MATHEMATICAL PROBLEMS IN COMPUTATION THEORY
BANACH CENTER PUBLICATIONS, VOLUME 21
PWN – POLISH SCIENTIFIC PUBLISHERS
WARSAW 1988

# CONSTRUCTION OF PARALLEL PROGRAMS WITH NETWORKS OF COOPERATING PROCESSES

JEAN-PIERRE BANATRE

*IRISA-INSA and INRIA-Rennes Campus de Beaulieu, Rennes Cédex, France*

A cooperation scheme allowing the construction of systems of concurrent processes is introduced. In a first part, an informal presentation of the cooperation scheme is given, then its formal properties in terms of Generalized Hoare Logic are described. In a second part, the use of the cooperation scheme is illustrated by several examples. The conclusion suggests several fields needing further investigation, for example the distributed implementation of the cooperation scheme.

## 1. Introduction

The subject of constructing programs suitable for parallel execution gains more and more interest. Two types of parallelism may be considered:

(i) implicit parallelism, which consists of interpreting in a parallel fashion some high level formalism, e.g. logic or functional specification. The programmer is not concerned with the actual design of a parallel algorithm, his/her only task is providing specifications,

(ii) explicit parallelism, where the programmer uses appropriate control and data structures to express parallel solutions to problems.

In this paper, we are concerned with parallelism of type (ii). Actually, a parallel program may be seen as the description of a set of processes and of their interactions. Interactions are usually modelled by so-called cooperation schemes. Among well-known cooperation schemes, let us mention the procedure call used when a process ask for a service from another process, the rendez-vous used when synchronous behaviour of processes is needed,

Key Words. *Parallel programming, processes, cooperation scheme, correctness, termination.*

the parallel composition which allows for concurrent execution of set of processes. It is our belief that the very nature of a problem makes more natural the use of one or another cooperation scheme in order to design a solution.

The present paper describes a cooperation scheme operating as follows: imagine that the solution $V$ to a problem is a vector of $n$ values $V = (v_1, \ldots, v_n)$. In order to compute $V$ a process $P_i$ is in charge of finding $v_i$, $\forall i \in [1, n]$ and by successive approximations $V_0$, $V_1$, ..., $V_k$, ... the solution $V$ is found. In order, to compute $V_i$ from $V_{i-1}$ interactions between processes are necessary. The cooperation scheme presented in this paper is a possible model for such interactions. In rough approximation, one can think that this cooperation scheme may be well adapted to perform numerical calculations based on relaxation methods, where a trial solution is first proposed and calculation adjusts the trial in order to find a solution. Of course, a major problem has to be tackled: the convergence of the relaxation method. In the same way, our cooperation scheme may lead to undesired infinite computations, if special care is not taken when choosing the initialization of the trial solution ($V_0$). Section 2 gives an informal presentation of the proposed cooperation scheme. Its logical properties are described and termination aspects are dealt with in Section 3. Several examples are developed in Section 4 and finally Section 5 contains a brief review and discussion.

## 2. A scheme for process cooperation

**2.1. The cooperation law.** Consider a set $S$ of processes $p_1, \ldots, p_n$. Each $p_i$ is provided with a "weight" $w_j$. The cooperation law obeyed by $S$ may be defined as follows:

Two processes $p_i$ and $p_j$ $(i \neq j)$ of $S$ may *communicate* if and only if a given predefined binary relationship $R(w_i, w_j)$ holds.

When communication is possible ($R(w_i, w_j)$ holds), $p_i$ and $p_j$ exchange atomically their weights, i.e., $p_i$ receives $w_j$ and $p_j$ receives $w_i$. After this exchange, $p_i$ executes a function $f_i(w_i, w_j)$ which may eventually change $w_i$ and $p_j$ executes a function $f_j(w_j, w_i)$ which may eventually change $w_j$. Upon termination of $f_i$ (resp. $f_j$), $p_i$ and $p_j$ are ready for new interactions with other processes belonging to $S$.

So processes $p_i$'s are cyclic, the executions of a cycle being governed by the possibility for $p_i$ of being "coupled" with another process $p_k$ (i.e., $R(w_i, w_k)$ holds).

STEADY STATE. The system $S$ is said to be in a *steady state* if the following condition holds: $\forall i, j, i \neq j, \neg R(w_i, w_j)$. In this state, there is no possibility of further interaction.

**2.2. Programming notation.** Weights are tuples whose type may be defined as a tuple of types: (*type* 1, ..., *type n*). A weight is defined as a tuple of typed identifiers, for example (*integer*: *s*, real: *r*, *string*: *u*). A component of a weight value is referred to by its identifer. The structure of a process is as follows:

$p_i$::*[ < ⟨definition of $w_i$⟩
◇
⟨definition of $f_i(w_i, w_j)$⟩
]

$w_i$, ($w_j$) refers to the weight of the process coupled with $p_i$, ($p_j$) for the execution of the current computation step for which $R(w_i, w_j)$ holds.

**2.3. Example.** Consider the problem of sorting 5 (different) names into alphabetical order. For example, the names may be:

GEORGE, ANDREW, JOHN, PETER, MIKE.

A process is attached to each name. The weight of the process is a couple (name, position), position being initially set up to 5 (number of names to be sorted).

The text of a process is:

```
*[(name, position)
  ◇
  if name < name
  then position:= position-1
  fi
]
```

Overbarred identifiers are used in the text of a process $p$ (with weight $w$) to refer to elements of the weight $\bar{w}$ of the process currently coupled with $p$ (i.e., such as $R(w, \bar{w})$ is true).

The relationship $R(w, \bar{w})$ governing interactions between processes is position = position. Thus two processes possessing the same position may exchange their weights and make a computation step.

The evolution of the system of processes may be figured as:

(GEORGE, 5) (ANDREW, 5) (JOHN, 5) (PETER, 5) (MIKE, 5)

(GEORGE, 5) (ANDREW, 4) (JOHN, 5) (PETER, 5) (MIKE, 4)

(GEORGE, 5) (ANDREW, 3) (JOHN, 4) (PETER, 5) (MIKE, 4)

(GEORGE, 4) (ANDREW, 3) (JOHN, 3) (PETER, 5) (MIKE, 4)

(GEORGE, 3) (ANDREW, 2) (JOHN, 3) (PETER, 5) (MIKE, 4)

(GEORGE, 2) (ANDREW, 2) (JOHN, 3) (PETER, 5) (MIKE, 4)

(GEORGE, 2) (ANDREW, 1) (JOHN, 3) (PETER, 5) (MIKE, 4)

On the above figure, a process is represented by its weight, and two communicating processes are linked by a horizontal line. Of course, this is one among the possible paths leading to a solution. This algorithm is non-deterministic as there is no indication on the way cooperating couples are selected.

One can see that when the system of processes reaches its steady state, the second position of the weight represents the alphabetical position of the name.

However, caution has to be taken so that the initialization of the weights may actually lead to a steady state representing a solution.

## 3. The logic of this cooperation scheme

**3.1. Generalized Hoare Logic.** In order to describe the formal properties of the above cooperation scheme, we use the Generalized Hoare Logic (GHL) as developed in [4]. GHL is a generalization of Hoare Logic for sequential programs to concurrent programs. It allows the derivation of invariance properties of programs from properties of their components.

Three predicates may be associated with a program fragment $\pi$.

**at** $(\pi) \Rightarrow$ "control resides at entry point of $\pi$",

**in** $(\pi) \Rightarrow$ "control resides somewhere in $\pi$",

**after** $(\pi) \Rightarrow$ "control resides at a point immediately following $\pi$".

These predicates are used to describe control states. In [4], GHL is described for a simple programming language. An inference rule is given for each construction of the language, making it possible to derive properties of statements from properties of components. Among the most important rules, let us mention the following:

$\pi$ is a program fragment made of $n$ components $\pi_1, \ldots, \pi_n$.

DECOMPOSITION RULE. If $I$ is an invariance relation verified by every $\pi_i$, then

$$\frac{I\{\pi_1\}I, \ldots, I\{\pi_n\}I}{I\{\pi\}I}.$$

CONJUNCTION RULE.

$$\frac{I_1\{\pi\}I_1, \ldots, I_n\{\pi\}I_n}{\bigwedge_{i=1,n} I_i\{\pi\}\bigwedge_{i=1,n} I_i}.$$

Finally, the logic of a program is represented by an invariance property $P$, and the method for proving $P$ is as follows:

(i) prove that pre $\Rightarrow P$, where pre is the precondition of the program,

(ii) prove that any state reached during execution satisfies $P$,

(iii) prove that $P \Rightarrow$ post, post being the postcondition reached at the exit point.

Let us now apply GHL in order to define the properties of our cooperation scheme.

**3.2. Formal properties of the cooperation scheme.** A process $\pi_i$ is represented by:

$$\begin{aligned} \pi_i :: \ & \pi_{0i}: \ *[w_i \\ & \qquad \diamond \\ & \pi_{1i}: \ \langle f(w_i, w_i) \rangle \\ & \pi_{2i}: \ ] \end{aligned}$$

where: $\pi_{ki}$, $k \in [0, 2]$ represent control points. The notation $\langle\ \rangle$ indicates that the execution of $f(w_i, w_j)$ is atomic [4].

Assuming that the cooperation scheme is governed by a relation $R(w_i, w_j)$, it is possible to state the following logical properties:

LOGICAL PROPERTIES OF THE PROPOSED SCHEME. (1) Symmetry of the communication condition

$$R(w_i, w_j) = R(w_j, w_i).$$

(2) Invariance property of function $f$

$$\begin{gathered} R(w_i, w_j) \wedge \mathbf{at}(\pi_{1i}) \wedge I, \\ \{\langle f(w_i, w_j) \rangle\}, \\ \mathbf{at}(\pi_{2i}) \wedge I. \end{gathered}$$

(3) Communication rule

$$\begin{gathered} R(w_i, w_j) \wedge \mathbf{at}(\pi_{0i}) \wedge \mathbf{at}(\pi_{0j}) \wedge I, \\ \{\langle \text{exchange of weights} \rangle\}, \\ R(w_i, w_j) \wedge \mathbf{after}(\pi_{0i}) \wedge \mathbf{after}(\pi_{0j}) \wedge I. \end{gathered}$$

(4) Process $\pi_i$ is cyclic

$$\mathbf{after}(\pi_{2i}) \equiv \mathbf{at}(\pi_{0i}).$$

(5) $f$ is atomic

$$\mathbf{after}(\pi_{1i}) \equiv \mathbf{at}(\pi_{2i}).$$

(6) Steady state.

