Centrum voor Wiskunde en Informatica

REPORT *RAPPORT*

One incursion in the non-supervised generation of concepts

D. Riaño

Computer Science/Department of Algorithmics and Architecture

Report CS-R9472 December 1994

P.O. Box 94079, 1090 GB Amsterdam (NL)
Kruislaan 413, 1098 SJ Amsterdam (NL)
Telephone +31 20 592 9333
Telefax +31 20 592 4199

One Incursion in the Non-Supervised Generation of Concepts

David Riaño
Universitat Rovira i Virgili, Carretera Salou s/n, 43006 Tarragona, Spain
E-mail: fibcls09@lsi.upc.es, drianyo@etse.urv.es

CWI
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands

Abstract

Domains described in the form of a set of conjunctive rules can hide internal concepts which, when used in the description, the final explanation of the domain becomes more natural and understable. Here, a methodology for automatically extract concepts from flat conjunctive rules is presented. The methodology is analyzed and refined to accomplish minimality and representativeness tasks, and an easy transformation is also supplied to supervise the process of automatic concept generation.

AMS Subject Classification (1991): 68T05, 68T30.
CR Subject Classification (1991): I.2.2, I.2.4, I.2.6.
Keywords & Phrases: grammar minimization, and-or knowledge, concept learning, knowledge compactation, prototyping.
Note: This work was completely developed in the Centrum voor Wiskunde en Informatica (CWI), afdeling Algoritmiek en Architectuur, Kruislaan 413, 1098 SJ Amsterdam, The Netherlands. E-mail: david@cwi.nl; during the three months' stay of the author in this research center and thanks to the 6th Erasmus Help (1994-95), No. E-4060-11.

1. Introduction

In the area of *Conceptual Learning* the examples and counterexamples in the input are used to modelize the underlying concept. The learned *concept* can be, then, represented in a wide variety of forms: *rules*, *frames*, *decision trees*, etc. The preference of one or other representation depends directly on several factors like the way in which the sample is given, the nature of the knowledge we want to represent and the use we want to do of it.

Even in the case we have come to a decision about the representation, some problems remain unsolved. One of the most important comes from the idea of *representativity*. The most of the time one same knowledge can be expressed in a wide variety of forms within the same model of representation. Furthermore, this bears some indirect facts like the problems of comparing and unifying knowledges, that must be carried out often with sophisticated *consensus* techniques.

In the area of knowledge representation by *rules* such problems arise with a particular interpretation. In the first hand we have to decide which rules are permited (*conjunctive*, *DNF*, *CNF*, etc.) and, in the other hand, which of the possible representations is kept to

Report CS-R9472
ISSN 0169-118X
CWI
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands

be the *prototype* of the concept. In this sense the *DNF* expression AB+AC+BC could be also represented following any of the logic expressions:

$$\mathrm{A}\ (\mathrm{B} + \mathrm{C}) + \mathrm{B}\ \mathrm{C}$$
$$(\mathrm{A} + \mathrm{C})\ \mathrm{B} + \mathrm{A}\ \mathrm{C}$$
$$\mathrm{A}\ \mathrm{B} + (\mathrm{A} + \mathrm{B})\ \mathrm{C}$$

One more thing to have into consideration concerns the features of the learning algorithm. Features like *time consumption* (it is not the same to produce *conjunctions* than *DNFs* than *CNFs*) [6] or *arbitrary considerations* (it is not the same to produce conjuntive rules where the only decision to make is to consider whether one descriptor is within the rule or not, and where the algorithms are well studied [2] [3] [5], than the fact of deducing CNF expressions where algorithms are not so evident [1] [4]).

In this work one method for the generation of *and-or expressions* as a consequence of a learning process is introduced. And-or expressions in the form,

```
<and-or expression> ::= (AND <and-or expression>+) |
                        (OR  <and-or expression>+) |
                        <descriptor>
```

generalizes the idea of *conjunctive*, *DNF*, and *CNF* expressions in the sense of that each of them is an *and-or expression.*

In the last lines it has been argued the advantages of the algorithms producing conjunctive expressions in front of algorithms producing other kind of expressions. Why then we keep interested in producing such other expressions? The answer has to be search in the idea of *redundancy.* The learning algorithm producing conjunctive expressions [5] has to be runned three times to learn the concept AB+AC+BC, and when this is learned it needs to make reference six times to the *set of descriptors*, where surprisingly only three descriptors are available (i.e. {A, B, C}). The use of more complex expressions permits lighten and reduce this redundancy. It is all a matter of saving.

