Project *Surmise Relations between Tests*

Documentation of the `libsrbi` Library

Cord Hockemeyer*

August 30, 2000

Institut für Psychologie
Karl–Franzens–Unversität Graz, Austria

*E–Mail: Cord.Hockemeyer@kfunigraz.ac.at

This text was produced with LaTeX 2$_\varepsilon$. A PostScript version was produced with `dvips` and a PDF version with PDFLaTeX.

Contents

1 Overview

This text documents the `libsrbi` library developed within the project *Surmise Relations between Tests* (SRBT; status of August 30, 2000) performed at the Department of Psychology at the University of Graz, Austria. The project is financed by the Austrian National Science Fund (FWF) under the project number P12726–SOZ. Additionally, there exist documentations of the mathematical part of the project and of its psychological application. The `libsrbi` library is built upon the `libkst` Library developed by Cord Hockemeyer at the Department of Psychology, University of Technology at Braunschweig, Germany (1).

The complete software documentation has fifth parts. In the first part, we will document design decisions and basic algorithms used in the software. Decisions on methods and tools are documented in Chapter 3. In Chapter 4, you find definitions of interfaces, especially of file formats. In Chapter 5, important algorithms used in the software are described as far as they are not already contained in the mathematical documentation. This first part of the documentation will be quite free in its format.

In the second to fourth part, user documentation of the library (i. e. for programmers) will be written separately, thus allowing to use a special format, the man pages, which have a long tradition on UNIX systems. As a consequence, library documentation will be available in a number of formats (man pages, PostScript, PDF, HTML). The second part documents a small number of general utility functions and verbosity state variables which are independent of knowledge space theory. The third part contains functions which deal with *plain* sets while the fourth part contains such functions dealing with families of sets (i. e. knowledge structures on items). Finally, the fifth part also contains functions which deal with knowledge spaces. These functions, however, are intended for internal use only and, therefore, should only be applied within the libraries. Their documentation is provided only for the programmers of the SRBI and SRBT libraries.

Part I

General documentation

2 Introduction

One important aspect of making design decisions is the pre-knowledge that the work of programming will be taken over by someone else at November 1, 1998. It should be possible for the successor to adapt and change decisions according to their personal experiences as easily as possible.

Another point underlying many decisions is the form of the software to be developed: To a large percentage, it consists of procedures handling, evaluating and/or processing information about knowledge spaces and surmise relations. A main part will have the form of a library of functions (see also Section 3.2). This fact which is a consequence of the tasks specified for the software part of the project was very important in many decisions documented below.

3 Tools and methods used in the development and documentation of software

This chapter describes the basic decisions on tools and methods used in the development and documentation of the software. This comprises mainly two decisions: the way of combining source code and documentation, and the selection of programming language, compiler, and development environment.

3.1 Documentation as part of the development process: Literate Programming

The consistent documentation of larger software packages under development is a major critical problem. This topic grows in importance if maintenance and development is to be performed by several people as it will be the case in the current project. Strong support for solving this problem lies in the concept of *literate programming* developed by Knuth (2; 3). Literate programming denotes the idea of having code and documentation in one source file and, thus, making it easier to have both at the same level of development. Knuth developed the language `WEB` (6) for concurrent programming and documenting in Pascal. A well–known program written in `WEB` is the typesetting program TEX (4; 5).

Later, the concept of `WEB` was also transferred to other programming languages (see, e. g. (7)). The approach of `WEB` requires the mastery of four languages: `WEB`, TEX (or LATEX), the programming language, and the natural documentation language (van Ammers and Kramer). An important strength of this approach is the support of structured programming: whether the programmer develops their code in top–down, bottom–up, or middle–out method — at each level of abstraction they can simply de-

note chunks of the next lower level by symbolic names. The code is `tangled`[1] together automatically.

A second approach is the documentation in the program source by comments which are extracted automatically. While many systems working in this manner rely completely on the comments in the source, there are at least two systems which also (partially) use the program source code itself: in `c2man` (c2man), e. g., parameters of functions are extracted from the source and, therefore, need not to be specified in the comments additionally. A second example is the `javadoc`[2] tool which is part of SUN's JAVA Development Kit.

Both approaches have their advantages. In literal programming, there is stronger support for structured programming and, thus, for the development process itself. On the other side, it requires learning and using an additional programming language. This is one of the reasons for a decision toward the second approach of just automatically evaluating comments. Another reason is the assumption that algorithms and their development will largely be documented with respect to the mathematical part of the project. Finally, documentations following the second approach is aimed towards the applying programmer of function libraries. For maintaining and extending the libraries, all necessary information should be included in this (external) part of the documentation explaining basic principles — many parts of the libraries will be quite simple, anyway.

As a result of this decision process, the software documentation will be twofold: this part contains decisions and concepts regarding data structures and algorithms. In the two subsequent, semi–automatically produced parts, the application programs and the programming interfaces between library functions and application programs will be described, respectively.

3.2 Programming languages, compilers, and development environment

This section covers a number of decisions about tools etc.:

- Programming language(s)
- Compiler(s) and development environment(s)

[1] The name of the program producing Pascal Source Code from the `WEB` source code is `tangle` in the original `WEB` system; the program `weave` produces TEX source from the `WEB` source.

[2] See `http://java.sun.com/products/jdk/1.0.2/tools/solaris/javadoc.html`.

- Versioning

3.2.1 Programming language

As most promising choices, the languages C, C++, and Java were considered. Out of this group, the author has chosen C as his central programming language for the following reasons: First, C has the highest level of standardization and, therefore, promises the best chances for portability. Opposed to that, both, C++ and Java are still in a process of development end enhancement. As a consequence, there are many compilers (and, in the case of Java, virtual machines) supporting different versions and functionalities.

In the case of C++, even the semantics of the language itself is subject to differ within different versions of the same compiler. In the case of Java, the main problem lies within the lacking backward compatibilities of classes and methods. In both languages, these problems may even occur within one major version of the compiler or development environment. Besides that, Java disqualifies itself as the main programming language within the project, since it is quite slow due to the concept of the virtual Java machine.

To obtain as much portability as possible two rules should be considered during the programming: (i) the usage of the language and of libraries should be restricted to pure ANSI C neglecting all proprietary extensions provided by compiler and development environment, and (ii) interfaces (C header files) should enable using the libraries also from C++ programs.

As far as script languages are appropriate, the author currently would consider shell scripts (preferably in Bourne shell syntax, otherwise in `bash` or `ksh` syntax), or `perl`, `sed` or `awk` scripts.

3.2.2 Compilers and development environments

Based on the selection of the programming language, the selection of a compiler is quite easy: Due to the resources available, the GNU C compiler (`gcc`) will be used. This compiler also has the advantages that it is available on practically any platform and that, on many platforms, it produces even better optimized programs than native C compilers. Besides that, it is a free product and, therefore, we can be sure that no licensing problems will occur.

The compilation of modules will be controlled using the `make` utility which is also freely available for practically any platform. It may well be possible that during the development the author decides to rely on advanced make features like the ability for recursion which is found, e. g., in the GNU `make` utility.

Decisions on tools for configuring software distributions (considering, e. g. ports to other platforms) may be made at a later point of time. This will, of course, also depend on general decisions about software distribution. Nevertheless, at least portability to different UNIX systems should be tested from time to time by compiling the complete software on different UNIX platforms available locally, e. g. Solaris, IRIX, DG–UX, or Linux.

Other development tools
The usage of other development tools (e. g. editors, development environments, or debuggers) are subject to each programmers personal preferences.

3.2.3 Versioning

The decision on versioning control is one of the hardest of those described herein. Currently (August 14, 1998), the author has decided not to use any versioning control system. Nevertheless, he feels committed to include version information in sources and documentation. If, later however, the need for using a versioning control system arises, e. g. because of different programmers working in parallel, this decision should be changeable easily. In this case, the author currently would suggest using the RCS system, since it is supported also by many other software systems, e. g. for documentation.

However, from the release of a first version for (internal) use on, a production version and a development version should be maintained in parallel. Thus, there should always be a working version available.

3.3 Portability

Portability of the software is a major topic for the program development within the project. It should be possible to port software to any major platform with only little difficulties. This will be supported through two steps:

1. Restriction to standard ANSI/ISO C and
2. Frequent test compilations on different compilers and systems.

Prior experiences show that even self–restriction to the ANSI/ISO C standard can not guarantee portability. This is partly due to those areas where the C language specification allows space for different interpretations and, therefore, for differently acting compiler implementations. Such problems will hopefully be detected and eliminated

by frequently compiling the software on different platforms. Currently, the following platforms are available:

- Sun SPARC-station–20, Solaris 2.5.1, gcc 2.7.2
- Sun Ultra–2, Solaris 2.6, gcc 2.8.1
- SGI IPxx, IRIX64 6.2, native C compiler
- PC PPro, Linux 2.0.27, gcc 2.7.2

4 Interfaces, main data structures, and file formats

In this chapter, many decisions will be introduced in comparison to earlier decisions made in the development of libraries for the work with knowledge spaces (henceforth called `libkst`; see, e. g. (1)). One important goal is the maintenance of compatibility between old and new formats, tools, and libraries. As a consequence, it is likely that, e. g., new libraries will get new names to enable the parallel usage of old and new libraries.

4.1 Interfaces

This section describes general principles in the design of function interfaces.

Verbosity and debugging flag

Many of the KST–Tools use an optional verbosity flag to determine the extensity of information printed by the programs. In order to propagate the effects of such a flag down to the library routines, the `libsrbi` library provides an `int verbose` (cf. Section 6.1) which may be evaluated by the library functions. Programs using the library may either set this flag or define it on their own.

Analogously to the `verbose` flag, there exists also the possibility for a `debug` flag (cf. Section 6.2).

4.2 Main data structures

4.2.1 Bitsets

The following discussion is based on the assumption that sets are represented as bit vectors internally.

Regarding functions dealing with sets, there are basically three possibilities to ensure that always correct universal set sizes are used:

1. Defining the universal set size as a global variable. This has the disadvantage of restricting flexibility. It may well occur that set theoretical functions are needed for subsets of different universal sets.

2. Defining a set data type which includes the size of the universal set. This approach has the disadvantage that, in families of sets, it may become necessary to store the same number very often.

3. Passing the size of the universal set to each set function.

The last way may result in higher computation times but it seems to be the best solution. Besides that, it is also the way used successfully in the old KST libraries.

Basic bitset size

An important question is, of course, which integer data type should be underlying the bitset vector. Bitsets will be represented as vectors of int. This will describe 32 bit integer values on practical all systems. However, the explicit size is declared via preprocessor `#define` commands in the `srbi-set.h` header file.

This decision was mainly a decision against using 64 bit technology. It might be worth rethinking that decision when 64 bit technology and programming have stabilized but, currently, it seems to be too less standardized.

4.2.2 Knowledge structures

The old libraries for the work with knowledge spaces do not use a complete structure describing a knowledge space. Instead, a `simplespace`, i. e. a vector of `int` numbers representing the data matrix together with two separate `int` numbers specifying the number of items and the number of states are used. This simple structure can easily become a source of error, especially in the cooperation of multiple programmers. Therefore, new data structures are used containing complete information for a knowledge space (or for a basis) and thus reducing the probability of programming errors.

4.2.3 Surmise relations

There exists a KST–Tool called `dep-matrix` printing a surmise relation as a matrix of size $Q \times Q$. The new `libsrbi` library defines a datatype `srbi` containing such a matrix together with information about the number of items.