All processes are at their initial control point and no more couple may be formed:

$$\bigwedge_{i=1,n} \mathbf{at}(\pi_{0i}) \wedge \neg \bigvee_{k,l=1,n} R(w_k, w_l).$$

(7) Initially all processes $\pi_i$ start simultaneously. If we denote by $\pi$, the set of processes $\pi_i$, $i \in [1, n]$ then the initialization condition is:

$$\mathbf{at}(\pi) \Rightarrow \bigwedge_{i=1}^{n} \mathbf{at}(\pi_{0i}).$$

These properties characterize the proposed cooperation scheme. In particular, they might be used in conjunction with the usual Hoare logic for sequential programs to prove properties of concurrent programs built according to our scheme.

**3.3. About termination.** Many problems which can be easily solved using our cooperation scheme, are such that the occurrence of a steady state (Property 6) implies the discovery of a solution – as an illustration see Example 2.3. It is then of prime importance to show that according to some given initial conditions the system of processes will converge.

A usual tool for proving termination of programs is the well-founded set: a set of elements and an ordering ">" defined on these elements, such that they can be no infinite descending sequence of elements. The idea for proving the termination of a sequential program (process) is to find a termination function that maps process variables into a well-founded set, the value of the termination function being successively decreased throughout the computation. Natural numbers under the $>$ = ordering are often used for proving termination of iterations [3]. In [2], multiset ordering is shown to be well-founded and used for proving termination of production systems. Multisets are like sets, but may contain multiple occurrences of the same element. Consider two multisets of natural numbers $M_1$ and $M_2$, the relationship $M_1 \gg M_2$ holds if $M_2$ can be obtained from $M_1$ by replacing one or more elements of $M_1$ by any finite sequence of natural numbers, each of which being smaller than the replaced one (more details in [2]).

Our idea consists of applying this multiset ordering for proving termination of parallel programs built according to our cooperation scheme. To each process $p_i$ is associated a termination function $t_i$ which maps the weight $w_i$ into the set of natural numbers under usual ordering. So $w_i$ will be mapped successively onto the following values $\{t_{i1}, \ldots, t_{ik}, \ldots\}$ such that, $\forall u$, $v \in N$, $u > v \Rightarrow t_{iu} > t_{iv}$. As every $p_i$, $i \in [1, n]$ is provided with a function $t_i$, then the initial state of the system of processes $p_i$, $i \in [1, n]$ will be mapped through functions $t_i$, $i \in [1, n]$ into the multiset $w_i = \{t_{i1}, \ldots, t_{in}\}$. Any subsequent state $w_i = \{t_{i1}, \ldots, t_{in}\}$ will be such that $w_i \ll w_1$ and any state $w_j$ derived from $w_i$ will be such that $w_j \ll w_i$. Thus, we have a simple mean for proving that parallel programs built according to our cooperation scheme terminate (or reach a steady state).

Coming back to example of Section 2.3, let us associate to every process $p_i$ the function $t_i$ defined as follows:

$t_i$: $\{\text{names}\} \times N \to N$,

$t_i(w_i) = \text{position}_i$.

The initial configuration of the system is represented by the multiset $\{5, 5, 5, 5, 5\}$. Given the form of the function $f_i(w_i, w_j)$, any application of this function results in decreasing either $\text{position}_i$ or $\text{position}_j$. So any configuration $w_2$ derived from $w_1$ will be lower than $w_1$ and so on and so forth. The set of configurations possesses a lower bound $w = \{1, 2, 3, 4, 5\}$ and this remak concludes the termination proof.

## 4. Applications of the proposed cooperation scheme

**4.1. Name sorting.** Let us give the invariance properties of Example 2.3. A possible invariant $I$ is the following:

$$I \cong \forall i,\ (\text{position}_i \neq n \Rightarrow \exists\ name_j\ (position_j = \text{position}_i + 1 \Rightarrow \text{name}_j > \text{name}_i)) \wedge (\forall k \in [1, n],\ 1 \leqslant \text{position}_k \leqslant n).$$

One can check easily, that the initialization is correct, actually it is clear that: $I_0 \cong (\forall k \in [1, n], \text{position}_k = n) \Rightarrow I$. It is also clear that the execution of a computation step keeps $I$ true. Finally, when a steady state is reached (as suggested in Section 3.3), the following conditions hold.

steady state $\Rightarrow (\forall i, j \in [1, n],\ i \neq j,\ \text{position}_i \neq \text{position}_j)$.

$I \wedge$ steady state $\Rightarrow$ names are correctly sorted.

**4.2. A parallel pretty printer.** This problem is concerned with the construction of a parallel pretty printer for programs made out of conditional clauses which may be nested at any level.

The following grammar describes the kind of conditional clauses we consider:

⟨conditional clause⟩::=

**if** ⟨clause⟩
  **then** ⟨clause⟩
  **else** ⟨clause⟩
**fi**

⟨clause⟩::= ⟨conditional clause⟩ ⟨expression without conditional clause⟩.

The purpose of the program to be constructed is to perform indentation of symbols according to given rules. We do not consider any constraint concerning line length.

*4.2.1. Conditions to be met by a pretty printed text.* In order to deal with the possibility of nested conditional clauses, let us define the nesting level. The nesting level is a number representing the static nesting depth. Let us give a simple example:

```
if
then      The enclosing conditional clause
| if      possesses nesting level 1,
| then    and enclosed conditional clauses possess
| else    nesting level 2 and 3.
| fi
else
| if
| then
| else
| fi
fi
```

Actually, the nesting level is associated with each symbol of conditional clauses, so:

(**if**, 1)
(**then**, 1)
(**if**, 2)
(**then**, 2)
(**else**, 2)
(**fi**, 2)
(**else**, 1)
(**if**, 3)
(**then**, 3)
(**else**, 3)
(**fi**, 3)
(**fi**, 1)

A program as represented on a screen or on a sheet of paper may be seen as an array of $n \times m$ symbols. The position of a symbol may be identified by its coordinates $(i, j)$. We define the indentation between two symbols by the difference in their second coordinate (the column).

The following table defines the indentation rules associated with conditional clauses:

| $T$ | (if, $k+r)(i+1)$ $\{r \geq 1\}$ | (then, $k)(i+r)$ $\{r \geq 1\}$ | (else, $k)(i+r)$ $\{r \geq 1\}$ | (fi, $k)(i+r)$ $\{r \geq 3\}$ |
|---|---|---|---|---|
| (if, $k)(i)$ | 1 | 2 | / | 0 |
| (then, $k)(i)$ | 1 | / | 0 | / |
| (else, $k)(i)$ | 1 | / | / | / |
| (fi, $k)(i)$ | / | / | / | / |

As an example, line 1 of table 1 can be read as follows: Given a symbol **if** of level $k$ with coordinates $(i, j)$,

(i) the coordinates of the **if** symbol beginning a conditional clause immediately nested within the considered conditional clause are $(i+1, j + T[(\textbf{if}, k), (\textbf{if}, k+1)])$, i.e., $(i+1, j+1)$,

(ii) the coordinates of the **then** symbol belonging to the same conditional clause are $(i+r, j+T([\textbf{if}, k), (\textbf{then}, k)]$, i.e., $(i+r, j+2)$, with $r >= 1$),

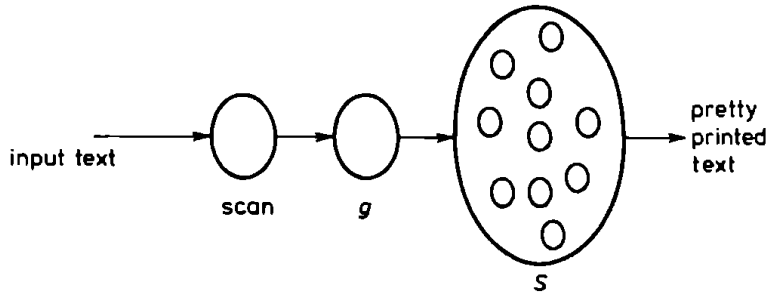
(iii) a **fi** symbol is situated on the same column as the opening **if** symbol, so if the coordinates of the **if** symbol are $(i, j)$ that of the corresponding *fi* symbol are $(i+r, j)$, with $r \geqslant 3$.

**4.2.2. Parallel solution to the problem.** The complete solution uses the following processing elements:

(i) a process, **scan**, which inputs the source text and produces and encoded version of it, where the nesting level is attached to each symbol.

(ii) A system $S$ of processes $p_i$'s. A process $p_i$ being associated with every symbol to be pretty-printed.

We can consider, that as far as **scan** progresses, new processes $p_i$'s are added to S. Actually, scanning a new symbol implies the creation of a new $p_i$ attached to this symbol. Assuming that these processes are generated by a process $g$, the overall organization of our solution is given by the following figure:




The system $S$ is built according to our cooperation scheme: the weight attached to every process of the system is a triple $(s, l, c)$, $s$ stands for the couple (symbol, nesting number), $l$ for line and $c$ for column.

Processes $p_i$ and $p_j$ possessing weights $w_i$ and $w_j$ cooperate if $(c_i - c_j) \neq T[s_i, s_j]$, and after the exchange of weights, the two computation steps performed by $p_i$ and $p_j$ ensure that the relation $(c_i - c_j) = T[s_i, s_j]$ becomes true. If the initialization is correct and if termination is ensured, then a steady state is reached, the set of weights will represent the solution.

In order to construct the solution, we have only to find the communication condition $R(w, \bar{w})$ and the text of processes, i.e., the function $f(w, \bar{w})$.

COMMUNICATION CONDITION. The condition $R(w, \bar{w})$ is directly derived from table $T$:

$R(w, \bar{w}) =$

**case** $(s, \bar{s})$ **of**
$((\textbf{if}, k), (\textbf{if}, k+1))$: $c \neq \bar{c}-1 \wedge l = \bar{l}-1$,
$((\textbf{if}, k), (\textbf{then}, k))$: $c \neq \bar{c}-2$,
$((\textbf{then}, k), (\textbf{if}, k+1))$: $c \neq \bar{c}-1 \wedge l = \bar{l}-1$,
$((\textbf{else}, k), (\textbf{if}, k+1))$: $c \neq \bar{c}-1 \wedge l = \bar{l}-1$,
$((\textbf{if}, k), (\textbf{fi}, k))$: $c \neq \bar{c}$,
$((\textbf{then}, k), (\textbf{else}, k))$: $c \neq \bar{c}$
**endcase**

The notation **case of ... endcase** has an obvious meaning: it allows us to discriminate on the symbols in order to describe precisely condition $R$.

