|T],C) :- cartesian_product_1(Elem,T,X),
                                      append([[Elem,H]],X,C).
%
% print_as_table(Max_items,Items,List) outputs list List in form of a table
% ------------------------------------ with Max_items entries per row
  print_as_table(M,Col,I,[H|T]) :- I<M, write(H),
                                   name(H,H1),card(H1,H_length),
                                   Blanks is Col - H_length, tab(Blanks),
                                   X is I+1,
                                   print_as_table(M,Col,X,T).
  print_as_table(M,Col,M,L)     :- nl,  print_as_table(M,Col,0,L).
  print_as_table(_,_,_,[])      :- nl.
%
% read_word(W) reads from the current input stream, W is the string
% ------------ of characters from the current position to the next

%                "terminating character".
  read_word(W) :- read_char_list(Chars),
                  name(W,Chars).
  read_char_list(Chars) :- get0(C), rest_char_list(C,Chars).
  rest_char_list(C,[])  :- terminating_char(C), ! .
  rest_char_list(C,[C|Chars]) :- read_char_list(Chars).
% the terminating characters: <return>
  terminating_char(10).
```

Fig. A-10: consult-file p_programs (cont.)

```
% fragments systems: definitions
% =============================
%
% x_fragment(F) holds if F is a X-fragment
% -------------
  x_fragment(F)      :- x_fragments(X,Y),member(F,Y).
%
% o_fragment(F) holds if F is an O-fragment
% -------------
  o_fragment(F)      :- o_fragments(X,Y),member(F,Y).
%
% create facts s_fragment(F):  s_fragment(F) holds if F is a S-fragment
%                              -------------
  :- edge(X,F),edge(Y,F),X\=Y,
     not clause(s_fragment(F),true), asserta(s_fragment(F)), fail.
%
% create facts e_fragment(F):  e_fragment(F) holds if F is an entry-fragment
%                              -------------
  :- fragment(F), not edge(_,F), asserta(e_fragment(F)), fail.
%
% x_o_e_fragment(F) succeeds, if F is an X-, O- or entry-fragment
% -----------------
  x_o_e_fragment(F) :- x_fragment(F).
  x_o_e_fragment(F) :- o_fragment(F).
  x_o_e_fragment(F) :- e_fragment(F).
% x_e_fragment(F)   succeeds, if F is an X- or O-fragment
% ---------------
  x_e_fragment(F)   :- x_fragment(F).
  x_e_fragment(F)   :- e_fragment(F).

%
```

Fig. A-11: consult-file fg_general

```
% no_dominator(F,G) succeeds if there is a path in graph G from some
% -----------------           entry-fragment such that F is not element of
%                             this path.
  no_dominator(F,G)    :- F\=G, e_fragment(G).
  no_dominator(F,G)    :- F\=G, edge(X,G), no_dominator(F,X).
%
%
% sucx(F,L) succeeds, if L is the set SUCX(F)  of definition 6
% ---------
  sucx(F,L)          :-  findall(X,check_x_frags(F,X),Lx),
                         no_duplicates(Lx,L).
%
% check_x_frags(F,G) succeeds, if fragment G has X-fragments such that
% ------------------               each path from set E to G contains F  and
%                                  (F=G or there is a R1-path from F to G)
  check_x_frags(F,G) :- x_fragments(G,[_|_]),
                        not no_dominator(F,G),
                        sucx_check(F,G).
%
% sucx_check(F,G) succeeds if F=G or there is a R1-path from fragment F to G
% ---------------
  sucx_check(F,F).
  sucx_check(F,G) :- not o_fragment(G), not x_fragment(G),
                     edge(X,G),
                     sucx_check(F,X).
%
% r1_path(F,G,Path) succeeds if: list Path is a R1-Path from fragment F
% -----------------                                      to fragment G
  r1_path(F,G,[F,G])  :- edge(F,G),not x_fragment(G),not o_fragment(G).
  r1_path(F,G,[F|T])  :- edge(F,X),
                         not(x_fragment(X)),not o_fragment(X),
                         r1_path(X,G,T).
%
```

Fig. A-11: consult-file fg_general (cont.)

