TECHNISCHE UNIVERSITÄT MÜNCHEN

INSTITUT FÜR INFORMATIK

Sonderforschungsbereich 342:
Methoden und Werkzeuge für die Nutzung
paralleler Rechnerarchitekturen

DBFW: A Simple DataBase FrameWork for the Evaluation and Maintenance of Automated Theorem Prover Data (incl. Documentation)

Peter Jakobi, Andreas Wolf

TUM-I9747
SFB-Bericht Nr. 342/28/97 A
November 97

TUM–INFO–11-I9747-130/1.–FI

Anforderungen an: Prof. Dr. A. Bode
Sprecher SFB 342
Institut für Informatik
Technische Universität München
D-80290 München, Germany

Druck: Fakultät für Informatik der
Technischen Universität München

DBFW: A Simple DataBase FrameWork for the Evaluation and Maintenance of Automated Theorem Prover Data (incl. Documentation)

Peter Jakobi, Andreas Wolf
Technische Universität München
Institut für Informatik
D-80290 München

e-mail: {jakobi,wolfa}@informatik.tu-muenchen.de

November 19, 1997

Abstract

This paper describes a simple yet generic database implementation framework for medium sized datasets, as they occur during tests and applications of automated theorem provers. The implementation covers automatic extraction of database objects from a set of text files, a text-based interface for simple database operations, and a tool for document, report and Webpage generation. This paper refers to a database of *SETHEO* proof data (*Setheodb*) as an example. It concludes with the description of *DBFW* as a part of the interactive proof system *ILF* [6]. The paper also serves as offline documentation for the framework.

1. Introduction

In the field of automated theorem proving (*ATP*), a lot of time and resources are often spent on tuning proof systems by evaluation of different parameter settings. In many (in our experience, almost all) cases, the output of such prover runs is used only to answer one specific question, even if the data has the potential to be reused. A database containing the results of such runs, together with an interface to access the stored data in an easy to automate manner, can make this possible. The database approach also saves disk space, as only relevant data is stored.

ATP systems are often used within the context of a workbench for constructing and checking proofs. Here, it is of interest for the designer of the *ATP* system to get information about the tasks given to the prover in an real environment. That feedback can be used to determine directions for further development of the prover. The information also helps to refine parameter settings for specific domains and user requirements.

In this paper, we introduce *DBFW*, a Perl database implementation framework. The framework provides

- support for automatic extraction of database objects from text files,
- a query interface,
- maintenance support for database files, and
- support for easy conversion to or from *DBFW* database formats.

Section 2 describes the starting point for the creation of the framework and its design goals. Section 3 deals with implementation and usage; the model elimination style theorem prover *SETHEO* [12] is used as a show-case application. Sections 4-6 offer hints for porting, describe available database formats and their patterns, and sketch further applications[1].

Throughout the paper, the *Setheodb* format for *SETHEO* [12] run data is used for examples and as a reference application for the framework. *Setheodb* stores the relevant information extracted from the log files of *SETHEO* program runs. *Setheodb* should be seen only as an example. The adaption of the scripts to another *ATP* system involves little effort, provided that the system produces textual output containing the relevant data.

We refer a reader interested in an introduction to the calculus of Tableaux to the books of Beth [3], Smullyan [16], and Fitting [9]. The special context of model elimination is considered in the book of Loveland [13] and in the paper of Stickel [17].

[1] An extended version including program and porting documentation is part of the *DBFW* distribution.

2. Problem Description and a Possible Solution

This section describes the usual, slightly chaotic, approach to so-called Data-Mining[1]. We define some requirements for improving the efficiency of the Data-Mining process and offer a solution meeting these requirements.

2.1 Data-Mining, classical approach

In our research group, the model elimination style theorem prover *SETHEO* [12] is being developed. Over a long period of testing, tens of thousands of *SETHEO* log files have been produced. These files fill several Gigabytes of disk space. Using these runs, we compare the performance of *SETHEO* with various sets of parameter settings. Considering the amount of available log files, we obviously should have been able to replace many runs of new experiments by data from log files of previous similar runs.

Subjectively selected and extracted data has proven to be difficult to reuse for the scientist extracting the data. This holds doubly for reusing a colleague´s data. So, in the past, almost all experiments had to be repeated to answer a new question, even if similar runs had already been done.

If you consider that testing a small set of parameters on a problem library like the *TPTP* [18] means a sleepless night for up to 100 processors, it is worth recycling the old and useful, but often chaotic and (previously) difficult to access, data.

Another consideration is an *ATP* system installed outside the developing group: it is difficult to obtain feedback and data on the tasks (not) proved. For developers however, this data is important in adapting a prover to the user´s application domain.

2.2 Requirements

To allow better use of the available data, we postulate the following requirements for the database:

- Extraction of (almost) all possibly relevant data from each log file. Here the creator of the database format has to select the “relevant” data and provide patterns for its extraction from the log files (see Section 3.1).
- Storing the data efficiently in terms of required space, time for access, and accessibility of the data.
- Simplicity of
  - format,
  - query tools (including possible modifications by *DBFW* users),
  - conversion into different database formats.
- Ease of automation of database queries and modifications.

[1] For a discussion of Data-Mining and Knowledge Discovery in Databases (*KDD*), see the November 96 issue of CACM. Our approach is taking place earlier and “mines” the raw (full-text) data in order to extract data-sets for a database.

2.3 Solution

To meet these requirements, we selected the following key concepts:

- An object-oriented database file format. Each object corresponds to a single line of the database file. This line stores all object tags. Each tag starts with the tag ID extended into a unique string followed by the tag value. An example object and its tags[2]:

```
_%Formula=SET001-1 _%Env=host sj30 datestamp 950804183600 _%Log=/LOGS/LOGFILE1
```

- A software-IC approach: a set of filters that can be combined within pipes. One of the filters (the extractor) will accept a set of log files and automatically extract the corresponding set of objects as a new database file.

[2]For easier parsing of an object´s line in the database, tag IDs are prefixed by _%.

3. Implementation of the Database Framework

We decided to use the script language Perl for the implementation. Perl is well suited for small text processing scripts, due to its very powerful regular expression matching on strings. A script language has the additional advantage of allowing quick changes and modifications. The Perl interpreter is sufficiently fast[1] for processing medium sized databases.

The framework consists of these main parts:

- Extractor *dbs_gen.p*[2]
- Query-Interface *db_cat.p*
- Document Generator *db_m5.p*

There are still other scripts in the framework. They offer some additional functionality not described here, or present predefined queries as examples. Scripts with the prefix *db_* are not database-format specific. Scripts with the prefix *dbs_* contain some specific code for the *Setheodb* format. Use the option -h to obtain online help.

3.1 dbs_gen.p: The Extractor

The extractor is a script for automatic, database-format-specific extraction of object tag values from text files. The input for *dbs_gen.p* is a list of log files. The output is a list of objects written to `stdout`, one for each log file. Command line options are available to define values for individual tags.

Extraction is performed by "patterns" defined in a separate file. For each tag, an associated pattern is defined to parse the log file with respect to the pattern and to extract the relevant information for the tag. Patterns are implemented as Perl code blocks. Tags with an empty value are omitted in the data base. The extracted object for each log file consists of the remaining tags. Example objects and a partial list of tag IDs are given at the end of Section 3.

dbs_gen.p restarts every `$jobsize` log files to conserve memory. If you wish to process more than `$jobsize` files, you **must** supply the log file names in a file instead of simply using `stdin`. `$jobsize` is currently defined as 499 in *dbs_gen.p*.

Usage: `dbs_gen.p [OPTIONS] FILE`

Options include:

```
-f <patternfile>    load the sepcified pattern file
-t <tag> <value>    set default value for objects whose <tag> is not defined
-T <tag> <value>    force all tags <tag> to <value>; deletes it on value ''
```

Usage example: Generating a database and setting the `User` tag to specify the owner of the runs to extract (option `-T`).

```
dbs_gen.p -T User 'Peter Jakobi' < file_of_logfilenames > database
```

If the -f option is not given, the generator loads its patterns from a file in the same directory and the same name as itself but with suffix *.pat*.

[1]We abused the framework on a database file of 80 MB. The answer time of several minutes is acceptable for a database of this size, especially if queries are first tested on smaller sample databases. The later described scripts *dbs_gen.p, db_split.p* and *db_cat.p* use only a few MB of memory independent of the database size.

[2]Scripts with the prefix *db_* are generic, while scripts with the prefix *dbs_* contain some specific code for the *Setheodb* format.

A log files may also contain so-called *embedded tag* lines: By default, these lines consist of a single tag starting in column 1, optionally prefixed by `%db␣` or `#db␣`. Embedded tags are copied directly into the object for this log file.

Example:

```
_%Formula=SET001-1
#db       _%Formula=SET001-1
%db                _%Formula=SET001-1
```

3.2 db_cat.p: The Query-Interface

The query-interface is the core script of the framework. *db_cat.p* is used for database maintenance and querying the database. *db_cat.p* reads a database from file or `stdin` and writes a modified database to `stdout`. Modifications include the selection and de-selection of objects, changing the contents of selected objects, and the removal of certain tags for all selected objects. As the shell command for a *db_cat.p* query can be fairly complicated, more complex queries should be stored as shell scripts for future use.