The distribution of this work from here and forth is as follows. In section 2 a *methodology* for learning and-or expressions in the task of conceptual learning is introduced. Section 3 studies some considerations to conclude with a new algorithm to produce the *minimal* rule set.

Moreover, introducing a simple selecting criteria, the method drives to the *canonical representation* of the concept trained. This is what section 4 is about. Last section is introduced as a control measure of the well functioning of the algorithms. Usually minimal solutions and best solutions are not the same, and this is specially true in bizarre domains, where redundancy removal can cause undesirable loss of clearness.

2. Methodology

Learning *conjunctive* expressions is the simplest and more honest way of learning. Algorithms are simple and understable, two properties that are, at the same time, shared with the results obtained. Partially, this is the reason for having so much methods aiming at this particular solution [2], [3], [5].

Other desirable property of those methods concerns the replying time. Those methods are proved [4] to be faster than other methods producing *k-term-DNF*, *k-DNF*, or *k-CNF* expressions, but, in the other hand, rough repetitive expressions are obtained and no save of memory is pursued in the process of learning.

This unconcerned behaviour presents fast representation drawbacks that complementary (or alternative) methodologies try to mend.

Here we present a complementary methodology on the form of rules factorization.

The *factorization* of a set of rules is achieved after two clearly separated phases. In the first one (obtain the *factorization list*) the *interesting* factors are found. In the last phase (obtain the *factorized grammar*) the *minimal* and-or representation for the set of rules is produced considering only the interesting factors in the previous phase.

The explanation is done first introducing a clearifying example and lately given the formal description of the methodology and the algorithms.

2.1 Previous Remarks

Before going on some considerations about the notation used in the rest of this work are deserved. The first consideration concerns the way the set of rules is represented. The fact of having several conjunctive expressions, $\{ce_1, ce_2, ..., ce_n\}$, describing the same concept, C, is represented in the form $C \leftarrow ce_1 + ce_2 + ... + ce_n$, which despite the similarity with traditional grammars representation (the concepts of set of rules and grammar are used interchangeably along this work), there are clear differences between them.

These differences are reflexed in the two properties about grammars here that traditional grammars do not own: commutativity and absorption.

Conjunctive expressions, ce_i, are build up from basic symbols using the conjunctive operator, $\bullet$, sometimes also called combination operators, which allows *commutativity.* Unlike, the combination of symbols in traditional grammars is done with the concatenation operator, $\cdot$, which does not permit commutativity.

Absorption is a second important property that grammars here owe to the conjunctive operator. This property stablish that combining one symbol with a conjunctive expression that already contains that symbol does not modify the expression. Again, this property is not shared by the model of traditional grammars with the concatenation operator.

One last consideration is about the concept of *factor.* For one grammar G, we define *factor* as any of the conjunctive expressions contained in any of the rules of G. One factor is *interesting* when it appears in more than two conjunctive expressions for the same rule, e.g. rule R $\leftarrow$ ABC + AB describes six possible factors, {A, B, C, AB, AC, ABC}, but only three of them are interesting for factorization purposes, {A, B, AB}.

2.2 Introductory Example

In this example one concept R is tried to be expressed in terms of some descriptors represented by the letters in the alphabet. The learning process produces six conjunctive expressions for R, all of them in the set of rules

$$\{\text{ABCD} \rightarrow \text{R}, \text{ABCE} \rightarrow \text{R}, \text{ABCF} \rightarrow \text{R}, \text{AHILK} \rightarrow \text{R}, \text{AHIMK} \rightarrow \text{R}, \text{AJK} \rightarrow \text{R}\},$$

and represented for the sake of shortness in the form,

$$\text{R} \leftarrow \text{ABCD} + \text{ABCE} + \text{ABCF} + \text{AHILK} + \text{AHIMK} + \text{AJK}$$

Interesting factors are those appearing more than twice in the same production. One method for clearly find the interesting factors is examplified in the enclosed table.

