T U M

I N S T I T U T   F Ü R   I N F O R M A T I K

tabl – A Tool for efficient Compilation of probabilistic Constraints

Volker G. Fischer / Manfred Schramm

TUM-I9636
November 96

T E C H N I S C H E   U N I V E R S I T Ä T   M Ü N C H E N

TUM-INFO-11-96-I9636-200/1.-FI

Druck: Fakultät für Mathematik und
Institut für Informatik der
Technischen Universität München

tabl – A Tool for efficient Compilation of probabilistic Constraints

Volker G. Fischer / Manfred Schramm

Institut für Informatik
Technische Universität München, Germany

Keywords

Reasoning, Reasoning under Incomplete Knowledge, Probabilistic Reasoning, MaxEnt, Binary Decision Diagrams

Abstract

We present a tool for preparing reasoning under incomplete probabilistic knowledge. This tool is used within our system PIT (Probability Induction Tool) but can as well be used for other purposes, whenever an efficient representation of probabilistic knowledge is useful.

Contents

Chapter 1

Introduction

1.1 Background

Reasoning under incomplete knowledge is one of the most expanding research topics in the last few years. In general, there are three major problems that have to be solved:

- First, we have to find an appropriate language to represent and handle our knowledge. This representation should allow us to express all we need and should be easy to read or change.
- Second, we have to use a well-founded theory of decisions under incomplete knowledge which avoids to introduce any personal bias when completing the knowledge.
- Third, we have to derive the answers to our questions from the completed knowledge and make it readable for the user.

We are convinced that probability theory, supported by well-known additional principles like indifference, independence and the method of Maximum Entropy will defend its strong position within those tasks. Its justification (and many examples for modelling common sense reasoning) is described in [Greiner & Schramm, 1994]. We therefore present a tool for preparing reasoning under incomplete probabilistic knowledge.

We call this tool '*tabl*' which stands for 'tabular compiler'. It is used within our system PIT[1] (Probability Induction Tool) but can as well be used for other purposes, whenever an efficient representation of probabilistic knowledge is useful.

Basically we transform a knowledge-database that contains probabilistic constraints into a (underdeterminated) system of linear equations with additional constraints for the solution. Any solution of this system fulfils the demands of the original database. The task of *tabl* is to produce a minimal, easy to solve system of equations by using indifference principles.

In the following subsection we shortly recall the necessary technical concepts.

[1]see [Ertel et al., 1996, Schramm & Schulz, 1996].

1.2 Technical Terms

In this section we will give a short introduction to the concepts of probability models on propositional interpretations. For a more formal description we refer to [Greiner & Schramm, 1994, Greiner & Tinhofer, 1996] or any introduction to reasoning with probabilities.

Let $R = \{a_1 \dots a_r\}$ be a finite set of propositional variables (**atoms, attributes**) and let $Form(R)$ be the set of propositional formulas in these variables (using e.g. the operators $\{\neg, \wedge, \vee, \rightarrow\}$ for *not*, *and*, *or* and the *material implication*). A literal is either an atom or a negated atom. An **interpretation** of the variables is a mapping $I : R \rightarrow \{\mathit{true},\ \mathit{false}\}$, so there are 2^r interpretations. An interpretation can be extended to a function $I : Form(R) \rightarrow \{\mathit{true},\ \mathit{false}\}$ by the usual rules for the application of the operators. We also call an interpretation a **possible world** and denote it by the unique full conjunction[2] of literals that evaluates this interpretation to *true*. A propositional specification S is a set of formulas. An interpretation that evaluates all formulas from S to true is called a **propositional model** of S.

Now let $\Omega = \{\omega_1, \dots, \omega_m\}$ be a finite set of disjoint **elementary events** (i.e. only one of the ω_i can occur at any one time or situation). A **probability measure** on Ω is a function $P : 2^\Omega \rightarrow [0, 1]$ with

$$\sum_{\omega \in \Omega} P(\omega) = 1 \quad \text{and} \quad P(A) := \sum_{\omega \in A} P(\omega) \text{ for } A \in 2^\Omega .$$

A **linear**[3] **probabilistic constraint** (or simply **constraint** from here on) is a statement of the form $P(A) \in J$ (**unconditional constraint**) or $P(A|B) \in J$ (**conditional constraint**), where $A, B \subseteq \Omega$ and J is a subinterval of $[0, 1]$[‡]. A **probability model (P-Model)** for a set of constraints DB is a probability measure which fulfils the constraints from DB.

We now choose the interpretations for a set of propositional variables as the elementary events of a probability measure. We can then identify a propositional formula with the set of its (propositional) models. Thus, probabilistic constraints can be written as (conditional) probabilities on propositional formulas. Probabilistic constraints with the probability 0 or 1 still act like purely propositional statements. We call them **logical constraints**.

If we have a unique P-Model for a set of constraints, we can easily compute the probability of a proposition by summing the probabilities of its models. We can also calculate a conditional probability $P(a|b)$ by comparing the total probability of the models of $a \wedge b$ to the probability of models of b ($a, b \in Form(R)$). Unfortunately, in most cases the constraints of a specification do not, without further principles, determine a unique P-Model, but rather a (infinite) set of P-Models. In the PIT-environment we select a single, representative model from this set by applying the principle of **maximum entropy** (**MaxEnt**).

[2]A **full conjunction** is a conjunction of literals where each atom from R appears either in atomic form or in negated form.

[3]see Section 2.1.

[‡]We write $P(\dots) = p$ for the special case of $J = [p, p]$.

1.3 Notation

The theory and notation used further on is based on the two already published reports [Greiner & Schramm, 1994] and [Greiner & Schramm, 1995]. All important theorems and proofs can be found there. To simplify the reading of this report we repeat the necessary theory where it is needed without giving a detailed description or proof.

To distinguish between normal and special text we use different fonts:

- The input source-text and the output of the compiler is written in `typewriter font`. Following symbols are declared ($a, b \in Form(R)$):

'`-`'	:	`p(-a)` meaning $p(\neg a)$.
'`*`'	:	`p(a*b)` meaning $p(a \wedge b)$.
'`+`'	:	`p(a+b)` meaning $p(a \vee b)$.
'`->`'	:	`p(a->b)` meaning $p(\neg a \vee b)$, respectively $p(a \rightarrow b)$.
'`|`'	:	`p(a|b)` meaning $p(a \mid b)$.
'`-|>`'	:	`p(a -|> b)` meaning $p(b \mid a)$.
'`#`'	:	lines beginning with `#` are ignored (comments).
'`p(e),p(f)`'	:	the default events `e` and `f` are used in all error messages.

- Text with a special meaning like commands or semantic information is written in *emphasised font*.

Within the next sections the following variables are used:

a, b, c, d	*formulas*, whereas *formulas* are all expressions 'e' or '$e \mid d$' or 'e `-|>` d' with $e, d \in Form(R)$.
x, y	interval borders.
r	number of attributes (respectively variables) in the database.
l	number of constraints in the database.
n	number of unknown variables of the system of equations ($n \leq 2^r$, i.e. size of vector p).
δ_i	$\delta_i \in [x, y]$. Variable for the interval-constraint of row i.
A	unmodified matrix.
$\hat{*}$	variable $*$ after applying the 'weak' indifference principle.
$\tilde{*}$	variable $*$ after applying the 'strong' indifference principle.

1.4 A first Example

In order to illustrate the general proceeding we start with a small example.

1.4.1 Database

Default-Knowledge:

- normally <u>an</u>imals do not <u>fl</u>y.
- <u>bi</u>rds are animals.
- normally birds fly.

Desired conclusion:

Animals, which are no birds, normally do not fly.

1.4.2 Probabilistic Modelling

Regarding the events (over the reference space of all beings)

- `an` = 'beings that are animals'
- `bi` = 'beings that are birds'
- `fl` = 'beings that can fly'

we can model this information in a probabilistic way in a format that is accepted by *tabl* as follows:

```
%verbose # ADD COMMENTS IN THE OUTPUT FILE
%noindif # STRONG INDIFFERENCE OFF (EXPLANATION LATER)

# EVENTS: BEINGS THAT ARE ANIMALS OR BIRDS OR CAN FLY
   var an,bi,fl;

# CONSTRAINTS:
   # normally animals do not fly
   P( an -|> -fl ) = (0.5,1.0];
   # birds are animals
   P( bi -|> an )  = 1.0;
   # normally birds fly
   P( bi -|> fl )  = (0.5,1.0];

# QUERY: DO MOST ANIMALS, WHICH ARE NOT BIRDS, FLY ?
   Q( an * -bi -|> fl )=(0.5,1];
.
```

We say that in a given P-Model a query is **true**, if its probability lies within the given interval (in this case between 0.5 and 1.0).

With the definitions given in Section 1.2 we are able to perform the transformation of the first example.

1.4.3 Transformation

In the first example holds $R := \{\texttt{an}, \texttt{bi}, \texttt{fl}\}$ and therefore Ω consists of eight elements:

$$\Omega := \{(\texttt{an}, \texttt{bi}, \texttt{fl}) = \omega_1, (\texttt{an}, \texttt{bi}, \neg\texttt{fl}) = \omega_2, \dots, (\neg\texttt{an}, \neg\texttt{bi}, \neg\texttt{fl}) = \omega_8\}$$

Please note, that this enumeration defines also an implicit ordering of the full conjunctions.
The first database entry (**db-constraint**) 'P(an -|> -fl)=(0.5,1.0]' (meaning '$P(\neg\texttt{fl} \,|\, \texttt{an}) = \delta_1$' and '$\delta_1 \in (0.5, 1.0]$') can be seen as a linear equality depending on the elementary probabilities $P(\omega_1), \dots, P(\omega_8)$:

$$P(\neg\texttt{fl} \,|\, \texttt{an}) = \delta_1 \qquad \Leftrightarrow \qquad P(\neg\texttt{fl} \cap \texttt{an}) = \delta_1 P(\texttt{fl}) \qquad \Leftrightarrow$$

$$\Leftrightarrow \quad (1 - \delta_1) \sum_{\omega_i \in \texttt{an} \cap \neg\texttt{fl}} P(\omega_i) \quad - \quad \delta_1 \sum_{\omega_j \in \texttt{an} \cap \texttt{fl}} P(\omega_j) = 0 \tag{1.1}$$

together with the **I**(nterval)**-condition** $\delta_1 \in (0.5, 0.1]$.