TEXT OF PROCESSES. The text of processes belonging to system $S$ is also directly derived from table $T$. Actually, if two processes are such that $R(w, \bar{w})$ is **false**, then after exchange of their weigths and execution of a computation step, $R(w, \bar{w})$ becomes **true**.

Let us express $f(w, \bar{w})$ in the same generic way as we expressed $R(w, \bar{w})$

$f(w, \bar{w}) =$
**case** $(s, \bar{s})$ **of**
$((\textbf{if}, k), (\textbf{if}, k+1))$:
**if** $l > \bar{l}$
**then** $c := \bar{c}+1$
**fi**
$((\textbf{if}, k), (\textbf{then}, k))$:
**if** $l > \bar{l}$
**then** $c := \bar{c}+2$
**fi**,
$((\textbf{then}, k), (\textbf{if}, k+1))$:
**if** $l > \bar{l}$
**then** $c := \bar{c}+1$
**fi**,
$((\textbf{else}, k), (\textbf{if}, k+1))$:
**if** $l > \bar{l}$
**then** $c := \bar{c}+1$
**fi**,
$((\textbf{if}, k), (\textbf{fi}, k))$:
**if** $l > \bar{l}$
**then** $c := \bar{c}$
**fi**,

((**then**, $k$), (**else**, $k$)):
  **if** $l > \bar{l}$
    **then** $c := \bar{c}$
  **fi**
**end case**

*4.2.3. An example of functioning of the algorithm.* Consider the configuration to of the source text.

$\gamma_0$: **if**
**then**
**if**
**then**
**else**
**fi**
**else**
**fi**

$\gamma_0$ is transformed by the scanner into $\gamma_1$ which has the following form:
$\gamma_1$: {(**if**, 1), (**then**, 1), (**if**, 2), (**then**, 2), (**else**, 2), (**fi**, 2), (**else**, 1)(**fi**, 1)}.

One process per symbol is generated by $g$. The text of these processes are derived from $f(w, \bar{w})$. Then, this network of processes cooperate according to the communication condition $R(w, \bar{w})$. Configurations $\Delta_i$, $i \in [0, 4]$ give one among the possible set of configurations leading to a steady state. Two communicating processes are linked by a vertical line and a process is represented by its weight.

$\Delta_0$: ((**if**, 1), 0, 0)
((**then**, 1), 1, 0)
((**if**, 2), 2, 0)
((**then**, 2), 3, 0)
((**else**, 2), 4, 0)
((**fi**, 2), 5, 0)
((**else**, 1), 6, 0)
((**fi**, 1), 7, 0)

$\Delta_1$: ((**if**, 1), 0, 0)
((**then**, 1), 1, 2)
((**if**, 2), 2, 0)
((**then**, 2), 3, 2)
((**else**, 2), 4, 0)
((**fi**, 2), 5, 0)
((**else**, 1), 6, 0)
((**fi**, 1), 7, 0)

$\Delta_2$: ((**if**, 1), 0, 0)
((**then**, 1), 1, 2)
((**if**, 2), 2, 3)
((**then**, 2), 3, 2)
((**else**, 2), 4, 2)
((**fi**, 2), 5, 0)
((**else**, 1), 6, 0)
((**fi**, 1), 7, 0)

$\Delta_3$: ((**if**, 1), 0, 0)
((**then**, 1), 1, 2)
((**if**, 2), 2, 3)
((**then**, 2), 3, 5)
((**else**, 2), 4, 2)
((**fi**, 2), 5, 0)
((**else**, 1), 6, 2)
((**fi**, 1), 7, 0)

$\Delta_4$: ((**if**, 1), 0, 0)
((**then**, 1), 1, 2)
((**if**, 2), 2, 3)
((**then**, 2), 3, 5)
((**else**, 2), 4, 5)
((**fi**, 2), 5, 3)
((**else**, 1), 6, 2)
((**fi**, 1), 7, 0)

$\Delta_4$ represents a steady state and the text can be correctly pretty-printed.

*4.2.4. Invariance properties and termination.* A possible invariant for our solution is the following:

$$I \equiv ((\exists c_u\colon c_u > K) \Rightarrow (\exists w_i, w_j, l_j = l_i + r \wedge c_j = c_i + T[S_i, S_j])) \wedge K \leqslant c_i, c_j \leqslant \mathbf{max},$$

$r$ depends on the couple of symbols which are considered (see table $T$).

As far as initialization is concerned, we have chosen: $I_0 = \forall w_i, c_i = K$.

$K$ being a constant indicating the column number of the first **if** symbol, in the example, we have chosen $k = 0$. **max** is an upper bound of the maximum number of columns necessary to represent the pretty printed text, for example, **max** can be chosen as 2* number of symbols.

Clearly, $I_0 \Rightarrow I$, so the initialization is correct. It is also clear that the execution of a computation step keeps $I$ true (see $f(w, \bar{w})$). Finally, when a steady state is reached the following conditions hold:

**steady state** $\Rightarrow \forall i, j \in [1, n], i \neq j, \neg R(w_i, w_j)$

$I \wedge$ **steady state** $\Rightarrow$ **the conditions for pretty printing are realized.**

As far as termination is concerned, we can choose the following termination functions (one per process $p_i$):

$$t_i\colon w_i \to N, \quad t_i(w_i) = c_i.$$

It is clear that following every exchange, one and only one out of the two computation steps will *increase* a column number. So given a multiset $w_l = \{c_1, \ldots, c_n\}$ representing the column numbers reached after some computation steps, it is obvious that following a new exchange (and a new computation step), $w_{l+1}$ will be greater than $w_l$. Now, as the maximum distance (in terms of columns) between two symbols is 2, we can state that the maximum breadth of the pretty printed text will be less than $K + 2*n$, if $n$ is the number of symbols and if the first **if** is placed in column $K$.

This concludes the termination proof, as the set of processes $S$ generates an increasing sequence of multisets and as an upper bound for this sequence has been exhibited.

**4.3. Other applications.** This section introduces solution to problems where termination considerations is not of prime importance. Its purpose is only to suggest other fields of application for the proposed cooperation scheme.

*4.3.1. Identifier binding in compilation.* The problem consists of associating an identifier use with its declaration, as it appears in block-structured languages. For example, consider the following PASCAL-like program skeleton:

```
      begin
(1)   a: integer;
(2)   b: real;
      ...
(3)   a:= a+7;
(4)   b:= 6.1
      end
```

Declarations of identifiers $a$ and $b$ appear in lines (1) and (2). They are used in lines (3) and (4). A parallel compiler could operate as follows:

While processing the source text, a process generator creates an instance of process **decl** when finding a declaration and an instance of process **use** when finding an identifier use.

The shape of these processes is as follows:

```
decl::*[ (id, type, block) ◇skip]
 use::*[ (id, block) ◇
           # produces code from the properties of the declaration #;
         (id, block):= (skip, skip)
       ]
```

The condition $R(w, \bar{w})$ which will allow the process **use** to be supplied with the properties of the declaration can be expressed as:

$$R(w_{\text{decl}}, w_{\text{use}}) \equiv \mathbf{id}_{\text{decl}} = \mathbf{id}_{\text{use}}.$$

In this example, two types of processes constitute the system, so it is necessary to be able to discriminate their weights – $\text{id}_{\text{decl}}$ refers to the id part of $w_{\text{decl}}$ and $\text{id}_{\text{use}}$ to the id part of $w_{\text{use}}$. The last instruction of the process use can be viewed as a destruction instruction as it is not possible to match **skip** with any identifier (i.e., $\forall w_{\text{decl}}, \neg R(w_{\text{decl}}, (\mathbf{skip}, \mathbf{skip}))$. So any instance of process **use** "commits suicide" after code production.

This very elegant solution can be compared with that proposed in [1]. In this latter paper, a data structure (varisized set of events) is still necessary in order to represent identifier tables. This data structure is completely hidden by the cooperation. This scheme with the present solution.

*4.3.2. An infinite clock.* Consider the system of two processes $\pi_1$ and $\pi_2$ built as follows:

```
π1::*[(value) ◇                    π2::*[(value) ◇
        if value = 0                       if value = 0
          then value:= 1                     then value:= 1
          else value:= 0                     else value:= 0
        fi                                 fi
    ]                                  ]
```

With the condition $R(w, \bar{w}) = |\text{value} - \overline{\text{value}}| = 1$, this system of processes will go into an infinite computation, provided that the initialization is correct (i.e., one out of the two processes has its weight to 1 and the other its weight to 0). So, processes $\pi_1$ and $\pi_2$ will produce the infinite sequence $(01^\omega, 10^\omega)$, if $\pi_1$ is initialized to 0 and $\pi_2$ to 1.

## 5. Discussion and conclusions

This paper proposes a cooperation scheme and its application to the construction of parallel programs. Logical properties of this cooperation scheme are given in terms of Generalized Hoare Logic and particular emphasis is put on termination aspects. Finally, some examples are developed which should suggest a method for parallel program construction.

As far as further research is concerned, several aspects need investigations, such as:

(1) identifying the class of problems which may naturally be solved by using the proposed cooperation scheme. We think of numerical calculations using relaxation methods, but one can also think of computations using cellular automata. Section 4 develops examples taken from non-numerical computing (pretty-printer, compilation);

(2) pursuing efforts on concurrent programming methodology. Actually, the ideas presented in this paper constitute a good starting point, but further studies are still needed in particular when dealing with non convergent systems of processes;

(3) building and experimenting prototype implementations. Some work has been done in this area, but we still need further research in order to produce efficient distributed implementations. In particular, we have to deal with the problem of "distributing" the evaluation of condition $R$ which appears as a "global" condition in our solutions.

## References

[1] F. André, J. P. Banâtre, J. P. Routeau, *A Multiprocessing Approach to Compile time Symbol Resolution*, ACM, TOPLAS 3, 1 (Jan. 1981), 11–23.

[2] N. Dershowitz, Z. Manna, *Proving termination with multiset ordering*, CACM 22, 8 (Aug. 1979), 465–476.

[3] E. W. Dijkstra, *A discipline of programming*, Prentice Hall (1976).

[4] L. Lamport, F. Schneider, *The "Hoare logic" of CSP, and All That*, ACM, TOPLAS 6, 2 (April 1984) 281–296.