The construction of this table is, for the example, as follows: keeping in mind along the whole process the rule set R ← ABCD + ABCE + ABCF + AHILK + AHIMK + AJK; the columns in the table are those letters in the alphabet which are present in more than two rules, e.g. {A, B, C, H, I, K}. Starting with the λ factor and combining it with all the elements heading the columns, the first row is obtained. For each element (or factor) in the row, the number of rules where this factor appears is saved, e.g. C3 in the third column means that the factor C occurs in three rules (R ← ABCD + ABCE + ABCF). All the new factors with more than two appearances (here the case of two is also considered) are added in the λ-column as future rows to consider. The rest, *clear* cells, are not considered.

•	A	B	C	H	I	K
λ	A6	B3	C3	H2	I2	K3
A6	A6	AB3	AC3	AH2	AI2	AK3
B3	AB3	B3	BC3	BH0	BI0	BK0
C3	AC3	BC3	C3	CH0	CI0	CK0
H2	AH2	BH0	CH0	H2	HI2	HK0
I2	AI2	BI0	CI0	HI2	I2	IK2
K3	AK3	BK0	CK0	HK0	IK2	K3
AB3	AB3	AB3	ABC3	ABH0	ABI0	ABK0
AC3	AC3	ABC3	AC3	ACH0	ACI0	ACK0
AH2	AH2	ABH0	ACH0	AH2	AHI2	AHK0
AI2	AI2	ABI0	ACI0	AHI2	AI2	AIK2
AK3	AK3	ABK0	ACK0	AHK0	AIK2	AK3
BC3	ABC3	BC3	BC3	BCH0	BCI0	BCK0
HI2	AHI2	BHI0	CHI0	HI2	HI2	HIK2
IK2	AIK2	BIK0	CIK0	HIK2	IK2	IK2
ABC3	ABC3	ABC3	ABC3	ABCH0	ABCI0	ABCK0
AHI2	AHI2	ABHI0	ACHI0	AHI2	AHI2	AHIK2
AIK2	AIK2	ABIK0	ACIK0	AHIK2	AHIK2	AHIK2
HIK2	AHIK2	BHIK0	CHIK0	HIK2	HIK2	HIK2
AHIK2	AHIK2	ABHIK0	ACHIK0	AHIK2	AHIK2	AHIK2

The *interesting* factors already considered are never reconsidered, this is what *soft* cells represent; e.g. IK2 for column I produces IK2 • I = IK2 (*absorption* property), which already exists as row.

When no more combinations are possible the interesting factors are those in the λ-column. In the example,

{A6, B3, C3, , H2, I2, K3, AB3, AC3, AH2, AI2, AK3, BC3, HI2, IK2, ABC3, AHI2, AIK2, HIK2, AHIK2}.

With this set of factors, the `value` function, and the initial rule set, the factorization process is reachable.

At each step the best factor is chosen and the set of rules factorized. The decision of which is the *best* factor is taken using the `value` function; in the example it is computed using the product of the number of rules containing the factor minus one, times the length of the factor. These values are all displayed as a superindex of the factors, e.g. AC^4_3 means that factor AC appears in three rules, R $\leftarrow$ ABCD + ABCE + ABCF, and the `value` function takes the value four (= (3-1)*2, where 2 is the lenght of AC).

Those values have to be reconsidered each time one factorization is done.

The next lines are to show the whole factorization process:

Before iteration (26 symbols):

$$\begin{aligned}
R &\leftarrow \mathrm{ABCD} + \mathrm{ABCE} + \mathrm{ABCF} + \mathrm{AHILK} + \mathrm{AHIMK} + \mathrm{AJK} \\
L_0 &= \{\ \mathrm{ABC}^6_3,\ \mathrm{A}^5_6,\ \mathrm{AB}^4_3,\ \mathrm{AC}^4_3,\ \mathrm{AHIK}^4_2,\ \mathrm{AK}^4_3,\ \mathrm{BC}^4_3,\ \mathrm{AHI}^4_3,\ \mathrm{AIK}^3_2,\ \mathrm{HIK}^3_2,\ \mathrm{AH}^2_2, \\
&\quad\ \mathrm{AI}^2_2,\ \mathrm{B}^2_3,\ \mathrm{C}^2_3,\ \mathrm{HI}^2_2,\ \mathrm{IK}^2_2,\ \mathrm{K}^2_3,\ \mathrm{H}^1_2,\ \mathrm{I}^1_2\}
\end{aligned}$$