4.3 File formats

Previous experiences with tools dealing with knowledge structures have shown that a classical UNIX assumption — *The users have to know what they are doing; it is their responsibility to ensure that parameters specified in program calls make sense* — caused many problems with users not having an appropriate technical background. As a consequence, tools developed in this project will use different file formats which simplify the identification of file format problems. Nevertheless, the new programs and functions will accept both, the old and the new formats. Besides that, there will be conversion programs. These new formats (and the functions handling them) will step by step also be introduced into already existing tools for the work with knowledge spaces. Thus it will be guaranteed that all programs developed with the support by the author will be compatible in their file formats.

4.3.1 Old file formats

Currently, there exist three main file formats:

- ASCII knowledge space files,
- binary knowledge space files, and
- basis files.

The basic pattern underlying all three file type can be described as follows:

Header
Number of Items
Number of States

Data Matrix
A Matrix with one column per item and one row per state stating if and how the respective item is an element of the respective state.

In knowledge space files, the data matrix contains a '1' if an item is an element of a state and a '0' otherwise. These matrices can be represented in ASCII text format or in binary format. In the latter case, one bit is used per item; a complete state, however, always uses a multiple of 32 bits.

In basis files, two different cases of items being elements of states are distinguished. The data matrix contains a '1' if a state is minimal for the item regarded (i. e. for the item q and the state B, we have $B \in \sigma(q)$ in mathematical terms), and it contains a '2' if the state contains the item without being minimal for it (i. e. $q \in B$ but $B \notin \sigma(q)$).

For several programs, also answer pattern files are used. They are not mentioned separately in the enumeration above, since they are in the same format as knowledge space files.

4.3.2 Changes in the file formats

As already mentioned above, the main reason for defining new file formats were problems observed in using tools for the work with knowledge spaces. Basically, three kinds of errors were observed:

Content mismatch: Users are confusing space files, basis files, and answer pattern files.

Format errors: Users are confusing ASCII and binary formats for knowledge space and answer pattern files. Some tools definitely require the one or the other format, for other tools it may be specified by options which the users forget to apply.

Endian errors: The third kind of error which is quite hard to detect results from exchanging binary files between different hardware platforms. This is due to the so–called Endian–problem, i. e. some machines store integer numbers with the most significant byte (MSB) first while others store the least significant byte (LSB) first[1]. Most UNIX workstations use an MSB–first order ('4-3-2-1') while the Intel PC processors (80x86 successors, and compatibles) use an LSB–first order ('1-2-3-4'). These differences make sharing binary files quite difficult and, therefore, easily produce errors and / or require extra computations.

As a consequence, the file formats are changed by inserting a leading format line. This line contains all necessary information (content type, version specification, content

[1] Actually there are even more forms. If the four bytes of an 32-bit-integer are numbered from '1' (LSB) to '4' (MSB) then not only the orders '1-2-3-4' and '4-3-2-1' may occur but also orders like '3-4-1-2' or '2-1-4-3'.

encoding, Endian information, etc.; see appendix B.1). Functions reading files shall test for such a line and, in case of nonexistence, make guesses depending on the contents. In this case, a warning may be issued telling the user about the deprecated file format and about the result of the guessing process.

4.3.3 Additional information about file contents

It may be useful to include into structure files information about the items, the prevenance of the structure etc. In the moment, this is supported through the provision of a comment syntax. These comments, however, are not stored when reading a file and, therefore, can not be re–stored in automatically produced filesd, either.

In future versions this should be changed by including an additional pointer to an abstract `info` structure in all knowledge space data types. For such an info structure, a (sub) library could be written which provides all necessary functionality. Functions of this library would be evoked, e. g. from the IO functions of `libsrbi` and `libsrbt`.

5 Special algorithms and methods

5.1 Selecting a student sample from large knowledge spaces

Sometimes knowledge spaces are too large to be stored in memory completely. If the space is available as a file, however, it may be useful to select a sample directly from that file. Such a procedure would read the header information from a knowledge space file, randomly select a sequence of state numbers, sort this sequence, and copy the corresponding knowledge states from the file into a `data` structure.

Another possibility, which would need even more computing resources, however, would be a double construction of the knowledge space from the basis using the Ganter algorithm (`ganter_basis()`, see Section 19.3, or `ganter_structure()`, see Section 19.4, function). The first construction would be used to determine the size of the knowledge space. Afterwards, again a sequence of state numbers would be randomly selected and chosen. Finally, the space would be constructed a second time storing all states selected in the second step into a `data` structure. Such a procedure would be needed for knowledge spaces which are even too large to be stored on hard disks. One should consider, however, that it might also be impossible to compute these spaces within reasonable time at all.

5.2 Special methods in memory menagemant

Since prior personal experiences have shown that various memory functions are often malfunctional, the functions `calloc()`, and `memcpy()` have not been used but their functionality has been reprogrammed with other commands wherever needed.

Part II

General utility functionalities

This part documents a small number of general utility functions and variables defined in the `srbi-util.h` header file.

6 Verbosity flags

6.1 verbose

NAME

verbose — flag indicating verbosity mode

SYNOPSIS

```
#include <srbi-util.h>

extern int verbose;
```

DESCRIPTION

The verbose flag indicates a verbosity mode.

6.2 debug

NAME

debug — flag indicating debug mode

SYNOPSIS

```
#include <srbi-util.h>

extern int debug;
```

DESCRIPTION

The verbose flag indicates a debugging mode.

7 Utility functions

7.1 random

NAME

random — Generating random numbers with larger domain.

SYNOPSIS

```
#include <srbi-util.h>

extern long random(void);
```

DESCRIPTION

On many systems, the rand() function from stdlib generates only 15bit integer numbers. In such cases random() should produce 30bit integers.

This function uses stdlib's rand() function and, therefore, can be influenced through stdlib's srand() function.

7.2 urandom

NAME

urandom — Generating random numbers with larger domain.

SYNOPSIS

```
#include <srbi-util.h>

extern unsigned int urandom(void);
```

DESCRIPTION

On many systems, the rand() function from stdlib generates only 15bit integer numbers. In such cases urandom() should produce 32bit integers.

This function uses stdlib's rand() function and, therefore, can be influenced through stdlib's srand() function.

Part III

Functions dealing with *plain* sets

The functions documented in this part are defined in the `srbi-set.h` header file. These functions are divided into eight groups: creating and deleting sets, changing sets' contents, I/O functions, set operating functions, set operating procedures, accumulating set operating procedures, set predicates, and functions for determining cardinalities.

8 Creating and deleting bitsets

8.1 `new_bitset`

NAME

new_bitset — Create a new and empty bitset

SYNOPSIS

```
#include <srbi-set.h>

extern bitset new_bitset(int set_size);
```

PARAMETERS

int set_size
: Size of the set to be created.

DESCRIPTION

Create a new and empty bitset.

8.2 `random_set`

NAME

random_set — Create a new bitset with randomly selected elements.

SYNOPSIS

```
#include <srbi-set.h>
```

```
extern bitset random_set(int set_size);
```

PARAMETERS

int set_size
Size of the bitset.

DESCRIPTION

This function uses stdlib's rand() routine. Therefore, it can be influenced through stdlib's srand() routine.

8.3 delete_bitset

NAME

delete_bitset — Delete a bitset and return storage to the system

SYNOPSIS

```
#include <srbi-set.h>

extern void delete_bitset(bitset a);
```

PARAMETERS

bitset a
Bitset to be deleted.

DESCRIPTION

Delete a bitset and return storage to the system.

8.4 copy_bitset

NAME

copy_bitset — Make a copy of a bitset.

SYNOPSIS

```
#include <srbi-set.h>

extern bitset copy_bitset
(
    bitset orig,
    int set_size
);
```

PARAMETERS

bitset orig
Original bitset.

int set_size
Size of the bitset.

DESCRIPTION

New memory is allocated.

9 Changing bitsets

9.1 set_excl

NAME

set_excl — Delete an item from a bitset

SYNOPSIS

```
#include <srbi-set.h>

void set_excl
(
    int item,
    bitset a,
    int set_size
);
```

DESCRIPTION

Delete an item from a bitset.

9.2 set_incl

NAME

set_incl — Add an item to a bitset

SYNOPSIS

```
#include <srbi-set.h>
```

```
void set_incl
(
    int item,
    bitset a,
    int set_size
);
```

DESCRIPTION

Add an item to a bitset.

9.3 set_bitset

NAME

set_bitset — Copy a bitset into another bitset

SYNOPSIS

```
#include <srbi-set.h>

extern int set_bitset
(
    bitset copy,
    bitset orig,
    int set_size
);
```

PARAMETERS

bitset copy
Destination bitset.

bitset orig
Original bitset.

int set_size
Size of the bitset.

DESCRIPTION

Copy a bitset into another bitset.

10 I/O functions for bitsets

10.1 ascii_read

NAME

ascii_read — Read a bitset from an ASCII file.

SYNOPSIS

```
#include <srbi-set.h>

extern bitset ascii_read
(
    FILE *in,
    int set_size
);
```

DESCRIPTION

New memory is allocated.

10.2 ascii_write

NAME

ascii_write — Write a bitset to an ASCII file.

SYNOPSIS

```
#include <srbi-set.h>
```

```
extern int ascii_write
(
    FILE *out,
    bitset a,
    int set_size
);
```

DESCRIPTION

Write a bitset to an ASCII file.

10.3 bin_read

NAME

bin_read — Read a bitset from an binary file.

SYNOPSIS

```
#include <srbi-set.h>

extern bitset bin_read
(
    FILE *in,
    int set_size
);
```

DESCRIPTION

New memory is allocated.

10.4 bin_write

NAME

bin_write — Write a bitset to an binary file.

SYNOPSIS

```
#include <srbi-set.h>

extern int bin_write
(
    FILE *out,
    bitset a,
    int set_size
);
```

DESCRIPTION

Write a bitset to an binary file.

11 Set operating functions

11.1 `complement`

NAME

complement — Determine the complement of a bitset

SYNOPSIS

```
#include <srbi-set.h>

extern bitset complement
(
    bitset orig,
    int set_size
);
```

DESCRIPTION

Determine the complement of a bitset.

11.2 `set_union`

NAME

set_union — Compute the union of two bitsets with a function

SYNOPSIS

```
#include <srbi-set.h>
```

```
extern bitset set_union
(
    bitset a,
    bitset b,
    int set_size
);
```

DESCRIPTION

Compute the union of two bitsets with a function.

11.3 section

NAME

section — Compute the intersection of two bitsets with a function

SYNOPSIS

```
#include <srbi-set.h>

extern bitset section
(
    bitset a,
    bitset b,
    int set_size
);
```

DESCRIPTION

Compute the intersection of two bitsets with a function.

11.4 set_diff

NAME

set_diff — Compute the set difference of two bitsets with a function

SYNOPSIS

```
#include <srbi-set.h>

extern bitset set_diff
(
    bitset a,
    bitset b,
    int set_size
);
```

DESCRIPTION

Compute the set difference of two bitsets with a function.

11.5 symm_diff

NAME

symm_diff — Compute the symmetrical set difference of two bitsets with a function

SYNOPSIS

```
#include <srbi-set.h>

extern bitset symm_diff
(
    bitset a,
    bitset b,
    int set_size
);
```

DESCRIPTION

Compute the symmetrical set difference of two bitsets with a function.

12 Set operating procedures

12.1 comp_union

NAME

comp_union — Compute the union of two bitsets with a procedure

SYNOPSIS

```
#include <srbi-set.h>

extern void comp_union
(
    bitset result,
    bitset a,
    bitset b,
    int set_size
);
```

DESCRIPTION

Compute the union of two bitsets with a procedure.

12.2 comp_section

NAME

comp_section — Compute the intersection of two bitsets with a procedure

SYNOPSIS

```
#include <srbi-set.h>

extern void comp_section
(
    bitset result,
    bitset a,
    bitset b,
    int set_size
);
```

DESCRIPTION

Compute the intersection of two bitsets with a procedure.