```
% predecessors(F,Pred) succeeds if: list Pred is set of predecessors of
% --------------------             fragment F.
  predecessors(F,Pred) :- findall(X,edge(X,F),Pred).
%
% provide_omega_set: constructs OMEGA-set unless it exists already.
% -----------------
  provide_omega_set :- clause(omega_set(_,_),_).
  provide_omega_set :- nl, write(' set OMEGA being constructed'),
                       omega.
%
% provide_constraints: constructs the inherent constraints unless
% -----------------                                  done already.
  provide_constraints :- clause(constraint(dummy,dummy,[dummy]),true).
  provide_constraints :- constraints.
%
% write_boolean_exprs(T,P): prints the pairs [l,r] of list P as
% ----------------------          implications l ==> r  if T=impl
%                                 equivalences l <==> r if T=eqiv
  write_boolean_exprs(_,[]).
  write_boolean_exprs(Type,[H|T])  :- write_boolean_expr(Type,H),
                                      write_boolean_exprs(Type,T).
  write_boolean_expr(equiv,[L1,L2]):- nl,
                                      write(L1),
                                      write(' <==> '),
                                      write(L2).
  write_boolean_expr(impl,[L1,L2]) :- nl,
                                      write(L1),
                                      write('  ==> '),
                                      write(L2).
```

Fig. A-11: consult-file fg_general (cont.)

2. FSA-user interaction

FSA is an interactive system. Figures A-12, A-13 and A-14 show excerpts from a FSA session, where the example fragment system (cf. fig. A-4) is analyzed; cf. sections 6 and 7.

The remainder of this section lists the PROLOG program implementing the FSA user interface.

```
 available FSA commands (class=1):
end   : termination of FSA
1     : determination of CF-representations
2     : determination of the set of characteristic fragments
3     : display set OMEGA
4     : determination/display of the inherent constraints
5     : determination of the inherent restrictions
6     : FSA-logging

 command: 1


determination of CF-representations

 The Cf-representations for all fragments of the fragment system:


 C(1) = [1.1,1.2,4.1,4.2.1,4.2.2,1.4,8.1,8.2,10.1,10.2]
 C(1.1) = [1.1]
 C(1.2) = [1.2]
 C(1.3) = [4.1,4.2.1,4.2.2]
 C(1.4) = [1.4]
 C(1.5) = [8.1,8.2]
 C(1.6) = [10.1,10.2]
 C(2) = [1.1]
 C(2.1) = [2.1]
 C(3) = [4.1,4.2.1,4.2.2]
 C(4) = [4.1,4.2.1,4.2.2]
 C(4.1) = [4.1]
 C(4.2) = [4.2.1,4.2.2]
 C(4.2.1) = [4.2.1]
 C(4.2.2) = [4.2.2]
 C(5) = [2.1,1.1,1.4]
 C(5) = [6.1,6.2,5.2]
 C(5.1) = [6.1,6.2]
 C(5.2) = [5.2]
 C(5.3) = [5.3]
 C(6) = [6.1,6.2]
 C(6.1) = [6.1]
 C(6.2) = [6.2]
 C(7) = [5.2]
 C(8) = [8.1,8.2]
 C(8.1) = [8.1]
 C(8.2) = [8.2]
 C(8.3) = [8.3]
 C(9) = [1.2]
 C(9.1) = [9.1]
 C(10) = [10.1,10.2]
 C(10.1) = [10.1]
 C(10.2) = [10.2]
 C(10.3) = [10.3]
 C(11) = [1.1]
 C(12) = [1.2]
 C(13) = [2.1]
 C(14) = [9.1]
 C(15) = [8.1]
 C(16) = [8.2]
 C(17) = [10.1]
 C(18) = [10.2]
 C(19) = [8.3]
 C(20) = [10.3]
 C(21) = [4.2.2]
 C(22) = [2.1,9.1,4.1,4.2.1,4.2.2,8.3,10.3]
 C(22.1) = [2.1,9.1,4.1,4.2.1,4.2.2,8.3,10.3]
```

Fig. A-12: Determination of CF-representations

```
 available FSA commands (class=1):
end  : termination of FSA
1    : determination of CF-representations
2    : determination of the set of characteristic fragments
3    : display set OMEGA
4    : determination/display of the inherent constraints
5    : determination of the inherent restrictions
6    : FSA-logging

 command: 2

 available FSA commands (class=2):
end  : termination of FSA
1    : The characteristic fragments according to COROLLARY 3:
2    : The characteristic fragments according to ALGORITHM 1:

 command: 1

the characteristic set according to COROLLARY 3:

1.1     1.2     1.4     4.1     4.2.1
4.2.2   5.2     6.1     6.2     8.1
8.2     10.1    10.2    2.1     5.3
8.3     9.1     10.3
```

Fig. A-13: Determination of characteristic set

```
 available FSA commands (class=1):
end  : termination of FSA
1    : determination of CF-representations
2    : determination of the set of characteristic fragments
3    : display set OMEGA
4    : determination/display of the inherent constraints
5    : determination of the inherent restrictions
6    : FSA-logging

 command: 5


 the inherent restrictions:
[2.1,1.1,1.4] <==> [6.1,6.2,5.2]
[2.1]  ==> [1.1]
[5.3]  ==> [2.1,1.1,1.4]
[5.3]  ==> [6.1,6.2,5.2]
[8.3]  ==> [8.1,8.2]
[9.1]  ==> [1.2]
[10.3]  ==> [10.1,10.2]
 end of restrictions


 available FSA commands (class=1):
end  : termination of FSA
1    : determination of CF-representations
2    : determination of the set of characteristic fragments
3    : display set OMEGA
4    : determination/display of the inherent constraints
5    : determination of the inherent restrictions
6    : FSA-logging

 command: end


         e n d    o f    F S A
```

Fig. A-14: Determination of inherent restrictions

```
action(cmnd('end'),
       class(0),
       expl('termination of FSA')).
action(cmnd('end'),
       class(1),
       expl('termination of FSA')).
action(cmnd('end'),
       class(2),
       expl('termination of FSA')).
action(cmnd('end'),
       class(3),
       expl('termination of FSA')).
action(cmnd('init'),
       class(0),
       expl('initialization')).
action(cmnd('1'),
       class(1),
       expl('determination of CF-representations')).
action(cmnd('2'),
       class(1),
       expl('determination of the set of characteristic fragments')).
% subcommands:
  action(cmnd('1'),
         class(2),
         expl('The characteristic fragments according to COROLLARY 3:')).
  action(cmnd('2'),
         class(2),
         expl('The characteristic fragments according to ALGORITHM 1:')).
%
action(cmnd('3'),
       class(1),
       expl('display set OMEGA')).
%
action(cmnd('4'),
       class(1),
       expl('determination/display of the inherent constraints')).
%
action(cmnd('5'),
       class(1),
       expl('determination of the inherent restrictions')).
%
```

Fig. A-15: FSA user interface

```
action(cmnd('6'),
       class(0),
       expl('FSA-logging')).
action(cmnd('6'),
       class(1),
       expl('FSA-logging')).
% subcommands:
  action(cmnd('1'),
         class(3),
         expl('logging on (log_file: fsa_log)')).
  action(cmnd('2'),
         class(3),
         expl('logging off')).
%
%
:-    consult(p_programs),
      nl, write('\f'), tab(35),
      write(' F S A '), nl(2), tab(10),
      write('A PROLOG implemention of algorithms related to fragment systems'),
      nl, tab(25), write('programmed by Franz J. Polster'),
      nl(2), write(' to proceed enter:  run.  !'), nl(2).
%
action_class(0).

%
% ====================================================================
  run :-  repeat,
          prompt_for_action(Class),
          read_action(Cmnd),
          execute(Cmnd,Class),
          fail.
% ====================================================================
%
%
execute(Cmnd,Class) :-
                        clause(logging_on,true),
                        tell(fsa_log),
                        perform_action(Cmnd,Class),
                        tell(user), ! .
execute(Cmnd,Class) :- perform_action(Cmnd,Class).
```

Fig. A-15: FSA user interface (cont.)

```
prompt_for_action(Class) :- action_class(Class),
                            write_menue(Class),
                            nl(2), write(' enter command: ').
write_menue(Class):- nl(3), write('available FSA commands'),
                     action_class(Class),
                            write(' (class='),write(Class),write('):'),
                     action(cmnd(Cmnd),class(Class),expl(Text)),
                     write_cmnd(Cmnd), write(Text),
                     fail.
write_menue(Class):- clause(logging_on,true),
                     tell(fsa_log),
                     nl(3), write('\f available FSA commands'),
                     action_class(Class),
                            write(' (class='),write(Class),write('):'),
                     action(cmnd(Cmnd),class(Class),expl(Text)),
                     write_cmnd(Cmnd), write(Text),
                     fail.
write_menue(_)    :- tell(user).
write_cmnd(Cmnd) :- nl,write(Cmnd),
                    build_list(Cmnd,X), card(X,Cmnd_length),
                    X_blanks is 5-Cmnd_length,
                    tab(X_blanks), write(': ').
 build_list(X,L) :- name(X,L).
 build_list(X,L) :- number(X,L).
%
%
read_action(X)    :- read_word(X), read_log(X), ! .
read_log(X)       :- clause(logging_on,true),
                     tell(fsa_log),
                     nl(2), write(' command: '), write(X), nl(2),
                     tell(user).
read_log(_).
%
```

Fig. A-15: FSA user interface (cont.)