1st iteration (22 symbols): ABC^6_3 is chosen.

$$\begin{aligned}
R &\leftarrow \mathrm{ABC}X_1 + \mathrm{AHILK} + \mathrm{AHIMK} + \mathrm{AJK} \\
X_1 &\leftarrow \mathrm{D} + \mathrm{E} + \mathrm{F} \\
L_1 &= \{\ \mathrm{AHIK}^4_2,\ \mathrm{AK}^4_3,\ \mathrm{A}^3_4,\ \mathrm{AHI}^3_2,\ \mathrm{AIK}^3_2,\ \mathrm{HIK}^3_2,\ \mathrm{AH}^2_2,\ \mathrm{AI}^2_2,\ \mathrm{HI}^2_2,\ \mathrm{IK}^2_2,\ \mathrm{K}^2_3,\ \mathrm{H}^1_2,\ \mathrm{I}^1_2\}
\end{aligned}$$

2nd iteration (20 symbols): AHIK^4_2 is chosen.

$$\begin{aligned}
R &\leftarrow \mathrm{ABC}X_1 + \mathrm{AHIK}X_2 + \mathrm{AJK} \\
X_1 &\leftarrow \mathrm{D} + \mathrm{E} + \mathrm{F} \\
X_2 &\leftarrow \mathrm{L} + \mathrm{M} \\
L_2 &= \{\mathrm{A}^2_3,\ \mathrm{AK}^2_2,\ \mathrm{K}^1_2\}
\end{aligned}$$

3rd iteration (20 symbols): A^2_3 is chosen.

$$\begin{aligned}
R &\leftarrow \mathrm{A}X_3 \\
X_1 &\leftarrow \mathrm{D} + \mathrm{E} + \mathrm{F} \\
X_2 &\leftarrow \mathrm{L} + \mathrm{M} \\
X_3 &\leftarrow \mathrm{BC}X_1 + \mathrm{HIK}X_2 + \mathrm{JK} \\
L_3 &= \emptyset
\end{aligned}$$

The factorization process stops when no more factors are available. The set of rules at this point is an and-or *reduction* of the initial concept, which in this example is

$$\boxed{\begin{aligned}
R &\leftarrow \mathrm{A}X_3 \\
X_1 &\leftarrow \mathrm{D} + \mathrm{E} + \mathrm{F} \\
X_2 &\leftarrow \mathrm{L} + \mathrm{M} \\
X_3 &\leftarrow \mathrm{BC}X_1 + \mathrm{HIK}X_2 + \mathrm{JK}
\end{aligned}}$$

2.3 The Algorithm

The transformation followed in the former example is formalized in two algorithms, one producing the list of *interesting* factors, and the second transforming the initial set of conjunctive rules into the *minimal* set of and-or rules, or in the worst, into a reduction of the initial set of rules.

Phase 1: Obtain Factorization list

<u>input</u>: $G=(N,T,P\subseteq N\times T^*)$ grammar
<u>code</u>: BASE ← All those terminal symbols appearing in more than one production of the grammar, G.
OPEN ← pairs *(terminal-symbol, No. appearances)* for all the elements in BASE.
CLOSED ← empty set.
repeat the following steps *until* OPEN is empty:

- Take one element out of OPEN, name it `item`.
- *Combine* `item` with all the elements in BASE.
- *First Removal*: do not consider those combinations not appearing more than one time in the grammar G.
- *Second Removal*: do not consider those combinations already in OPEN or CLOSED.
- *Add* the surviving combinations to CLOSED.

<u>output</u>: CLOSED set of *(factor, No. appearances)* pairs.