12.3 comp_set_diff

NAME

comp_set_diff — Compute the set difference of two bitsets with a procedure

SYNOPSIS

```
#include <srbi-set.h>

extern void comp_set_diff
(
    bitset result,
    bitset a,
    bitset b,
    int set_size
);
```

DESCRIPTION

Compute the set difference of two bitsets with a procedure.

12.4 comp_symm_diff

NAME

comp_symm_diff — Compute the symmetrical set difference of two bitsets with a procedure

SYNOPSIS

```
#include <srbi-set.h>

extern void comp_symm_diff
(
    bitset result,
    bitset a,
    bitset b,
    int set_size
);
```

DESCRIPTION

Compute the symmetrical set difference of two bitsets with a procedure.

13 Accumulative set operating procedures

13.1 acc_union

NAME

acc_union — Compute the accumulated union of two bitsets

SYNOPSIS

```
#include <srbi-set.h>

extern void acc_union
(
    bitset result,
    bitset op,
    int set_size
);
```

DESCRIPTION

Compute the accumulated union of two bitsets.

13.2 acc_section

NAME

acc_section — Compute the accumulated intersection of two bitsets

SYNOPSIS

```
#include <srbi-set.h>
```

```
extern void acc_section
(
    bitset result,
    bitset op,
    int set_size
);
```

DESCRIPTION

Compute the accumulated intersection of two bitsets.

13.3 acc_set_diff

NAME

acc_set_diff — Compute the accumulated set difference of two bitsets

SYNOPSIS

```
#include <srbi-set.h>

extern void acc_set_diff
(
    bitset result,
    bitset op,
    int set_size
);
```

DESCRIPTION

Compute the accumulated set difference of two bitsets.

13.4 acc_symm_diff

NAME

acc_symm_diff — Compute the accumulated symmetrical set difference of two bitsets

SYNOPSIS

```
#include <srbi-set.h>

extern void acc_symm_diff
(
    bitset result,
    bitset op,
    int set_size
);
```

DESCRIPTION

Compute the accumulated symmetrical set difference of two bitsets.

14 Set predicates

14.1 is_element

NAME

is_element — Test if a bitset contains an item

SYNOPSIS

```
#include <srbi-set.h>

extern int is_element
(
    int item,
    bitset a,
    int set_size
);
```

DESCRIPTION

Test if a bitset contains an item.

14.2 emptyset

NAME

emptyset — Test if a set is empty

SYNOPSIS

```
#include <srbi-set.h>
```

```
extern int emptyset
(
    bitset a,
    int set_size
);
```

DESCRIPTION

Test if a set is empty.

14.3 equal

NAME

equal — Test if two subsets are equal

SYNOPSIS

```
#include <srbi-set.h>

extern int equal
(
    bitset a,
    bitset b,
    int set_size
);
```

DESCRIPTION

Test if two subsets are equal.

14.4 subset

NAME

subset — Test if a bitset is a subset of another

SYNOPSIS

```
#include <srbi-set.h>

extern int subset
(
    bitset a,
    bitset b,
    int set_size
);
```

DESCRIPTION

Test if a bitset is a subset of another.

14.5 subseteq

NAME

subseteq — Test if a bitset is subset of or equal to another

SYNOPSIS

```
#include <srbi-set.h>

extern int subseteq
(
    bitset a,
    bitset b,
    int set_size
);
```

DESCRIPTION

Test if a bitset is subset of or equal to another.

15 Cardinalities

15.1 `cardinality`

NAME

cardinality — Determine the cardinality, i.

SYNOPSIS

```
#include <srbi-set.h>

extern int cardinality
(
    bitset a,
    int set_size
);
```

DESCRIPTION

E. The number of elements of a bitset.

15.2 `union_size`

NAME

union_size — Determine the size of the union of two bitsets

SYNOPSIS

```
#include <srbi-set.h>
```

```
extern int union_size
(
    bitset a,
    bitset b,
    int set_size
);
```

DESCRIPTION

Determine the size of the union of two bitsets.

15.3 section_size

NAME

section_size — Determine the size of the intersection of two bitsets

SYNOPSIS

```
#include <srbi-set.h>

extern int section_size
(
    bitset a,
    bitset b,
    int set_size
);
```

DESCRIPTION

Determine the size of the intersection of two bitsets.

15.4 set_diff_size

NAME

set_diff_size — Determine the size of the ordinary set difference of two bitsets

SYNOPSIS

```
#include <srbi-set.h>

extern int set_diff_size
(
    bitset a,
    bitset b,
    int set_size
);
```

DESCRIPTION

Determine the size of the ordinary set difference of two bitsets.

15.5 symm_diff_size

NAME

symm_diff_size — Determine the size of the symmetrical set difference of two bitsets

SYNOPSIS

```
#include <srbi-set.h>

extern int symm_diff_size
(
    bitset a,
    bitset b,
    int set_size
);
```

DESCRIPTION

Determine the size of the symmetrical set difference of two bitsets.

Part IV

Functions dealing with knowledge spaces

The functions documented in this part are defined in the `srbi-space.h` and `filetype.h` header files. The functions from `srbi-space.h` are divided into four groups, functions for creating and deleting knowledge spaces, I/O functions, functions for converting knowledge spaces from one representation to another, and functions to select sub spaces and to simulate students. Additionally, `srbi-space.h` defines variables which may be applied for controlling the verbosity of programs using these functions.

The `filetype.h` header file contains a number of type, variable, and function definitions regarding types and formats of files.

16 Creating, copying, and deleting knowledge spaces

16.1 new_space

NAME

new_space — Allocate memory for a knowledge space.

SYNOPSIS

```
#include <srbi-space.h>

extern space *new_space
(
    int q_size,
    int s_size
);
```

PARAMETERS

int q_size
Number of items.

int s_size
Number of states.

DESCRIPTION

Allocate memory for a knowledge space.

16.2 new_basis

NAME

new_basis — Allocate memory for a basis.

SYNOPSIS

```
#include <srbi-space.h>

extern basis *new_basis
(
    int q_size,
    int b_size
);
```

PARAMETERS

int q_size
Number of items.

int b_size
Number of clauses.

DESCRIPTION

Allocate memory for a basis.

16.3 new_structure

NAME

new_structure — Allocate memory for a knowledge structure.

SYNOPSIS

```
#include <srbi-space.h>

extern structure *new_structure
```

```
(
    int q_size,
    int s_size
);
```

PARAMETERS

int q_size
Number of items.

int s_size
Number of states.

DESCRIPTION

Allocate memory for a knowledge structure.

16.4 new_data

NAME

new_data — Allocate memory for a data set.

SYNOPSIS

```
#include <srbi-space.h>

extern data *new_data
(
    int q_size,
    int s_size
);
```

PARAMETERS

int q_size
Number of items.

int s_size
Number of patterns.

DESCRIPTION

Allocate memory for a data set.

16.5 new_patterns

NAME

new_patterns — allocate memory for a pattern set.

SYNOPSIS

```
#include <patterns.h>

patterns *new_patterns
(
    int q_size,
    int p_size
);
```

DESCRIPTION

@Param q_size number of items @param p_size number of patterns.

16.6 new_srbi

NAME

new_srbi — Allocate memory for a surmise relation.

SYNOPSIS

```
#include <srbi-space.h>

extern srbi *new_srbi(int q_size);
```

PARAMETERS

int q_size
Number of items.

DESCRIPTION

Allocate memory for a surmise relation.

16.7 `free_space`

NAME

free_space — Return memory used by a space to the system.

SYNOPSIS

```
#include <srbi-space.h>

extern void free_space(space **s);
```

DESCRIPTION

Return memory used by a space to the system.

16.8 `free_basis`

NAME

free_basis — Return memory used by a basis to the system.

SYNOPSIS

```
#include <srbi-space.h>

extern void free_basis(basis **b);
```

DESCRIPTION

Return memory used by a basis to the system.

16.9 free_structure

NAME

free_structure — Return memory used by a structure to the system.

SYNOPSIS

```
#include <srbi-space.h>

extern void free_structure(structure **s);
```

DESCRIPTION

Return memory used by a structure to the system.

16.10 free_data

NAME

free_data — Return memory used by a data set to the system.

SYNOPSIS

```
#include <srbi-space.h>

extern void free_data(data **s);
```

DESCRIPTION

Return memory used by a data set to the system.

16.11 free_patterns

NAME

free_patterns — Return memory used by a pattern set to the system

SYNOPSIS

```
#include <patterns.h>

void free_patterns(patterns **p);
```

DESCRIPTION

@Param p patterns set.

16.12 free_srbi

NAME

free_srbi — Return memory used by a surmise relation

SYNOPSIS

```
#include <srbi-space.h>

extern void free_srbi(srbi **r);
```

DESCRIPTION

Return memory used by a surmise relation.

16.13 copy_space

NAME

copy_space — Make a copy of an existing knowledge space.

SYNOPSIS

```
#include <srbi-space.h>

extern space *copy_space(space *orig);
```

DESCRIPTION

Make a copy of an existing knowledge space.

16.14 copy_basis

NAME

copy_basis — Make a copy of an existing basis.

SYNOPSIS

```
#include <srbi-space.h>

extern basis *copy_basis(basis *orig);
```

DESCRIPTION

Make a copy of an existing basis.

16.15 copy_structure

NAME

copy_structure — Make a copy of an existing knowledge structure.

SYNOPSIS

```
#include <srbi-space.h>

extern structure *copy_structure(structure *orig);
```

DESCRIPTION

Make a copy of an existing knowledge structure.

16.16 copy_data

NAME

copy_data — Make a copy of an existing data set.

SYNOPSIS

```
#include <srbi-space.h>

extern data *copy_data(data *orig);
```

DESCRIPTION

Make a copy of an existing data set.

16.17 copy_patterns

NAME

copy_patterns — make a copy of a set of patterns.

SYNOPSIS

```
#include <patterns.h>

patterns *copy_patterns(patterns *orig);
```

DESCRIPTION

The necessary memory is allocated. @Param orig original patterns to be copied @return pointer to copied patterns.

16.18 random_data

NAME

random_data — Create a random data set.

SYNOPSIS

```
#include <srbi-space.h>

extern data *random_data
(
    int q_size,
    int s_size
);
```

PARAMETERS

int q_size
Number of items.

int s_size
Number of patterns.

DESCRIPTION

All subsets of items are equally probable. Repetitions are possible.

17 Extracting and setting knowlede states

17.1 copy_state

NAME

copy_state — Copy a single knowledge state from a knowledge space.

SYNOPSIS

```
#include <srbi-space.h>

extern bitset copy_state
(
    space *s,
    int position
);
```

PARAMETERS

space *s
Space from which the state is to be copied.

int position
Index of the state to be copied.

DESCRIPTION

New memory is allocated.

17.2 get_state

NAME

get_state — Get a single knowledge state from a knowledge space.

SYNOPSIS

```
#include <srbi-space.h>

extern bitset get_state
(
    space *s,
    int position
);
```

PARAMETERS

space *s
Space from which the state is to be copied.

int position
Index of the state to be copied.

DESCRIPTION

No new memory is allocated.

17.3 set_state

NAME

set_state — Set a knowledge state within a knowledge space.

SYNOPSIS

```
#include <srbi-space.h>

extern int set_state
```

```
(
    space *s,
    int position,
    bitset b
);
```

PARAMETERS

space *s
Knowledge space.

int position
Index of the state to be set.

bitset b
New value of the state.

DESCRIPTION

Set a knowledge state within a knowledge space.

17.4 copy_pattern

NAME

copy_pattern — Copy a single answer pattern from a data set.