With this transformation and the commitment '$P(a) = P(a \,|\, \Omega)$' for all $a \in Form(R)$ the database (consisting of $l = 3$ db-constraints) can be written as a linear (generally underdeterminated) system of equations:

$$A \cdot p = z \quad , \quad A \in \mathbb{R}^{2^r \cdot l} \quad \text{and} \quad z = (0, \dots, 0), \quad p = (P(\omega_1), \dots, P(\omega_8)) \tag{1.2}$$

Adding the normalisation constraint '$\sum_{i=1}^{2^r} P(\omega_i) = 1.0$' any solution of (1.2) that fulfils the I-conditions is a possible P-Model.

1.4.4 Output

tabl basically performs the transformation of the database. So the expected output of the first example would be:

ω_i: (This section does not appear in the output of *tabl* !)									
ω_1:	ω_2:	ω_3:	ω_4:	ω_5:	ω_6:	ω_7:	ω_8:		
an	an	an	an	¬an	¬an	¬an	¬an		
bi	bi	¬bi	¬bi	bi	bi	¬bi	¬bi		
fl	¬fl	fl	¬fl	fl	¬fl	fl	¬fl		
$P(\omega_1)$	$P(\omega_2)$	$P(\omega_3)$	$P(\omega_4)$	$P(\omega_5)$	$P(\omega_6)$	$P(\omega_7)$	$P(\omega_8)$	=	I-condition
A:								z:	
1	1	1	1	1	1	1	1	1	
$-\delta_1$	$1-\delta_1$	$-\delta_1$	$1-\delta_1$	0	0	0	0	0	$\delta_1 \in (0.5, 1]$
$1-\delta_2$	$1-\delta_2$	0	0	$-\delta_2$	$-\delta_2$	0	0	0	$\delta_2 = 1.0$
$1-\delta_3$	$-\delta_3$	0	0	$1-\delta_3$	$-\delta_3$	0	0	0	$\delta_3 \in (0.5, 1]$
Query:									
0	0	$1-\delta_q$	$-\delta_q$	0	0	0	0	0	$\delta_q \in (0.5, 1]$

The first line of A represents the normalisation constraint, the following three lines are the linear equations of the db-constraints. The query itself is also represented as a linear equation.

We want to emphasise that the result of this representation includes all information of the original database. Any solution (P-Model) of the underdeterminated system of equations (1.2) fulfilling the I-conditions implies an (correct) answer to the query. The query is true in a chosen P-Model if its probability lies within the given I-constraint of δ_q. (This still leaves the problem to select one of the possible P-Models).

The possibilities of the unmodified transformation are limited. A major problem of this representation is caused by the exponential growth of the Matrix A to the number of attributes $(\mathbf{O}(2^r) * \mathbf{O}(l))^{\ddagger}$. Therefore the calculation of databases consisting of 25 or more attributes exceed the capabilities of the current (and possibly future) computers by far.

An improvement of this situation is to distinguish only between these sets of possible **worlds**[4] that are structural different. This idea is strongly motivated by the widely known **principle of indifference**, that we quote in an informal description from [JAYNES, 1978]: *"as far as the available evidence gives us no reason to consider the proposition a either more or less likely than b, then the honest way to describe that state of knowledge is: $P(a) = P(b)$".*

If we demand this principle in our representation, we can see immediately that ω_7 and ω_8 fulfil a sufficient condition for indifference, because their corresponding columns in A are equal (further on the equality of columns within A will be denoted as **'weak' indifference**)

We can reduce the system of equations (1.2) by one column by setting the indifferent probabilities $P(\omega_7)$ and $P(\omega_8)$ equal. In our representation the equality can be expressed by adding the column of $P(\omega_8)$ to the column of $P(\omega_7)$ and deleting the column of $P(\omega_8)$. This **merging** is also possible for three or more equal columns.

The most simple way to express the summation of (equal) columns within A (indifferent from the query-lines, that always have to be added) is to note the multiplication factor in a separate vector called **multiplicator-line**[5].

A further diminishing of A is possible by analysing **logical constraints**. Regarding the third row of A as its linear equation

$$-P(\omega_5) - P(\omega_6) = 0 \tag{1.3}$$

leads to the conclusion that '$P(\omega_5) = P(\omega_6) = 0$'. In other words, the 5th and 6th column of the system of equations can be deleted and the resulting row of the logical constraint (consisting only of zero entries) holds no further information and can also be removed from A.

[‡] **O** denotes the Landau-symbol.

[4] see Section 1.2.

[5] Another reason for the multiplicator-line lies in the use of BDDs (see Section 3.1).

The unmodified transformation combined with the two possible simplifications (merging columns by indifference and deleting rows and columns by logical constraints) generates the final matrix $\hat{A}$[6] together with the corresponding query-line:

$P(\omega_7)$ $P(\omega_8)$	$P(\omega_3)$	$P(\omega_1)$	$P(\omega_4)$	$P(\omega_2)$	=	I-condition
multiplicator-line:						
2	1	1	1	1		
$\hat{A}$:					$\hat{z}$:	
1	1	1	1	1	1	
0	$-\delta_1$	$-\delta_1$	$1-\delta_1$	$1-\delta_1$	0	$\delta_1 \in (0.5, 1]$
0	0	$1-\delta_3$	0	$-\delta_3$	0	$\delta_3 \in (0.5, 1]$
Query:						
0	$1-\delta_q$	0	δ_q	0	0	$\delta_q \in (0.5, 1]$

Please note, that the order of columns does not influence the result, as long as the each matrix column stays consistent with the corresponding column of the queries.

Now we regard the real output of *tabl*:

```
# CREATED 10.12 AT 14:35 (MET)

# NUMBER OF: COLUMNS ROWS KOMPLEX GRID SQP-POINT SQP-INT QUERIES
 5  3  0  0  1  2  1

# EPS: 0.00000000

# ORDER:DB-CONSTRAINT ':' SUM,RANK,LOWER-BOUND,UPPER-BOUND,TYPE{NUMBER}
# MATRIX-ELEMENTS : (n,m) = n*1 + m*delta_i

# L_CONSTRAINTS FROM INPUT-LINE-NUMBERS :  11

# MULTIPLICATOR-LINE:
2  1  1  1  1

# NORMALISATION CONSTRAINT:
(1,0) (1,0)  (1,0)  (1,0)  (1,0)   :   1   0 -1.00000000 -1.00000000 S
# RESULT FROM INPUT-LINE-NUMBER 9 :
(0,0) (0,-1) (0,-1) (1,-1) (1,-1)  :   0   1  0.50000000  1.00000000 S
# RESULT FROM INPUT-LINE-NUMBER 13 :
(0,0) (0,0)  (1,-1) (0,0)  (0,-1)  :   0   2  0.50000000  1.00000000 S

# QUERY FROM INPUT-LINE-NUMBER 16 :
(0,0) (1,-1)  (0,0) (0,-1) (0,0)   :   0  -1  0.00000000  1.00000000 Q
```

[6] respectively the final system of equations $\hat{A} \cdot \hat{p} = \hat{z}$.

Ignoring the first few lines that contain only informations about sizes (e.g. the matrix size for later applications) and comments the result of the transformation starts with the multiplicator-line after the comment '`# MULTIPLICATOR-LINE:`'. The next lines represent the actual matrix $\hat{A}$. Each line is constructed the following way (here for the second row):

$$\underbrace{\texttt{(0,0) (0,-1) (0,-1) (1,-1) (1,-1)}}_{\text{matrix row}} : \underbrace{\texttt{0}}_{\hat{z}_2} \underbrace{\texttt{1}}_{\text{rank}} \underbrace{\texttt{0.50000000 1.00000000}}_{\delta_2 \in [0.5,1]} \underbrace{\texttt{S}}_{\text{type}} \qquad (1.4)$$

tabl encodes the expression '$\mathbf{n} \cdot 1 + \mathbf{m} \cdot \delta_i$' as '`(n,m)`'. $\hat{z}_2$ denotes the second element of the vector $\hat{z}$. Ignoring the entries 'rank' and 'type' the output equals the expected transformation.

1.5 Results

The unmodified 8×4 Matrix has been reduced to a 5×3 Matrix. This effect increases for larger databases, especially if another, 'strong' indifference demand is used.

The idea behind *tabl* is to perform the transformation very efficiently. In the first example we calculated the matrix $\hat{A}$ by subsequently reducing A as far as possible. Further on we make an entirely different approach. We create $\hat{A}$ directly by 'unfolding' only this columns of A that are not 'weak' indifferent. This is efficiently possible by using **B**inary **D**ecision **D**iagrams (BDDs) for the internal representation of different worlds. Therefore the growth of A is influenced only by the content of information in the database. In a database consisting of propositional logic constraints *tabl* acts as a theorem prover.

The second task of *tabl* consists of checking the input for (eventually hidden) contradictory constraints in the database. Additionally *tabl* produces a diversity of useful warnings that help to understand the results of the database-transformation.

As a final result it can be said, that the complexity (size and time) of the output produced by *tabl* depends only on the 'complexity' of the input and not on the number of attributes or number of constraints used. Our tests for real problems show that up to 25 attributes and 100 (non-logical) constraints (logical constraints are 'good-natured' in most of the cases) are solvable within reasonable time.

The output of *tabl* contains all important information of the original database and all information that is needed for further processing.