Phase 2: Obtain the Factorized Grammar

<u>input</u>: G initial *flat* grammar
F set of the pairs *(factors, No. appearances)* to consider.
`value`: $F \rightarrow \Re$, bijective function measuring the goodness of the factors.
<u>code</u>: FG ← G
L ← F
repeat the following steps *until* the list of factors L is empty:

- Take the "*best*" factor out of L according to the `value` function, name it `best`.
- *Factorize* grammar FG using the `best` factor.
- *First Removal*: remove from L all the factors that do not appear more than one time in the production of the grammar FG.
- *Second Removal*: remove from L all the factors that contain completely the `best` factor (e.g. AB contains both factors, A and B).

<u>output</u>: FG factorized grammar (equivalent to G).

Some `value` *functions*

1. *lexicographic order*: the `value` function has an internal ordered representation of all the possible descriptors, $\mathcal{D} = \{d_1 < d_2 < ... < d_n\}$. Then the set of factors $F \subseteq D^+$ define a `value` function such that decreases for factors appearing later on when ordered like in a dictionary.

2. *general order*: introduced to avoid the limitation of working only with finite sets of descriptors. If $\mathcal{D}$ is enumerable, then F is also enumerable, and the `value` function defines a general order for D^+.

3. *grammar-directed order*: under this name several `value` functions are defined. The idea here is to consider several features of the factors (*No. of appearances in the rule set*, *length of the factor*, *No. of rules needed to generate such factor*, etc.). All those features can be combined under several forms to define the `value` function. Consider the following interesting combinations:

   (a) Pure: `value`(factor) = *No. appearances*(factor), `value`(factor) = *length*(factor).

   (b) Combined: `value`(factor) = *No. appearances*(factor) + *length*(factor), `value`(factor) = (*No. appearances*(factor) - 1) * *length*(factor)

   and compare their behaviour for the same rule set in table 0.1 (the number of symbols for each representation is a measure of the degree of compactation for each `value` function).

4. *dependence-directed order*: in this case the `value` function does not only consider the benefit of the feasible factorizations for coming to a decision, but also the loss of factorizeness. The `value` function is defined in the form:
   `value`(factor) = *gain of factorizing*(factor) - *loss of factorizing*(factor)

   While the measure of *gain* is easily obtained with the expression
   ((*No. appearances*(factor) - 1) * *length*(factor) - 2),

   the *loss* is computed as the *gain* of the *best* dependent alternative factor g that the actual factor f thwarts to be applicable, this is

   $$\max_{\substack{g \in S \\ g \ \mathbf{dep} \ f}} \{((\mathit{No.\ appearances}(f) - 1) * \mathit{length}(f) - 2)\}$$

   where g **dep** f (*dependence*) means that factors f and g share some symbol. An example of *dependence-directed order* is supplied in table 0.2. Although this *order* avoids some problems in the above methods the *minimal* description is not ensured.

3. Minimality

The *complexity* of a set of rules is measured in this work as the number of symbols used. In this sense, for two sets of rules describing the same concept, we will define the *best* one as the less *complex*. This is, the one whose description contains less symbols; and from here the interest in the search of the minimal representation. Unfortunately, *minimality* is not ensured by any of the `value` functions supplied in the previous section, and it is hardly thinkable to find this function in a non-recursive form,

Lexicographic order (26 symbols)	*Inverse lexicographic order* (23 symbols)
$R \leftarrow \text{A}X_1$ $X_1 \leftarrow \text{B}X_2 + \text{H}X_4 + \text{JK}$ $X_2 \leftarrow \text{C}X_3$ $X_3 \leftarrow \text{D} + \text{E} + \text{F}$ $X_4 \leftarrow \text{I}X_5$ $X_5 \leftarrow \text{K}X_6$ $X_6 \leftarrow \text{L} + \text{M}$	$R \leftarrow \text{K}X_1 + \text{ABC}X_3$ $X_1 \leftarrow \text{HI}X_2 + \text{AJ}$ $X_2 \leftarrow \text{A}X_4$ $X_3 \leftarrow \text{D} + \text{E} + \text{F}$ $X_4 \leftarrow \text{L} + \text{M}$
No. appearances (23 symbols)	*Factor length* (21 symbols)
$R \leftarrow \text{A}X_1$ $X_1 \leftarrow \text{BC}X_2 + \text{K}X_3$ $X_2 \leftarrow \text{D} + \text{E} + \text{F}$ $X_3 \leftarrow \text{H}X_4 + \text{J}$ $X_4 \leftarrow \text{I}X_5$ $X_5 \leftarrow \text{L} + \text{M}$	$R \leftarrow \text{A}X_3$ $X_1 \leftarrow \text{L} + \text{M}$ $X_2 \leftarrow \text{D} + \text{E} + \text{F}$ $X_3 \leftarrow \text{K}X_4 + \text{BC}X_2$ $X_4 \leftarrow \text{HI}X_1 + \text{J}$
No. + length (21 symbols)	*(No. - 1)* length* (20 symbols)
$R \leftarrow \text{A}X_1$ $X_1 \leftarrow \text{BC}X_2 + \text{K}X_4$ $X_2 \leftarrow \text{D} + \text{E} + \text{F}$ $X_3 \leftarrow \text{L} + \text{M}$ $X_4 \leftarrow \text{HI}X_3 + \text{J}$	$R \leftarrow \text{A}X_3$ $X_1 \leftarrow \text{D} + \text{E} + \text{F}$ $X_2 \leftarrow \text{L} + \text{M}$ $X_3 \leftarrow \text{BC}X_1 + \text{HIK}X_2 + \text{JK}$