SYNOPSIS

```
#include <srbi-space.h>

bitset copy_pattern
(
    data *s,
    int position
);
```

PARAMETERS

data *s
Data set from which the answer pattern is to be copied.

int position
Index of the answer pattern to be copied.

DESCRIPTION

New memory is allocated.

17.5 get_pattern

NAME

get_pattern — Get a single answer pattern from a data set.

SYNOPSIS

```
#include <srbi-space.h>

extern bitset get_pattern
(
    data *d,
    int position
);
```

PARAMETERS

data *d
Data set from which the answer pattern is to be copied.

int position
Index of the answer pattern to be copied.

DESCRIPTION

No new memory is allocated.

17.6 set_pattern

NAME

set_pattern — Set a answer pattern within a data set space.

SYNOPSIS

```
#include <srbi-space.h>

extern int set_pattern
(
    data *d,
    int position,
    bitset b
);
```

PARAMETERS

data *d
Data set.

int position
Index of the answer pattern to be set.

bitset b
New value of the answer pattern.

DESCRIPTION

Set a answer pattern within a data set space.

18 I/O functions for knowledge spaces on items

18.1 load_basis

NAME

load_basis — load a basis from a file

SYNOPSIS

```
#include <srbi-space.h>

basis *load_basis(const char filename[]);
```

PARAMETERS

const char filename[ ]
Name of the file to be loaded.

DESCRIPTION

This function loads a basis from a file. It determines automatically which type of data is stored in the file and performs a squeeze operation on the data if necessary.

RETURNS

Pointer to resulting basis.

18.2 load_space

NAME

load_space — load a space from a file

SYNOPSIS

```
#include <srbi-space.h>

space *load_space(const char filename[]);
```

PARAMETERS

const char filename[ ]
Name of the file to be loaded.

DESCRIPTION

This function loads a space from a file. It determines automatically which type of data is stored in the file and performs a constr operation on the data if necessary.

RETURNS

Pointer to resulting space.

18.3 load_structure

NAME

load_structure — load a structure from a file

SYNOPSIS

```
#include <srbi-space.h>

structure *load_structure(const char filename[]);
```

PARAMETERS

const char filename[ ]
Name of the file to be loaded.

DESCRIPTION

This function loads a structure from a file.

RETURNS

Pointer to resulting structure.

18.4 load_data

NAME

load_data — load a data set from a file

SYNOPSIS

```
#include <srbi-space.h>

data *load_data(const char filename[]);
```

PARAMETERS

const char filename[ ]
Name of the file to be loaded.

DESCRIPTION

This function loads a data set from a file.

RETURNS

Pointer to resulting data set.

18.5 load_patterns

NAME

load_patterns — load a pattern set from a file.

SYNOPSIS

```
#include <patterns.h>

patterns *load_patterns(const char filename[]);
```

DESCRIPTION

This function loads a pattern set from a file. @Return pointer to resulting pattern set. @Param filename[] name of the file to be loaded.

18.6 load_srbi

NAME

load_srbi — load a surmise relation from a file

SYNOPSIS

```
#include <srbi-space.h>

srbi *load_srbi(const char filename[]);
```

PARAMETERS

const char filename[ ]
Name of the file to be loaded.

DESCRIPTION

This function loads a surmise relation from a file.

RETURNS

Pointer to resulting surmise relation.

18.7 save_basis

NAME

save_basis — write a basis to a file

SYNOPSIS

```
#include <srbi-space.h>

int save_basis
(
    basis *b,
    filetype mode,
    const char filename[]
);
```

PARAMETERS

basis *b
Basis to be stored.

filetype mode
Format to be used.

const char filename[ ]
Name of the file to be saved.

DESCRIPTION

This function writes a basis to a file using the new basisfile format.

RETURNS

Error code.

18.8 save_space

NAME

save_space — write a knowledge space to a file

SYNOPSIS

```
#include <srbi-space.h>

int save_space
(
    space *s,
    filetype mode,
    const char filename[]
);
```

PARAMETERS

space *s
Space to be stored.

filetype mode
Format to be used.

const char filename[ ]
Name of the file to be saved.

DESCRIPTION

This function writes a knowledge space to a file using the new spacefile format.

RETURNS

Error code.

18.9 save_structure

NAME

save_structure — write a knowledge structure to a file

SYNOPSIS

```
#include <srbi-space.h>

int save_structure
(
    structure *s,
    filetype mode,
    const char filename[]
);
```

PARAMETERS

structure *s
Structure to be stored.

filetype mode
Format to be used.

const char filename[ ]
Name of the file to be saved.

DESCRIPTION

This function writes a knowledge structure to a file using the new structurefile format.

RETURNS

Error code.

18.10 save_data

NAME

save_data — write a data set to a file

SYNOPSIS

```
#include <srbi-space.h>

int save_data
(
    data *s,
    filetype mode,
    const char filename[]
);
```

PARAMETERS

data *s
Data to be stored.

filetype mode
Format to be used.

const char filename[ ]
Name of the file to be saved.

DESCRIPTION

This function writes a data set to a file using the new datafile format.

RETURNS

Error code.

18.11 save_patterns

NAME

save_patterns — write a pattern set to a file.

SYNOPSIS

```
#include <patterns.h>

int save_patterns
(
    patterns *pa,
    filetype mode,
    const char filename[]
);
```

DESCRIPTION

This function writes a set of answer patterns to a file using the new patternfile format. @Return error code. @Param pa patterns to be stored @param mode format to be used @param filename[] name of file to be saved.

18.12 save_srbi

NAME

save_srbi — write a surmise relation to a file

SYNOPSIS

```
#include <srbi-space.h>

int save_srbi
(
    srbi *r,
    const char filename[]
);
```

PARAMETERS

srbi *r
Surmise relation to be stored.

const char filename[ ]
Name of the file to be saved.

DESCRIPTION

This function writes a surmise relation to a file using the new relationfile format.

RETURNS

Error code.

18.13 write_patterns

NAME

write_patterns — write a set of answer-patterns to stdout.

SYNOPSIS

```
#include <patterns.h>

int write_patterns(patterns *pa);
```

DESCRIPTION

Write a set of answer patterns to stdout, using '1' for a correct solved item, '0' for a wrong solution and 'x' for a not answered item. @Return -1, if an error occured, 0 else.

18.14 write_srbi

NAME

write_srbi — write a surmise relation as a matrix

SYNOPSIS

```
#include <srbi-space.h>

int write_srbi
(
    FILE *f,
    srbi *m
);
```

PARAMETERS

FILE *f
: FILE to write into.

srbi *m
: Matrix to be written.

DESCRIPTION

This function writes the matrix of a surmise relation to a stream. No header information are written.

RETURNS

Error code.

19 Conversion of various data structures

19.1 `constr_basis`

NAME

constr_basis — close a basis under union

SYNOPSIS

```
#include <srbi-space.h>

space *constr_basis(basis *b);
```

PARAMETERS

basis *b
: Basis to be closed.

DESCRIPTION

This function computes the closure under union of the basis specified. Currently, Dowling's (1993) algorithm is used.

RETURNS

Pointer to resulting knowledge space.

19.2 constr_structure

NAME

constr_structure — close a structure under union

SYNOPSIS

```
#include <srbi-space.h>

space *constr_structure(structure *s);
```

PARAMETERS

structure *s
Structure to be closed.

DESCRIPTION

This function computes the closure under union of the structure specified. Currently, Dowling's (1993) algorithm is used.

RETURNS

Pointer to resulting knowledge space.

19.3 ganter_basis

NAME

ganter_basis — close a basis under union using the Ganter algorithm

SYNOPSIS

```
#include <srbi-space.h>

int ganter_basis
(
    basis *b,
```

```
    filetype mode,
    const char filename[]
);
```

PARAMETERS

basis *b
Basis to be closed.

filetype mode
File format.

const char filename[ ]
Filename for resulting space.

DESCRIPTION

This function takes a basis and computes its closure under union using the algorithm developed by Ganter. The result is directly stored into a file.

RETURNS

Number of states of the resulting knowledge space. Negative in case of an error.

Warning
Not yet implemented!

19.4 ganter_structure

NAME

ganter_structure — close a structure under union using the Ganter algorithm

SYNOPSIS

```
#include <srbi-space.h>

int ganter_structure
(
```

```
    structure *s,
    filetype mode,
    const char filename[]
);
```

PARAMETERS

structure *s
Structure to be closed.

filetype mode
File format.

const char filename[ ]
Filename for resulting space.

DESCRIPTION

This function takes a structure and computes its closure under union using the algorithm developed by Ganter. The result is directly stored into a file.

RETURNS

Number of states of the resulting knowledge space. Negative in case of an error.

Warning
Not yet implemented!

19.5 squeeze

NAME

squeeze — determine basis of a structure

SYNOPSIS

```
#include <srbi-space.h>

basis *squeeze(structure *s);
```

PARAMETERS

structure *s
Knowledge structure to be squeezed.

DESCRIPTION

This function computes the minimal elements of a given knowledge structure. The set of minimal elements is the basis of the space which would be the result of closing the structure under union.

RETURNS

Pointer to the resulting basis.

19.6 dep_matrix

NAME

dep_matrix — determine SRBI matrix from a basis

SYNOPSIS

```
#include <srbi-space.h>

srbi *dep_matrix(basis *b);
```

PARAMETERS

basis *b
Basis describing the surmise relation.

DESCRIPTION

This function takes a basis, checks whether it describes a surmise relation, and stores it as a SRBI structure.

RETURNS

Pointer to the resulting SRBI structure. NULL if the structure cannot be generated, e.g. If the basis does not describe a surmise relation.

19.7 clause_set

NAME

clause_set — Determine the set of states of a basis

SYNOPSIS

```
#include <srbi-space.h>

structure *clause_set(basis *b);
```

DESCRIPTION

Allocate memory for a simplespace and fill it with the complete clauses which are given in partitioned form in the basis_struct structure.

19.8 get_clause

NAME

get_clause — Determine a clause

SYNOPSIS

```
#include <srbi-space.h>

extern bitset get_clause
(
    basis *b,
    int index
);
```

PARAMETERS

basis *b
The basis where the clause is found in.

int index
The position of the clause in the basis.

DESCRIPTION

Clauses, i.e. Knowledge states which are members of the basis, are partitioned into two subsets: the set of items for which the clause is minimal and the set of items which are elements of the clause but for which the clause is not minimal. Therefore we have specific functions in the spacelib.

19.9 clause_cardinality

NAME

clause_cardinality — Compute the size of a clause

SYNOPSIS

```
#include <srbi-space.h>

int clause_cardinality
(
    basis *b,
    int index
);
```

PARAMETERS

basis *b
The basis where the clause is found in.

int index
The position of the clause in the basis.

DESCRIPTION

Clauses, i.e. Knowledge states which are members of the basis, are partitioned into two subsets: the set of items for which the clause is minimal and the set of items which are elements of the clause but for which the clause is not minimal. Therefore we have specific functions in the spacelib.

RETURNS

Return codes: >= 0 Size of the clause -1 basis is NULL pointer -2 index > basis->b_size.

19.10 patterns2data

NAME

patterns2data — convert pattern set to data set.

SYNOPSIS

```
#include <patterns.h>

data *patterns2data(patterns *p);
```

DESCRIPTION

This function takes a pattern set and converts it to a data set assuming that all unanswered items are not mastered. @Param p patterns structure @return pointer to resultin data set.

19.11 data2patterns

NAME

data2patterns — convert data set to pattern set

SYNOPSIS

```
#include <patterns.h>

patterns *data2patterns(data *d);
```

DESCRIPTION

This function is mere a cast operator. In the resulting pattern set, all items are considered to be answered, i.e. The un-answered matrix is set to zero. @Param d data set @return pointer to resulting patterns set.