*Presented to the semester*
*Mathematical Problems in Computation Theory*
*September 16 – December 14, 1985*

MATHEMATICAL PROBLEMS IN COMPUTATION THEORY
BANACH CENTER PUBLICATIONS, VOLUME 21
PWN – POLISH SCIENTIFIC PUBLISHERS
WARSAW 1988

# THE COMPLEXITY OF CLASSIFICATION PROBLEMS

L. BUDACH

*Laboratoire Informatique Théorique et Programmation, Université Paris, Institut de Programmation, Paris*
on leave of absence from
*Humboldt-Universität, Sektion Mathematik, Berlin, GDR*

## 1. Information systems and classification problems

**1.1.** An *information system* (see [8]) is a quadrupel $S = (X, A, V, r)$, where $X$, $A$, $V$ are finite sets and $r$ is a mapping of $X \times A$ into $V$. The set $X$ is interpreted as the set of all *objects* under consideration, $A$ is the set of all *attributes*, and $V$ is the set of *descriptors*. The mapping $r$ is the so-called *information function*.

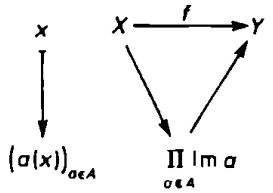
Let $a \in A$ be an attribute; $a$ defines a function

$$X \to V, \quad x \to r(x, a),$$

of $X$ into $V$. We assume that different attributes define different functions. This enables us to identify the attribute $a$ with the corresponding function and to write $a(x)$ instead of $r(a, x)$. Let $\operatorname{Im} a = \{a(x) | x \in X\}$ be the image of the function $a$, then $a$ can be considered as a function of $X$ onto $\operatorname{Im} a$. By abuse of language we consider $A$ to be a set of functions $a$: $X \to \operatorname{Im} a$ and write $S = (X, A)$ instead of $S = (X, A, V, r)$. Let $f$: $X \to Y$ be a mapping. We call $f$ to be *dependent on* $S$ if the following condition is satisfied:

If $a(x_1) = a(x_2)$ for all $a \in A$, then $f(x_1) = f(x_2)$, $x_1, x_2 \in X$.

The mapping $f$ is dependent on $S$ iff there is a function $\prod_{a \in A} \operatorname{Im} a \to Y$, such that the following diagram is commutative:



A tripel $C = (X, A, f)$ such that $(X, A)$ is an information system, and $f$: $X \to Y$ is a function dependent on $(X, A)$ is called a *classification problem.* The function $f$ is called the *classifying function* or the *classification.*

By some technical reasons we will assume throughout the following paper, that all information systems satisfy the following condition:

$S$ is fully faithfull, i.e., the function $X \to \prod_{a \in A} \operatorname{Im} a$ is bijective. This implies that all functions $f$: $X \to Y$ are dependent on $S$.

### 1.2. Examples

**1.2.1.** Every boolean function $f$: $\{0, 1\}^n \to \{0, 1\}$ defines a classification problem. The underlying information system consists of $X = \{0, 1\}^n$ as set of objects, $A = \{1, 2, \ldots, n\}$ is the set of attributes, $V = \{0, 1\}$ is the set of descriptors, $r = X \times A \to V$ is the selection function defined by $r((x_1, \ldots, x_n), i) = x_i$, and $f$ is the classifying function.

**1.2.2.** Let $\Sigma$ be a finite aphabet and $L$ be a language over $\Sigma$. $L$ is a subset of $\Sigma^*$ and defines a subset $L^n := L \cap \Sigma^n$ for any natural number $n$. Let $f_n$: $\Sigma^n \to \{0, 1\}$ be the characteristic function of $L^n$, i.e.,

$$f_n(w) = \textit{if } w \in L^n \textit{ then } 1 \textit{ else } 0.$$

Therefore $L$ defines for every $n$ a classification problem with information function $f_n$ and the underlying information system $(\Sigma^n, \{1, 2, \ldots, n\}, \Sigma, r)$ with $r((\sigma_1, \ldots, \sigma_n), i) = \sigma_i$.

**1.2.3.** Let $V$ be a finite set, called the set of vertices and let $v_0$, $v_e$ be two distinguished vertices of $V$. Take $V' = V - \{v_e\}$. A *V-Maze* is a function $d$: $V' \times \{0, 1\} \to V$ (see [1], [2], [11]). A $V$-maze can be considered as a directed graph $\Gamma(d)$, $V$ being the set of vertices and $E = \{(v, d(v, i)) \mid v \in V', i \in \{0, 1\}\}$ the set of edges. This graph has two distinguished nodes $v_0$ and $v_e$. The outdegree of all nodes different from $v_e$ is two and $v_e$ is a sink of this graph, its outdegree being zero. The $V$-maze $d$ is called to be threadable, if there is a path in $\Gamma(d)$ connecting $v_0$ with $v_e$.

Let $X = \operatorname{Map}(V' \times \{0, 1\}, V)$ be the set of $V$-mazes. These are the objects of the following information system:

$A = V' \times \{0, 1\}$ is the set of attributes,

$V$ is the set of descriptors and

$r$: $X \times A \to V$ is the function defined by $r(d, (v, i)) := d(v, i)$.

$\operatorname{MAZES}(V) := (X, A, V, r)$ is an information system which is the underlying information system of the classification problem $\operatorname{GAP}(V, v_0, v_e)$ the classifying function $t$ of which is defined by

$$t(d) := \textit{if } d \textit{ is threadable then } 1 \textit{ else } 0.$$

Since all $\operatorname{GAP}(V, v_0, v_e)$ with $\#(V) = n$ are isomorphic we write $\operatorname{GAP}(n)$

instead of GAP$(V, v_0, v_e)$. Without restriction of generality we can assume $V = \{1, 2, \ldots, n\}$ and $v_0 = 1$, $v_e = n$.

## 2. Questionnaires or classifying graphs

**2.1.** A procedure to classify via a given classifying function are the questionnaires introduced by C. Picard [9] (see also [2] and [4]). A questionnaire or a classifying graph over a given information system $S = (X, A)$ is a quintuple $F = (Q, Y, \alpha, \delta, q_0)$, where

$Q$ is a finite set, the set of nodes;

$Y$ is a finite set with $Y \cap A = \emptyset$ and $\alpha$: $Q \to Y \cup A$ is a mapping; the nodes of act $F := \alpha^{-1}(A)$ are called questions and the nodes of term $F := \alpha^{-1}(Y)$ are the results;

$\delta = \{\delta_q | \ q \in \text{act } F\}$, $\delta_q$: $\text{Im}\,\alpha(q) \to Q$ describes the strategy of posing questions;

$q_0$ is the initial node, i.e., that node, in which all enquiries get started.

$X$ operates partially on $Q$ by the following function:

$$\text{act } F \times X \to Q, \quad (q, x) \mapsto qx := \delta_q((\alpha(q))(x)).$$

This action can be interpreted as follows: let $q$ be a question; then $\alpha(q)$ is an attribute, i.e., a mapping $\alpha(q)$: $X \to V$. To pose question $q$ to the object $x \in X$ means to apply $\alpha(q)$ on $x$. The answer of $x$ to the question $q$ is $\alpha(q)(x)$. This answer implies a new question or a result, namely $\delta_q(\alpha(q)(x))$ which we called $qx$. The partial action of $X$ on $Q$ can be extended to $X^*$, the free monoid generated by $X$. For $x \in X$ take $qx^{n+1} := (qx^n)x$ if $qx^n \in \text{act } F$. Let $\xi_F$: $X \to Y$ be the following function

$$\xi_F(x) := \textit{if } q_0 x^n \in \text{term } F \textit{ then } \alpha(q_0 x^n) \textit{ else not defined}.$$

The sequence $q_0, q_0 x, q_0 x^2, \ldots, q_0 x^n \in \text{term } F$ describes the strategy of $F$ in asking questions: after having asked for the attribute $\alpha(q_0 x^m)$, $m < n$, $F$ gets the answer $(\alpha(q_0 x^m))(x)$ which makes $F$ move to the node $q_0 x^{m+1}$ if $m+1 < n$ or the result of the enquiry $\xi_F(x) = \alpha(q_0 x^n)$.

If $F$ is free of cycles (more precisely if the directed graph

$$\Gamma(F) = \left(Q, \{(g, \delta_q(i)) | \ q \in \text{act } Q, \ i \in \text{Im}\,\alpha(q)\}\right)$$

is free of cycles), then $qx^n \neq q$ for all $q \in \text{act } F$, $x \in X$. In this case $\xi_F$ is fully defined on $X$. So if we assume that $F$ is free of cycles, then $\xi_F$ is a fully defined function, which can be proved to be always (i.e., also in case when we do not assume that $S$ is fully faithful) dependent on $S$. We say that $F$ is a solution of a classification problem $C = (X, A, f)$ if $\xi_F = f$.

**2.2.** Easy to verify that every classification problem admits a solution $F$ which is, moreover, a tree (for the easy proof of this fact we refer the reader

to [2]). In [2] and [4] we introduced different measures for the complexity of classifying graphs. One of these measure was the size of $F$:

$$\text{size}(F) = \#(Q).$$

Let $C = (X, A, f)$ be any classification problem. We introduce two numbers:

$$\text{size}(C) = \min \{\text{size}(F) |\ \xi_F = f\}, \quad \text{and}$$

$$\text{Size}(C) = \min \{\text{size}(F) |\ \xi_F = f \text{ and } \Gamma(F) \text{ is a tree}\}.$$

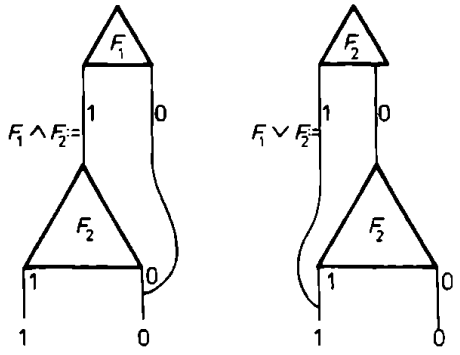
One of the most interesting and outstanding problems in theoretical computer science is the determination of the "small size" size$(C)$ of certain classification problems $C$. Though also the determination of Size$(C)$ is not easy, it can be done in certain cases. The following section presents some results concerning these questions. The detailed proofs of these results can be read in [2].

## 3. Classifying graphs for GAP($n$) and related problems

**3.1.** Assume $Y = \{0, 1\}$ and let $F_1$, $F_2$ be classifying graphs with

$$F_i = (Q_i, Y, \alpha_i, \delta_i, q_{i0}), \quad i = 1, 2.$$

We define $F_1 \wedge F_2$ and $F_1 \vee F_2$ in the following straightforward manner:




Obviously

$$\text{size}(F_1 \wedge F_2) = \text{size}(F_1 \vee F_2) = \text{size}(F_1) + \text{size}(F_2) - 2$$

and

$$\xi_{F_1 \vee F_2} = \xi_{F_1} \vee \xi_{F_2}, \quad \xi_{F_1 \wedge F_2} = \xi_{F_1} \wedge \xi_{F_2}.$$

**3.2.** Consider the following classification problems:

In 1.2.3 we introduced already the information system MAZES$(V)$ and the classification problem GAP$(V, v_0, v_e)$. The latter classification problem can be considered as a special case of the following classification problem.