Table 0.1: Transformations of the set of rules for several *grammar-directed orders*

The initial set of rules

$$\mathrm{R} \leftarrow \mathrm{AB} + \mathrm{AC} + \mathrm{AD} + \mathrm{AE} + \mathrm{BF} + \mathrm{CG} + \mathrm{DH} + \mathrm{EI}$$

where $\mathrm{B} = \mathrm{B}_1 \bullet \mathrm{B}_2 \bullet \mathrm{B}_3 \bullet \mathrm{B}_4$, $\mathrm{C} = \mathrm{C}_1 \bullet \mathrm{C}_2 \bullet \mathrm{C}_3 \bullet \mathrm{C}_4$, $\mathrm{D} = \mathrm{D}_1 \bullet \mathrm{D}_2 \bullet \mathrm{D}_3$, and $\mathrm{E} = \mathrm{E}_1 \bullet \mathrm{E}_2 \bullet \mathrm{E}_3$.

Table of mutual dependences

	A	B	C	D	E
A	1	1	1	1	1
B	1	1	0	0	0
C	1	0	1	0	0
D	1	0	0	1	0
E	1	0	0	0	1

value(A) = 1 - max{2,2,1,1} = -1
value(B) = 2 - max{1} = 1
value(C) = 2 - max{1}= 1
value(D) = 1 - max{1} = 0
value(E) = 1 - max{1} = 0

The table of dependences contains *one* in row i, column j if factors f_i and f_j are mutually dependent, and *zero* otherwise. The computation of the **value** function for the factor f_i only considers those alternative factors f_j such that the table contains *one* in row i, column j. And from all of them only keeps the greatest (max).

The *grammar-directed order* obtains a worst result than the *dependence-directed order.*

factor: A_4^3 (36 symbols)

$$\mathrm{R} \leftarrow \mathrm{AW} + \mathrm{BF} + \mathrm{CG} + \mathrm{DH} + \mathrm{EI}$$
$$\mathrm{W} \leftarrow \mathrm{B} + \mathrm{C} + \mathrm{D} + \mathrm{E}$$

grammar-directed order

factors: $\mathrm{B}_2^1 \cdot \mathrm{C}_2^1 \cdot \mathrm{D}_2^0 \cdot \mathrm{E}_2^0 \cdot$ (31 symbols)

$$\mathrm{R} \leftarrow \mathrm{BW} + \mathrm{CX} + \mathrm{DY} + \mathrm{EZ}$$
$$\mathrm{W} \leftarrow \mathrm{A} + \mathrm{F}$$
$$\mathrm{X} \leftarrow \mathrm{A} + \mathrm{G}$$
$$\mathrm{Y} \leftarrow \mathrm{A} + \mathrm{H}$$
$$\mathrm{Z} \leftarrow \mathrm{A} + \mathrm{I}$$

dependence-directed order

Table 0.2: Transformations of the set of rules for *dependence-directed order*

$$\texttt{value}(f,S,F) \;=\; (No.\ appearances(f)-1)*length(f)-2+ \max_{g\in Filter(f,S,F)}\{\texttt{value}(g, Factorize(S;f), Filter(f,S,F))\}$$

with the basic case $\texttt{value}(f,S,\emptyset)=0$.