20 Deriving knowledge spaces with certain properties

20.1 `wellgrade`

NAME

wellgrade — Compute a wellgraded basis

SYNOPSIS

```
#include <srbi-space.h>

basis *wellgrade(basis *b);
```

DESCRIPTION

This fuction computes the basis of a wellgraded space which is a superset of the space described through the basis given as a parameter. Wellgradedness can not be reached through a closure operator since, for a non-wellgraded knowledge space, there does not exist a *unique* smallest wellgraded knowledge space as a superset.

Here, the decision is to partition basis elements causing non-wellgradedness into a *prerequisite subset* and a *new items* subset. In the resulting wellgraded basis, each non-wellgraded element is substituted by the family of all sets of the form *prerequisite subset + single new item*.

21 Selecting sub spaces and simulating students

21.1 mask_basis

NAME

mask_basis — Reduce a basis to a subset of items

SYNOPSIS

```
#include <srbi-space.h>

extern basis *mask_basis
(
    basis *b,
    bitset mask
);
```

PARAMETERS

basis *b
Original basis.

bitset mask
Item set of reduced basis.

DESCRIPTION

Reduce a basis to a subset of items.

21.2 mask_space

NAME

mask_space — Reduce a knowledge space to a subset of items

SYNOPSIS

```
#include <srbi-space.h>

extern space *mask_space
(
    space *s,
    bitset mask
);
```

PARAMETERS

space *s
Original space.

bitset mask
Item set of reduced space.

DESCRIPTION

Reduce a knowledge space to a subset of items.

21.3 mask_data

NAME

mask_data — Reduce a data set to a subset of items

SYNOPSIS

```
#include <srbi-space.h>

extern data *mask_data
```

```
(
    data *d,
    bitset mask
);
```

PARAMETERS

data *d
Original data set.

bitset mask
Item set of reduced patterns.

DESCRIPTION

Reduce a data set to a subset of items.

21.4 `equal_sample`

NAME

equal_sample — select sequence of states under equal distribution

SYNOPSIS

```
#include <srbi-space.h>

data *equal_sample
(
    structure *s,
    int size
);
```

PARAMETERS

structure *s
Source structure of the sample.

int size
Size of the sample.

DESCRIPTION

This function selects from a knowledge space a sequence of knowledge states assuming equal distribution. This is to be used as simulated data. Some noise (careless errors and lucky guesses) may be added via the mk_noisy() function.

This function uses stdlib's rand() routine. Therefore, it can be influenced through stdlib's srand() routine.

RETURNS

Pointer to the selected data.

21.5 mk_noisy

NAME

mk_noisy — add noise to a set of data

SYNOPSIS

```
#include <srbi-space.h>

data *mk_noisy
(
    data *d,
    int *beta,
    int *eta
);
```

PARAMETERS

data *d
Original 'clean' data set.

int *beta
Beta values in per thousand.

int *eta

Eta values in per thousand.

DESCRIPTION

This function adds noise, i.e. Careless errors and lucky guesses to a data set. Noise is added according to the model developed by Falmagne and Doignon.

This function uses stdlib's rand() routine. Therefore, it can be influenced through stdlib's srand() routine.

RETURNS

Pointer to noisy data.

22 File type and version (`filetype.h`)

22.1 `structtype_names`

NAME

structtype_names — Strings describing the various structure types.

SYNOPSIS

```
#include <filetype.h>

extern const char *structtype_names[];
```

DESCRIPTION

Strings describing the various structure types.

22.2 `srbt_version`

NAME

srbt_version — String containing SRBT mark and version.

SYNOPSIS

```
#include <filetype.h>

extern const char srbt_version[];
```

DESCRIPTION

Version 2.0 marks the new format header lines. Version 1.0 is to be used for the ‘old´ KST format.

22.3 type_of_file

NAME

type_of_file — Checks for the type of an SRBI / SRBT file.

SYNOPSIS

```
#include <filetype.h>

extern filetype type_of_file(const char *filename);
```

PARAMETERS

const char *filename
Name of file to be tested.

DESCRIPTION

Type_of_file() opens a file and tries to find out which type of SRBI / SRBT data are stored in the file. It is closed afterwards, again.

RETURNS

Type of file including type of contents and additional attributes. Filetypes contain in the lower eight bits the structtype, in the next eight bits there are format specifications, and the next 16 bits contain the wordsize for binary files.

WARNING

Old-style binary files are always identified as data files since we can assume none of the stricter conditions to be fulfilled for knowledge structures or knowledge spaces.

Part V

Internal IO functions for knowledge spaces

The functions documented in this part are defined in the `internal/srbi-space-io.h` header file. They are devided into two groups, functions for knowledge spaces and functions for bases. Since the internal representations for knowledge spaces, knowledge structuresw, and data sets are basically identical, they are not distinguished on this level.

23 I/O functions for knowledge spaces on items

23.1 aopeni_space

NAME

aopeni_space — Open a spacefile in ASCII format for reading

SYNOPSIS

```
#include <srbi-space-io.h>

extern FILE *aopeni_space
(
    const char *filename,
    int *q_size,
    int *s_size
);
```

DESCRIPTION

Open a spacefile in ASCII format for reading.

23.2 aopeno_space

NAME

aopeno_space — Open a spacefile in ASCII format for writing

SYNOPSIS

```
#include <srbi-space-io.h>

extern FILE *aopeno_space
(
    const char *filename,
    int q_size,
    int s_size
);
```

DESCRIPTION

Open a spacefile in ASCII format for writing.

23.3 ard_space

NAME

ard_space — Read a complete knowledge space in ASCII format.

SYNOPSIS

```
#include <srbi-space-io.h>

extern space *ard_space(const char *filename);
```

DESCRIPTION

Read a complete knowledge space in ASCII format.

23.4 awr_space

NAME

awr_space — Write a complete knowledge space in ASCII format.

SYNOPSIS

```
#include <srbi-space-io.h>

extern int awr_space
(
    const char *filename,
    space *s
);
```

PARAMETERS

const char *filename
Name of spacefile.

space *s
Pointer to space structure.

DESCRIPTION

Write a complete knowledge space in ASCII format.

23.5 bopeni_space

NAME

bopeni_space — Open a spacefile in binary format for reading

SYNOPSIS

```
#include <srbi-space-io.h>

extern FILE *bopeni_space
(
    const char *filename,
    int *q_size,
    int *s_size
);
```

DESCRIPTION

Open a spacefile in binary format for reading.

23.6 bopeno_space

NAME

bopeno_space — Open a spacefile in binary format for writing

SYNOPSIS

```
#include <srbi-space-io.h>

extern FILE *bopeno_space
(
    const char *filename,
    int q_size,
    int s_size
);
```

DESCRIPTION

Open a spacefile in binary format for writing.

23.7 brd_space

NAME

brd_space — Read a complete knowledge space in binary format.

SYNOPSIS

```
#include <srbi-space-io.h>

extern space *brd_space(const char *filename);
```

DESCRIPTION

Read a complete knowledge space in binary format.

23.8 bwr_space

NAME

bwr_space — Write a complete knowledge space in binary format.

SYNOPSIS

```
#include <srbi-space-io.h>

extern int bwr_space
(
    const char *filename,
    space *s
);
```

PARAMETERS

const char *filename
: Name of spacefile.

space *s
: Pointer to space structure.

DESCRIPTION

Write a complete knowledge space in binary format.

24 I/O functions for bases on items

24.1 openi_basis

NAME

openi_basis — Open a basisfile for reading

SYNOPSIS

```
#include <srbi-space-io.h>

extern FILE *openi_basis
(
    const char *filename,
    int *q_size,
    int *b_size
);
```

DESCRIPTION

Open a basisfile for reading.

24.2 openo_basis

NAME

openo_basis — Open a basisfile for writing

SYNOPSIS

```
#include <srbi-space-io.h>
```

```
extern FILE *openo_basis
(
    const char *filename,
    int q_size,
    int b_size
);
```

DESCRIPTION

Open a basisfile for writing.

24.3 rd_basis

NAME

rd_basis — Read a complete basisfile.

SYNOPSIS

```
#include <srbi-space-io.h>

extern basis *rd_basis(const char *filename);
```

PARAMETERS

const char *filename
Name of basisfile.

DESCRIPTION

Read a complete basisfile.

24.4 read_basis_element

NAME

read_basis_element — For the basis files we need topic specific reading routines,

SYNOPSIS

```
#include <srbi-space-io.h>

extern int read_basis_element
(
    FILE *in,
    bitset minimal,
    bitset nonminimal,
    int q_size
);
```

DESCRIPTION

I.e. Functions to determine whether a set is minimal for an item or not. The result is just an error code.

24.5 wr_basis

NAME

wr_basis — Write a complete basisfile.

SYNOPSIS

```
#include <srbi-space-io.h>

extern int wr_basis
(
    const char *filename,
    basis *b
);
```

PARAMETERS

const char *filename
Name of basisfile.

basis *b
Pointer to basis structure.

DESCRIPTION

Write a complete basisfile.

24.6 write_basis_element

NAME

write_basis_element — Similar for writing

SYNOPSIS

```
#include <srbi-space-io.h>

extern int write_basis_element
(
    FILE *out,
    bitset minimal,
    bitset nonminimal,
    int q_size
);
```

DESCRIPTION

Similar for writing.

25 Basic I/O functions

25.1 `read_int`

NAME

read_int — Read an integer value from binary stream with endian corrections.

SYNOPSIS

```
#include <srbi-space-io.h>

int read_int
(
    FILE *f,
    structtype endian
);
```

PARAMETERS

FILE *f
File to read from.

structtype endian
Do we have to change endian?.

DESCRIPTION

Read an integer value from binary stream with endian corrections.

25.2 read_bb

NAME

read_bb — Read a bitset_basis value from binary stream with endian corrections.

SYNOPSIS

```
#include <srbi-space-io.h>

bitset_basis read_bb
(
    FILE *f,
    structtype endian
);
```

PARAMETERS

FILE *f
File to read from.

structtype endian
Do we have to change endian?.

DESCRIPTION

Read a bitset_basis value from binary stream with endian corrections.

25.3 a_read_state

NAME

a_read_state — Read a state from an ASCII file.

SYNOPSIS

```
#include <srbi-space-io.h>

int a_read_state
```

```
(
    FILE *f,
    bitset bs,
    int q_size
);
```

PARAMETERS

FILE *f
File to read from.

bitset bs
Already allocated bitset.

int q_size
Size of set.

DESCRIPTION

Read a state from an ASCII file.

RETURNS

Error code.

25.4 b_read_state

NAME

b_read_state — Read a state from a binary file.

SYNOPSIS

```
#include <srbi-space-io.h>

int b_read_state
(
    FILE *f,
    bitset bs,
```

```
    int q_size,
    structtype endian
);
```

PARAMETERS

FILE *f
File to read from.

bitset bs
Already allocated bitset.

int q_size
Size of set.

structtype endian
Do we have to change endian?.

DESCRIPTION

Read a state from a binary file.

RETURNS

Error code.

26 Not yet really in order within documentation

26.1 `read_patterns_element`

NAME

read_patterns_element — For the files with patterns we need topic specific reading routines,

SYNOPSIS

```
#include <srbi-space-io.h>

extern int read_patterns_element
(
    FILE *in,
    bitset solved,
    bitset unanswered,
    int q_size
);
```

DESCRIPTION

I.e. Functions to determine whether an item was not solved correctly or not answered. The result is just an error code.

26.2 write_patterns_element

NAME

write_patterns_element — Similar for writing

SYNOPSIS

```
#include <srbi-space-io.h>

extern int write_patterns_element
(
    FILE *out,
    bitset solved,
    bitset unanswered,
    int q_size
);
```

DESCRIPTION

Similar for writing.