Our processing of $\hat{A}$ within PIT selects the (unique) P-Model P^* that has maximum entropy. P^* can also be seen as the unique solution that adds only minimal additional information to the database. The output of *tabl* allows to find P^* even in huge examples (80 to 100 constraints) within a few seconds.

tabl is designed as a tool that provides the information contained in a (probabilistic) database in a very efficient way by using the principle of indifference. Other applications that need a way to process a probabilistic database can use this output as a database-representation as well.

Chapter 2
The Input

2.1 General Structure

The *tabl* input-format is similar to a procedural programming language (like Pascal). The precise EBNF input-format description for syntactical correct databases can be found in Appendix A.1.

A *tabl* input-file consists of 5 separate successive parts:

1. options

   Right now the following options are available:

   %verbose : given this option the output will contain comments for each output-line.

   %noindif : this option switches off the strong indifference check.

   %eps=*precision* : declares the precision of open intervals[1][2].

   %*filename* : *filename* specifies the output-filename[3].

2. variable-declaration

   All used variables (complies the 'attributes' of Section 1.4.2) of the database have to be declared here. Up to 69 different variables can be used.

   Example:

   ```
   var smoker,miner,die_early;
   ```

   declares the variables: 'smoker', 'miner' and 'die_early'.

3. database

   The database-part contains the actual probabilistic information (i.e. constraints). The general way to express a constraint is:

   `P(`*formula*`) = `*interval* `;`

   Any well formed *formula*[4] containing variables, '`-`', '`*`', '`+`', '`->`, '`( )`','`|`' and '`-|>`' (respecting the natural operator precedence) can be used. The symbols '`|`' and '`-|>`'

[1] see Section 2.3.
[2] default value: 0.0 .
[3] default: *stdout*.
[4] see Section 1.3.

can only be used once per *formula*.
For *interval* any rational number or any (open, half-closed or closed) sub-interval (in this case '=' denotes '∈') between 0.0 and 1.0 is possible.

Examples:

```
P( smoker ) = 0.7 ;
P( die_early | smoker ) = (0.7,1.0] ;
P( die_early | -smoker * -miner) = (0.1,0.3) ;
```

4. 'conclusion' and 'queries'

   The (only) 'conclusion' and the 'queries' are formalised the same way as probabilistic constraints[5]. The conclusion is noted as: '`C(`*`formula`*`) = `*`interval`*` ;`'[6] and the queries are noted as '`Q(`*`formula`*`) = `*`interval`*` ;`'[7].

   For example we may ask, if most of the people that die early are smokers:

   ```
   Q( smoker | die_early ) = (0.5,1.0] ;
   ```

5. end of database

   The token '`.`' indicates the end of the database.

Now we can regard the complete input-file '*example.tabl*'[8]:

```
# NON MONOTONIC REASONING ON THE HEALTH OF SMOKERS AND MINERS
# THE TRANSFORMATION FROM LANGUAGE TO INTERVALS IS JUST FOR DEMONSTRATION.


%verbose  # COMMENTS ON
%noindif  # STRONG INDIFFERENCE OFF

var smoker,miner,die_early;

# 70% OF THE PEOPLE ARE SMOKERS
P( smoker ) = 0.7 ;
# SMOKERS (GENERALLY) DIE EARLY
P( die_early | smoker ) = (0.7,1.0] ;
# HEALTHY PEOPLE (GENERALLY) DO NOT DIE NOT EARLY
P( die_early | -smoker * -miner) = (0.1,0.3) ;

# ARE MOST PEOPLE THAT DIE EARLY SMOKERS ?
Q( smoker | die_early ) = (0.5,1.0] ;
# ARE MOST PEOPLE THAT DIE EARLY MINERS ?
Q( miner | die_early ) = (0.5,1.0] ;
.
```

[5] The differences between conclusion and query are described in Section 2.5.
[6] '`C(`*`formula`*`);`' if no interval is needed.
[7] '`Q(`*`formula`*`);`' if no interval is needed.
[8] The leading line-numbers of all following examples are just for documentation-purposes.

With this general introduction it should be possible to formalise most of the needed information. Additionally *tabl* offers some more sophisticated constructions that are described the following sections.

2.2 Rank Concept

In some situations it may be necessary to determine that two different $formulas$ (a and b) have exactly the same probability '$P(a) = \delta_a$, $P(b) = \delta_b$, $\delta_a = \delta_b$'. We can express this information in *tabl* as follows:

```
# P(a) AND P(b) ARE EQUAL AND BETWEEN (0.5,1]
P(a)= P(b)= (0.5,1];
```

tabl assigns every matrix-row a special (primary different) number called **rank**[9]. The fact that two probabilities are equal is expressed in the output by equal rank-numbers of the corresponding matrix rows.
The rank concept leads to a few additional consequences for the consistency of databases that are described in Section 4.1.2. Using the rank concept we recommend to switch off the strong indifference algorithm[10], because we have it not yet adapted to the rank-concept.

2.3 Interval Borders

In the database-input any combination of open or closed (probability) intervals is allowed. An open interval (x, y) is treated as a closed interval $[x+\epsilon, y-\epsilon]$ (analogous for half-open intervals). By default ϵ is 0.0 (so there is no difference between open and closed intervals) but in some applications it may be necessary not to reach the open interval-border. Therefore the option '`%eps=`*number*' changes open interval-borders by adding (subtracting) the amount *number* from the given border-value.

Example: The input

```
%eps=0.001
var a;
p(a)=(0.3,0.7);
.
```

produces the following result:

```
2  2  0  0  1  1  0
1  1

(1,0)  (1,0)   :   1   0 -1.00000000 -1.00000000 S
(0,-1) (1,-1)  :   0   1  0.30100000  0.69900000 S
```

[9] see (1.4).
[10] using the '`%noindif`'-option.

2.4 Compilation

The general installation guidance of *tabl* is provided in Appendix C. To compile the input-file of Section 2.1, '*example.tabl*', (which does not include an output-filename-option) into a file '*example.dat*' we would type:

> \> *tabl example.tabl example.dat*

```
WELCOME TO THIS VERSION V2.17 04.12.96
INTERNAL31
WORKING...


TASK COMPLETED
C U
```

tabl creates a file *example.dat* with this content:

```
# CREATED 10.12 AT 14:40 (MET)

# NUMBER OF: COLUMNS ROWS KOMPLEX GRID SQP-POINT SQP-INT QUERIES
 5  4  0  0  2  2  2

# EPS: 0.00000000

# ORDER:DB-CONSTRAINT':'SUM,RANK,LOWER-BOUND,UPPER-BOUND,TYPE{NUMBER}
# MATRIX-ELEMENTS : (n,m) = n*1 + m*delta_i


# MULTIPLICATOR LINE:
 2  1  1  2  2

# NORMALISATION CONSTRAINT:
(1,0)  (1,0)  (1,0)  (1,0)  (1,0)  :  1  0 -1.00000000 -1.00000000 S
# RESULT FROM INPUT-LINE-NUMBER 11 :
(0,-1) (0,-1) (0,-1) (1,-1) (1,-1) :  0  1  0.70000000  0.70000000 S
# RESULT FROM INPUT-LINE-NUMBER 13 :
(0,0)  (0,0)  (0,0)  (0,-1) (1,-1) :  0  2  0.70000000  1.00000000 S
# RESULT FROM INPUT-LINE-NUMBER 14 :
(0,0)  (0,-1) (1,-1)  (0,0) (0,0)  :  0  3  0.10000000  0.30000000 S

# QUERY FROM INPUT-LINE-NUMBER: 18
(0,-1)  (0,0) (0,-1)  (0,0) (2,-2) :  0  -1 0.50000000  1.00000000 Q
# QUERY FROM INPUT-LINE-NUMBER: 20
(1,-1)  (0,0) (0,-1)  (0,0) (1,-2) :  0  -1 0.50000000  1.00000000 Q
```

2.5 PIT - Instructions

In this section we shortly describe some input-instructions that are used within the PIT-environment.

The PIT-environment allows a worst case analysis of a (unique) sentence (called conclusion) on a set of MaxEnt Models (obtained by choosing points in the intervals of the constraints[11]). While the queries (the questions of the user) have the format '`Q(`*formula*`)`', the conclusion has the format '`C(`*formula*`)`'. It is demanded that if a conclusion exists it has to stand before the first question in the input-file.

The worst case analysis runs on two different optimisation algorithms; each algorithm uses its own constraints. These two different types of constraints are distinguished by marking the equality sign with an additional '`K`' if the constraint is used for the worst case analysis.

The difference between conclusion and query (respectively between normal- and komplex-constraint) in the output of *tabl* consists only in the different types[12] '`C`' and '`Q`' (respectively '`S`' and '`K`') at the end of the row.

Example:

```
var a,b,c;

p(a)   =K (0.5,1.0];
p(a+b) =  (0.5,1.0];

c(b -|> -a) =(0.5,1.0];
q(b -|> -a) =(0.5,1.0];
q(a*c -|> b)=(0.5,1.0];
.
```

produces:

```
3  3  1  0  1  1  2

2  2  4

(0,-1) (0,-1) (1,-1)  :   0   1  0.50000000  1.00000000 K
(1,0)  (1,0)  (1,0)   :   1   0 -1.00000000 -1.00000000 S
(0,-1) (1,-1) (1,-1)  :   0   2  0.50000000  1.00000000 S

(0,0)  (2,-2) (0,-2)  :   0  -1  0.50000000  1.00000000 C
(0,0)  (2,-2) (0,-2)  :   0  -1  0.50000000  1.00000000 Q
(0,0)  (0,0)  (1,-2)  :   0  -1  0.50000000  1.00000000 Q
```

Please note, that the output lists at first the komplex-constraints and then all other constraints followed by the conclusion and the queries.

[11] described in [SCHRAMM & SCHULZ, 1996].

[12] see (1.4)

Chapter 3

Generating the Output Matrix

3.1 The 'weak' Indifference Principle

As described in Section 1.4 the transformation (1.2) of the original matrix A by demanding the principle of indifference leads to a new matrix $\hat{A}$ that consists only of different columns.