3.1 Some considerations about the functions Factorize and Filter

- The function *Factorize(S,X)* produces a grammar which is equivalent to S and where one of the productions has suffered the following transformation:

$$\textit{before}\text{: } \mathrm{R} \leftarrow \boxed{\boxed{\mathrm{X}}\ \mathrm{x}_1 + \ldots + \boxed{\mathrm{X}}\ \mathrm{x}_n} + \ldots$$

$$\begin{aligned} \textit{after}\text{: } \mathrm{R} &\leftarrow \overbrace{\boxed{\mathrm{X}}}^{l}\ \mathrm{X} + \ldots \\ \mathrm{X} &\leftarrow \mathrm{x}_1 + \ldots + \mathrm{x}_n \end{aligned}$$

  and where the loss of complexity is $\delta_{size} = size_{before} - size_{after} = (n-1)*l-2$ symbols.

- Every factorization is followed by a reconsideration of all the possible future *interesting* factors (function $Filter(f,S,F)$). This is, all the factors in F which are supersets of the `pivoting` factor f are removed, since they will not appear in the grammar as a whole ever more. And for the factors in F being subsets of the `pivot`, their goodness `value` is reduced $(|pivot|-1)*l$ unities. For example, the list of factors $\{\text{ABC}^6_3,\ \text{A}^5_6,\ \text{AB}^4_3,\ \text{ABCD}^4_2,\ \ldots\}$, factorizes with `pivot` ABC (the one with greatest `value` as the superindex shows), and modifies the above set to $\{\text{A}^3_4,\ \text{AB}^0_1,\ \ldots\}$. Finally the elements with `value` less than or equal to 2 are also removed.

$$\begin{aligned} \textit{before}\text{: } \mathrm{R} &\leftarrow \boxed{\boxed{\mathrm{X\ X'}}\ \mathrm{x}_1 + \ldots + \boxed{\mathrm{X\ X'}}\ \mathrm{x}_n} + \boxed{\mathrm{X'\ x_1'}} + \ldots + \boxed{\mathrm{X'\ x_p'}} + \ldots \\ \mathrm{Y} &\leftarrow \boxed{\mathrm{X'\ y_1}} + \ldots + \boxed{\mathrm{X'\ y_p}} + \ldots \end{aligned}$$

$$\begin{aligned} \textit{after}\text{: } \mathrm{R} &\leftarrow \boxed{\mathrm{X\ X'}}\ \mathrm{X} + \boxed{\mathrm{X'\ x_1'}} + \ldots + \boxed{\mathrm{X'\ x_p'}} + \ldots \\ \mathrm{Y} &\leftarrow \boxed{\mathrm{X'\ y_1}} + \ldots + \boxed{\mathrm{X'\ y_p}} + \ldots \\ \mathrm{X} &\leftarrow \mathrm{x}_1 + \ldots + \mathrm{x}_n \end{aligned}$$

  Elements factorized, and all their internal combinations, also deserve reconsideration, e.g. having the concept R $\leftarrow$ AB + AC + BD, and factorizing with A reduces the goodness of factorizing B.

3.2 A New Algorithm for Minimal Productions

The next algorithm is a transformation of the one in page 6 aiming at *minimality*. None of the `value` functions has been proved to produce the *minimal* set of rules in terms of number of symbols, and an alternative focusing is implemented in the form of a *backtracking algorithm* with pruning for ensuring minimality.

Phase 2 bis: Obtain the Minimal Factorized Grammar

input: G grammar
F set of the pairs *(factors, No. appearances)* to consider.
code: B $\leftarrow$ G
k $\leftarrow$ No. of symbols in G
for all factor f $\in$ F:
- *Compute* $Factorize(f, G)$ and $Filter(f, G, F)$. Name the respective results G_f and F_f, and apply this algorithm to factorize the grammar G_f with factors in F_f. Let factorized grammar be T.
- *If* grammar T is *better* than grammar B, swap them and compute k $\leftarrow$ No. of symbols in T.

output: B factorized grammar (equivalent to G).