```
perform_action(init,0) :-
    nl, write(' initialization:'),
    nl, write(' enter name of file with the specification of fragment system:'),
    read_word(F),call(consult(F)),
    consult(fg_general),
    consult(fg_omega_sets),
    consult(cf_repr),
    consult(fg_char_frags),
    consult(fg_c_set_cor),
    consult(fg_constraints),
    new_action_class(1),
    nl, write(' end of initialization'), nl(2), ! .
%
perform_action('1',1)  :- nl, write('determination of CF-representations'),
                          conversation(1,1,F),
                          cf_representation(F), ! .
 conversation(1,1,F) :- telling(X), X\=user, tell(user),
                        nl, write('enter fragment name: '), read_word(F), nl,
                        tell(X).
 conversation(1,1,F) :- nl, write('enter fragment name: '), read_word(F), nl.
cf_representation(F) :- fragment(F),
                        findall(L,c(F,L),L_reprs),
                        nl(2), write('The CF-representations:'),
                        write_reprs(F,L_reprs).
cf_representation(_) :- nl(2),
                        write(' The Cf-representations for all fragments'),
                        write(' of the fragment system:'), nl(2),
                        fragment(X),
                        findall(L,c(X,L),L_reprs),
                        write_reprs(X,L_reprs),
                        fail.
cf_representation(_).
write_reprs(F,[H|T]) :- nl, write(' C('),write(F),write(') = '), write(H),
                        write_reprs(F,T).
write_reprs(_,[]).
%
```

Fig. A-15: FSA user interface (cont.)

```
perform_action('2',1)  :-new_action_class(2), ! .
%
perform_action('1',2)  :- nl, write('the characteristic set'),
                              write(' according to COROLLARY 3:'),
                          nl,
                          char_frags_cor(Char_fragments),
                          print_as_table(5,8,5,Char_fragments),
                          new_action_class(1), ! .
%
perform_action('2',2)  :- nl, write('the characteristic set'),
                              write(' according to ALGORITHM 1 (step 3):'),
                          nl,
                          write('(Very time-consuming! Have a coffee break!)'),
                          nl,
                          char_frags_alg(Char_fragments),
                          print_as_table(5,8,5,Char_fragments),
                          new_action_class(1), ! .
%
perform_action('3',1)  :- provide_omega_set,
                          nl, write(' The elements of set OMEGA:'),
                          write_omegas, ! .
        write_omegas :- omega_set(X,R), R\=dummy,
                          nl, tab(1),write(X), tab(5), write('root is: '),
                          write(R),
                          fail.
        write_omegas :- nl(2), write(' end of set OMEGA').
%
perform_action('4',1)  :- nl(2),
                          write(' the inherent constraints:'),
                          provide_constraints,
                          write_constraints, ! .
   write_constraints :- constraint(rc1,F,L),
                          write_boolean_exprs(equiv,[[[F],L]]),
                          fail.
   write_constraints :- constraint(rc2,F,[F,G]),
                          write_boolean_exprs(impl,[[[F],[G]]]),
                          fail.
   write_constraints :- nl,write(' end of constraints').
%
```

Fig. A-15: FSA user interface (cont.)