Implemented concepts of the relational data model are:

1. *Selection* - select objects for a new database file
2. *Projection* - select tags to keep or remove. This affects all selected objects.

Operations on multiple database files, such as *Joins*, are not implemented. Selections are performed before the projection.

Usage: `db_cat [OPTIONS] FILE`

The most important options are:

```
-d <FILE>           write all de-selected objects to FILE
-h                  online help, format descriptions, etc
-lowmem             memory efficient, slower processing
```

Select database objects - multiple selections are combined by logical *And* and performed in sequence:

```
-s <TAGID> <REGEXP> select objects with tags TAGID containing text or
                    regular expression REGEXP (case insensitive)
-S <PERLEXP>        select objects matching PERLEXP
```

See the UNIX manual page *perlre.1* for information on Perl regular expressions. PERLEXPs are arbitrary Perl expressions, from Perl regular expression statements (REGEXP) to complete blocks. They allow arbitrary side effects including the modification of objects.

Project selected objects - may occur at most once:

```
-P <PERLEXP>        project objects using tags matching PERLEXP
                    (match may include tag value; slow)
```

Without -s/-S/-P options, *db_cat.p* behaves similar to *grep*: The first argument is taken to be a REGEXP.

Usage examples:

- de-select every object with a Formula tag starting with SYN
  `db_cat.p -S '!/_%Formula=SYN/' db`
- select formulae starting with SYN, print only Formula tags
  `db_cat.p -s Formula SYN -P /_%Formula=/ db`
- change all tags in all objects with the tag ID "User" to the new ID "Owner"
  `db_cat.p -S s/_%User=/_%Owner=/ db`
- implement the query *a And b Or c* (*a*, *b*, *c*: PERLEXP)
  `db_cat.p -S` *a* `db -S` *b* `db > db.result`
  `db_cat.p -S` *c* `db >> db.result`

```
# cat db.result # or - removing duplicates:
sort db.result | uniq
```

- `dbs_mathquery1.p` - a complex query written in Perl that generates a datafile for further evaluation using *Mathematica* [22]. See also 6.2.

Further examples are given throughout this paper, in the *dbs_q** scripts and as part of the test suite (see *db.test/README*).

3.2.1 Multi-line Objects

Normally, objects consist of exactly one line. For easier editing and viewing however, multi-line objects are partially supported. They are intended as an alternative object representation suitable for editing objects using standard text editors. Multi-line objects are delimited by a sequence of at least two linefeeds[3]. Most of the other framework scripts also support a subset of these switches.

Options:

```
-h                   display usage / help text
-m                   rudimentary multi-line object mode
                     - object delimeter \n\n+
(-mi/-mo)            - -mi / -mo input/output-only  multi-line objects
                     - avoid whitespace-only lines within an object
                     - *DON'T* split an object by inserting empty lines
                     - *DON'T* add whitespace to object delimeters!
                     - use empty lines to separate comments and objects
                     - multi-line objects are supported by:
                       db_cat.p, db_ed.p
```

Example:

```
db_cat.p -mo . db > db.edit
vi db.edit
db_cat.p -mi . db.edit > db
```

3.3 db_m5.p: The Document Generator

The generator creates document files from databases. Both document layout and content are stored as objects in database files, and can be reused later. At the core, *db_m5.p* is a depth-first macro processor that allows certain objects to contain Perl code blocks and also to loop within a set of selected objects. A live example returning interactive HTML forms is available on the *DBFW* home-page on the WWW [8].

The layout specification is provided as a set of so-called group objects. Group objects are ordinary objects that use special semantics for some tags when interpreted by the `@@grp;` grouping command. Important commands, variables and database object tags are shown below in Figures 3.1 - 3.3.

db_m5.p´s expressiveness is derived from its three key features:

1. Depth-First Macro Expansion

   Evaluation of the input database starts at the specified object (defaulting to the object matching the regular expression `/_%Description=Main/` in the Perl array `@db`) by expanding the `_%Data` tag: Any embedded commands are recursively expanded [4] and the resulting text replaces the command string. If a complex command contains a "*", its input is expanded first before applying the command itself. The macro expansion can be requested explicitly by calling &expand().

[3] `db_fix.p -W` "validates" a multi-line database, assuming that all empty lines are object delimeters.

[4] Expansion takes place in the order shown in Figure 3.1: At first, all occurring @@nop´s are expanded, ... until at last, all occurring group objects are expanded. During each expansion, any embedded commands in the returned string are either recursively expanded or stripped, if further expansion is disallowed. Thus, commands usually *should not* be used to compute names or data of other commands.

Commands:	
@@nop;	nop (protects empty lines in input text, ...)
@@//;	line comment
@@\n;	newline
@@code;	run rest of attribute as Perl code
@@inc*$VARIABLE;	expand variable contents in output
@@inc*#ATTRIBUTE;	expand attribute contents in output
@@perl*$VARIABLE;	run variable contents as Perl code
@@perl*#ATTRIBUTE;	run attribute contents as Perl code
@@obj*#SELECTION;	include results of the selected object
@@grp*#SELECTION;	start a new group using the selected group object, optionally processing a set of objects

Figure 3.1: *db_m5.p* commands

The most important group object local variables:	
%object	attributes of the active object
%group	attributes of the active group object
%data	all attributes on the path to the current object
%groupdata	all attributes of group objects on the path
$grprc	variables containing accumulated text
$rc	variable containing accumulated text (local to `&expand()`)

Figure 3.2: *db_m5.p* variables

Some commands may additionally define a new *LaTeX*-like environment which leads us to the second key feature.

2. Grouping

   Each group object is invoked by a `@@grp;` command. The group object defines a new dynamic environment and optionally loops through a set of objects, applying the group´s `_%Data` and `_%PreProc / _%PostProc` tags to the data of each object `%object`.

3. Embedded Code

   Commands can explicitely refer to Perl code blocks and include their results into the accumulated output. In addition, the group objects defining the document structure can optionally run Perl statements to modify their output. Perl code blocks are evaluated within the current group´s environment and have access to various data about the visited objects. See the online help for a detailed list of variables.

See the online help text and the examples for more information on predefined variables and functions.

Usage: `db_m5.p [OPTIONS] FILE`

Attributes (tags) of group objects:	
File	expands to output filename ('¿X' to append to X)
LoopSelect	expands to SEL; loops over SELected objects
LoopSort	expands to a Perl sorting order
Error	expands to a Perl code block printing the error message $msg
	$grprc contains the text for output
LoopPreProc	expanded before loop
PreProc	expanded before expanding Data of the current object
Data	text content to expand, possibly containing commands
PostProc	expanded after processing Data of the current object
LoopPostProc	post-processing after finishing the loop

Figure 3.3: *db_m5.p* tags

Invoking *db_m5.p*:

- explicit calling: `db_m5.p -m db.test/m5.demo`
- implicit calling using `#!db_m5.p` as first line of an executable database file.
- with a wrapper like the cgi example *dbs_m5cgi.p*.

In addition, *db_m5.p* may be invoked automatically by adding a new MIME-type to your httpd.

The following example sets up a small interactive query interface on the World Wide Web. It is also included in both the distribution and the *DBFW* home page.

The first object is the root object for this document. The PreProc tag explicitely reads the example *Setheodb* database into the Perl array @DB_public. Combined with using only non-recursive commands on @DB_public, the script cannot compromise the security of the local host. Thus secure processing of foreign database files is possible. The Data tag simply prints a WWW form and calls the second object to include the names of the user-selected formulae.

```
# example inputs: _%Formula=NUM or simply NUM

_%Description=cgi
 _%PreProc=@@code; open(fhlocal, '/home/setheo/DATABASE/bin/db.test/db.mo');
           @DB_public=<fhlocal>; close fhlocal;
           $pat=$db_cgi::args{'SEARCH'}; $pat=&m5_screenregexp($pat);
           $pat='.' if (!($pat));
 _%Data=<HTML>
@@nop;
<BODY>
@@nop;
<FORM METHOD="GET" ACTION="@@inc$act;">
Select objects with (restricted) REGEXP:
<INPUT NAME="search" VALUE="@@inc$pat;" SIZE=50><BR><BR>
<INPUT TYPE="submit" VALUE="  Search  ">   <INPUT TYPE="reset" VALUE="  Reset  ">
</FORM><p>
@@nop;
List of Formula tags of objects matching @@inc$pat; in database:
<UL>
@@grp*#Description,'Loop1b$',DB;
</UL><p>
@@nop;
</BODY></HTML>
```

The second object is a simple loop that returns the Formula name of each data object selected by the LoopSelect tag. As the currently active object is available in the `%object` hash, inclusion of the variable `$object{Formula}` is sufficient to return the value of the active object´s Formula tag. The final "nop" command simply protects the linefeed after the HTML list element tag. Otherwise, this linefeed might be misinterpreted by Perl to mean the end of the current object (see Section 3.2.1).

```
_%Description=Loop1b
 _%LoopSelect='/@@inc$pat;/','','DB_public'
 _%Data=      <LI>@@inc$object{Formula};
@@nop;
```