26.3 read_item_info

NAME

read_item_info — Functions for reading item- and line- infos from a file

SYNOPSIS

```
#include <srbi-space-io.h>

long read_item_info
(
    FILE *f,
    int **info,
    int q,
    char *buf
);
```

DESCRIPTION

Functions for reading item- and line- infos from a file.

Appendix

Please note that documentation in the appendices A and B is not updated (semi–) automatically with the source (opposed to the previous manpages and to Appendix D). In any case of contradiction, the manpages are probably more accurate and up–to–date. Please note further that Appendix C is explicitly partially superseded by new developments and that it is included here only because these parts have not yet been newly documented.

A Data structures

A.1 Bitsets

The `srbi-set.h` file defines two data types for bitsets:

```
typedef unsigned int bitset_basis;
typedef bitset_basis *bitset;
```

These data types simply serve to ensure an independency from the word size of the hardware used.

A.2 Knowledge structures and surmise systems

A.2.1 Structure type

The type `structtype` is used to describe the type of a structure. Currently, there exist seven different types: bases, knowledge spaces, knowledge structures, surmise relations, data, disjoint partition, and partition[1]: The `internal/filetype.h` header file defines

```
typedef unsigned int structtype;
```

together with the following types:

```
#define UNKNOWN          0
#define BASIS            1
#define SPACE            2
#define STRUCTURE        3
#define RELATION         4
#define DATA             5
#define DISJ_PART        6
#define PARTITION        7
```

[1]The structtypes *disjoint partition* and *partition* are defined for potential use within the to–be–written SRBT–Library.

Thus, it is possible to define new structure types, easily.

! **Note** that the structype numbers must be limited to a maximal value of 254 due to the use of `structtype` in the `filetype` encoding (cf. App. A.2.2). The value of `READ_ERROR`=255 is used as a virtual structure (and file) type, e. g. for non–existant files.

A.2.2 Filetype

Files are specified by their contents and their encoding. The `filetype` datatype is defined as

```
typedef unsigned int filetype;
```

in the `internal/filetype.h` header file.

The `filetyp` is to be regarded as a bitmask. The lower eight bits are reserved for storing the content type by means of `structtype` codes (cf. App. A.2.1). The upper bits describe the encoding. Currently, the following encoding parameters are defined:

```
#define OLD_FORMAT    0x0100
#define BINARY_FILE   0x0200
#define BIG_ENDIAN    0x0400
```

Further encoding descriptors should be defined in as way to use the `filetype` as a bistmask.

A.2.3 Knowledge space

Knowledge spaces are represented by a `struct` containing the number of items and the number of knowledge states and a matrix, internally represented as a vector, where the knowledge states are stored.

```
typedef struct {
  int          q_size;         /* Number of items */
  int          s_size;         /* Number of states */
  bitset_basis *matrix;        /* Binary data matrix */
} space;
```

A.2.4 Knowledge structure and data set

Knowledge structures and data (i. e. answer patterns) are in fact the same structures as knowledge spaces. The are simply defined separately to facilitate clear distinctions in the progress of programming.

```
typedef struct {
  int          q_size;         /* Number of items */
  int          s_size;         /* Number of states */
  bitset_basis *matrix;        /* Binary data matrix */
} structure;


typedef struct {
  int          q_size;         /* Number of items */
  int          d_size;         /* Number of states */
  bitset_basis *matrix;        /* Binary data matrix */
} data;
```

A.2.5 Basis

The basis structure is similar to the structures already introduced above. The matrix containing the basis elements, however, is split up into two matices distinguishing whether a basis element is minimal or not for a contained item.

```
typedef struct {
  int          q_size;         /* Number of items */
  int          b_size;         /* Number of states */
  bitset_basis *minimal;       /* Matrix of minimal items */
  bitset_basis *nonminimal;    /* Matrix of non-minimal items */
} basis;
```

A.2.6 Surmise Relation

Surmise relations can be stored as a quadratic matrix describing whether or not an item i is a prerequisite for an item j. Since the size of this matrix is prespecified as the square of the number of items, we can use a bytewise storage instead of a bitwise one. The idea for this structure was developed based on the `dep-matrix` program. The choice of `char` as basic type for `matrix` is only a question of memory usage. The cells of the matrix contain `'\0'` or `'\1'`, and not `'0'` and `'1'`, respectively.

```
typedef struct {
  int          q_size;         /* Number of items */
  char         **matrix;       /* Matrix of relation */
} srbi;
```

B Additional man pages

This Annex contains additional manpages describing general principles which have been written manually and have not been produced as an automatic excerpt from the source code.

B.1 srbifile — SRBI File formats in the SRBT project

Last changed: 27 July 1999

B.1.1 Synopsis

#SRBT `[version] [structtype] [encoding] [endian] [wordsize] [comment]`

B.1.2 Description

This manpage describes the format and header of files for surmise relations between items (SRBI). With the `libsrbi` library, the default file formats have changed by an additional header line (also for binary files) containing all information necessary for interpreting the file contents. The main corpus, however, keeps unchanged as described in the basisfile (Sect. C.1, p. 151), patternfile (Sect. C.3, p. 153) , and spacefile (Sect. C.2, p. 152) manpages.

There are some new filetypes now: The former `patternfile` is now called `datafile`. The term `patternfile` is now used for a filetype diffrentiating between un-answered and falsely answered items. While the latter are encoded through a '0' (zero), the former are encoded through an 'X'.

Additionally, the `relationfile` was introduced as a completely new filetype. A `relationfile` is always in ASCII format. After the format header and possible comments (see below) there comes a line with the number of items followed by a quadratic

matrix with one row and column for each item. The cell at the *i* th row and *j* th column contains a '1' if item *j* is a prerequisite of item *i* and a '0' otherwise.

B.1.3 Usage

#SRBT `[version] [structtype] [encoding] [endian] [wordsize] [comment]`

#SRBT This string at the very beginning of the first line of a file denotes that the file is for use with the `libsrbi` or `libsrbt` libraries from the SRBT project.

version Version number of the file format.

structtype This string describes the type of the data contained in the file. Six structtypes are currently supported in the SRBI library:
`basisfile`
`spacefile`
`structurefile`
`relationfile`
`datafile`
`patternfile`

Since basisfiles and patternfiles are always stored as ASCII text files, the following, optional file header fields are only valid for datafiles, spacefiles, and structurefiles.

encoding The `encoding` information specifies whether the data are stored in `ASCII` or in `binary` form.

endian The `endian` information specifies (for binary files only) whether the storing computer has a `LITTLE` or a `BIG` endian processor. Default is `BIG` endian.

wordsize States are stored as a multiple of `wordsize` bits. The default value is 32.

comment The possibility to specify a comment is primarily provided for use with the ASCII files. The first line may contain a comment separated by a hash sign (#). Subsequent lines beginning with a hash sign (#) may also contain comments. This also holds if the first line does not contain a comment. **Note** that all comments must be specified before starting the *real* data.

For **binary** files, the format header line and optional comments are closed by a `NUL` (i.e. `0x00`) character.

B.1.4 Remarks

The new *SRBI* file format was developed in order to ensure that users do not mingle different file type specifications. This used to become a major problems with users having only little computer experience.

If SRBI files are created manually, one should provide as many header information as possible.

Any tools should should be able also to read files in the old format. If a file of the wrong type (in the new format) is passed to a program, it may either be rejected (with an appropriate error message) or converted (optionally issuing an additional warning that a file of wrong type was passed over).

B.1.5 Warning

Please note that the introductionary string #SRBT is used for `srbifiles`, too.

B.1.6 See also

basisfile (Sect. C.1, p. 151), patternfile (Sect. C.3, p. 153) , srbtfile (Sect. B.2, p. 147) spacefile (Sect. C.2, p. 152)

B.2 srbtfile — SRBT File formats

Last changed: 5 May 1999

B.2.1 Synopsis

#SRBT `[version] [structtype] [encoding] [endian] [wordsize] [comment]`

B.2.2 Description

This manpage describes extensions of the SRBI file format srbifile (Sect. B.1, p. 145). The main extension lies within the definition of additional content types.

B.2.3 Usage

#SRBT `[version] [structtype] [encoding] [endian] [wordsize] [comment]`

#SRBT This string at the very beginning of the first line of a file denotes that the file is for use with the `libsrbt` or `libsrbi` libraries.

version Version number of the file format.

structtype This string describes the type of the data contained in the file. Seven structtype are currently supported:
`basisfile`
`spacefile`
`structurefile`
`relationfile`
`datafile`
`disjointpartition`
`partition`

Since basisfiles are always stored as ASCII text files, the following, optional file header fields are only valid for patternfiles, spacefiles, and structurefiles. Currently, it has not been decided which encodings will be valid for partitions.

encoding The `encoding` information specifies whether the data are stored in `ASCII` or in `binary` form.

endian The `endian` information specifies (for binary files only) whether the storing computer has a `LITTLE` or a `BIG` endian processor. Default is `BIG` endian.

wordsize States are stored as a multiple of `wordsize` bits. The default value is 32. Specification of `wordsize` requires a specification of `endian`.

comment The possibility to specify a comment is primarily provided for use with the ASCII files. The first line may contain a comment separated by a hash sign (#). Subsequent lines beginning with a hash sign (#) may also contain comments. **Note** that all comments must be specified before starting the *real* data.

For **binary** files, the format header line and optional comments are closed by a `NUL` (i.e. `0x00`) character.

B.2.4 Remarks

The new *SRBT* file format was developed in order to ensure that users do not mingle different file type specifications. This used to become a major problems with users having only little computer experience.

If SRBT files are created manually, one should provide as many header information as possible.

Any tools should should be able also to read files in the old format. If a file of the wrong type (in the new format) is passed to a program, it may either be rejected (with an appropriate error message) or converted (optionally issuing an additional warning that a file of wrong type was passed over).

B.2.5 See also

basisfile (Sect. C.1, p. 151), patternfile (Sect. C.3, p. 153) , srbifile (Sect. B.1, p. 145) , spacefile (Sect. C.2, p. 152)

C Old formats

This Annex contains manpages for the file formats used with the old `libkst` library and the knowledge space tools. They are included here since they are mentioned within the manpages of App. B. The patternfile described below corresponds to the datafiles in the SRBI library. !!

C.1 basisfile — Format of basisfiles (v1.0)

Last changed: 11 July 1996

C.1.1 Description

A `basisfile` is an ASCII file describing the basis of a knowledge space. It has the following format:

The first line contains the number of items in the knowledge domain.

The second line contains the number of basis elements.

The following lines contain the basis elements building a matrix where the columns describe the items and the rowsdescribe the basis elements. In each cell of this matrix there is a '0' if the basis element does not contain the item, a '1' if it is a clause for the item and a '2' otherwise.

For any set of knowledge states, a basis according to this specification can be computed with the basis (1K) program.

C.1.2 Version information

This manpage describes version v1.0 of the basisfile format. The format changes in v2.0 by additional format and meta information header lines (see srbifile (Sect. B.1, p. 145)).

C.1.3 See also

basis (1K), patternfile (Sect. C.3, p. 153) , spacefile (Sect. C.2, p. 152) , srbifile (Sect. B.1, p. 145)

C.2 spacefile — Format of spacefiles (v1.0)

Last changed: 04 June 1996

C.2.1 Description

There are two different possibilities to store a `spacefile` — as an ASCII file or as a binary file. Both types describe knowledge spaces in a very similar manner:

ASCII file format

The first line contains the number of items in the knowledge domain. The second line contains the number of knowledge states.

The following lines contain the knowledge states building a matrix where the columns describe the items and the rows describe the knowledge states. In each cell of this matrix there is a '1' if the knowledge state does contain the item, and a '0' otherwise.