The mathematical representation of a constraint

$$P(b\,|\,a)=\delta \quad\Leftrightarrow\quad [\mathbf{1}-\boldsymbol{\delta}]\cdot P(a\cap b)+[-\boldsymbol{\delta}]\cdot P(a\cap\neg b)+[\mathbf{0}]\cdot P(\neg a)=0 \qquad (3.1)$$

splits the space Ω into three disjoint sets of elementary events, which differ by their factors (i.e. **matrix-entries**) in the representation of the system of equations (3.1):

$$\begin{array}{lcl} \omega_i \in a\cap b & \Rightarrow & \text{matrix-entry } [1-\delta] \\ \omega_i \in a\cap\neg b & \Rightarrow & \text{matrix-entry } [-\delta] \\ \omega_i \in \neg a & \Rightarrow & \text{matrix-entry } [0] \end{array}$$

tabl creates the matrix $\hat{A}$ by building up a tree T starting with the root Ω (see Figure 3.1). Each constraint increases the height of T by splitting every leaf t in a maximum of three different sets ($t\cap ab$,$t\cap a\neg b$,$t\cap\neg a$) representing the different matrix-entries ($[1-\delta]$,$[-\delta]$,$[0]$). Empty sets can be deleted, because they represent the impossible event. The internal representation of sets is efficiently possible by using BDDs[‡] [1]. Therefore the list of matrix-entries along the path from the root to a leaf represents one column of $\hat{A}$. (In fact we need to store only the leafs of T together with their list of matrix-entries).

A further simplification is possible by regarding logical constraints. We know that a logical constraint does not add a row to A, but only leads to deleting columns.

$$\begin{array}{lll} \text{case 1:} & P(b\,|\,a)=0 & \Leftrightarrow P(a\cap b)=0 \\ \text{case 2:} & P(b\,|\,a)=1 & \Leftrightarrow P(a\cap\neg b)=0 \end{array} \qquad (3.2)$$

As a consequence of this we need to cut all leaves of T with $\neg(a\cap b)$ (respectively $\neg(a\cap\neg b)$).

The fact that a logical constraint does not increase the number of leaves in T, an unconditional constraint increases the number of leafs by at most two and a conditional constraint

[‡] see [Bryant, 1986].

[1] The BDD-software-package used is written by David E. Long (see Appendix C).

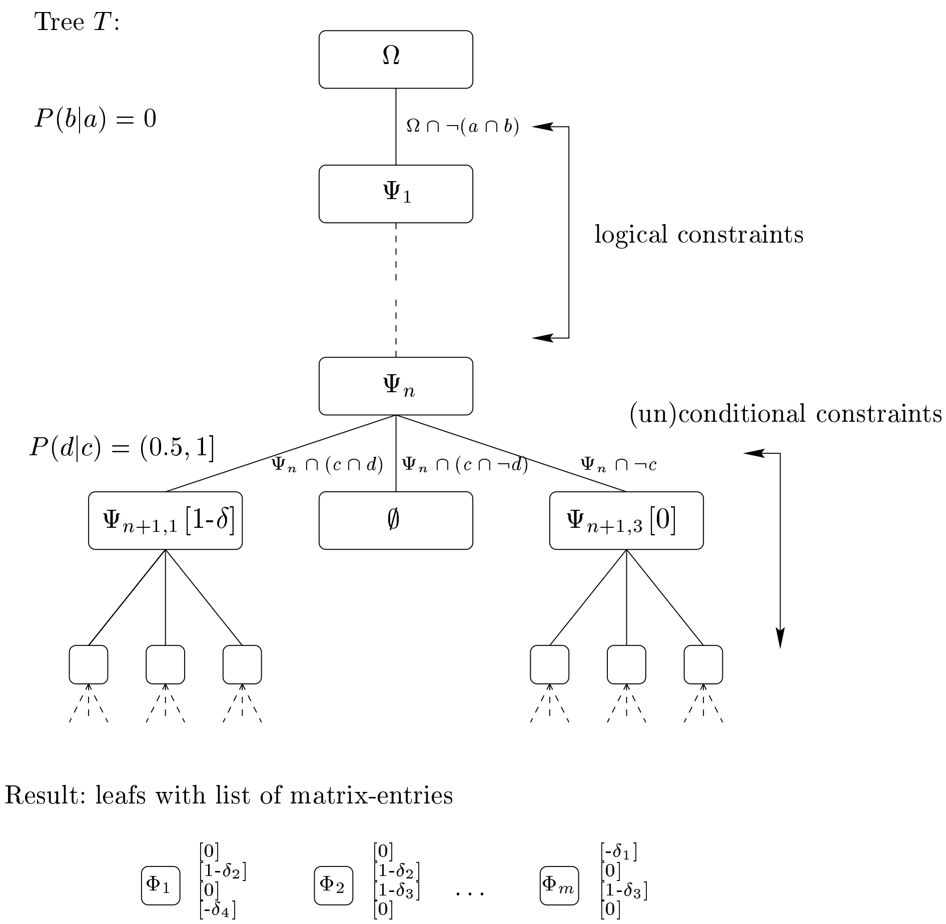

Figure 3.1: Creation of the tree T

increases the number of leafs by at most three, suggests a sorting of constraints. To minimize the number of necessary cuts between the sets we keep the growth of T as small as possible by first applying the logical constraints on T, followed by the unconditional constraints and at last the conditional constraints. The multiplicator-line entry M_i for each column i equals the number of elementary events[2] $|\Phi_i|$[‡].

The factor F_i of the column i of a query '`Q(d|c)=...`' can be calculated by counting the number of elementary events for each of the factors $[1 - \delta_q]$ and $[-\delta_q]$:

$$F_i = |\Phi_i \cap (c \cap d)| \cdot [1 - \delta_q] + |\Phi_i \cap (c \cap \neg d)| \cdot [-\delta_q] \tag{3.3}$$

The worst case complexity of the growth of leafs in T is therefore $\mathbf{O}(3^l)$ (which always has to be smaller then 2^r). Our results show that in 'realistic' databases the growth factor is smaller then $\mathbf{O}(1.5^l)$.

[2]i.e. full conjunctions.
[‡]The precision is limited due to the internal representation as the C-type *float*.

3.2 The 'strong' Indifference Principle

The equality of columns is a sufficient condition for indifference ($\hat{A}$ therefore contains only different columns). But the demand of the principle of indifference is more far-reaching then this simple equality.
We use an algorithm that is able to detect another type of indifference in the system of equations. We call this type of indifference '**strong**' indifference. A detailed description and proof can be found in Appendix D.

In short terms, we look for a set of rows $\sigma_r > 1$ and a set of columns $\sigma_c > 1$ within $\hat{A}$ for that we can show, that there always exists an indifferent[3] solution in which the rows of σ_r are equal[4]. Analogous to the weak indifference, the 'strong' indifferent columns can be merged (considering the multiplicator-line).

Another requirement for the indifference of the rows[5] i and j consists in the equality of the intervals for δ_i and δ_j. If the strong indifference between two rows fails due to unequal intervals a warning is displayed. Before applying the strong indifference algorithm it is necessary to '**normalise**'[6] the matrix $\hat{A}$ by changing some of the db-constraints into the db-constraints of their reverse event, which however does not influence the content of information in the database.

The result of the strong indifference algorithm is the final output-matrix $\tilde{A}$.

If the application of the 'strong' indifference is not wished, it can be switched off by the `%noindif`-option in the input-file.

3.3 Results

The combined two indifference principles can lead to an extreme simplification of the system of equations. For demonstration we present a result from the PIT-environment:

To evaluate the performance of *tabl* we used the tool in a medical expert-system for perceiving poisons[7]. The database consisted of 20 variables and 123 constraints. The output-matrix had the size 2125×40 (reduced from a worst case 1048576×123 matrix and a weak indifferent 8353×70 matrix).

The compilation was done in about half a minute (using a Linux Pentium-133 PC).

[3] meaning indifferent in the set of columns (respectively the events corresponding to the columns) σ_c.
[4] so we can reduce σ_r to one representing constraint in $\tilde{A}$.
[5] respectively constraints.
[6] see footnote in Section 4.2.
[7] see Appendix E.

Chapter 4

Checks, Errors & Warnings

4.1 Consistency Checks

One task of *tabl* is to detect inconsistencies in the database. In the following sections, we shortly describe the checks that are performed by *tabl*. The error- and warning-messages that are displayed should help the user to find the reason(s) of the detected inconsistencies.

For easier understanding we use simple examples that show the idea behind each check. The interesting parts of the input-files (with line-numbers) and compiler-messages are printed for each example.

4.1.1 Reverse Events

If the database includes information about an event a, it automatically includes information about the event $\neg a$. For example '$p(a) \in [0.8, 1.0]$' implies '$p(\neg a) \in [0.0, 0.2]$'. If the database contains the additional information '$p(\neg a) \in [0.0, 0.1]$' it would change the former probability-interval of a to '$p(a) = [0.9, 1.0]$'. *tabl* searches for reverse events in the database and adapts their intervals. Afterwards one of the two events is deleted. An error occurs if the intersection between the intervals is empty.

Example:

```
5    p(a) =(0.8,1.0];
6    p(-a)=(0.3,1.0];
```

produces:

```
[WARNING]  LINE 5 AND 6 ARE REVERSE EVENTS (NOW EQUAL)
[ERROR  1] SEMERROR : CONTRADICTORY CONSTRAINTS,
           LINE 5 AND 6 ARE EQUAL EVENTS, BUT HAVE DIFFERENT PROBABILITIES
```

4.1.2 Rank Consequences

The rank concept makes it possible to assign two or more events exactly the same probability. As a further consequence it may be necessary to merge primary different ranks to a new one.