In order to invoke *db_m5.p* with these 0 objects from the http daemon, a small wrapper is sufficient:

```
#!/usr/local/dist/bin/perl

$0=~/([^\/]+)$/; $script=$1;
$path = "/home/setheo/DATABASE/bin/db.doc/html";
$binpath= "/home/setheo/DATABASE/bin";
$act  = "http:/cgi-bin/user-cgi/jakobi/$script";
system("cd $path ;
        $binpath/db_m5.p -m -s Description cgi -e '\$act=\"$act\"' m5demo.txt");
exit 0;
```

The output of asking for all formulae of *TPTP* domain *SYN* in the example database, as seen by the *Lynx* browser:

```
Select objects with (restricted) REGEXP:
  -------------------------------------------------

  SYN_______________

  List of Formula tags of objects matching SYN in database:
    * SYN127-1
    * SYN128-1
    * SYN129-1
    * SYN130-1
    * SYN132-1
```

Another example is `db_m5.p -m db.test/m5.demo`: A *db_m5.p* database file splitting a *Setheodb* database into a set of HTML files.

db_m5.p is a powerful extraction tool, however it is suitable only for fairly small databases, as it is inefficient in both space and time. If you want to trade setup time for larger throughput, have a look at COHTML [5] or PHP [15].

3.4 Other scripts

This sections offers a short overview of the more interesting of the remaining scripts and example queries. Invoke the scripts with option `-h` or have a look at the comments in the source for further information.

- *db_ed.p* - reads a database completely into memory and allows interactive editing of selected objects in multi-line mode. An example is given in section 3.5.3.
- *db_fix.p* - database cleanup and testing.

  Multi-line databases are supported by simply changing Perl´s input record separator. As a result, adding a blank to an empty line can already accidentally glue two objects together: An usage example that prints a warning for every possible instance of this "feature": `db_fix.p -w mdb`
- *dbs_form.p* - an example of formatting a *Setheodb* database for human-only readable output.
- *db_split.p* - splits a database of single-line objects into many database files. The name of the database for each object is computed by a user-supplied Perl expression. When using the -lowmem option, `stdin` cannot be used for input of more than `$jobsize` objects. An example is given in section 3.5.3 and in the test suite *db_test/README*.
- *dbs_q** - some predefined queries for the *Setheodb* format.

3.5 Example: *Setheodb* - a Database Format for SETHEO Runs

3.5.1 Application/Background: The Model Elimination Prover *SETHEO*

Model Elimination, as described by Loveland [13], can be seen as a special kind of Tableau Calculus (see the book of Smullyan [16]) that works directly with the clausal form of a formula. Furthermore, the duplication of partial proofs used repeatedly in the tableau can be prevented using some kind of *factorization* (lemma generation) [11]. In general, Model Elimination is a goal oriented top down procedure.

The output of *SETHEO* is a PROLOG list (in a file with the suffix *.tree*) describing the proof tree with references to the clauses involved in the inferences. The most important statistical information, like success, number of inferences, proof depth and proof time, can be extracted from the log file.

Above, we have described the output of *SETHEO* in such an extended way to demonstrate a generic view of the output of theorem provers. If the user wants to adapt *DBFW* to his own theorem prover, only some patterns of the extractor have to be adapted, but the generated information and the tags used will nearly be the same. So only a few changes in the framework are necessary.

3.5.2 *SETHEO* Data Files

This section shows example files for the *TPTP* file `MSC006-1.lop`. The commands used for this example are:

- `inwasm -foldup -cons MSC006-1` # preprocessing/compiling with constraints
- `sam -cons -dr MSC006-1` # proving with constraints and iterative deepening

1. ***Lop*** **Formula** `MSC006-1.lop` **(Input)**

```
#--------------------------------------------------------------------------
# File     : MSC006=NonObv-1 : TPTP v1.2.0. Released v1.0.0.
# Domain   : Miscellaneous
# Problem  : A "non-obvious" problem
# Version  :
# English  : Suppose there are two relations, P and Q. P is transitive,
#             and Q is both transitive and symmetric.
#             Suppose further the "squareness" of P and Q: any two things
#             are related either in the P manner or the Q manner. Prove
#             that either P is total or Q is total.

# Refs     : Pelletier F.J., and Rudnicki P. (1986), Non-Obviousness,
#             In Wos L. (Ed.), Association for Automated Reasoning
#             Newsletter (6), Association for Automated Reasoning, Argonne,
#             Il, 4-5.
# Source   : [Pelletier & Rudnicki, 1986]
# Names    : nonob.lop [SETHEO]

# Status   : unsatisfiable
# Syntax   : Number of clauses    :    6 (   1 non-Horn;   2 unit;   5 RR)
#             Number of literals   :   12 (   0 equality)
#             Maximal clause size  :    3
#             Number of predicates :    2 (   0 propositional; 2-2 arity)
#             Number of functors   :    4 (   4 constant; 0-0 arity)
#             Number of variables  :   10 (   0 singleton)
#             Maximal term depth   :    1

# Comments : Rudnicki says "I think that what you call the non-obvious
#             problem from our write-up with Jeff should be attributed
#             to J. \Lo\'{s} (in LaTeX)." and "J. \Lo\'{s}  is in LaTeX,
#             and it is the name of my Polish prof that told me that.
#             English approximation of his name can be typed as J. Los.".
#          : tptp2X: -fsetheo:sign MSC006-1.p
#--------------------------------------------------------------------------
# p_transitivity, hypothesis.
p(X, Z) <-
    p(X, Y),
    p(Y, Z).

# q_transitivity, hypothesis.
q(X, Z) <-
    q(X, Y),
    q(Y, Z).

# q_symmetry, hypothesis.
q(Y, X) <-
    q(X, Y).

# all_related, hypothesis.
p(X, Y);
q(X, Y) <- .

```

```
# p_is_not_total, hypothesis.
 <- p(a, b).

# prove_q_is_total, conjecture.
 <- q(c, d).

#---------------------------------------------------------------------------
```

2. stdout/stderr Output of *inwasm*

```
inwasm V4.0 [wasm-less] Copyright TU Munich (June 96)
command line: /home/setheo/bin.solaris/inwasm -cons -foldup MSC006-1
codegen: 116 [19997] labels, 0 colls [343 accesses]
Assembler optimization: 67 labels read, 25 labels output
MSC006-1.hex generated in  0.04 seconds
Parsing input-file.
Preprocessing: Generating weak-unification.
Preprocessing: purity
      Message: Deleted clauses : None.
      Message: Making sortarray.
Preprocessing: orbranch reordering
Preprocessing: inserting tautology constraints
      Message: 5 tautology-constraints generated.
Preprocessing: inserting subsumption constraints
      Message: 2 subsumption-constraints generated.
Preprocessing: removing redundant constraints
      Message: 5 constraints deleted.
Preprocessing: fanning
Preprocessing: inserting reduction steps
Preprocessing: inserting symmetry constraints
      Message: 2 symmetry-constraints generated.
Preprocessing: removing redundant constraints
      Message: 4 constraints deleted.
Preprocessing: subgoal reordering
Codegeneration.
```

3. *sam .log* File MSC006-1.log

```
SAM V3.3 Copyright TU Munich (December 22, 1995)

Options : -cons -dr MSC006-1

using antilemma-constraints
using regularity-constraints
using tautology-constraints
using subsumption-constraints

Start proving...

-d:    2  time     < 0.01 sec  inferences =       9  fails =       7
-d:    3  time     < 0.01 sec  inferences =      30  fails =      23
-d:    4  time =     0.02 sec  inferences =     100  fails =      79
-d:    5  time =     0.04 sec  inferences =     365  fails =     291
-d:    6  time =     0.12 sec  inferences =    1465  fails =    1244
-d:    7  time =     0.33 sec  inferences =    5505  fails =    4916

                    ******** SUCCESS ************

Number of inferences in proof      :       20
        - E/R/F/L                  :       17/       2/       1/       0
Intermediate free variables        :        5
Intermediate inferences            :       26
Intermediate open subgoals         :        6
Generated antilemmata              :       46
Number of unifications             :     7474
    - E/R/F/L                      :     3452/    2665/    1357/       0
Number of generated constraints    :     3877
    - anl/reg/ts                   :       99/    1612/    2166
```