```
perform_action('5',1)  :- nl(2),
                          write(' the inherent restrictions:'),
                          provide_constraints,
                          restrictions, ! .
%
perform_action('6',0)  :- asserta(old_action_class(0)), new_action_class(3), ! .
perform_action('6',1)  :- asserta(old_action_class(1)), new_action_class(3), ! .
perform_action('1',3)  :- asserta(logging_on),
                            old_action_class(C), retract(old_action_class(_)),
                            new_action_class(C), ! .
perform_action('2',3)  :- old_action_class(C), retract(old_action_class(_)),
                            new_action_class(C), ! ,
                            tell(user),
                            retract(logging_on).
%
perform_action(end,1):- perform_action(end,0).
perform_action(end,2):- perform_action(end,0).
perform_action(end,3):- perform_action(end,0).
perform_action(end,0):-
                          nl(3), tab(10),
                          write('e n d     o f     F S A'),
                          nl(3),
                          tell(fsa_log), told,
                          end.
%
perform_action(debug,_) :- ! , debug.
perform_action(nodebug,_) :- ! , nodebug.
perform_action(X,_)   :- nl, write('command '),write(X),
                              write(' unknown or not allowed').
%
%
new_action_class(Class) :- retract(action_class(_)),
                            asserta(action_class(Class)).
```

Fig. A-15: FSA user interface (cont.)

REFERENCES

1. Aho, A.V.; Hopcroft, J.E.; Ullman, J.D.: The Design and Analysis of Computer Algorithms. Addison-Wesley Publishing Company, 1974

2. Aho, A.V.; Ullman, J.D.: The Theory of Parsing, Translation and Compiling (vol. 1). Prentice-Hall, Inc. Englewood Cliffs, New Jersey, 1972

3. Cooprider, L.W.: The Representation of Families of Software Systems. PhD-Thesis, CMU-CS-79-116, Department of Computer Science, Carnegie-Mellon University, April 1979

4. Freeman, P.: Reusable Software Engineering: Concepts and Research Directions. In: Freeman, P.; Wasserman, A.I. (ed.): Tutorial on Software Design Techniques (4th edition), p. 63-76. IEEE Computer Society Press, 1983

5. Gordon, R.D.: The Modular Application Customizing System. IBM Systems Journal 19,4(1980), p.521-541

6. Hecht, M.S.: Flow Analysis of Computer Programs. North Holland, New York, 1977

7. Nehmer, J.: Betriebssysteme für Kleinrechner. Angewandte Informatik, No. 1, 1977, p. 1-14

8. Nilsson, N.J.: Problem-Solving Methods in Artificial Intelligence. McGraw Hill Inc., 1971

9. Parnas, D.L.; Handzel, G.; Würges, H.: Design and Specification of the Minimal Subset of an Operating System Family. IEEE Transactions on Software Engineering 2,4 (Dec. 1976) p. 301-307

10. Polster, F.J.: Generierbare Datenbanksysteme. Doctoral Dissertation, University of Karlsruhe, Fakultät für Informatik, Fed. Rep. Germany, Dec. 1982.

11. Polster, F.J.: Dedication of general database software to specific applications. Proceedings IEEE Computer Societies 7th International Computer Software and Applications Conference (COMPSAC '83), Chicago, Nov 7-11, 1983, p. 201-210

12. Polster, F.J.: Reuse of Software through Generation of Partial Systems. Report KfK 3964, Kernforschungszentrum Karlsruhe, Fed. Rep. Germany, Sept. 1985

13. Polster, F.J.: Reuse of Software through Generation of Partial Systems. A concise version of [12], accepted for publication in IEEE Transactions on Software Engineering.

14. Polster, F.J.: Adaptation of Program Systems Through Code Selection. Kernforschungszentrum Karlsruhe, Fed. Rep. Germany, submitted for publication

15. Polster, F.J.: Generable Database Management Systems. To appear in Angewandte Informatik, 1985

16. Rüb, W.; Schrott, G.: Automatische Generierung problemangepaßter Prozeßrechner-Betriebssysteme. Angewandte Informatik, No. 1, 1980, p.7-17

17. Schrott, G.: Generation of dedicated realtime operating systems by dialogue. Proc. IFAC/IFIP Workshop on Real Time Programming, p. 145-151. Eindhoven, Netherlands. June 1977

18. Shaw, M.: Reduction of Compilation Costs Through Language Contraction. CACM 17,5 (May 1974), p. 245-250

19. Tichy, W.F.: Software Development Control Based on System Structure Description. PhD-Thesis, CMU-CS-80-120, Department of Computer Science, Carnegie-Mellon University, Jan. 1980

20. Tichy, W.F.: A Data Model for Programming Support Environments and its Application. In: Schneider, H.-J.; Wasserman, A.I. (eds.): Automated Tools for Systems Design, p.31-48. North-Holland Publishing Company, 1982.

21. Clocksin, W.F.; Mellish, C.S.: Programming in Prolog. Springer-Verlag Berlin Heidelberg New York, 1981

22. Leibrandt, U.; et al.: IF/Prolog User's Manual, Version 2.0. InterFace Computer GmbH, München, Federal Republic of Germany, 1984