Theorem 1 *The algorithm used to obtain the factorization list produces all the factors needed to minimize the base of rules.*

Proof: Minimal representations are only factorizations of the original set of rules. Let's prove then that the algorithm obtains all the possible factors. Applying induction on the length of the factors: all factors of length one are obtained since they are the result of combining λ with all the terminals which appear in more than two rules (first iteration of the algorithm). Factors of length $n+1$ are in the general form $f_{n+1} = \alpha \bullet f_n$, where f_n is of length n and consequently the algorithm produces it (induction), and α is a symbol that must appear in more than two of the original rules since f_{n+1} is a factor that contains this symbol. This is, f_n is combined with α at some point in the algorithm.□

Theorem 2 *The algorithm used to obtain the factorization list ends.*

Proof: Since the set of rules contains a finite number of finite rules, the set of symbols (subset fo the alphabet) used to define such rules is also finite. In this case, the number of possible combinations of symbols appearing in the λ-column is

$$1 \leq \textit{No. of interesting factors} \leq \sum_{i=1}^{n} \binom{n}{i}$$

Furthermore, the algorithm avoid reconsideration of factors already considered.□

4. Canonic Representation

One of the problems already mentioned in the introduction is the fact of not only obtain one minimal grammar but also obtain the same grammar independently of the order in which rules are given.

The problem comes from the fact that, for one same concept, several descriptions can exist, all of them fulfilling the request of minimality. This is, algorithms achieving the minimal description have to be supplied with an alternative mechanism to decide whether the solution is representative or not.

For algorithms applying the `value` functions in page 7, a second valuation criteria is sometimes needed. This second criteria is used when the `value` function for two different factorizations is the same (alternative representations) and only one of them must be chosen.

In the examples of this work, the second criteria function has been the one giving the least factor in lexicographic order.

5. The Supervised Version

Each one of the factorizations performed on the set of rules can be considered as the learning of a new concept within the domain studied. In this sense, when the concept R $\leftarrow$ ABC + ABD is factorized into $\{$R $\leftarrow$ ABX, X $\leftarrow$ C + D$\}$, the underlying idea is that concept R is defined in terms of the new concept X. Of course, if the factorizations are performed automaticaly, the appearing concepts are out of control, and '*no sense*' descriptions may arise (unsupervised method). In other cases, we can be not interested in all the concepts arising along the minimal set of rules construction, but only in a subset of them (supervised method).

Fortunately, transforming the algorithm for unsupervised concept generation into a supervised algorithm is trivial. The only modification to the algorithms in this work is to introduce a command that, before each new factorization is achieved, notifies the new concept and waits for confirmation (the new concept has been recognized) or refusal (the concept is senseless). In the first case, the *expert* confirms the new concept, and the factorization is performed. In the other case, the next factorization is considered.

6. Acknowledgements

This work is the result of a conversation with Peter Grünwald who stablished the basis for the initial versions of the algorithms here. I must show my gratitude to Jeroen van Maanen for having played the critical part of this work and for having spent some efforts proving some properties that finally are not considered or proved to be false. I am also grateful to Prof. Doct. Paul Vitányi, leader of the research group wherein this work was developed and the person who accepted me at the CWI. Finally, I thank Prof. Doct. Ulises Cortés, at the UPC, and Doct. Vicenç Torra, at the URV, all their efforts to arrange my bureaucratic problems.

References

1. Blum A., Khardon R., Kushilevitz E., Pitt L., Roth D., *On Learning Read-k-Satisfy-j DNF*, Proceedings on the 7th Annual ACM Conference on Computational Learning Theory (COLT'94), New Jersey (1994).
2. Cendrowska J., *PRISM: An Algorithm for Inducing Modular Rules*, Int. Journal on Man-Machine Studies 27 (1987).
3. Quinlan J. R., *Simplifying Decision Trees*, Int. Journal on Man-Machine Studies 27 (1987).
4. Riaño D., *Automatic Knowledge Generation from Data in Classification Domains*, Master Thesis, Facultat d'Informàtica de Barcelona (UPC), Barcelona (1994).
5. Riaño D., Research Report CWI, Amsterdam (1994).
6. Valiant L. G., *A Theory of the Learnable*, Communications of the ACM 27 (1984).