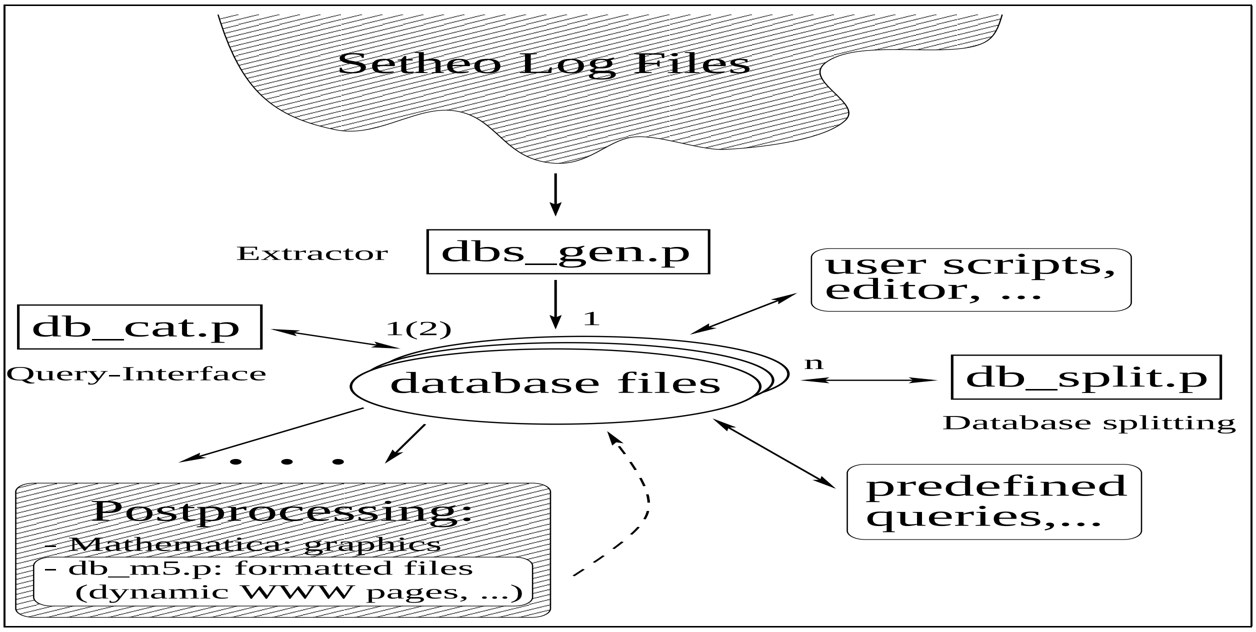

Figure 3.4: The software-IC compatibility map

```
Number of fails                  :     6560
    - unification                :     2078
    - depth bound                :     1366
    - constraints                :     3116
        - anl/reg/ts             :       99/     485/    2532
Number of folding operations     :      492
    - one level                  :       81
    - root                       :      115
Instructions executed            :    24466
Abstract machine time (seconds)  :     0.53
Overall time          (seconds)  :     0.58
```

4. *.tree* **File** `MSC006-1.tree`[5]

```
[
[~query__,[ 0 , ext__(0.1,5.1) ] ,[
  [ query__ ] ,
  [~p(a,b),[ 1 , ext__(5.2,1.1) ] ,[
    [ p(a,b) ] ,
    [~p(a,d),[ 2 , ext__(1.2,1.1) ] ,[
      [ p(a,d) ] ,
      [~p(a,c),[ 3 , ext__(1.2,4.1) ] ,[
        [ p(a,c) ] ,
        [q(a,c),[ 4 , ext__(4.2,3.2) ] ,[
          [ ~q(a,c) ] ,
          [q(c,a),[ 5 , ext__(3.1,2.2) ] ,[
            [ ~q(c,a) ] ,
            [~q(a,d),[ 6 , ext__(2.3,4.2) ] ,[
              [ q(a,d) ] ,
              [p(a,d), [ 7 , red__(4.1,2 ) ]] ] ],
            [q(c,d),[ 8 , ext__(2.1,6.2) ] ,[
              [ ~q(c,d) ] ] ] ] ] ] ] ] ],
      [~p(c,d),[ 9 , ext__(1.3,4.1) ] ,[
        [ p(c,d) ] ,
        [q(c,d),[ 10 , ext__(4.2,6.2) ] ,[
          [ ~q(c,d) ] ] ] ] ] ] ],
    [~p(d,b),[ 11 , ext__(1.3,1.1) ] ,[
      [ p(d,b) ] ,
      [~p(d,c),[ 12 , ext__(1.2,4.1) ] ,[
```

```
          [ p(d,c) ] ,
          [q(d,c),[ 13 , ext__(4.2,3.2) ] ,[
            [ ~q(d,c) ] ,
            [q(c,d),[ 14 , ext__(3.1,6.2) ] ,[
              [ ~q(c,d) ] ] ] ] ] ] ] ],
        [~p(c,b),[ 15 , ext__(1.3,4.1) ] ,[
          [ p(c,b) ] ,
          [q(c,b),[ 16 , ext__(4.2,3.2) ] ,[
            [ ~q(c,b) ] ,
            [q(b,c),[ 17 , ext__(3.1,2.3) ] ,[
              [ ~q(b,c) ] ,
              [~q(d,b),[ 18 , ext__(2.2,4.2) ] ,[
                [ q(d,b) ] ,
                [p(d,b), [ 19 , red__(4.1,11 ) ]] ] ],
              [q(d,c), [ 20 , fac__(2.1,13 ) ]] ] ] ] ] ] ] ] ] ] ] ] ]
    ].
```

3.5.3 DBFW at Work: Software-IC Compatibility Map

The "execution" graph in Figure 3.4 shows most possible filter combinations for the framework. Filters may be connected by a pair of directional arrows that go through an intermediate database file. The numbers at the arrows denote the number of new database files generated by a filter[6].

An example for combining the filters:

```
   cd db.test/log
1. find . -name \*.log\* -print | dbs_gen.p > db1
2. db_cat.p -s Formula SYN db1 -d db2 > db3
3. db_ed.p -i db3 db4
      s Formula SYN127
      x
4. wc db4
5. db_split.p -l -i 's/^.*_%Formula=(\S+).*$/$1/' db4 | wc
```

Here we assume that log files have the suffix *.log*. The first line generates a database file *db1* for the given log files. The next line splits this database into *db2* and *db3*. The later database file contains the objects of the TPTP domain *SYN*. The third line interactively edits (*db_ed.p*) all objects for formula *SYN127* and writes the changed database back to *db4*. Using a standard UNIX command, line 4 returns the number of objects in *db4*. The number of lines is identical to the number of objects, as the database generated above contains exactly one object per line, no comments, and no empty lines. The final line splits *db4* into individual databases for each formula and lists the names of the different formulae, which are again counted by the UNIX command *wc*. Note that several objects can occur for a single formula and that no unique object ID has been defined within *Setheodb*.

This example demonstrates some of the scripts and their interaction with each other using database files as a common data format. The integration of the framework into the standard UNIX environment is also covered. The example is included in the distribution.

3.5.4 Tag Description

The *Setheodb* format provides tags to describe

- the environment, date and owner _%User of the run,
- the command line parameters and other data on the *SETHEO* subprograms (*inwasm, wasm, sam*),
- the name of the proven formula, and
- the proof itself.

[5] Whitespace before closing square brackets has been compressed.

[6] For the moment, there are no filters implemented that require multiple input files. However, the UNIX commands *sort* and *comm* can be used for testing multiple database files on common objects.

TAGID	CONTENT
_%Formula=	formula name (*TPTP*-style name)
_%Env=	host, datestamp of the log file, ...
_%Comment=	comment
_%User=	owner of the log file
_%Errors=	list of detected error strings
_%IVersion=	*inwasm* (parser) version
_%IPar=	*inwasm* parameter
_%WVersion=	*wasm* (compiler) version
_%WPar=	*wasm* parameter
_%SVersion=	*sam* (inference machine) version
_%SPar=	*sam* parameter
_%Time=	runtime (cpu-relative)
_%Inf=	inferences (all and sorted by types)
_%Unif=	unifications
_%Fail=	fails
_%IterDepth=	greatest depth (in the proof tree) visited (-d)
_%IterInf=	greatest number of inferences made (-i)
_%IterLocInf=	combination bound, local inferences (-loci)
_%TVersion=	*TPTP* version
_%FormState=	state (if TPTP)
_%FormCl=	number of clauses
_%FormLit=	number of literals
_%Result=	result (if tag defined: exactly one of TIMEFAIL—TOTALFAIL—BOUNDFAIL—SUCCESS—ERROR, otherwise *sam* has had trouble or was not called)
_%Log=	name of the object´s log file

Figure 3.5: List of standard tags of the *Setheodb* format

Additional tags may be defined as necessary for each object (maintenance, version control, semi-private information for the user, ...). Tag IDs are extended into strings matching a REGEXP (`_%\S+=`) that by definition may not occur within tag values. Figure 3.5 shows the current list of tags for the *Setheodb*-format. Tags may also be omitted[7].

Example for an object:

```
_%Formula=COL013-1 _%Result=ERROR _%SVersion= _%Env= sj22, 30. Februar 95
```

Feel free to add new tags to this list. The database scripts are mostly independent of the tag IDs used. Unique object ID tags are not used at the moment.

The database file consists of comments (lines starting with '#' in column 1), empty lines and objects. Objects are structured into attributes or tags: `_%<ID>=<VALUE>`. Each tag can contain arbitrary text (except text matching the Perl REGEXP `_%\S+=`). To allow use of simple regular expressions in queries, please use the tags within your objects in the order given above[8].