Binary file format

The file contains a sequence of `long integer` numbers. The first two numbers give the number of items and the number of knowledge states. The following `long integers` build bitsets, one per knowledge state. A bitset consists of as many `long integers` as are needed to represent the item set. This number of `long integers` needed can be computed as `(ItemNo + BitsPerLong - 1) / BitsPerLong` where `BitsPerLong` is the machine specific number of bits used to store a `long integer` number.

C.2.2 Warning

Using a binary `spacefile` on different hardware platforms may produce unexpected results since there may be different byte orders and therefore different bit orders.

C.2.3 Version information

This manpage describes version v1.0 of the basisfile format. The format changes in v2.0 by additional format and meta information header lines (see srbifile (Sect. B.1, p. 145)).

C.2.4 See also

basisfile (Sect. C.1, p. 151), patternfile (Sect. C.3, p. 153) , srbifile (Sect. B.1, p. 145)

C.3 patternfile — Format of answer pattern files (v1.0)

Last changed: 04 June 1996

C.3.1 Description

Answer pattern files have the same format as knowledge space files. There are two different possibilities to store a `patternfile` — as an ASCII file or as a binary file. Both types describe knowledge spaces in a very similar manner:

ASCII file format

The first line contains the number of items in the knowledge domain. The second line contains the number of answer patterns.

The following lines contain the knowledge states building a matrix where the columns describe the items and the rows describe the answer patterns. In each cell of this matrix there is a '1' if the answer pattern does contain the item, and a '0' otherwise.

Binary file format

The file contains a sequence of `long integer` numbers. The first two numbers give the number of items and the number of knowledge states. The following `long integers` build bitsets, one per answer pattern. A bitset consists of as many `long integers` as are needed to represent the item set. This number of `long integers` needed can be computed as `(ItemNo + BitsPerLong - 1) / BitsPerLong` where `BitsPerLong` is the machine specific number of bits used to store a `long integer` number.

C.3.2 Warning

Using a binary `patternfile` on different hardware platforms may produce unexpected results since there may be different byte orders and therefore different bit orders.

C.3.3 Version information

This manpage describes version v1.0 of the basisfile format. The format changes in v2.0 by additional format and meta information header lines (see srbifile (Sect. B.1, p. 145)).

C.3.4 See also

basisfile (Sect. C.1, p. 151), spacefile (Sect. C.2, p. 152) , srbifile (Sect. B.1, p. 145)

D Interfaces — C header files

This appendix contains — as a final reference — the C header files which build the interface between the `libsrbi` library and the programmer applying and extendingthe library.

D.1 srbi-set.h

```
/* srbi-set.h - functions and datatypes for work with sets */

/* This library was developed within the Project "Surmise
 * Relations Between Tests" (SRBT) at the Department of
 * Psychology, University of Graz, Austria. It is mainly
 * based on an older library developed at the Section
 * of Mathematical and Social Psychology, University of
 * Technology, Braunschweig, Germany.
 *
 * (C) 1995, 1998 Cord Hockemeyer (CHockemeyer@acm.org)
 *
 */
#ifndef _SRBI_SET_H_INCLUDED_
#define _SRBI_SET_H_INCLUDED_

#ifdef __cplusplus
extern "C" {
#endif



/*
 * The following line is machine dependent.
 */
typedef unsigned int bitset_basis;
```

```
typedef bitset_basis *bitset;


#define BITS_PER_BB    32L /** Bits used for a integer      **/
                           /** Should be a power of 2       **/
#define LD_BPB          5L /** Binary logarithm of BPB      **/
#define BPB_M1         (BITS_PER_BB - 1L)
                           /** Should be a binary 11...11 **/
#define BYTES_PER_BB   (sizeof(bitset_basis))



/**
*** Functions creating and deleting extended bitsets
**/

/* Create a new and empty bitset */
extern bitset new_bitset
    (int set_size /* Size of the set to be created   */
);

/* Delete a bitset and return storage to the system */
extern void delete_bitset
    (bitset a      /* Bitset to be deleted             */
);

/* Make a copy of a bitset. New memory is allocated. */
extern bitset copy_bitset
    (bitset orig, /* Original bitset                  */
     int set_size /* Size of the bitset               */
);

/* Copy a bitset into another bitset */
extern int set_bitset
    (bitset copy,     /* Destination bitset */
     bitset orig,      /* Original bitset     */
     int set_size     /* Size of the bitset */
```

```
);

/* Create a new bitset with randomly selected elements.
 * This function uses stdlib's rand() routine. Therefore,
 * it can be influenced through stdlib's srand() routine.
 */
extern bitset random_set
    (int set_size /* Size of the bitset                */
);


/**
*** I/O functions
***
*** The *read()- functions allocate the memory used for their
*** results.
**/
#include <stdio.h>

/* Read a bitset from an ASCII file.
   New memory is allocated.          */
extern bitset ascii_read(FILE *in, int set_size);

/* Write a bitset to an ASCII file. */
extern int ascii_write(FILE *out, bitset a, int set_size);

/* Read a bitset from an binary file.
   New memory is allocated.          */
extern bitset bin_read(FILE *in, int set_size);

/* Write a bitset to an binary file. */
extern int bin_write(FILE *out, bitset a, int set_size);

/**
*** Functions for the connection of extended bitsets
*/

/* Compute the intersection of two bitsets with a function */
extern bitset section(bitset a, bitset b, int set_size);
```

```
/* Compute the union of two bitsets with a function */
extern bitset set_union(bitset a, bitset b, int set_size);

/* Compute the set difference of two bitsets with a function */
extern bitset set_diff(bitset a, bitset b, int set_size);

/* Compute the symmetrical set difference of two bitsets with a function
extern bitset symm_diff(bitset a, bitset b, int set_size);


/* Compute the intersection of two bitsets with a procedure */
extern void comp_section(bitset result, bitset a,
                         bitset b, int set_size);

/* Compute the union of two bitsets with a procedure */
extern void comp_union(bitset result, bitset a,
                           bitset b, int set_size);
/* Compute the set difference of two bitsets with a procedure */

extern void comp_set_diff(bitset result, bitset a,
                          bitset b, int set_size);

/* Compute the symmetrical set difference of two bitsets with a procedure
extern void comp_symm_diff(bitset result, bitset a,
                           bitset b, int set_size);


/* Compute the accumulated intersection of two bitsets */
extern void acc_section(bitset result, bitset op, int set_size);

/* Compute the accumulated union of two bitsets */
extern void acc_union(bitset result, bitset op, int set_size);

/* Compute the accumulated set difference of two bitsets */
extern void acc_set_diff(bitset result, bitset op, int set_size);

/* Compute the accumulated symmetrical set difference of two bitsets */
extern void acc_symm_diff(bitset result, bitset op, int set_size);
```

```
/**
*** Functions to extend/reduce extended bitsets
**/

/* Add an item to a bitset */
void set_incl(int item, bitset a, int set_size);

/* Delete an item from a bitset */
void set_excl(int item, bitset a, int set_size);

/* Determine the complement of a bitset */
extern bitset complement(bitset orig, int set_size);

/**
*** Predicates for extended bitsets
**/

/* Test if a bitset contains an item */
extern int is_element(int item, bitset a, int set_size);

/* Test if a set is empty */
extern int emptyset(bitset a, int set_size);

/* Test if two subsets are equal */
extern int equal(bitset a, bitset b, int set_size);

/* Test if a bitset is a subset of another */
extern int subset(bitset a, bitset b, int set_size);

/* Test if a bitset is subset of or equal to another */
extern int subseteq(bitset a, bitset b, int set_size);

/**
*** Properties of extended bitsets
**/

/* Determine the cardinality, i.e. the number of elements of a bitset */
```

```
extern int cardinality(bitset a, int set_size);

/* Determine the size of the symmetrical set difference of two bitsets */
extern int symm_diff_size(bitset a, bitset b, int set_size);

/* Determine the size of the ordinary set difference of two bitsets */
extern int set_diff_size(bitset a, bitset b, int set_size);

/* Determine the size of the intersection of two bitsets */
extern int section_size(bitset a, bitset b, int set_size);

/* Determine the size of the union of two bitsets */
extern int union_size(bitset a, bitset b, int set_size);



#ifndef TRUE
#define TRUE 1
#define FALSE 0
#endif


/**
*** Some useful macros for determining bitpositions,
*** bitset sizes etc.
***
*** word_no(i) is the number of words needed to store i bits.
*** word_cnt(i) ist the number of the word where bit i is stored.
*** bit_pos(i) is the bitposition of bit i in bitset[wordcnt(i)].
*** bit_mask(i) is the bitmask to set/test/... bit i.
**/
#define word_no(i)   (((i) + BPB_M1) >> LD_BPB)
#define word_cnt(i) ((i) >> LD_BPB)
#define bit_pos(i)   ((i) & BPB_M1)
#define bit_mask(i) (1L << bit_pos(i))


#ifdef __cplusplus
```

```
}
#endif

#endif
```