Example 1:

```
5   p(a) =
6   p(b) = (0.5,1];
7
8   p(-a)=
9   p(c) = (0.3,1];
```

produces:

```
[WARNING]  LINE 5 AND 8 ARE REVERSE EVENTS (NOW EQUAL)
[WARNING]  LINE 5 AND 8 ARE EQUAL EVENTS
```

So this output equals the output of the following database:

```
5   p(a)=
6   p(b)=
7   p(c)=(0.5,0.7];
```

Another aspect is the equality of reverse events, e.g. '$p(a) = p(\neg a)$' leads to '$p(a) = p(\neg a) := 0.5$'.

Example 2:

```
5   p(a) =
6   p(-a) = [0.5,1.0];
7
8   p(b)=
9   p(-b) = 0.3;
```

produces[1]:

```
[WARNING]  LINE 5 AND 6 ARE REVERSE EVENTS WITH p(e)=0.5
[WARNING]  LINE 5 AND 6 ARE EQUAL EVENTS
[ERROR  1] p(e)==p(-e) => p(e)==0.5 IS NOT IN INTERVAL
           LINE 8 AND 9 ARE REVERSE EVENTS WITH p(e)=0.5
```

[1]The error- and warning-messages always use the default event '`e`' and '`f`' indifferent from the real variable-name.

4.1.3 Logical Constraints

Logical constraints can influence the probability of other constraints as well. *tabl* detects those influences that change other constraints into logical constraints.

Example 1:

```
5    p(a) = 1.0;
6    p(b) = 1.0;
7    p(c -|> -a) =[0,0.5);
8    p(c -|> a*b)=[0,0.5);
```

produces:

```
[WARNING]  LINE 7 HAS PROBABILITY 0
[ERROR  1] SEMERROR : p(e) WITH PROBABILITY 1 FOUND, WHERE 1 IS NOT IN
                      INTERVAL
           ERROR IN LINE : 8
```

4.1.4 Undefined Constraints

We define a conditional constraint as **undefined** if the condition has the probability 0. *tabl* detects undefined constraints in the database and stops with an error. If a conclusion or query is undefined, *tabl* displays a warning. An undefined conclusion or query consists only of '(0,0)' matrix-entries in the output-file.

Example 1:

```
5    p(a)=1.0;
6    p(b | -a)=(0,0.5];
```

produces:

```
[ERROR  1] SEMERROR : p(e-|>f) FOUND, WHERE p(e)=0
           ERROR IN LINE : 6
```

Example 2:

```
5    p(a)=1.0;
6
7    q(b | -a)=(0,0.5];
```

produces:

```
[WARNING]  SEMERROR : p(e-|>f) FOUND, WHERE p(e)=0
           UNDEFINED QUERY IN LINE 7
```

4.1.5 Contradictory Constraints

tabl also checks for some more sophisticated integrity conditions:

Example 1:

```
5   p( a* b) =0;
6   p( a*-b) =0;
7   p(-a* b) =0;
8   p(-a*-b) =0;
```

produces:

```
[MATRIX ERROR] CONTRADICTORY CONSTRAINT NEAR LINE 8
```

Example 2:

```
5   p( a* b) =0;
6   p( a*-b) =0;
7   p(-a* b) =0;
8   p(-a*-b) =[0,0.5];
```

produces:

```
[ERROR  1] SEMERROR : p(e) WITH PROBABILITY 1 FOUND, WHERE 1 IS NOT IN
                      INTERVAL
           ERROR IN LINE : 8
```

This ability is limited to logical constraints. For example *tabl* does not detect the following contradiction:

```
5   p( a * b)=[0,0.1];
6   p( a *-b)=[0,0.1];
7   p(-a * b)=[0,0.1];
8   p(-a *-b)=[0,0.1];
```

However the PIT-environment is able to detect this contradiction immediately.

4.2 Strong Indifference Warnings

The strong indifference searches for indifferent columns due to a number of rows given in the database. An additional precondition for merging two rows i and j by strong indifference consists in the equality of the intervals of δ_i and δ_j.

If the strong indifference fails due to the inequality[2] of the two intervals, *tabl* displays a warning.

Example:

```
5   p( a * b)=(0.3,1.0];
6   p( a *-b)=(0.4,1.0];
```

produces this message:

```
LINES 5 AND 6 COULD POSSIBLY BE EQUAL !
```

and this output:

```
 2  1  1

(1,0)  (1,0)  (1,0)   :   1   0 -1.00000000 -1.00000000 S
(0,-1) (0,-1) (1,-1)  :   0   1  0.30000000  1.00000000 S
(0,-1) (1,-1) (0,-1)  :   0   2  0.40000000  1.00000000 S
```

Now we modify the example by adapting the two intervals:

```
5   p( a * b)=(0.3,1.0];
6   p( a *-b)=(0.3,1.0];
```

and are getting this output:

```
 2  1

(1,0)  (2,0)   :   1   0 -1.00000000 -1.00000000 S
(0,-1) (1,-2)  :   0   1  0.30000000  1.00000000 S
```

This adaption allows *tabl* to merge the two constraints due to strong indifference. In many cases such a modification leads to a significant reduction of the system of equations.

4.3 Operation Order

tabl performs a number of consistency-checks during the creation of the output-file. The order of these checks is very important for the efficiency of the compilation. The following operations are performed:

[2]The intervals are normalised before comparison. This is necessary because $p(a)$ and $p(\neg a)$ describe the same event. Therefore the normalisation selects the (unique) representation (either $p(a)$ or $p(\neg a)$) for each db-constraint '$p(a) = \delta_i$' in which the upper interval-border of δ_i has the bigger value.

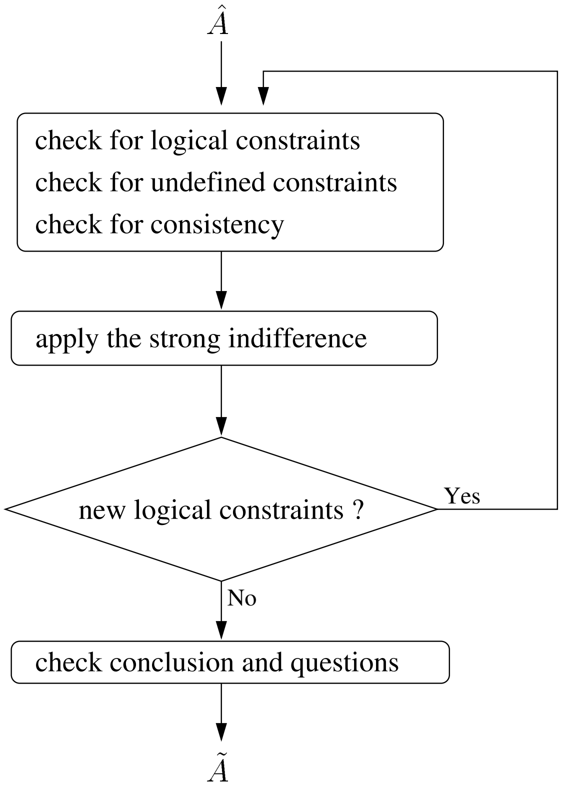

Figure 4.1: Operation order

- check for logical constraints

  Searches for rows i that imply logical constraints[3]. If a logical constraint is found and its probability 0 (respectively 1) lies in the probability-interval of δ_i the necessary columns and the row i are removed from the system of equations[4].
  Worst case complexity of this check: $\mathbf{O}(\hat{n} \cdot \hat{l})^{\S}$.

- check for undefined constraints

  Searches for undefined constraints ($p(b \mid a)$ with $p(a) = 0$) in the current system of equations and in the removed (logical) constraints. For logical constraints we check if there exists at least one event (i.e. full conjunction) within the condition a that does not have probability $0^{\ddagger}$. For all rows of the matrix we just have to verify that there exists at least one column that does not have a '`(0,0)`' entry.
  Worst case complexity: $\mathbf{O}(\hat{n} \cdot \hat{l})$.

[3] i.e. all rows i whose columns j contain either only '$k_{i,j}$`*(0,1)`' entries (probability 0) or only '$k_{i,j}$`*(1,-1)`' entries (probability 1), $k_{i,j} \in \mathbb{N}_0^+$.
[4] Naturally this also happens to rows that have equal rank.
[§] The dimension of the matrix $\hat{A}$ is: width $\hat{n} \leq 2^r$ and height $\hat{l} \leq l$.
[‡] This operation uses the internal BDD-representation.

- check for consistency

  Checks for equal[5] (respectively reverse[6]) events in the matrix. If such an equality is found, *tabl* adapts the probability-intervals and ranks. All but one of the equal rows are removed from the system of equations.
  Worst case complexity: $\mathbf{O}(\hat{n} \cdot \hat{l}^2)$.

- apply the strong indifference[7]

  Worst case complexity: $\mathbf{O}(\hat{n}^2 \cdot \hat{l}^2)$.

The most expensive operation is the calculation of the strong indifference matrix. To make the input for this algorithm as small as possible, all other operations (which can reduce the system of equations) are performed before.

The consistency check as well as the strong indifference algorithm produces a matrix $\tilde{A}'$ that could contain new logical constraints. These logical constraints again cause a reduction of columns and rows within $\tilde{A}'$ which eventually leads to new strong indifferent columns.

Therefore it is necessary to repeat this cycle of operations until $\tilde{A}'$ contains no more logical constraints.

Example: (*'b4.tabl'*[8])

```
%verbose
%b4.dat

var qu,re,pa,po,ha;

p(pa -|> -ha) = 1;
p(qu -|> pa ) = (0.5,1];
p(re -|> ha ) = (0.5,1];
p(pa -|> po ) = (0.5,1];
p(ha -|> po ) = (0.5,1];

q(qu*-re -|> pa) = (0.5,1];
.
```

[5] The two rows of the matrix simply have to be identical.

[6] The representation of the reverse event of a row can easily be calculated and then compared with all other rows of the matrix.

[7] see Appendix D.

[8] taken from [Greiner & Schramm, 1994].