See `db_cat.p -h setheo` and the patterns in *dbs_setheodb.pat* for further information.

3.5.5 A Few Lines of the Database

In this example, we will generate a database from the files *db.doc/example.log, db.doc/example1.log*[9]:

```
echo example.log ; echo example1.log ) | dbs_gen.p | db_cat.p -mo .
```

The `-mo` switch of *db_cat* adds a line-feed after each tag to allow easier editing of database files using standard text editors. Normally all tags of an object would be concatenated into a single line. The final dot of the

[7] A missing tag in itself may be information: An object without _%Result is generated for invalid log files, anything from a core-dump-used-as-a-log-file to abnormally aborted *SETHEO*-runs.

[8] This order should be identical to the one used for the extractor patterns defined in *dbs_setheodb.pat*.

[9] These example log files are cut down *SETHEO* logs, modified to give a human readable hostname that is suitable for publication. They also demonstrate the setting of additional tags from within log files (`DUMMY-TAG`). The example files are contained in the distribution of *DBFW* [8].

db_cat.p command is necessary: it is a trivial REGEXP selecting all objects in the database. The command above prints these two (multi-line) objects:

```
_%Formula=SET001-2
 _%Env=This_run's_hostname 960705140514
 _%User=jakobi
 _%CountLop=1
 _%IVersion=3.2
 _%IPar=-cons -linksubs
 _%Log=example.log
 _%DUMMY-TAG=DUMMY-VALUE

_%Formula=SET001-3
 _%Env=This_run's_hostname 960705140438
 _%User=jakobi
 _%CountLop=1
 _%IVersion=3.2
 _%IPar=-cons -linksubs
 _%Log=example1.log
 _%DUMMY-TAG=DUMMY-VALUE
```

4. Porting and Maintenance

This sections offers information and hints to future users of the framework.

Porting of the non-*Setheodb* scripts should require changing only some executable paths, most notably the path of Perl 5 binary (look for `/home` or `/usr` strings in the scripts). The framework has been tested on HP-UX, Linux and Solaris 2.

4.1 Copyright, Warranty

The framework is (C) TU München. It is provided under the terms of the *GNU General Public License*, a copy is provided in the distribution *db.doc/LICENSE*. For the copyright of the WWW-Server demo based on the network security scanner *Satan*, please see file *db.www/copyright.html*. Additional release notes can be found in *db.doc/README*.

This means - among other things - that there is no warranty on the product and that we will accept no liabilities.

However, consider this:

- You backup your database files regularly (**don´t you!?**).
- You can use the text editor of your choice to access the data in the database files.
- You can use standard UNIX tools like grep to access your database.
- With a few lines of Perl, you can convert your database into any format you desire, for example. an input format for a newly bought relational database. Similarly, you can use *db_m5.p* to preprocess the export format of an RDBMS.

4.2 Support / How to Get

Though there is no formal support. The authors are interested in improving the framework and in coordinating efforts of users to port, improve or integrate the the framework in other packages. Please send an eMail to one of the authors or to `setheo@informatik.tu-muenchen.de`. Thank you.

ILF: For further information on *ILF* contact `dahn@mathematik.hu-berlin.de`.

SETHEO homepage:

- `http://wwwjessen.informatik.tu-muenchen.de/ setheo/`
- `ftp://ftp.informatik.tu-muenchen.de/local/lehrstuhl/jessen/Automated_Reasoning/SETHEO/`

Framework homepage:

- `http://wwwjessen.informatik.tu-muenchen.de/~setheo/database_framework/`
- `ftp://ftp.informatik.tu-muenchen.de/local/lehrstuhl/jessen/Automated_Reasoning/SETHEO/database/`

4.3 Files and Documents

The files to install can be found in directory . (or *bin*). Copy all files and links to a directory and add it to your $PATH variable. The sub-directories are optional.

An overview of the files, directories and naming conventions follows in the Figures 4.1 and 4.2.

File/Dir	Description
db.doc/	various documents and readme files, including *Setheodb*-specific documents in German: • *admin.txt* (adding new objects and maintaining the *Setheodb* database), • *db.ger.html* plus examples (a short introduction talk from early in 1996, see HTML source of the talk for additional notes; provided as is). **This directory also contains the input data for most of the examples included in this paper.**
db.test/	a small test suite for basic functionality tests - read and run the *README* file.
db.tool/	support programs like countlop. These tools are not part of the framework itself.
db.www/	a WWW interface example (unsupported; call *html.pl* from within the directory)
*db_**	framework scripts and data files, format independent
*dbs_**	*Setheodb*-specific scripts and data files (mostly intended as examples ; *dbs_q** - some query examples)
*dbi_**	*ILF*-specific scripts and data files
*db*_*.frm*	format description (for display by *db_cat.p*)
*db*_*.pat*	extraction pattern for specific formats (for *db_gen.p*, suitably symlinked or explicitely requested)
*db*_*.tpl*	new entry template (for *db_ed.p*, suitably symlinked or explicitely requested)

Figure 4.1: Files in the installation directory tree

Script/Packages	Description
db_cat.p	query-interface, also a Perl package
db_cgi.p	cgi support functions, to be used as a Perl package
db_ed.p	allows editing selected objects
db_fix.p	cleanup of database files, object delimeters, etc
dbs_form.p	prints the objects of a *Setheodb* database in a more structured way
db_gen.p	automated extractor script (for *Setheodb* symlink to *db_gen.p*)
db_m5.p	automated document generation using a description contained in a database, also a Perl package (cgi wrapper: *dbs_m5cgi.p*)
db_split.p	splits a database into several database files according to a Perl expression

Figure 4.2: Scripts and Perl packages

4.4 Requirements

- a UNIX-like enviroment
- *Perl5*. *Perl5.001m* or higher is required for correct globbing in the WWW-example. After replacing `quotemeta`, namespace commands and REGEXP-look-ahead-constructs ("`(?X...)`"), the scripts should also run under *Perl4.036*. The scripts are tested under Solaris 2, HpUX and Linux. The scripts do not require UNIX specifics, so the framework itself should be able to run under a DOS port of Perl with few modifications. It may be necessary to disable the low memory hacks in some scripts that uses the `exec` call to split the processing of large databases in palatable chunks.

  Please mail your modifications to the authors! Thank you.

4.5 Porting

Most of the *dbs_** / *db.www* scripts are offered as example, as they are useful only in *Setheodb* context. Porting of the *db_** scripts should only require changing of some executable paths, most notably the Perl binary (look for `/home` or `/usr` strings).

Some scripts contain code to cope with large database files, sometimes using an internal variable `$jobsize` for defining the size of the chunk to operate on (usually 500 objects) [1]. *db_m5.p* is requires a port of Perl with extremly stable memory handling.

4.6 Designing a Database Format

Designing a new format is a straight forward process: write a list of tag IDs with their semantics and tell your users about this list. The list can be easily extended: simply add some new tags to some existing objects.

In case you want to implement an automatic extractor/conversion script, you should invest some time in designing your format. Tag IDs should be expressive, and the object structure should allow for easy conversion to or from other formats. If other users or scripts use your databases, your format should allow for easy processing of the information. Be certain to catch all relevant information in your database, especially if the original log files and other data are available only temporarily.

If you need relational operations like *Joins*, you should consider the use of the framework as an intermediate data-extraction step before storing the data into a final database.

To generate a new format, a user has to create (and edit) several resource files:

- a template file for creating the tags (suffix *.tpl*, optional, supported by *db_ed.p*)
- a file containing extraction patterns (suffix *.pat*, required for automatic extraction, used by *db_gen.p*)
- a format description (suffix *.frm*, optional, used by the online help facility of *db_cat.p*)

db_ed.p and *db_gen.p* load the resource file requested by the `-f` switch or autoload a resource files by changing the suffix of their invocation name. So *Setheodb´s dbs_gen.p* is simply a symbolic link to *db_gen.p* that configures the extractor for the *Setheodb* format. If you want to use the extractor, modifying the extraction patterns in a copy of *dbs_gen.pat* is going to be your main task in adapting the framework to a new environment.

If you really need to use different `$pattagstart` and `$patdatastart`[2] for various formats, you can define these variables differently for each format in an `if-elsif`-cascade depending on its basename.

4.6.1 The Extractor

db_gen.p uses extraction patterns (Perl code blocks) to extract the values of object attributes from the specified text files. The Perl code blocks implement your heuristics to extract the information you want. This may include side-effects for other patterns like truncating the input data to scan, etc. All extraction patterns are always run, even if the return value will be overridden by command line arguments later on.

See *dbs_gen.p* and *dbs_gen.pat* for examples.

Pseudocode of the extactor main loop generating the object:

- The current line of input is the name(s) of the input file(s) for parsing - read the file(s) into `$log` and `$olog`.
- Gather embedded tags into `%logtags`. Embedded tags can be overridden by option T.
- Gather command line arguments (defaults and overrides) and embedded tags in the hahses `%override` and `%defaults`.
- Run all extraction patterns, possibly using the corresponding values from `%override` or `%defaults`. Delete corresponding entries in the hashes `%override` and `%defaults`.