Call a $V$-maze $d$ $k$-threadable for a given natural number $k$, if it is threadable and if the path connecting $v_0$ with $v_e$ has a length smaller or equal to $k$. Let $t_k = t_k(V, v_0, v_e)$ be the characteristic function of the set of all $k$-threadable mazes, i.e.,

$$t_k(d) = \text{if } d \text{ is } k\text{-threadable then 1 else 0.}$$

Let $\mathrm{GAP}_k(V, v_0, v_e)$ be the corresponding classification problem. Obviously $\mathrm{GAP}(V, v_0, v_e) = \mathrm{GAP}_k(V, v_0, v_e)$ with $k = \#(V)-1$. It is easy to verify that one has the following equality:

$$t_{k+l} = V\{t_k(V, v_0, v_e) \wedge t(V, v, v_e) \mid v \in V,\ v \neq v_0,\ v \neq v_e\} \vee t_1(V, v_0, v_e).$$

From this equation one gets the following recursion formula for $s(k, n) := \mathrm{size}(\mathrm{GAP}_k(V, v_0, v_e))$ with $n := \#(V)$:

$$\begin{aligned} s(k+l, n) &\leqslant \sum_{i=2}^{n-1} \big(s(k, n)+s(l, n)-2\big)-2(n-3)+4-2 \\ &= (n-2)\big(s(k, n)+s(l, n)-4\big)+4. \end{aligned}$$

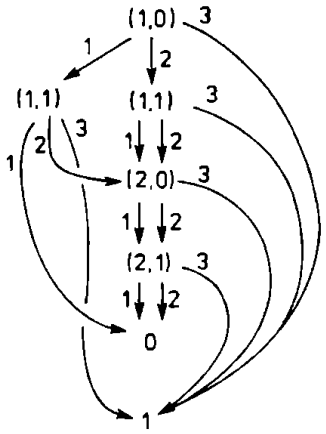
From this formula results:

$$s(2k, n) \leqslant 2(n-2)\big(s(k, n)-2\big)+4 \leqslant 2(n-1)s(k, n)$$

and this gives the following upper bound for $s(n) = \mathrm{size}(\mathrm{GAP}(n))$.

$$s(n) \leqslant 4n(n-1)^{\log n}.$$

**3.3.** Let us give an example: An optimal solution for the classification problem GAP(3), i.e., a classifying graph for GAP(3) with the minimal number $s(3) = 7$ of nodes is the following classifying graph:




**3.4.** The above example is the easiest case of the following solution $F$ for $\mathrm{GAP}(n)$ which is in general not optimal but better than 3.2, for $n$ small:

$$\mathrm{act}\, F = \{(U, n) \in 2^V \times N \mid 1 \in U,\ n \in U,\ \#(U)-1 \leqslant n < 2\#(U)\},$$

$\operatorname{term} F = \{0, 1\} = Y$,

$\alpha(U, n) :=$ $n$-th element of $U \times 2$ in lexicographical order,

$$\delta_{(U,n)} = \begin{cases} (U \cup \{y\}, n+1) & \text{if} \quad n+1 < 2 \# (U \cup \{y\}) \text{ and } y \neq v_e, \\ 1 & \text{if} \quad y = v_e, \\ 0 & \text{if} \quad n+1 = 2 \# (U \cup \{y\}), \end{cases}$$

$$q_0 = (\{v_0\}, 0).$$

It is easy to see, that

$$\operatorname{size} F = 2 + \sum_{i=0}^{n-2} \binom{n-1}{i} (i+2).$$

For $n = 3$ we get $\operatorname{size} F = 7$, and for $n = 4$, $\operatorname{size} F = 14$. The first value is optimal and we believe that also $s_4 = 14$. But already the proof of this fact seems to be hard. The importance of the numbers $s_n$ is demonstrated by the following theorem:

**3.5.** THEOREM. *Let* $\mathbf{L}$ *be the class of all languages, which can be recognized by a Turing machine with logarithmic tape and let* $\mathbf{NL}$ *be the class of all languages which can be recognized by a nondeterministic Turing machine in logarithmic tape. Obviously* $\mathbf{L} \subseteq \mathbf{NL}$. *In order that* $\mathbf{L} = \mathbf{NL}$ *it is necessary that* $s_n$ *is polynomial in* $n$.

The proof of this theorem can be found in [2].

## 4. Classifying trees

**4.1.** Let $S = (X, A)$ be an (fully faithful) information system. Let $\operatorname{trees}(S)$ be the set of all classifying trees over the given information system $S$. To every attribute $a \in A$ with $\operatorname{Im} \alpha = \{y_1, \ldots, y_n\}$ there is an $n$-ary function $a$: $\operatorname{trees}(S)^n \to \operatorname{trees}(S)$ which is defined in the following way: let $F_1, \ldots, F_n$ be elements of $\operatorname{tree}(S)$ with

$$F_i = (Q_i, Y_{i,i}, \delta_i, q_{i0});$$

then $F := a(F_1, \ldots, F_n) = (Q, Y, \alpha, \delta, q_0)$ is defined as follows:

$$Q = \{a\} \cup \Big(\bigcup_{i=1}^{n} Q_i \times \{i\}\Big), \qquad Y = \bigcup_{i=1}^{n} Y_i, \qquad \alpha(a) := a, \qquad \alpha(q, i) := \alpha_i(q).$$

Then we get $\operatorname{act}(F) = \{a\} \cup \Big(\bigcup_{i=1}^{n} \operatorname{act}(F_i) \times \{i\}\Big)$. The function $\delta$ is defined by $\delta_{(q,i)}(y) := (\delta_q(y), i)$ and $\delta_{a(i)} := (q_{i0}, i)$. The initial node $q_0$ of $F$ is defined by $q_0 := a$, which completes the definition of $F$. Suppose we are given a fixed set $Y$. To every element $y \in Y$ we define the following trivial classifying tree:

$$[y] := (\{y\}, \{y\}, 1_{\{y\}}, \emptyset, \{y\}),$$

consisting only of one node. For the next theorem we consider by set theoretic reasons only classifying trees $F = (Q, Y, \alpha, \delta, q_0)$ with a fixed set $Y$ of possible results of the classification.

**4.2.** THEOREM. *(trees$(S)$, $A)$ is a free algebra and the set of all $[y]$ forms a set of free generators.*

The proof of this theorem is given in [2].

**4.3.** Consider the following binary relation $\triangleq$ on the set of all classifying trees.

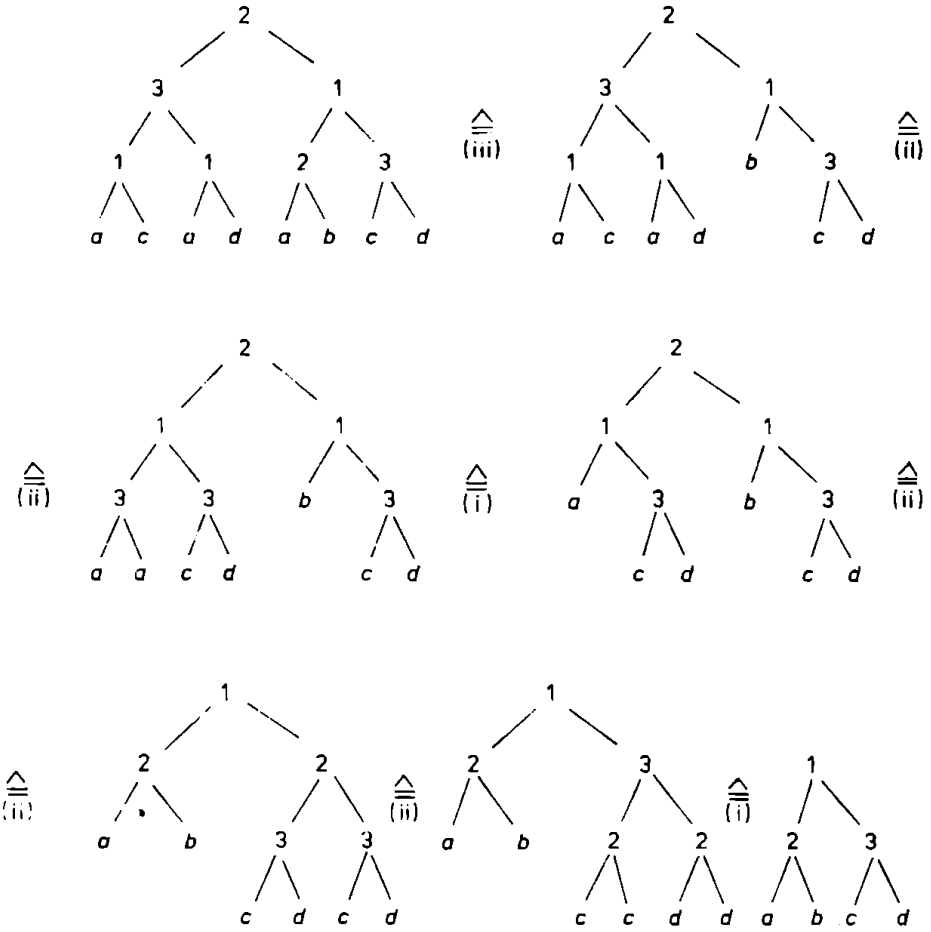
(i) *for $a \in A$, $y \in Y$ holds $a([y], \ldots, [y]) \triangleq [y]$,*

(ii) *if $a$, $b \in A$ then $a(b(F_{11}, \ldots, F_{1n}), \ldots, b(F_{m1}, \ldots, F_{mn})) \triangleq b(a(F_{11}, \ldots, F_{m1}), \ldots, a(F_{1n}, \ldots, F_{mn}))$;*

(iii) *Let $F = \mu(F_1, \ldots, F_n)$ and assume $F_i$ contains a* subtree $F' = \mu(F'_1, \ldots, F'_n)$. *Let $F''$ be the tree obtained from $F$ by replacing $F'$ by $F'_i$. Then $F \triangleq F''$.*

**4.4.** DEFINITION. The congruence relation $\equiv$ generated by $\triangleq$ will be called the *syntactic congruence* of classifying trees.