When compiled with the verbose[9] option '*tabl -v b4.tabl*', these messages are displayed:

```
WELCOME TO THIS VERSION V2.17 05.12.96
INTERNAL31
WORKING...

CALCULATED TREE HEIGHT: 4

CALCULATING BDD-TREE
 WORKING ON LINE : 6  TREE-WIDTH: 1
 WORKING ON LINE : 7  TREE-WIDTH: 1
 WORKING ON LINE : 8  TREE-WIDTH: 3
 WORKING ON LINE : 9  TREE-WIDTH: 8
 WORKING ON LINE : 10 TREE-WIDTH: 14
 EXIT TREE-WIDTH: 20

WORKING ON INTEGER-MATRIX
WORKING ON QUESTIONS

WORKING ON INDIFFERENCE
 CHECKING MATRIX
 NORMALISATION
 CHECKING INTERVALS
 CHECKING RECURSIVE
 COLUMNS REDUCED FROM 20 TO 11

CHECKING CONCLUSION AND QUESTIONS

WORKING ON OUTPUT

TASK COMPLETED AND WROTE INTO b4.dat
C U
```

[9] '*tabl -v*' prints the current status during compilation.

And the file '*b4.dat*' with this content is created:

```
# CREATED 5.12 AT 18:28 (MET)

# NUMBER OF: COLUMNS ROWS KOMPLEX GRID SQP-POINT SQP-INT QUERIES
11  3  0  0  1  2  1

# EPS: 0.00000000

# ORDER: DB-CONSTRAINT ':' SUM,RANK,LOWER-BOUND,UPPER-BOUND,TYPE {NUMBER}
# MATRIX-ELEMENTS : (n,m) = n*1 + m*delta_i

# L_CONSTRAINTS FROM INPUT-LINE-NUMBERS :   6

# MULTIPLICATOR LINE:
 2  1  1  1  1  1  1  1  2  1  1

# NORMALISATION CONSTRAINT:
(1,0) (2,0)  (2,0)  (4,0)  (2,0)  (2,0)  (2,0)  (2,0)  (1,0)  (2,0)  (2,0)
    :   1   0 -1.00000000 -1.00000000 S
# RESULT FROM INPUT-LINE-NUMBER  7 :
(0,0) (0,0)  (0,0)  (0,-2) (0,-1) (0,-1) (1,-1) (1,-1) (0,-1) (1,-2) (1,-2)
    :   0   1  0.50000000  1.00000000 S

# RESULT FROM INPUT-LINE-NUMBER  9 :
(0,0) (0,-1) (1,-1) (0,0)  (0,-1) (1,-1) (0,-1) (1,-1) (0,0)  (0,-1) (1,-1)
    :   0   2  0.50000000  1.00000000 S

# QUERY FROM INPUT-LINE-NUMBER: 12
(0,0) (0,0)  (0,0) (0,-2)  (0,-1) (0,-1) (1,-1) (1,-1) (0,0)  (0,0)  (0,0)
    :   0  -1  0.50000000  1.00000000 Q
```

Bibliography

[Bryant, 1986] Bryant, R. (1986). Graph based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691.

[Ertel et al., 1996] Ertel, W., Goller, C., Schramm, M., and Schulz, S. (1996). *SETTHEO/NN and PIT Evaluation Report*, volume ESPRIT Project 9119 MIX,S9.2. Technische Universität München.

[Greiner & Schramm, 1994] Greiner, M. and Schramm, M. (1994). *Nichtmonotones Schließen auf Wahrscheinlichkeitsmodellen, begründet durch die Prinzipien Indifferenz, Unabhängigkeit und maximale Entropie*, volume TUM-I9440. Technische Universität München.

[Greiner & Schramm, 1995] Greiner, M. and Schramm, M. (1995). *Non-Monotonic Reasoning on Probability Models: Indifference, Independence & MaxEnt, Part1: Overview*, volume TUM-I9509. Technische Universität München.

[Greiner & Tinhofer, 1996] Greiner, M. and Tinhofer, G. (1996). *Stochastik für Studienanfänger der Informatik*. Carl Hanser Verlag, München Wien.

[Jaynes, 1978] Jaynes, E. (1978). *Papers on Probability, Statistics and Statistical Physics*, chapter Where do we stand on Maximum Entropy?, pages 210–314. Kluwer Academic Publishers.

[Schramm, 1996] Schramm, M. (1996). *Indifferenz, Unabhängigkeit und maximale Entropie: Eine wahrscheinlichkeitstheoretische Semantik für Nicht-Monotones Schließen*, volume 4. CS Press, Dissertationen zur Informatik edition.

[Schramm & Schulz, 1996] Schramm, M. and Schulz, S. (1996). Combining propositional logic with maximum entropy reasoning on probability models. In Niemelae, I., editor, *Proceedings of the ECAI'96 Workshop on Integrating Nonmonotonic into Automated Reasoning Systems*. Universität Koblenz-Landau.

Appendix A

Input & Output Format

The syntax is described in EBNF-like notation:

[ ]*	means 'as many as you want'.
[ ]+	means 'at least once'.
$[\ ]^i$	means 'repeat i times'.
[ ]	means 'optional'.
{ }	is used for grouping.
'*c*'	means 'character(s) *c* is(are) expected'.
comment	means 'verbal (not formalised) comment'.

A.1 Input Format

The well formed *tabl* input is defined as :

TABL_INPUT	::=	[**OPTION**]* **VAR_DEF** [**CONSTRAINT**]+ [**CONCLUSION**] [**QUERY**]* '.'
OPTION	::=	'%verbose' | '%noindif' | '%eps' '=' **P_NUMBER** | '%*output-filename*'
VAR_DEF	::=	'var' **VARNAME** [',' **VARNAME**]* ';'
VARNAME	::=	***letter*** [***letter*** | ***digit***]*
CONSTRAINT	::=	[**P_EXPR**]+ [**I_TYPE**] **RESULT** ';'
P_EXPR	::=	{ **'P'** | **'p'** } '(' **FORMULA** ')' '='
FORMULA	::=	***any well formed 'formula' containing***: ***variables*** () - + * -> | -|>
I_TYPE	::=	{ 'K' | 'k' } | { 'G' | 'g' } '(' **N_NUMBER** ')'
N_NUMBER	::=	['0'|'1'|'2'|'3'|'4'|'5'|'6'|'7'|'8'|'9'|'8'|'9']+
RESULT	::=	**INTERVAL** | **P_NUMBER**
INTERVAL	::=	{ '['|'(' } **P_NUMBER** ',' **P_NUMBER** { ')'|']' }
P_NUMBER	::=	{ **'0'** | **'1'** } | [**'0.'** [**'0'**|**'1'**|...|**'9'**]+]

CONCLUSION ::= { 'C' | 'c' } '(' **FORMULA** ')' ['=' **RESULT**] ';'
QUERY ::= { 'Q' | 'q' } '(' **FORMULA** ')' ['=' **RESULT**] ';'

COMMENTS ::= **'#'** ***comment-text proceeds until '\n'***

Additionally the following semantics have to be kept in mind:

FORMULA respects the natural order of '`- * + -> | -|>`'. '`|`' or '`-|>`' can only be used once per **FORMULA**. The *output-filename* can be any Unix-filename that contains no special characters (like `%$?&`) and no spaces. Each option is allowed only once. A maximum of 69 variables is allowed. The name of a variable can have up to 32 characters and cannot contain any special characters. Comments are allowed everywhere in the input.

A.2 Output Format

The general output syntax of *tabl* without the '`%verbose`'-option is:

TABL_OUTPUT ::= '`:E`' |
INFO
MULT_LINE
[**MATRIX_LINE**]$^{\text{N\_ROWS}}$
[**CONCLUSION**]
[**QUERY**]$^{\text{N\_QUERIES}}$

INFO ::= **N_COLUMNS N_ROWS N_KOM N_GRID**
N_SQP_P N_SQP N_QUERIES
N_COLUMNS ::= ***column size of*** $\hat{A}$
N_ROWS ::= ***row size of*** $\hat{A}$
N_KOM ::= ***number of komplex-constraints***
N_GRID ::= ***number of grid-constraints (not yet needed)***
N_SQP_P ::= ***number of sqp-point-constraints***
N_SQP_I ::= ***number of sqp-interval-constraints***
N_QUERIES ::= ***number of queries***

MULT_LINE ::= [**N_NUMBER**]$^{\text{N\_COLUMNS}}$

MATRIX_LINE ::= **LINE N_INTERVAL RANK** { '`S`' | '`K`' | { '`I`' **N_NUMBER** } }
CONCLUSION ::= **LINE T_INTERVAL** '`C`'
QUERY ::= **LINE T_INTERVAL** '`Q`'

LINE ::= ['`(`' **N_NUMBER** '`,`' ['+'|'−'] **N_NUMBER** '`)`']$^{\text{N\_COLUMNS}}$
'`:`' **z_i**
z_i ::= '`1`' ***for the normalisation constraint, else*** '`0`'
N_INTERVAL ::= **P_NUMBER P_NUMBER**
T_INTERVAL ::= **N_INTERVAL** | { '`-1.0`'['`0`']∗ '`-1.0`'['`0`']∗ }

RANK	::=	**N_NUMBER**
N_NUMBER	::=	['0'|'1'|'2'|'3'|'4'|'5'|'6'|'7'|'8'|'9'|'8'|'9']+
P_NUMBER	::=	{**'0'**|**'1'**} | { **'0.'** [**'0'**|**'1'**|. . . |**'9'**]+ }
COMMENTS	::=	**'#'** ***comment-text proceeds until '\n'***

The only output ':E' indicates that 'an error has occurred'. The **INFO**-line specifies the size of the following output-text. In **LINE** the expression '(n,m)' represents the matrix element '$\mathtt{n} * 1 + \mathtt{m} * \delta_i$'. The matrix $\tilde{A}$ is ordered by first komplex-constraints followed by all sqp-point-constraints and at last all sqp-interval-constraints[1]. The interval-borders of the normalisation constraint are set to '-1.0'. The interval of all queries or the conclusion that have no **RESULT** are set to the interval $[0.0, 1.0]$. If the %verbose-option is specified in the input-file every output line has a comment. Therefore further programs can ignore all lines beginning with '#' or consisting of '\n'.