[1] In part, this was made necessary by the fact that Perl leaks memory when doing many evals. This occurs for versions from Perl 4 ranging upto at least 5.001m.

[2] These variables are used to extend the tag ID into a unique string. They are defined in *db_cat.p* and *db_gen.p*.

- Add remaining entries of `%override` and `%defaults` to the object.

When writing the extraction patterns, you should be careful to allow and watch for binary data such as core dumps that got somehow included into the set of input files to extract... The main loop currently restricts all tags to less than 128 characters, marking all binary characters as '?'.

To add a new set of extractor patterns in a file called *YourFormat.pat*, simply add a symlink with the same basename *YourFormat.p* to *db_gen.p*. The extractor automatically looks for its configuration in the file with its invocation name and a suffix of *.pat*. Alternatively, you can use the `-f` option to explicitely request a specific pattern file.

4.6.2 The Query-Interface as a Package

db_cat.p can be used as a package to process a database that has been read into a Perl array. The interface consists of the `@db_cat::db` array containing the input database. The array `@db_cat::db` and `@db_cat::del` (optional) are the output databases. `$db_cat::project` and `@db_cat::pattern` contain the patterns for *Projection* and *Selection*. If you wish to change the patterns, you have to reset `undefine $db_cat::main` first: The patterns are compiled into a dynamically defined subroutine, to reduce memory leakage during numerous `eval` calls.

For more information, see the

- variable definition section at the beginning of *db_cat.p* (package flags, etc.).
- usage example at the end of *db_cat.p*. A larger example is the script for interactive editing, *db_ed.p*.

Other packages within the framework are the document generator *db_m5.p* and *db_cgi.p*, which contains some support functions for cgi usage on the WWW. See the source of these scripts for more information.

4.6.3 Testing

The file *db.test/README* documents and implements a small testsuite.

4.7 Database Maintenance

Maintenance heavily depends on your use of the framework. If you simply use a modified extractor to obtain and massage the data for your database, maintenance is restricted to keep the extractor talking to the database in a language the database accepts.

If you don´t need the full relational model and the object metapher of the framework is sufficient, you can use the framework´s database files for permanent storage, using *db_cat.p* or your texteditor to access your objects.

Adding objects is easy: `cat dbnew >> db`.

Removing objects is only slightly more complex:

```
db_cat.p -s Formula SET db > dbnew
mv db dbold
mv dbnew db
```

This eliminates all objects with an attribute _%Formula containing the string "set" (case does not matter for option -s).

The challenge is what we like to call "aging" your objects. For *Setheodb* this means the regular removing of objects from old *SETHEO* versions. The file *db.doc/admin.txt* offers a commented maintenance session in German. Assigning object owners may help to identify obsolete objects. Enlightening your users to use embedded tags to better document their objects-to-be in the log files for the extractor may also help (intentions for this *SETHEO* run, etc). Maintenance in our case proves to be mostly a problem of defining and applying a processing policy for the contents of the database.

4.8 Finally: Bugs

Certainly. None known for the framework itself, excepting the number of changes per month per script, which is still quite high.

A word of warning about the multi-line object mode: The mode relies on the objects being separated by whitespace containing at least two linefeeds without any characters inbetween. It is very easy to accidentally glue several objects together when editing the database manually. Do not rely on this "feature" to put several text paragraphs into a tag - use some special character to distinguish the paragraph break from an empty line and object delimeter[3]. *db_fix.p* tries to fix any "empty" lines by removing all blanks and tabs. *db_ed.p* interprets arbitrary whitespace lines in objects returned from the editor as object delimeters. GNU *cat*´s `-A` option can be used in conjunction with *grep* to display lines containig questionable whitespace.

[3] *db_cat.p* also replaces the sequence `@@\n` by a linefeed in reading or writing an object.

5. Available Database Formats and Patterns

This section describes all available extractor patterns and database formats. Information on *ILF*-variants of the patterns and formats can be found in Section 6.3. See Section 3.5 for the *Setheodb* format.

For now, the database patterns are independent of each other and do not try to translate prover specific output into a generic format.

5.1 *Otter*: *dbx_otter.**

The *Otter* extractor patterns are based on version 3.0.4 (August 1995). The following description of *Otter*[14] is quoted from the version 3 online manual:

> *Otter* (Organized Techniques for Theorem-proving and Effective Research) is a resolution-style theorem-proving program for first-order logic with equality. *Otter* includes the inference rules binary resolution, hyperresolution, UR-resolution, and binary paramodulation. Some of its other abilities and features are conversion from first-order formulas to clauses, forward and back subsumption, factoring, weighting, answer literals, term ordering, forward and back demodulation, evaluable functions and predicates, and Knuth-Bendix completion. *Otter* is coded in C, is free, and is portable to many different kinds of computer.

The example below uses `tba_gg.in` as input, which can be found in the *Otter* testsuite.

```
otter < tba_gg.in > tba_gg.out 2>&1
echo "tba_gg.out" | db_gen.p -f dbx_otter.pat
```

```
_%Formula=tba_gg
  _%Env=root on kefk, Sun Mar  9 14:12:48 1997
  _%Version=Otter 3.0.4, August 1995
  _%Par=PAR1
  _%Time=0 (0 hr, 0 min, 0 sec wallclock)
  _%Length=8
  _%Level=6
  _%Cl_Stats=18 379 165
  _%Result=SUCCESS
```

5.2 *Discount*: *dbx_discount.**

The *Discount* patterns are based on version 2.1L, as maintained by Stephan Schulz. He describes *Discount*[7] as follows:

TAGID	CONTENT
_%Formula=	formula name
_%Env=	host, datestamp of the log file, ...
_%Comment=	comment
_%Errors=	list of detected error strings
_%Version=	version
_%Par=	parameter
_%Time=	runtime
_%Length=	length of proof
_%Cl_Stats=	various statistics
_%Result=	result (if tag defined: exactly SUCCESS

Figure 5.1: List of standard tags of the *Otter* format

TAGID	CONTENT
_%Formula=	formula name
_%Env=	host, datestamp of the log file, ...
_%Comment=	comment
_%Errors=	list of detected error strings
_%Version=	version
_%Par=	parameter
_%Time=	runtime
_%Length=	proof steps (discount facts)
_%Cl_Stats=	various statistics
_%Result=	result (if tag defined: exactly SUCCESS)

Figure 5.2: List of standard tags of the *Discount* format

The *Discount* system is a distributed equational theorem prover based on the teamwork method for knowledge-based distribution. It uses an extended version of unfailing Knuth-Bendix completion that is able to deal with arbitrarily quantified goals. *Discount* features many different control strategies that cooperate using the teamwork approach. Competition between multiple strategies, combined with reactive planning, results in an adaptation of the whole system to given problems, and thus in a very high degree of independence from user interaction.

Discount-2.1L branched off the main *Discount* line after version 2.0 and concentrated on implementing learning strategies suitable for the use in sequential mode. It is maintained by Stephan Schulz, <schulz@informatik.tu-muenchen.de>. Important features of *Discount-2.1L* are occasionally ported back to the main *Discount* line (now version 3.0).

Example:

```
# input file dsc_BOO001_1
discount dsc_BOO001_1 > dsc_BOO001_1.log
mextract  -n -n2 -s 2> dsc_BOO001_1.mextract_statistics
echo "dsc_BOO001_1.log" | db_gen.p -f dbx_discount.pat
```

The above command sequence generates the following object:

```
_%Formula=dsc_BOO001_1
 _%Ordering=XKBO inverse : 1 > multiply : 1 > a : 1
 _%Time=0.025 s / 0.000 s (real)
 _%Length=16 written; (read: 356 / first extraction: 16 )
 _%Cl_Stats=13 Regeln 1 Gleichungen 74 kritische Paare 0 kritische Ziele
            186 Reduktionen
 _%Result=SUCCESS
```

6. Other Applications

The file *db.doc/db.ger.html* contains a short talk (in German) on the framework, including the www interface and the *Mathematica* example. The HTML code of the talk contains additional notes to the "slides".

6.1 A Partial WWW Interface

A simple WWW interface demonstration is provided in the directory *db.www*. The script *html.pl* starts a small, personal Perl www server based on *Satan* (the network security scanner) and can be used with any browser. The server provides a restricted interface to some of the framework´s scripts. The example is unsupported and provided as-is.

Due to some queries´ memory requirements, it is useful to start a new server / browser pair for each user on the host (s)he is using, with this user´s priviledges.

The demonstration offers only a subset of all possible execution paths through the framework for *Setheodb* database files: for example, *db_cat.p* is called at most once.

More information is available in

- *db.www*, see also file *db.www/copyright.html*
- *db.doc/form.ps*
- *db.doc/result.ps*

Bugs: The Perl version must be at least 5.001m for successful globbing of ~USER-relative filenames.