Let us consider an example:



**4.5.** Two classifying trees $F_1$ and $F_2$ will be called *semantical equivalent*, $F_1 \sim F_2$, of their classifying functions are identical: $\xi_{F_1} = \xi_{F_2}$. It is obvious that classifying trees which are syntactical equivalent are semantical equivalent too. More interesting is the other direction which will be the main result of the following theorem, the proof of which will be found also in [2].

**4.6.** THEOREM. *Syntactical and semantical equivalence are equal, i.e., for all classifying trees holds: $F_1 \sim F_2$ if and only if $F_1 \equiv F_2$.*

WARNING. For this theorem the assumption that $S$ is fully faithful is of significant importance. See example 4.13 of [2].

## 5. Optimal trees for GAP($n$)

**5.1.** In 2.2 we introduced the notion of the "big size" of a classification problem $C = (X, A, f)$:

$$\mathrm{Size}(C) := \min \{\mathrm{size}(F) \mid \xi_F = f \text{ and } \Gamma(F) \text{ is a tree}\}.$$

A classifying tree $F$ is called an *optimal tree solution of* $C$ or for short an *optimal tree for* $C$ if $F$ is a solution of $C$ and if, moreover, $\mathrm{size}(F) = \mathrm{Size}(C)$. Since every classification problem has a solution which is a tree, every classification has an optimal tree solution. The example at the end of 4.4 gives evidence that it might not be easy to find an optimal tree for $C$ and, moreover, it may be rather difficult to decide whether a given classifying tree is an optimal solution. We will find in 7.7 a criterion which allows to answer the second question in certain cases.

Let us first consider the problem GAP($n$).

Let $\sigma(n) := \mathrm{Size}(\mathrm{GAP}(n))$. We intend now to give a recursion which allows to compute these numbers and to give lower bounds for $\sigma(n)$.

**5.2.** Let $V$ be a finite set with two distinguished elements $v_0$ and $v_e$. In 1.2.3 we introduced $V$-mazes as functions $d: V' \times \{0, 1\} \to V$ with $V' = V \setminus \{v_e\}$. A partial $V$-maze is a partial function $d: V' \times \{0, 1\} \xrightarrow{\supseteq} V$. As for $V$-mazes we can consider the directed graph $\Gamma(d)$ with $V$ being the set of vertices of $\Gamma(d)$ and $E$, the set of edges being defined by

$$E := \{(v, d(v, i)) \mid (v, i) \in \mathrm{dom}\, d\}.$$

Let $d$ be a partial $V$-maze. Let reach $d$ be the set of all $v \in V$ which are connected with $v_0$ by a path from $v_0$ to $v$. We call $d$ a *trunk* if $\mathrm{dom}\, d \subseteq (\mathrm{reach}\, d) \times \{0, 1\}$, and $d$ is called a *complete trunk* if in this inclusion equality holds. We call $d$ to be *connected* if $v_e \in \mathrm{reach}\ d$. We call $d$ *disconnected* if it is not connected, and *stably disconnected* if all extensions $d' \supseteq d$ of $d$ are disconnected.

Let $d$ be any partial maze. Define $\tilde{d}$ by

$$\tilde{d} := d\downarrow((\operatorname{reach} d) \times \{0, 1\}) \cap \operatorname{dom} d.$$

Obviously $\tilde{d}$ is a trunk and it is easy to see that $d$ is stably disconnected if and only if $\tilde{d}$ is a complete disconnected trunk.

Let $F$ be a tree solution of $\mathrm{GAP}(V, v_0, v_e)$ and let $q$ be any node of $F$. Let

$$q_0 \xrightarrow{x_0} q_1 \xrightarrow{x_1} q_2 \xrightarrow{x_2} \ldots \to q_n = q$$

be the path in $F$ connecting the initial node $q_0$ of $F$ with $q$. Since $\alpha(q_t) \in A = V' \times \{0, 1\}$, we have $\alpha(q_t) = (v_t, i_t)$. Consider the set $\{((v_t, i_t), x_t) \mid t = 0, 1, \ldots, n-1\}$ which we call $d_q$.

**5.3.** THEOREM. *If $F$ is an optimal tree for $\mathrm{GAP}(V, v_0, v_e)$, then $d_q$ is the graph of a partial function $d_q$: $V' \times \{0, 1\} \xrightarrow{\supset} V$, i.e., $d_q$ is a partial maze. Moreover:*

(i) *$d_q$ is a trunk for all nodes $q$ of $F$;*

(ii) *for all $q \in \operatorname{act} F$, $d_q$ is not complete nor connected;*

(iii) *for all $q \in \operatorname{term} F$ with $\alpha(q) = 1$, $d_q$ is connected;*

(iv) *for all $q \in \operatorname{term} F$ with $\alpha(q) = 0$, $d_q$ is complete and disconnected, i.e., stably disconnected.*

**5.4.** Every partial maze $d$ defines a point in the grid $N^2$ by

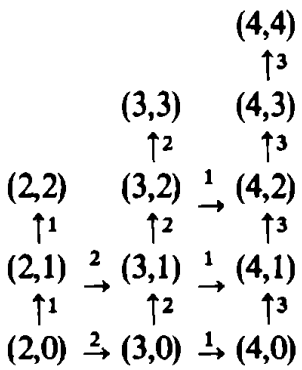
$$\eta(d) := (\#(\operatorname{reach} d) + 1,\ \#(\operatorname{dom} \tilde{d}) - \#(\operatorname{reach} d) + 1).$$

Every node $q$ of as optimal tree for $\mathrm{GAP}(V, v_0, v_e)$ defines a point $\eta(q) := \eta(d_q)$ of $N^2$.

Consider the following graph $A^n = (V^n, E^n)$:

$$V^n = \{(x, y) \in N^2 \mid 0 \leqslant y \leqslant x,\ 2 \leqslant x \leqslant n\} \cup \{1, 2, \ldots, 2(n-1)\},$$
$$E^n = \{((x, y), i, (x+1, y)) \mid i \in \{1, 2, \ldots, n-x\}\} \cup$$
$$\cup \{((x, y), i, (x, y+1)) \mid i \in \{1, 2, \ldots, x-1\},\ y < x\} \cup$$
$$\cup \{((x, y), 1, x+y-1) \mid y < x\}.$$

Take $A^4$ as an example where we have omitted in the pictorial representation the edges of the third kind between $(x, y)$ and $x+y-1$:



**5.5.** THEOREM. *If $F$ is an optimal solution for* GAP($n$), *then $F$ is a covering tree of the graph* $\mathbf{1}^n$. *This implies that* $\sigma(n) = \mathrm{Size}(F)$ *is equal to the number of simple pathes in* $\mathbf{1}^n$.

**5.6.** COROLLARY. $\sigma(n) = \Omega(n^n(n-2)!)$.

For details we refer the reader to [2].

## 6. Coloured posets

**6.1.** A finite poset is said to be *pure* if all maximal chains have the same length. A pure poset satisfies the Jordan–Dedekind condition: if $x$ and $y$ are two elements and if $x < y$, then $I_x = \{z|\ z \leqslant x\}$, $V_x = \{z|\ x \leqslant z\}$ and $[x, y] = \{z|\ x \leqslant z \leqslant y\}$ are pure.

The symbol "$\prec$" denotes the covering relation: $x \prec y$ if $x < y$ and if $x < z \leqslant y$ implies $z = y$. If $P$ is a finite pure poset, then a rank function $r$: $P \to \boldsymbol{N}$ can be defined as follows:

(i) If $P$ has a least element 0, then we define $r(0) = 0$, otherwise we define $r(x) = 1$ for all minimal elements $x$.

(ii) If $x \prec y$, then $r(y) = r(x) + 1$.

**6.2.** A finite simplicial complex $K$ is by definition a nonempty family of nonempty subsets called *simplexes* of a set $\{v\}$ of vertices such that

(i) any set consisting of exactly one vertex is a simplex;

(ii) any nonempty subset of a simplex is a simplex.

(For details we refer to [12].) The dimension of a simplex $s$, $\dim s$, is $\# s - 1$. The dimension of $K$, $\dim K$, is $\max\{\dim s|\ s \in K\}$. The maximal simplexes, i.e., those simplexes which are maximal under inclusion, are called facets. $K$ is said to be *homogeneously n-dimensional* if every simplex belongs to an $n$-simplex of $K$. So all facets are $n$-dimensional.

Every finite simplicial complex $K$ defines a finite poset $(K, \subseteq)$ the elements of which are the simplexes of $K$ and these are ordered by inclusion. If $K$ is homogeneously $n$-dimensional, then the corresponding poset is pure. Its rank function $\varrho$ satisfies obviously the following condition: $\varrho(s) = \dim s + 1$.

Let $P$ be an arbitrary ordered set. It defines a simplicial complex $\Delta(P)$ in the following way: The vertices of $\Delta(P)$ are the elements of $P$ and the simplexes of $\Delta(P)$ are nonempty subsets $\{x_0, x_1, \ldots, x_k\}$ of $P$ such that $x_0 < x_1 < \ldots < x_k$. If $K$ is a simplicial complex, then $K' = \Delta(K)$ ($K$ to be considered as poset) is the barycentric subdivision of $K$.

**6.3.** EXAMPLE. Let $S = (X, A, V, \varrho)$ be an information system with $N := \# A - 1$. We assume as usual that $S$ is fully faithful. An $S$-condition is

defined to be a partial function $c$: $A \xrightarrow{\supset} V$ satisfying $c(a) \in \operatorname{Im} a$ for all $a \in \operatorname{dom} c$. As usual $c$ can be considered as a subset of the product $A \times V$,

$$c = \{(a, v) |\ a \in \operatorname{dom} c,\ v = c(a)\}.$$

Consider the following simplicial complex:

(i) The set of vertices is $A \times V$.

(ii) The set $\operatorname{Cond}(S)$ of simplexes is the set of all $S$-conditions (considered as subsets of $A \times V$).

The facets of $\operatorname{Cond}(S)$ are the fully defined functions $c$: $A \to V$. They all are of dimension $N$. Hence $\operatorname{Cond}(S)$ is a homogeneously $N$-dimensional simplicial complex.

**6.4.** Let $P$ be a finite pure poset. A partial function $g$: $P \xrightarrow{\supset} Y$ is called a *precolouring of* $P$ and $(P, g, Y)$ is called a *precoloured poset* if

(i) all maximal elements of $P$ belong to the domain of $g$,

(ii) if $x < y$ and $x, y \in \operatorname{dom} g$, then $g(x) = g(y)$.