[1] see Section 2.5.

Appendix B
Errors & Warnings

Whenever an error occurs the compilation of the input file cannot be completed. Warnings are used to inform the user about any irregularities or compiler-caused changes in the database.

If possible the line number at which the error or warning has (probably) occurred is printed.

B.1 Errors

B.1.1 Syntax Errors

- general syntax-error

  If no well-formed[1] input file is given the compiler stops with:

  `PARSER-ERROR LINE` *`line`* `parse error`

  Hint: The absence of the '.' token at the end of the input-file also causes this error.

- output-filename

  If the output-filename is declared twice in the input file the compiler stops with:

  `OUTPUT-FILE ALREADY DECLARED :` *`filename`*

- variable-declaration

  If used variables are declared twice or not declared at all or miss-spelled, the compiler stops with:

  `VARIABLE ALREADY DECLARED :` *`variable`*
  `VARIABLE NOT DECLARED :` *`variable`*

- interval-borders

  If a given probability is no valid subinterval of $[0, 1]$ the compiler stops with:

  ```
  INTERVAL : LOWER BOUND > UPPER BOUND
  INTERVAL : LOWER BOUND < 0
  INTERVAL : UPPER BOUND > 1
  PROBABILITY OUT OF LIMITS
  ```

[1] see Appendix A.1.

Please remember the influence of the `%eps`-option which changes open interval borders.

B.1.2 Semantic Errors

Semantic errors are mostly caused by contradictory information in the database. A detailed description of some semantic errors can be found in Section 4.1. If a semantic error occurs, the output-file consists only of the token ':E'.

- contradictory constraints

  CONTRADICTORY CONSTRAINTS
  CONTRADICTORY CONSTRAINTS NEAR LINE *line_1*
  CONTRADICTORY CONSTRAINTS, LINE *line_1* AND *line_2* ARE EQUAL EVENTS
  BUT HAVE DIFFERENT PROBABILITIES

- rank consequences[2]

  p(e)==p(-e) -|> p(e)==0.5 IS NOT IN INTERVAL, LINE *line_1* AND *line_2* ARE
  REVERSE EVENTS WITH p(e)=0.5
  p(e) WITH PROBABILITY 0/1 FOUND, WHERE 0/1 IS NOT IN INTERVAL

- undefined probabilities

  p(e-|>f) FOUND, WHERE p(e)=0

B.2 Warnings

- new logical constraints

  LINE *line* HAS PROBABILITY 0/1

- rank consequences

  LINE *line_1* AND *line_2* ARE EQUAL EVENTS
  LINE *line_1* AND *line_2* ARE REVERSE EVENTS WITH p(e)=0.5
  LINE *line_1* AND *line_2* ARE REVERSE EVENTS (NOW EQUAL)

- conclusion and queries

  CONCLUSION HAS PROBABILITY 0/1
  UNDEFINED CONCLUSION
  QUERY IN LINE *line* HAS PROBABILITY 0/1
  UNDEFINED QUERY IN LINE *line*

- 'strong' indifference[3]

  LINES *line_1* AND *line_2* COULD POSSIBLY BE EQUAL !

[2] The error- and warning-messages always use the default event 'e' and 'f' indifferent from the real variable-name.

[3] see Appendix D.

Appendix C
Software-Installation

C.1 General

tabl should run on all Unix systems that support the GNU C/C++ compiler version 2.6.3 or higher. In fact the program should run with every C/C++ compiler that supports templates (preconfigured however is the GNU-Compiler).
Already tested platforms are:

- Linux 1.2.13, gcc 2.6.3
- Linux 2.0.11, gcc 2.7.2
- SUN Solaris 5.4 & 5.5, gcc 2.7.2

C.2 Additional Software Needed

tabl relies on this additional software-packages:

- **LEDA-R-3.3.c** (not included)

  is a software package (free for academic research and teaching) consisting of useful datatypes and algorithms.

  Available under: ftp.mpi-sb.mpg.de /pub/LEDA

- **BDD library** (included)

  is a software package for Binary Decision Diagrams written by David E. Long, email: long@research.att.com .

- **lex and yacc** (not included)

  This two programs (or compatible software) should be available on every Unix system.

C.3 Getting Started

1. Check if LEDA is installed on your system.
2. Go to the *tabl* main directory.
3. Check the user defined entries in the file *work/Makefile*.
4. Type *make*.

C.4 Installation

tabl consists mainly of four different programs:

tabl_bdd	an Unix-filter (for up to 31 variables)
tabl_bdd_int	an Unix-filter (for up to 69 variables)
tabl	a shell script for using the filters.
tabl_count	counts variables in a input-file.

The easiest way is to copy or link this four programs in your local or system bin-directory (f.e. *ln -s tabl ~/bin*).

The second possibility is to set the *TABL_PATH* environment variable to the tabl-source directory (f.e. '*export TABL_PATH=~/src/tabl*' for the korn-shell).

C.5 Commands

- *tabl [-v* | *-h] input-file [output-file]*

  compiles the *input-file* and writes the result in a file named *output-file* (if no '`%file-name`' option is given in the input-file). Options: '*-v*' switches on the verbose mode and '*-h*' prints a small help-file.

- *tabl_bdd [-v* | *-h]* < *input-file* > *output-file*

  can be used as a filter that reads its input from *stdin* and writes the output to *stdout*. It uses a fast integer type and can handle up to 31 variables.

- *tabl_bdd_int [-v* | *-h]* < *input-file* > *output-file*

  analogous to *tabl_bdd*, but can handle up to 69 variables.

- *tabl_count input-file*

  returns the number of variables in the input-file.

Appendix D
The 'strong' Indifference Algorithm

D.1 Proof

We will now

1. introduce a special partition on the columns and rows of the matrix $\hat{A}$. With the help of this partition we define the principle of strong indifference. The application of this principle leads us a new system of equations (D.1).

$$\tilde{A} \cdot \tilde{p} = \tilde{z} \tag{D.1}$$

2. show the consistency of the system (D.1) by explicitly constructing a solution $\tilde{p}$ from any solution $\hat{p}$ of the old system of equations.

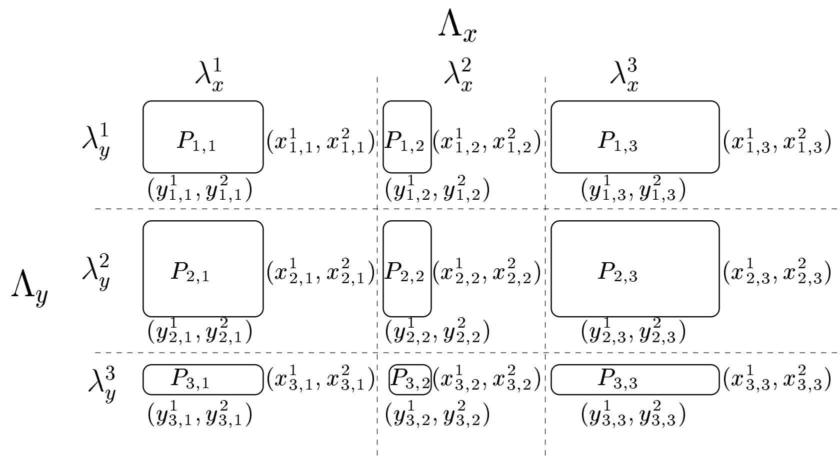

Figure D.1: Example for a 3×3 partition of $\hat{A}$ (sorted by partitions into sub-matrices $P_{i,j}$).

ad 1:

Starting with the system of equations

$$\hat{A} \cdot \hat{p} = \hat{z} \tag{D.2}$$

(leaving out the normalisation constraint and remembering the multiplicator-line) we look for a partition Λ_x into sets of columns and a partition Λ_y into sets of of rows having the following properties (see Figure D.1):

- For each set of columns $\lambda_x \in \Lambda_x$ and each set of rows $\lambda_y \in \Lambda_y$ holds:

$$(x^1_{\lambda_x,\lambda_y}, x^2_{\lambda_x,\lambda_y}) := \sum_{j\in\lambda_y} \hat{A}(j,i) \quad \text{is identical for all} \quad i \in \lambda_x \tag{D.3}$$

$$(y^1_{\lambda_x,\lambda_y}, y^2_{\lambda_x,\lambda_y}) := \sum_{i\in\lambda_x} \hat{A}(j,i) \quad \text{is identical for all} \quad j \in \lambda_y \tag{D.4}$$

- The number of sets in the partitions Λ_x and Λ_y is minimal, respectively the number of columns in each set of a partition $\lambda_x \in \Lambda_x$ and the number of rows in each set of a partition $\lambda_y \in \Lambda_y$ is maximal.

Now we consider the system of equations $\hat{A}\cdot\hat{p}' = \hat{z}$ which originates if we set the probabilities of the events ω_i within each set $\lambda_x \in \Lambda_x$ equal (meaning $\hat{p}'(\hat{\omega}_i) := p_{\lambda_x}$ for all $i \in \lambda_x$ [‡]). We get a new (equivalent) system of equations (D.5) from this system by merging[1] equal columns (i.e. the columns within each set $\lambda_x \in \Lambda_x$). The resulting rows within each set of rows $\lambda_y \in \Lambda_y$ have to be identical, so we can confine us to one representation per partition λ_y.