Submitting the form in Figure 6.1 corresponds to these commands:

```
db_cat.p -s Formula SET /home/setheo/DATABASE/DB/db.setheo |
   dbs_mathquery1.p
```

6.1.1 Another WWW-Interface

The framework is also suitable for use as a low-cost (read no-cost) database within a cgi-bin script for normal UNIX-based websites. See the section on the document generator and the example cgi wrapper *dbs_m5cgi.p* and database *db.test/m5.demo*.

6.2 Complex Queries and Graphical Post-Processing

More complex queries can be written in Perl using the query interface as a Perl package. For example, a data file returned by such a query can be read by *Mathematica* to generate a graphical answer (Figure 6.2). The script *dbs_mathquery1.p* in combination with the *Mathematica* notebook (*db.doc/math.gfx1.mb*) is an example for a fairly complex query returning a graphic display as answer.

Bugs: The notebook is a first attempt at graphical postprocessing using *Mathematica*. It is usable and can serve as a rough guide, however it´s no polished template for your applications. It is also a part of the German talk in *db.doc/db.ger.html*.

More information is available in

- *dbs_mathquery1.p*
- *db.doc/math.gfx1.mb*

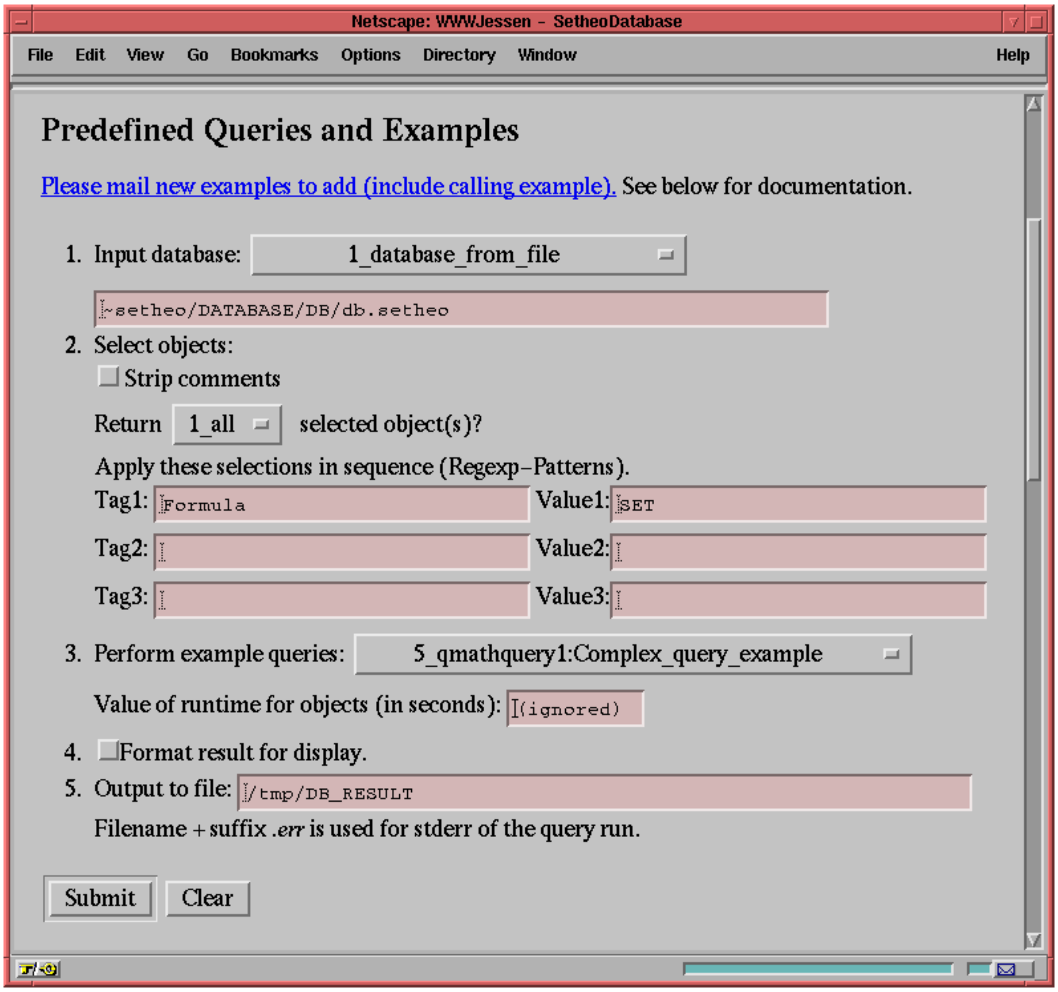

Figure 6.1: A request to the server

- *db.doc/math.gfx1.ma.real.ps*

Figure 6.2 shows the results of certain *SETHEO* runs depending on run time and *TPTP* problem domain. It is the result of applying the *Mathematica* notebook to the datafile returned by the form above. For the graphic shown here, you have to select all relevant runs from the database, for example by changing the contents of the field `value1` to ".".

6.3 Integrated Logical Functions (*ILF*)

We are currently integrating the framework into *ILF (Integrated Logical Functions [6])*, a front-end for several automated theorem provers. The framework is used within *ILF* to collect statistics on the success of the provers and their tactics. Beginning with *SETHEO*, we are adapting *DBFW* to cooperate with other provers integrated in *ILF*, such as *DISCOUNT* [2], *KoMeT* [4], *OTTER* [14] and *SPASS* [20].

6.3.1 Application/Background: ILF

We wanted to use the database tools to get some information about the relevant use of *SETHEO* in "real" contexts, as they occur while supporting mathematicians proving theorems or computer scientists verifying communication protocols. Therefore we integrated it into the system *ILF* [6] developed at the Humboldt-University at Berlin. *ILF* is a system that integrates automated theorem provers, proof tactics for interactive

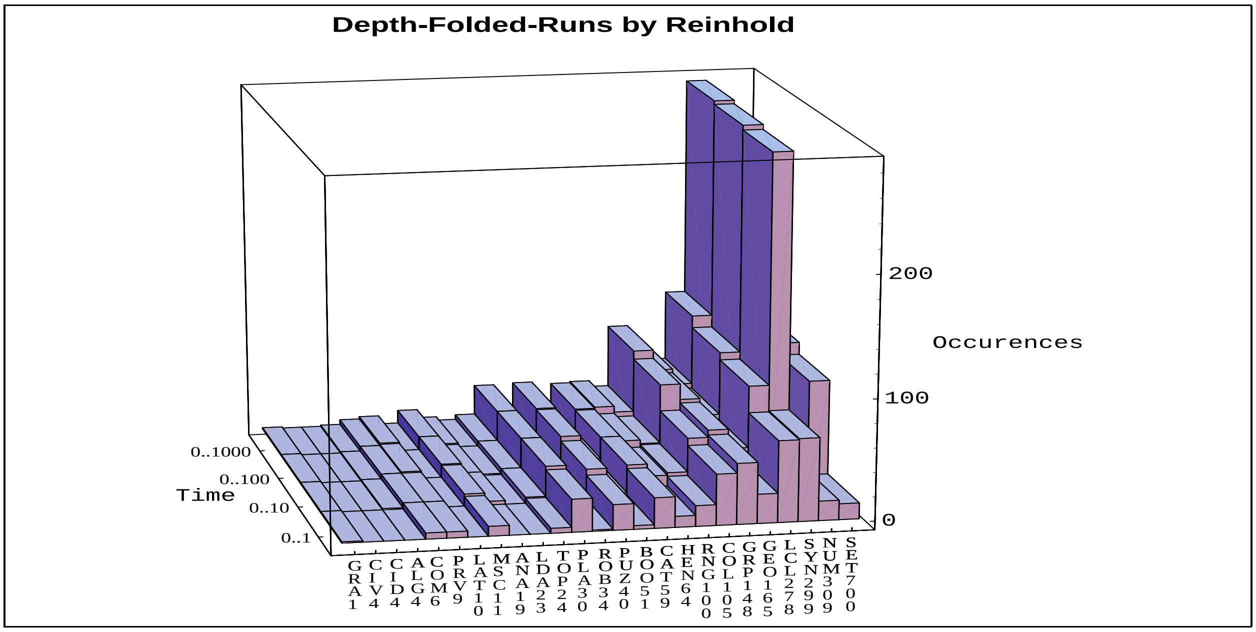

Figure 6.2: Graphical processing of query results using *Mathematica*

deductive systems and models within a graphical user interface.

Research in the field of theorem proving in many groups in several countries has created a lot of sophisticated tools including

- automated theorem provers for various logical calculi,
- rewrite systems,
- proof tactics,
- model finders and
- domain specific methods.

ILF is a tool that can be configured in many ways to *I*ntegrate all these *L*ogical *F*unctions. The common feature of these tools that is used for this integration is that they all can be used to modify a knowledge base.

ILF is applied on two different levels. It yields methods of testing the power of tools to support logically correct arguments in a specific field. Several ways to combine these tools in proof tactics can be tested rapidly. When a collection of useful proof tactics has been obtained, it can be encapsulated as a set of "rules of inference“ in a new interactive or automated deductive system. It is also possible to extend an existing system in this way. This new, more powerful system can be tailored to meet exactly the needs of an end user, making available just those procedures that his kind of problems demand.