If $x$ is an element of $\operatorname{dom} g$, then we say that $x$ is *coloured* and $g(x)$ is the *colour of* $x$.

Let $(P_i, g_i, Y_i)$ $(i = 1, 2)$ be two precoloured posets. A morphism of $(P_1, g_1, Y_1)$ into $(P_2, g_2, Y_2)$ is a pair $(\pi, \eta)$ consisting of an order preserving map $\pi$: $P_1 \to P_2$ and a function $\eta$: $Y_1 \to Y_2$ such that the diagram

$$\begin{array}{ccc} P_1 & \xrightarrow{g_1} & Y_1 \\ {\scriptstyle\pi}\downarrow & & \downarrow{\scriptstyle\eta} \\ P_2 & \xrightarrow[g_2]{} & Y_2 \end{array}$$

is commutative. This means more precisely: Whenever $x \in P_1$ is coloured, $\pi(x)$ is coloured and $g_2 \pi(x) = \eta g_1(x)$.

**6.5.** Let $(P, g, Y)$ be a precoloured poset. $g$ is a colouring of $P$ and $(P, g, Y)$ is called a *coloured poset* if in addition to properties (i) and (ii) in (6.4) the following property holds:

(iii) If $x \in P$ and if all $z$, covering $x$, are coloured and have the same colour $g(z) = y_0 \in Y$, then $x$ is coloured and $g(x) = y_0$.

Let $\max P$ be the set of all maximal elements of $P$. Condition (iii) is equivalent to either one of the following conditions:

(iii') If $x \in P$ and if all $z$ with $x < z$ are coloured and have the same colour $g(z) = y_0 \in Y$, then $x$ is coloured and $g(x) = y_0$.

(iii'') If $x \in P$ and if all $z \in V_x \cap \max P = \{y \in \max P |\ x \leqslant y\}$ have the same colour $g(z) = y_0 \in Y$, then $x$ is coloured and $g(x) = y_0$.

Obviously every precolouring $g$ can be extended in a unique way to a colouring $\bar{g}$ by the following procedure: $x \in \operatorname{dom} \bar{g}$ iff $\# g(V_x \cap \max P) = 1$. In this case there is a $y_0 \in Y$ with $g(V_x \cap \max P) = \{y_0\}$. Define $\bar{g}(x) := y_0$.

**6.6.** Every coloured poset $(P, g, Y)$ can be divided into disjoint parts,

$$P = \bigcup_{y \in Y} g^{-1}(y) \cup C\mathrm{dom}\, g,$$

where $C\mathrm{dom}\, g := P \setminus \mathrm{dom}\, g$ is the complement of $\mathrm{dom}\, g$ in $P$. Obviously all $g^{-1}(y)$ are ascending (open) subsets of $P$ and $C\mathrm{dom}\, g$ is a descending (closed) subset of $P$.

We define

$$\mathrm{Pure}(g, y) := g^{-1}(y),$$

$$\mathrm{Mix}(g) := C\mathrm{dom}\, g,$$

$$\mathrm{Pure}(g) := \bigcup_{y \in Y} \mathrm{Pure}(g, y) = \mathrm{dom}\, g.$$

**6.7.** EXAMPLE. Let $K = (S, Y, f)$ be a classification problem and let $S = (X, A, V, \varrho)$ be the underlying information system. As we have already seen in 6.3, $S$ defines a poset $\mathrm{Cond}(S)$. The maximal elements of $\mathrm{Cond}(S)$ are the facets of $\mathrm{Cond}(S)$ which in turn are the elements of $\bigtimes_{a \in A} \mathrm{Im}\, a$. Therefore we can define the function

$$g\colon \max \mathrm{Cond}(S) = \bigtimes_{a \in A} \mathrm{Im}\, a \xrightarrow{\varrho^{-1}} X \xrightarrow{f} Y,$$

i.e., a precolouring of $\mathrm{Cond}(S)$ which defines in turn by 6.5 a colouring $\bar{g}$ which we denote by $\mathring{f}$.

For every $y \in Y$ we define $\mathrm{Pure}(K, y) := \mathrm{Pure}(f, y) := \mathrm{Pure}(\mathring{f}, y)$ and $\mathrm{Pure}(K) := \mathrm{Pure}(f) := \mathrm{Pure}(\mathring{f})$ which are partially ordered sets and define therefore via $\Delta$ complexes which are subcomplexes of $\mathrm{Cond}'(S)$, the barycentric subdivision of $\mathrm{Cond}(S)$.

**6.8.** Let $S_i = (X_i, A_i, V_i, \varrho_i)$ $(i = 1, 2)$ be two information systems. A triple $\varkappa\colon X_1 \to X_2$, $\alpha\colon A_2 \to A_1$, $\nu\colon V_1 \to V_2$ is called a *homomorphism* $\sigma = (\varkappa, \alpha, \nu)\colon S_1 \to S_2$ if for all $x \in X_1$, $a \in A_2$ the following equality holds:

$$\varrho_2(\varkappa(x), a) = \nu\varrho_1(x, \alpha(a))$$

or if we consider attributes as functions:

$$a\varkappa(x) = \nu\alpha(a)(x).$$

$\mathrm{Mix}(K)$, which is defined by $\mathrm{Mix}(K) := \mathrm{Mix}(f) := \mathrm{Mix}(\mathring{f})$ is a simplicial complex, subcomplex of $\mathrm{Cond}(S)$.

The notions of Pure and Mix are motivated by the following: Let $c$ be a condition and $x$ an object of $X$. We say $x$ *satisfies* $c$ if for all $a \in \mathrm{dom}\, c$ holds $a(x) = c(x)$. Let $\mathrm{Sat}(c)$ be the set of all objects which satisfy $c$. Then $c \in \mathrm{Pure}(f)$ iff $\# f(\mathrm{Sat}(c)) = 1$, i.e., all $x$ satisfying $c$ are of "the same colour". Otherwise: $c \in \mathrm{Mix}(f)$ iff there are objects $x$ and $y$ satisfying $c$ with $f(x) \neq f(y)$.

Note that neither Pure nor Mix are in general functorial, in many important cases, however, they are:

**6.9.** PROPOSITION. *Let* $\lambda = (\varkappa, \alpha, \nu, \eta)$: $K_1 \to K_2$ *be a homomorphism of classification problems* $K_i = (S_i, Y_i, f_i)$, $S_i = (X_i, A_i, V_i, \varrho_i)$ *the underlying information systems. Suppose one of the following conditions to be satisfied:*

(i) $\alpha$ *is injective and* $\nu(\mathrm{Im}\,\alpha(a)) = \mathrm{Im}\,a$ *for all* $a \in A_2$.

(ii) $\varkappa$ *is surjective.*

(iii) $K_2 = K_1 \times K_1$ *and* $\lambda$ *is the diagonal map.*

*The mapping* $\lambda_*$ *which maps the* $S_1$*-condition* $c$ *onto* $\nu \subset \alpha$ *is a simplicial map of* $\mathrm{Mix}(K_1)$ *into* $\mathrm{Mix}(K_2)$. *Moreover,* $\lambda_*$ *is an order preserving map of* $\mathrm{Pure}(K_1)$ *into* $\mathrm{Pure}(K_2)$ *which in turn defines a simplicial map of the corresponding simplicial complexes* $\Delta(\mathrm{Pure}(K_i))$.

## 7. Topology of $\mathrm{Cond}(S)$, $\mathrm{Pure}(K)$, and $\mathrm{Mix}(K)$

**7.1.** Let $K$ be a simplicial complex. The geometric realization $|K|$ of $K$ is by definition the set of all functions $p$ defined over the set of vertices of $K$ with values in the interval $[0, 1] \subseteq \boldsymbol{R}$ satisfying the following conditions:

(a) $\mathrm{supp}\,p = \{v|\ p(v) \neq 0\} \in K$;

(b) $\sum_v p(v) = 1$.

**7.2.** PROPOSITION. *Assume* $S$ *to be completely fully faithful. Then for the* $i$*-th homology group of* $\mathrm{Cond}(S)$ *with coefficients in* $\boldsymbol{Z}$ *holds*

$$H_i(\mathrm{Cond}(S); \boldsymbol{Z}) \cong \begin{cases} \boldsymbol{Z} & \text{if } i = 0, \\ \boldsymbol{Z}^t & \text{if } i = N, \\ \{0\} & \text{otherwise,} \end{cases}$$

*where* $t = (m-1)^{N+1}$ *with* $m = \# V$, $N+1 = \# A$.

*If* $m = 2$ (case of boolean functions), *then* $|\mathrm{Cond}(S)|$ *is homeomorphic to the* $N$*-dimensional sphere.* (B. Graw [6] proved, moreover, that in general $\mathrm{Cond}(S)$ is shellable and $|\mathrm{Cond}(S)|$ is a bouquet of $t$ $N$-dimensional spheres.)

**7.3.** PROPOSITION. *Let* $K$ *be a classification problem with completely fully faithful underlying information system* $S$. *Then* $\mathrm{Mix}(K)$ *is a pure* $(N-1)$*-dimensional subcomplex of* $\mathrm{Cond}(S)$ *and* $|\Delta(\mathrm{Pure}(K))|$ *is homotopic to the complement of* $|\mathrm{Mix}(K)|$ *in* $|\mathrm{Cond}(S)|$.

**7.4.** PROPOSITION (Lefschetz Duality). *If* $S$ *is completely fully faithful and* $m = \# V = 2$, *then* $\mathrm{Mix}(K)$ *and* $\mathrm{Pure}(K)$ *are connected by the following isomorphism of the homology groups:*

$$H_i(\mathrm{Pure}(K); \boldsymbol{Z}) = H_{N-i}(\mathrm{Cond}(S), \mathrm{Mix}(K); \boldsymbol{Z}).$$

This theorem allows to compute the homology groups $H_i(\mathrm{Pure}(K); Z)$ knowing $H_i(\mathrm{Mix}(K); Z)$, and vice versa. To do this use the exact homology sequence

$$\ldots \to H_{i+1}(\mathrm{Cond}(S), \mathrm{Mix}(K); Z) \to H_i(\mathrm{Mix}(K); Z) \to H_i(\mathrm{Cond}(S); Z) \to H_i(\mathrm{Cond}(S), \mathrm{Mix}(K); Z) \to \ldots$$

and take into account Proposition 3.2.