The system of equations of the example in Figure D.1 is therefore:

$$\tilde{A}\cdot\tilde{p} = \begin{pmatrix} x^1_{1,1} - x^2_{1,1}\delta_1 & x^1_{1,2} - x^2_{1,2}\delta_1 & x^1_{1,3} - x^2_{1,3}\delta_1 \\ x^1_{2,1} - x^2_{2,1}\delta_2 & x^1_{2,2} - x^2_{2,2}\delta_2 & x^1_{2,3} - x^2_{2,3}\delta_2 \\ x^1_{3,1} - x^2_{3,1}\delta_3 & x^1_{3,2} - x^2_{3,2}\delta_3 & x^1_{3,3} - x^2_{3,3}\delta_3 \end{pmatrix} \begin{pmatrix} p_{\lambda_1} \\ p_{\lambda_2} \\ p_{\lambda_3} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \tag{D.5}$$

We can see immediately that each solution of (D.5) fulfils (D.2).

ad 2:

By the demand of the principle of indifference indifferent events have the same probability. Therefore we construct for each solution $\hat{p}_0$ of (D.2) a new solution $\hat{p}'_0$ of (D.2) in which the columns i (respectively the corresponding events ω_i) of each set $\lambda_x \in \Lambda_x$ are considered indifferent:

$$\hat{p}'_0(\hat{\omega}_i) := |\lambda_x|^{-1} \sum_{k\in\lambda_x} \hat{p}_0(\hat{\omega}_k) \text{for all } i \in \lambda_x{}^{\S} \tag{D.6}$$

Now we have to show that for each $\hat{p}'_0$ (and for each $\tilde{p}_0$ which represents the solution $\hat{p}'_0$ in (D.5)[2]) holds:

- $\tilde{A}\cdot\tilde{p}_0 = \tilde{z}$ (The strong indifferent (merged) P-Model $\tilde{p}_0$ is a solution of the strong indifferent equation-system $\tilde{A}\cdot\tilde{p} = \tilde{z}$).

- $\hat{A}\cdot\hat{p}'_0 = \hat{z}$ (The strong indifferent P-Model $\hat{p}'_0$ is a solution of the weak indifferent equation-system $\hat{A}\cdot\hat{p} = \hat{z}$). This has to be true due to **ad 1**.

[‡] $\hat{p}'(\hat{\omega}_i)$ denotes the i-th element of $\hat{p}_i$.

[1] Analogous to the 'merging' of Section 1.4.4, we have to add the corresponding columns.

[§] This assignment also equals the MaxEnt-distribution within λ_x.

[2] see Figure D.2.

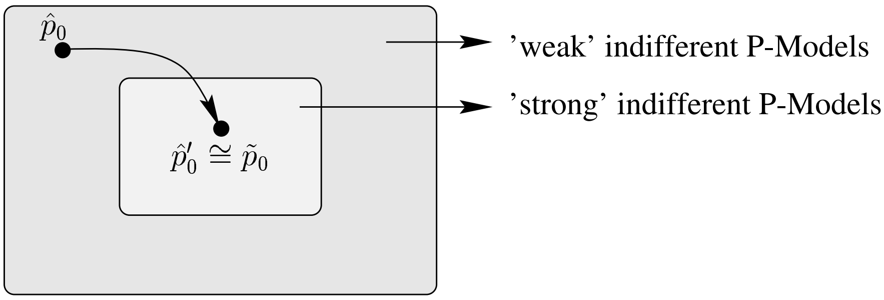

Figure D.2: Construction $\tilde{p}_0$ from $\hat{p}_0$.

To simplify the proof, we regard only the first set of the example in Figure D.1 (consisting of the sub-matrices $P_{1,1}$, $P_{1,2}$ and $P_{1,3}$).

If we add all rows of the partition λ_y^1 we get a new constraint that is valid for all P-Models $\hat{p}$:

$$
\begin{array}{rcl}
 & \sum_{k\in\lambda_x^1}(y_{1,1}^1, y_{1,1}^2)\cdot\hat{p}(\omega_k) + \ldots + \sum_{k\in\lambda_x^3}(y_{1,3}^1, y_{1,3}^2)\cdot\hat{p}(\omega_k) & = 0 \\
\Leftrightarrow & (y_{1,1}^1, y_{1,1}^2)\cdot|\lambda_x^1|\cdot\frac{1}{|\lambda_x^1|}\sum_{k\in\lambda_x^1}\hat{p}(\omega_k) + \ldots + (y_{1,3}^1, y_{1,3}^2)\cdot|\lambda_x^3|\cdot\frac{1}{|\lambda_x^3|}\sum_{k\in\lambda_x^3}\hat{p}(\omega_k) & = 0 \\
\Leftrightarrow & (y_{1,1}^1, y_{1,1}^2)\cdot|\lambda_x^1|\cdot\hat{p}'(\hat{\omega}_{\lambda_x^1}) + \ldots + (y_{1,3}^1, y_{1,3}^2)\cdot|\lambda_x^3|\cdot\hat{p}'(\hat{\omega}_{\lambda_x^3}) & = 0 \\
\Leftrightarrow & (x_{1,1}^1, x_{1,1}^2)\cdot|\lambda_y^1|\cdot\hat{p}'(\hat{\omega}_{\lambda_x^1}) + \ldots + (x_{1,3}^1, x_{1,3}^2)\cdot|\lambda_y^1|\cdot\hat{p}'(\hat{\omega}_{\lambda_x^3}) & = 0 \\
\Leftrightarrow & (x_{1,1}^1, x_{1,1}^2)\cdot\hat{p}'(\hat{\omega}_{\lambda_x^1}) + \ldots + (x_{1,3}^1, x_{1,3}^2)\cdot\hat{p}'(\hat{\omega}_{\lambda_x^3}) & = 0
\end{array}
$$

So we have proofed that the strong indifferent solution $\hat{p}'_0$ (respectively $\tilde{p}_0$) fulfils (D.5) and likewise (D.2).

From now on we can confine us to the strong indifferent P-Models represented by the solutions of (D.5) because we know that for each P-Model $\hat{p}_0$ of (D.2) we can construct another P-Model $\hat{p}'_0$ that fulfils the demand of the principle of indifference better and is a P-Model of (D.5).

q.e.d.

D.2 Remarks

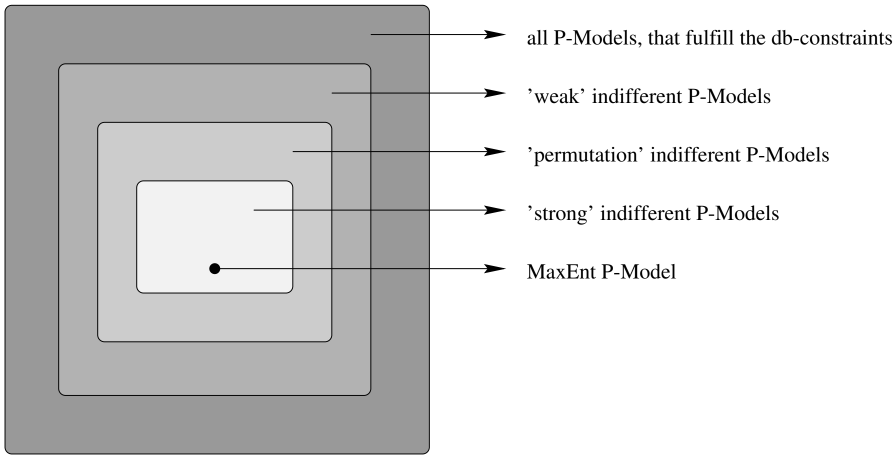

Figure D.3: Hierarchy of possible P-Models

The strong indifference algorithm includes the weaker condition that any permutation of rows followed by a permutation of columns that leaves the matrix unchanged is sufficient for indifference ('permutation'[3] indifference) of the set of permutated rows and columns (without proof).

Figure D.3 shows the hierarchy of the possible P-Models along the diminishing of the system of equations.

[3]see [Greiner & Schramm, 1995].

Appendix E

A Medical Expert-System Database (excerpt)

The physician determinated the probability of about 100 rules by using 7 qualitative steps (from 'almost never' up to 'almost always'). From this statements, the following file was generated. The questions are e.g. whether a patient with a number of symptoms is intoxicated with the poison 'adt' or with 'carbamate'. Symptoms with 3 possible values are mapped onto the combinations of two two-valued variables, where the 4th value is known to be zero.

```
% database.dat
var FC1, FC2, PAS1, PAS2, QRSIsnormal, QTIsnormal, ROTIsbrisk,
adt, alc, bar, ben, car, patientIscalm, pupils1, pupils2,
regardIsnormal, temp1, temp2, tonusIshypertonia, urineIsyes, phe;

#adjusting 3 valued variables

p( -temp1 * -temp2 ) = 0;

... (more rules follow here)


# specific Rules

P(tonusIshypertonia * -ROTIsbrisk * PAS1 * PAS2 -|> adt) = [0.65, 1];

... (more rules follow here)


# Rules under the condition of only adt
p( adt*-alc*-bar*-ben*-car*-phe -|> tonusIshypertonia ) = [0.75, 1.0];
p( adt*-alc*-bar*-ben*-car*-phe -|> urineIsyes ) = [0.75, 1.0];

... (more rules follow here)


# Rules under the condition of only carbamate
p( -adt*-alc*-bar*-ben*car*-phe -|> patientIscalm ) = [0.95, 1.0];
p( -adt*-alc*-bar*-ben*car*-phe -|> -ROTIsbrisk) = [0.75, 1.0];

... (more rules follow here)
```

```
# Queries

# Is the patient with the given symptoms intoxicated with adt  ?

q( temp1 * temp2 * patientIscalm * pupils1 * pupils2 * -tonusIshypertonia *
-ROTIsbrisk *  PAS1 * -PAS2 *  FC1 * -FC2  * QRSIsnormal * -QTIsnormal *
urineIsyes -|> adt) ;

# Is the patient with the given symptoms intoxicated with carbamate  ?

q( temp1 * temp2 * patientIscalm * pupils1 * -pupils2 * -tonusIshypertonia *
ROTIsbrisk *  -PAS1 * PAS2 *  FC1 * -FC2  * QRSIsnormal * QTIsnormal *
urineIsyes -|> car) ;
.
```