Perhaps the most challenging feature of *ILF* is its modularity. The power of *ILF* can be easily extended by integrating further systems and developing libraries of domain specific proof tactics. In fact, for an experienced PROLOG programmer, it is a matter of a few days to integrate a new system that has been developed somewhere else independently.

Within the DFG-Schwerpunkt "Deduktion" of the Deutsche Forschungsgemeinschaft the prover *SETHEO* was made available to *ILF* together with other automated provers.

ILF can be configured as a *ProofPad* to assist a user without special knowledge in automated theorem proving in editing elementary proofs, making the best possible use of the power of automated theorem provers.

Using *ILF*, there were solved problems in the domain of lattice ordered fields, communication protocols, and processor verification, using the support of *SETHEO*. The system is used not only by its creators, but also students (of mathematics) used it for their diploma thesis [21].

Within *ILF*, the context is quite different to that of the *TPTP*. The files are present only temporarily, so the data base has to be updated immediately. The filenames have no semantics, and comments contain at most a pointer to the goal and the names of the used axioms. Other information on predicates, formula length and number etc. have to be derived directly from analyzing the PROLOG formulae.

Interesting information will be here of another kind as in *TPTP*. For example, we want to know how many axioms were really used in the proof, which domain the problems belongs to, how many problems the prover was able to prove using which resources and so on.

6.3.2 Porting Log

- The document generator immediately stressed a buggy port of Perl 5 into core dumps. Furthermore, strange core dumps occured in shell scripts, which we traced down to a crippled `/bin/sh` version. Perl was subsequently upgraded to a stable 5.003 port, and we replaced the troublesome Bourne shell by calls to bash (or alternatively: ksh).
- The *Setheodb* database format was slightly changed for *ILF* (see *dbi_ilf_setheo.**).
  - Provers may attempt several different strategies in parallel for proof job, that is for each proof sought by *ILF*:
    * `ID` tag added: `_%ID=ILF.PROVER.USER.DATESTAMP(#N)`, where N is the node number in case there are several objects for a single proof job. Example:

      ```
      _%ID=ILF.SETHEO.jakobi.970215050255#1
      ```

    * `ST_ID` (a.k.a. Success Task): This tag has either the value SELF or the value of the ID tag of the first node returning a proof. Some tags of the success task object are appended to all other objects of the current job (prefix `ST_`).

      Note that the node number is depending on machine load! Note also that the distribution of proof tasks to machines should be randomized in order to be able to obtain meaningful statistics for large databases.
  - Some tags such as `USER` and `ENV` have been removed.
  - Standardized tags for all provers within *ILF*: `ID, Result`, in addition to any embedded tags provided by *ILF* itself.
- *dbi_mod.p* performs prover-independent database postprocessing.
  - Generation of `ST_` tags for jobs with parallel prover instances.
  - Appending selected objects to user-defined databases.
  - Support for prover development: The user may automatically mail user-selected objects to the prover developers.

  The user options can be set in the automatically created resource file *$USERILFHOME/.dbfwrc*, which also documents these features.

Example object for a single node from a parallel proof job using *SETHEO* (slightly shortened):

```
_%ID=ILF.SETHEO.jakobi.970314161005#5
 _%ST_ID=ILF.SETHEO.jakobi.970314161005#4
 _%Formula=ilf.9
 _%IVersion=3.3.1
 _%IPar=-cons
```

```
_%SVersion=3.3
_%SPar=-cons -batch -cputime 120 -cons -wdr -singledelay 2 -forcegr ...
_%FormCl=119
_%FormLit=309
_%FormLength=7
_%FormPred=16 allowedOp/1 contd/3 equal/2 function_like/3 gmyeq/2 gr/2 h/1 ...
_%ST_IPar=-cons _%ST_SPar=-cons -batch -cputime 120 -cons -wdr -dynsgreord 5 ...
```

The integration of *DBFW* into *ILF* is completed, with *SETHEO* as the first supported prover.

6.3.3 How to Add Support for A New Prover

Create a new or modify existing extractor patterns for the new prover. Then, add the following lines to the wrapper of the prover in question:

```
# PROVER      - directory name for the prover, e.g. setheo
# BASENAME    - basename for all files of the active ILF job
# JOBNR       - number of the ILF job (reserved)
# MAXTASK     - number of task running in parallel (1..N)
# SUCCESSTASK - number of task returning the first proof,
#               optional (one of 1..MAXTASK)
$database_script="$ENV{USERILFHOME}/dedsys/$PROVER/database.p";
$database_script="$ENV{ILFHOME}/dedsys/$PROVER/database.p"
   if not -x $database_script;
system "$database_script $BASENAME $JOBNR $MAXTASK $SUCCESSTASK
   # > /dev/null 2>&1";
```

Finally, adapt a copy *database.p*, the prover specific database wrapper. It sets the environment for *db_gen.p* and calls *dbi_mod.p* to postprocess the generated database.

7. Summary

The database framework implementation described in this paper is suitable for medium sized datasets that can be seen as a collection of objects. The relational operations *Selection* and *Projection* are available. *Join*, however, is not.

The framework provides tools to extract objects from text files, to modify collections of objects, to generate documents or reports from a set of objects, and to convert objects to different formats.

The conversion of *DBFW* database files to specific text formats is easy. Thus it is possible to use the framework as a front-end for arbitrary database systems, to extract objects from text files. The intermediate *DBFW* database can be converted and exported later on to a full-blown relational database system, such as *transbase* or *postgres*. Similarly, the framework can be used to post-process objects from other database systems for access via the World Wide Web.

The framework is a useful tool for supporting development, testing and application specific tuning of automated theorem provers. The success of *SETHEO* at the CADE-13 system competition [19] shows the promise of a tool that allows more efficient tuning and testing of a prover against an extensive library of proof tasks such as the TPTP[18] library.

The framework is available under GNU GPL on the World Wide Web[8].

8. References

[1] Denzinger J., Pitz W.: *Das DISCOUNT-System: Benutzerhandbuch.* University of Kaiserslautern, SEKI Working Paper SWP-92-16.

[2] Avenhaus J., Denzinger J., Fuchs M.: *Discount: A System for Distributed Equational Deduction.* Proceedings 6. RTA, pp. 397-402, Springer, 1995.

[3] Beth E. W.: *The Foundations Of Mathematics.* North-Holland, 1969.

[4] Bibel W., Brüning S., Egly U., Rath T.: *KoMeT.* Proceedings of CADE-12, Springer, 1994.

[5] COHTML: `http://gladiole.isbe.ch:8080`

[6] Dahn B. I., Gehne J., Honigmann Th., Wolf A.: *Integration of Automated and Interactive Theorem Proving in ILF.* Proceedings of CADE-14, Springer, 1997 (to appear).

[7] Denzinger J., Kronburg M., Schulz S.: *Discount - A Distributed and Learning Equational Prover.* Journal of Automated Reasoning, 1997 (to appear).

[8] DBFW: `http://wwwjessen.informatik.tu-muenchen.de/~setheo/database_framework/`

[9] Fitting M. C.: *First order logic and automated theorem proving.* Springer, 1990.

[10] Ibens O., Schumann J.: *SETHEO User Manual.* Technical Report, Technische Universität München, Institut für Informatik, 1997 (in preparation).

[11] Letz R., Mayr K., Goller Ch.: *Controlled Integration of the Cut Rule into Connection Tableau Calculi.* Journal of Automated Reasoning, 4 (1994).

[12] Letz R., Schumann J., Bayerl S., Bibel W.: *SETHEO: A High-Performance Theorem Prover.* Journal of Automated Reasoning, 8 (1992).

[13] Loveland D. W.: *Automated Theorem Proving: a Logical Basis.* North-Holland, 1978.

[14] McCune W.: *Otter 2.0.* Proceedings of the 10th CADE, pp. 663-664, Springer, Berlin, 1990.

[15] PHP: `http://www.vex.net/php`

[16] Smullyan R. M.: *First-Order Logic.* Springer, 1968.

[17] Stickel M. E.: *A Prolog Technology Theorem Prover.* New generation computing 2 (1984).

[18] Suttner C. B., Sutcliffe G., Yemenis T.: *The TPTP Problem Library.* Proceedings of CADE-12, Springer, 1994.

[19] Sutcliffe G., Suttner C. B.: *Special Issue: The CADE-13 ATP System Competition.* Journal of Automated Reasoning (18), 1997.

[20] Weidenbach C., Gaede B., Rock G.: *SPASS & FLOTTER.* Proceedings of CADE-13, Springer, 1996.

[21] Wolf, A.; Kmoch, A.: *Einsatz eines automatischen Theorembeweisers in einer taktikgesteuerten Beweisumgebung für die Hardware-Verifikation – Fallstudie* (in German). Technical Report, Technische Universität München, Institut für Informatik, 1997 (in preparation).

[22] Wolfram S.: *Mathematica: A System for Doing Mathematics by Computers.* Addison-Wesley, 1988.