Let

$$h_i(\mathrm{Pure}(K)) := \operatorname{rank} H_i(\mathrm{Pure}(K); Z), \quad h_i(\mathrm{Mix}(K)) := \operatorname{rank} H_i(\mathrm{Mix}(K); Z)$$

be the Betti numbers of $\mathrm{Pure}(K)$, $\mathrm{Mix}(K)$ respectively. Then Proposition 7.4 yields the following:

**7.5.** COROLLARY. *Assume* $N \geqslant 2$. *Under the assumptions of* 7.4 *the following equalities hold:*

$$\begin{aligned} h_0(\mathrm{Pure}(K)) &= h_{N-1}(\mathrm{Mix}(K)) + 1, \\ h_i(\mathrm{Pure}(K)) &= h_{N-1-i}(\mathrm{Mix}(K)) \quad \text{if} \quad N-1 > i > 0, \\ h_{N-1}(\mathrm{Pure}(K)) &= h_0(\mathrm{Mix}(K)) - 1, \\ h_i(\mathrm{Pure}(K)) &= 0 \quad \text{if} \quad i > N-1. \end{aligned}$$

**7.6.** Let $\mathrm{size}(C)$ be the size of a smallest classifying tree for the classification problem $C$. The following result gives evidence that classification problems which are difficult from the topological point of view are intractable from the computational standpoint.

**7.7.** THEOREM.

$$\mathrm{size}(C) \geqslant \frac{h_0(\mathrm{Pure}(C)) - 1}{m-1}.$$

**7.8.** COROLLARY. *If* $m = 2$, *then*

$$\mathrm{size}(C) \geqslant h_0(\mathrm{Pure}(C)) - 1 = h_{N-1}(\mathrm{Mix}(C)),$$

*i.e., classification problems with many "*$(N-1)$*-dimensional holes" are of high complexity.*

The following theorem is of great importance for the study of connections between different classification problems.

**7.9.** THEOREM. *Let* $\lambda\colon C_1 \to C_2$ *be a homomorphism of classification problems satisfying one of conditions* (i), (ii) *or* (iii) *of* 2.8. *Let* $D$ *be any sheaf* (cf. [5]) *of* $R$*-modules* ($R$ *an arbitrary ring*) *over* $\mathrm{Cond}(C_1)$. *There are two spectral sequences:*

$$I_2^{pq} = H^p(\mathrm{Pure}(C_2), R^q \lambda_* D) \Rightarrow I^n = H^n(\mathrm{Pure}(C_1), D)$$

*and*

$$II_2^{pq} = H^p(\mathrm{Mix}(C_2),\ R^q\lambda_* D) \Rightarrow II^n = H^n(\mathrm{Mix}(C_1),\ D).$$

*If $c$ is a condition of $C_1$, then the fiber of $R^q\lambda_* D$ in $c$ is given by*

$$(R^q\lambda_* D)_c = H^q(\lambda_*/c,\ D) = H^q(\lambda_*^{-1}(V_c),\ D),$$

*where $V_c = \{d \mid d$ a condition of $\mathrm{Pure}(C_2)$, $\mathrm{Mix}(C_2)$, respectively, and $c$ a subcondition of $d\}$.*

This theorem will be used in the next section to compute the Euler–Poincaré characteristic.

## 8. The Euler–Poincaré characteristic

**8.1.** Let $K$ be a finite simplicial complex. The alternating sum

$$\chi(K) = \sum_{i=0}^{\infty} (-1)^i \,\#\,\{s \in K \mid \dim s = i\} = \sum_{s \in K} (-1)^{\dim s},$$

i.e., the number of simplexes of even dimension minus the number of simplexes of odd dimension is a topological invariant because $\chi(K) = \sum_{i=0}^{\infty} (-1)^i \operatorname{rank} H_i(K; \mathbf{Z})$. This invariant is called the *Euler–Poincaré characteristic*. For a poset $P$ one defines $\chi(P) = \chi(\Delta(P))$. The Lefschetz duality (cf. 7.4) has the following straightforward consequence.

**8.2.** PROPOSITION. *Let $C$ be a classification problem and $S = (X, A, V, \varrho)$ the underlying information system which is assumed to be completely fully faithful, and let $\# V = 2$, $\# A - 1 = N$. Then*

$$\chi(\mathrm{Mix}(C)) - 1 = (-1)^{N+1}\big(\chi(\mathrm{Pure}(C)) - 1\big).$$

**8.3.** PROPOSITION. *The Euler–Poincaré characteristic of $GAP(N)$ satisfies the following properties:*

(i) $\chi\big(\mathrm{Pure}(\mathrm{GAP}(N),\ 0)\big) = h_0\big(\mathrm{Pure}(\mathrm{GAP}(N),\ 0)\big) = h_0\big(\mathrm{Pure}(\mathrm{GAP}(N))\big) - 1 = \Omega\big((N-2)!\,(n-1)^N\big)$;

(ii) $\mathrm{Mix}(\mathrm{GAP}(N))$ *is shellable* (cf. [6]) *and therefore* $\chi\big(\mathrm{Mix}(\mathrm{GAP}(N))\big) = 1 + \operatorname{rank} H_{2(N-2)}\big(\mathrm{Mix}(\mathrm{GAP}(N));\ \mathbf{Z}\big)$.

(For $N = 3$ one gets $\chi\big(\mathrm{Pure}(\mathrm{GAP}(3))\big) = 13$.)

The next theorem studies the relationship of the Euler–Poincaré characteristic along 'nice' homomorphisms.

**8.4.** THEOREM. *Let $\lambda\colon C_1 \to C_2$ be a homomorphism of classification problems satisfying one of conditions* (i), (ii) *or* (iii) *of* 6.9. *Let $c$ be an $S_2$-condition and define $\lambda_*/c$, $\lambda_* \backslash c$ in the following way:*

$$\lambda_*/c := \{d \in \mathrm{Mix}(C_1) |\ \lambda_*(d) \supseteq c\} \quad (\textit{defined if } c \in \mathrm{Mix}(C_2)),$$

$$\lambda_* \setminus c := \{d \in \mathrm{Pure}(C_1) |\ \lambda_*(d) \subseteq c\} \quad (\textit{defined if } c \in \mathrm{Pure}(C_2)).$$

*$\lambda_*/c$ and $\lambda_* \setminus c$ are called the fibers of $c$ along $\lambda$. Then*

$$\chi(\mathrm{Mix}(C_1)) = \sum_{c \in \mathrm{Mix}(C_2)} (-1)^{\dim c} \chi(\lambda_*/c),$$

$$\chi(\mathrm{Pure}(C_1)) = \sum_{c \in \mathrm{Pure}(C_2)} (-1)^{\mathrm{codim}\, c} \chi(\lambda_* \setminus c)(m-1)^{\mathrm{codim}\, c}.$$

This theorem allows to compute the Euler–Poincaré characteristic of Mix and Pure if one finds 'nice' homomorphisms onto easier classification problems, the fiber of which is also computable.

**8.5.** The next theorem studies the behaviour of classification problems if one puts one additional question: Let $C = (S, Y, f)$ be a classification problem, $y \in Y$, and let $a$ be an attribute of the underlying information system. If one extends $f$ by adding question $a$ in case of $y$ one gets the following classification $C \dotplus_y a = (S, Y \perp\!\!\!\perp \mathrm{Im}\, a, f \dotplus_y a)$, where range $(f \dotplus_y a) = Y \perp\!\!\!\perp \mathrm{Im}\, a$, and for an object $x$ one defines

$$(f \dotplus_y a)(x) := \begin{cases} f(x) & \text{if} \quad f(x) \neq y, \\ a(x) & \text{if} \quad f(x) = y. \end{cases}$$

Let $U$ be an ascending subset of $\mathrm{Cond}(S)$. Then we define $U \setminus a := \{c \in U |\ a \notin \mathrm{dom}\, c\}$. $U \setminus a$ is obviously a descending subset of $U$ (not of $\mathrm{Cond}(S)$ in general). One gets the following theorem:

**8.6.** THEOREM. *If $\chi(C) := \chi(\mathrm{Pure}(C))$, then*

$$\chi(C \dotplus_y a) = \chi(C) + \chi(\mathrm{Pure}(C, y) \setminus a).$$

**8.7.** It is possible to define a classification $f$ relative to a classification $g$ in such a way that

$$\chi(f) = \chi(g) + \chi(f|g)$$

if $f$ classifies finer than $g$. Moreover, one gets for $C_1 \times C_2$ the formula

$$\chi(C_1 \times C_2) = \chi(C_1) \cdot \chi(C_2).$$

These formulas resemble the well-known formulas of the classical Shannon entropy so that cum grano salis the Euler–Poincaré characteristic can be considered as kind of structural information.

## References

[1] L. Budach, *Two pebbles don't suffice*, In: *Mathematical Foundations of Computer Science* 1981; Proceedings, 10th Symposium Strebské Pleso, Czechoslovakia, 1981 (Eds.: J. Gruska, M. Chytil); Lecture Notes in Computer Science, 118; Berlin–Heidelberg–New York 1981; 578–589.

[2] –, *Klassifizierungsprobleme und das Verhältnis von deterministischer zu nichtdeterministischer Raumkomplexität*, Seminarbericht Nr. 68, Sektion Mathematik der Humboldt-Universität, 1985, 1–64.
[3] –, *Information und Rechnen*, In: *Zur Bedeutung der Information für Individuum und Gesellschaft*; Berichtsband der Wissenschaftlichen Konferenz zum Leibniz-Tag der Akademie der Wissenschaften der DDR, Berlin, 29–30.6.1983; 191–208.
[4] –, *A Lower Bound for the Number of Nodes in a Decision Tree*, EIK (to appear).
[5] R. Godement, *Topologie algébrique et théorie des faisceaux*, Hermann, Paris 1958.
[6] B. Graw, Personal communication, 1983.
[7] J. Kahn, M. Saks, *A topological approach to evasivenes*, Manuscript (1983), 1–37.
[8] W. Marek, Z. Pawlak, *Information storage and retrieval systems*, Mathematical Foundations. Theoret. Compt. Sci. 1 (1976), 331–354.
[9] C. F. Picard, *Theorie der Fragebogen*, Akademie-Verlag, Berlin 1973.
[10] P. Pudlàk, S. Žak, *Space complexity of computations*, Manuscript (1983), 1–30.
[11] W. Savitch, *Relations between nondeterministic and deterministic tape complexities*, Journal of Computer and System Science 4 (1970), 177–192.
[12] E. H. Spanier, *Algebraic Topology*, McGraw-Hill, 1966.
[13] R. P. Stanley, *Combinatorics and Commutative Algebra*, Birkhäuser, Boston Basel, Stuttgart 1983.

*Presented to the semester*
*Mathematical Problems in Computation Theory*
*September 16–December 14, 1985*