D.2 srbi-space.h

```
/* srbi-space.h - functions and datatypes for work with knowlege spaces */

/* This library was developed within the Project "Surmise
 * Relations Between Tests" (SRBT) at the Department of
 * Psychology, University of Graz, Austria. Partially it
 * is based on an older library developed at the Section
 * of Mathematical and Social Psychology, University of
 * Technology, Braunschweig, Germany.
 *
 * (C) 1995, 1998 Cord Hockemeyer (CHockemeyer@acm.org)
 */

#ifndef _SRBI_SPACE_H_INCLUDED_
#define _SRBI_SPACE_H_INCLUDED_

#include <stdio.h>
#include <srbi-set.h>
#include <filetype.h>

#ifdef __cplusplus
extern "C" {
#endif


typedef struct {
  int           q_size;          /* Number of items */
  int           s_size;          /* Number of states */
  bitset_basis *matrix;          /* Binary data matrix */
  int**        item_info;       /* Information numbers for each item */
  int**        line_info;       /* Information numbers for each line */
} space;

typedef struct {
```

```
  int           q_size;          /* Number of items */
  int           s_size;          /* Number of states */
  bitset_basis *matrix;          /* Binary data matrix */
  int**        item_info;        /* Information numbers for each item */
  int**        line_info;        /* Information numbers for each line */
} structure;

typedef struct {
  int           q_size;          /* Number of items */
  int           b_size;          /* Number of states */
  bitset_basis *minimal;         /* Matrix of minimal items */
  bitset_basis *nonminimal;      /* Matrix of non-minimal items */
  int**        item_info;        /* Information numbers for each item */
  int**        line_info;        /* Information numbers for each line */
  int           ordered;         /* Is this basis ordered by size? */
} basis;

typedef struct {
  int           q_size;          /* Number of items */
  char          **matrix;        /* Matrix of relation */
  int**        item_info;        /* Information numbers for each item */
} srbi;

typedef struct {
  int           q_size;          /* Number of items */
  int           d_size;          /* Number of states */
  bitset_basis *matrix;          /* Binary data matrix */
  int**        item_info;        /* Information numbers for each item */
  int**        line_info;        /* Information numbers for each line */
} data;

typedef struct {
  int           q_size;          /* Number of items */
  int           p_size;          /* Number of patterns */
  bitset_basis *solved;          /* Matrix of solved items */
  bitset_basis *un_answered;     /* Matrix of un-answered items */
  int**        item_info;        /* Identification number(s) for each item */
  int**        line_info;        /* Identification number(s) for each patter
```

```
} patterns;


typedef struct {
  int          q_size;          /* Number of items */
  char**       matrix;          /* Matrix of relation */
  int**        item_info;       /* Information numbers for each item */
} ihypoth;

/* load a basis from a file
 * This function loads a basis from a file. It determines
 * automatically which type of data is stored in the file
 * and performs a squeeze operation on the data if necessary.
 * Returns: Pointer to resulting basis.
 */
basis *load_basis(
  const char filename[]     /* Name of the file to be loaded */
);

/* load a space from a file
 * This function loads a space from a file. It determines
 * automatically which type of data is stored in the file
 * and performs a constr operation on the data if necessary.
 * Returns: Pointer to resulting space.
 */
space *load_space(
  const char filename[]     /* Name of the file to be loaded */
);

/* load a structure from a file
 * This function loads a structure from a file.
 * Returns: Pointer to resulting structure.
 */
structure *load_structure(
  const char filename[]     /* Name of the file to be loaded */
);

/* load a data set from a file
 * This function loads a data set from a file.
 * Returns: Pointer to resulting data set.
 */
```

```
data *load_data(
  const char filename[]     /* Name of the file to be loaded */
);


/* load a surmise relation from a file
 * This function loads a surmise relation from a file.
 * Returns: Pointer to resulting surmise relation.
 */
srbi *load_srbi(
  const char filename[]     /* Name of the file to be loaded */
);

/* write a basis to a file
 * This function writes a basis to a file using the new
 * basisfile format.
 * Returns: Error code.
 */
int save_basis(
  basis *b,                 /* Basis to be stored */
  filetype mode,            /* Format to be used */
  const char filename[]     /* Name of the file to be saved */
);

/* write a knowledge space to a file
 * This function writes a knowledge space to a file using
 * the new spacefile format.
 * Returns: Error code.
 */
int save_space(
  space *s,           /* Space to be stored */
  filetype mode,      /* Format to be used  */
  const char filename[]     /* Name of the file to be saved */
);

/* write a knowledge structure to a file
 * This function writes a knowledge structure to a file using
 * the new structurefile format.
 * Returns: Error code.
 */
```

```
int save_structure(
  structure *s,        /* Structure to be stored */
  filetype mode,       /* Format to be used   */
  const char filename[]     /* Name of the file to be saved */
);

/* write a data set to a file
 * This function writes a data set to a file using
 * the new datafile format.
 * Returns: Error code.
 */
int save_data(
  data *s,             /* Data to be stored */
  filetype mode,       /* Format to be used   */
  const char filename[]     /* Name of the file to be saved */
);

/* write a surmise relation to a file
 * This function writes a surmise relation to a file using
 * the new relationfile format.
 * Returns: Error code.
 */
int save_srbi(
  srbi *r,             /* Surmise relation to be stored */
  const char filename[]     /* Name of the file to be saved */
);

/* write a surmise relation as a matrix
 * This function writes the matrix of a surmise relation to a
 * stream. No header information are written.
 * Returns: Error code.
 */
int write_srbi(
  FILE *f,             /* FILE to write into */
  srbi *m              /* Matrix to be written */
);



/* Allocate memory for a knowledge space. */
```

```
extern space *new_space(int q_size,   /* Number of items */
                        int s_size    /* Number of states */
);

/* Return memory used by a space to the system. */
extern void free_space(space **s);

/* Make a copy of an existing knowledge space. */
extern space *copy_space(space *orig);

/* Copy a single knowledge state from a knowledge space.
 * New memory is allocated.
 */
extern bitset copy_state(
  space *s,     /* Space from which the state is to be copied */
  int position  /* Index of the state to be copied */
);

/* Get a single knowledge state from a knowledge space.
 * No new memory is allocated.
 */
extern bitset get_state(
  space *s,     /* Space from which the state is to be copied */
  int position  /* Index of the state to be copied */
);

/* Set a knowledge state within a knowledge space. */
extern int set_state(
  space *s,        /* Knowledge space                */
  int position,    /* Index of the state to be set */
  bitset b         /* New value of the state         */
);


/* Allocate memory for a knowledge structure. */
extern structure *new_structure(
  int q_size,   /* Number of items */
  int s_size    /* Number of states */
```

```
);

/* Return memory used by a structure to the system. */
extern void free_structure(structure **s);

/* Make a copy of an existing knowledge structure. */
extern structure *copy_structure(structure *orig);

/* Allocate memory for a data set. */
extern data *new_data(int q_size,   /* Number of items */
                      int s_size    /* Number of patterns */
);

/* Return memory used by a data set to the system. */
extern void free_data(data **s);


/* Create a random data set. All subsets of items are equally
probable. Repetitions are possible.                          */
extern data *random_data(int q_size,     /* Number of items */
                         int s_size      /* Number of patterns */
);

/* Make a copy of an existing data set. */
extern data *copy_data(data *orig);

/* Copy a single answer pattern from a data set.
 * New memory is allocated.
 */
bitset copy_pattern(
  data *s,        /* Data set from which the answer pattern is to be copied */
  int  position   /* Index of the answer pattern to be copied */
);

/* Get a single answer pattern from a data set.
 * No new memory is allocated.
 */
extern bitset get_pattern(
```

```
  data *d,      /* Data set from which the answer pattern is to be copied
  int position  /* Index of the answer pattern to be copied */
);

/* Set a answer pattern within a data set space. */
extern int set_pattern(
  data *d,        /* Data set                                    */
  int position,   /* Index of the answer pattern to be set */
  bitset b        /* New value of the answer pattern             */
);

/* Allocate memory for a basis. */
extern basis *new_basis(int q_size, /* Number of items */
                        int b_size  /* Number of clauses */
);

/* Return memory used by a basis to the system. */
extern void free_basis(basis **b);

/* Make a copy of an existing basis. */
extern basis *copy_basis(basis *orig);

/* Allocate memory for a surmise relation. */
extern srbi *new_srbi(int q_size); /* Number of items */

/* Return memory used by a surmise relation */
extern void free_srbi(srbi **r);



/* close a basis under union
 * This function computes the closure under union of the basis
 * specified. Currently, Dowling's (1993) algorithm is used.
 * Returns: Pointer to resulting knowledge space.
 */
space *constr_basis(
  basis *b            /* Basis to be closed */
```

```
);

/* close a structure under union
 * This function computes the closure under union of the structure
 * specified. Currently, Dowling's (1993) algorithm is used.
 * Returns: Pointer to resulting knowledge space.
 */
space *constr_structure(
  structure *s              /* Structure to be closed */
);

/* close a basis under union using the Ganter algorithm
 * This function takes a basis and computes its closure under
 * union using the algorithm developed by Ganter. The result
 * is directly stored into a file.
 * Returns: Number of states of the resulting knowledge space.
 *          Negative in case of an error.
 * Warning: Not yet implemented!
 */
  int ganter_basis(
  basis *b,                 /* Basis to be closed             */
  filetype mode,            /* File format                    */
  const char filename[]     /* Filename for resulting space */
);

/* close a structure under union using the Ganter algorithm
 * This function takes a structure and computes its closure under
 * union using the algorithm developed by Ganter. The result
 * is directly stored into a file.
 * Returns: Number of states of the resulting knowledge space.
 *          Negative in case of an error.
 * Warning: Not yet implemented!
 */
int ganter_structure(
  structure *s,             /* Structure to be closed         */
  filetype mode,            /* File format                    */
  const char filename[]     /* Filename for resulting space */
);

/* determine basis of a structure
 * This function computes the minimal elements of a given
 * knowledge structure. The set of minimal elements is the
```

```
 * basis of the space which would be the result of closing
 * the structure under union.
 * Returns: Pointer to the resulting basis.
 */
basis *squeeze(
  structure *s          /* Knowledge structure to be squeezed */
);

/* determine SRBI matrix from a basis
 * This function takes a basis, checks whether it describes a
 * surmise relation, and stores it as a SRBI structure.
 * returns pointer to the resulting SRBI structure. NULL if the
 * structure cannot be generated, e.g. if the basis does not
 * describe a surmise relation.
 */
srbi *dep_matrix(
  basis *b              /* Basis describing the surmise relation */
);

/* Reduce a knowledge space to a subset of items */
extern space *mask_space(space  *s,  /* Original space */
                         bitset mask /* Item set of reduced space */
);

/* Reduce a data set to a subset of items */
extern data *mask_data(data   *d,  /* Original data set */
                       bitset mask /* Item set of reduced patterns */
);

/* Reduce a basis to a subset of items */
extern basis *mask_basis(basis  *b,  /* Original basis */
                         bitset mask /* Item set of reduced basis */
);

/* Compute the size of a clause
 *
 * Clauses, i.e. knowledge states which are members of the basis,
 * are partitioned into two subsets: the set of items for which
 * the clause is minimal and the set of items which are elements
 * of the clause but for which the clause is not minimal. Therefore
```

```
 * we have specific functions in the spacelib.
 */
int  /* Return codes:
      * >= 0    Size of the clause
      * -1      basis is NULL pointer
      * -2      index > basis->b_size
          */
clause_cardinality
(
    basis *b,     /* The basis where the clause is found in  */
    int index     /* The position of the clause in the basis */
);

/* Determine a clause

 * Clauses, i.e. knowledge states which are members of the basis,
 * are partitioned into two subsets: the set of items for which
 * the clause is minimal and the set of items which are elements
 * of the clause but for which the clause is not minimal. Therefore
 * we have specific functions in the spacelib.
 */
extern bitset get_clause
(
    basis *b,/* The basis where the clause is found in  */
    int index            /* The position of the clause in the basis */
);


/* Determine the set of states of a basis
 *
 * Allocate memory for a simplespace and fill it with the complete
 * clauses which are given in partitioned form in the basis_struct
 * structure.
 */
structure *clause_set(basis *b);




/* Compute a wellgraded basis
 * This fuction computes the basis of a wellgraded space which
```

```
 * is a superset of the space described through the basis given
 * as a parameter. Wellgradedness can not be reached through a
 * closure operator since, for a non-wellgraded knowledge space,
 * there does not exist a \emph{unique} smallest wellgraded
 * knowledge space as a superset.
 *
 * Here, the decision is to partition basis elements causing
 * non-wellgradedness into a \emph{prerequisite subset} and a
 * \emph{new items} subset. In the resulting wellgraded basis,
 * each non-wellgraded element is substituted by the family
 * of all sets of the form \emph{prerequisite subset + single
 * new item}.
 */
basis *wellgrade(basis *b);




/* select sequence of states under equal distribution
 * This function selects from a knowledge space a sequence
 * of knowledge states assuming equal distribution. This is
 * to be used as simulated data. Some noise (careless errors
 * and lucky guesses) may be added via the mk_noisy() function.
 *
 * This function uses stdlib's rand() routine. Therefore,
 * it can be influenced through stdlib's srand() routine.
 *
 * Returns: Pointer to the selected data.
 */
data *equal_sample(
  structure *s,       /* Source structure of the sample     */
  int       size      /* Size of the sample                 */
);

/* add noise to a set of data
 * This function adds noise, i.e. careless errors and lucky
 * guesses to a data set. Noise is added according to the model
 * developed by Falmagne and Doignon.
 *
 * This function uses stdlib's rand() routine. Therefore,
 * it can be influenced through stdlib's srand() routine.
 *
```

```
 * Returns: Pointer to noisy data.
 */
data *mk_noisy(
  data *d,              /* Original 'clean' data set          */
  int  *beta,           /* Beta values in per thousand        */
  int  *eta             /* Eta values in per thousand         */
);



#ifdef __cplusplus
}
#endif

#endif
```

D.3 srbi-util.h

```
/* srbi-util.h - functions and variables for general use */

/* This library was developed within the Project "Surmise
 * Relations Between Tests" (SRBT) at the Department of
 * Psychology, University of Graz, Austria. Partially it
 * is based on an older library developed at the Section
 * of Mathematical and Social Psychology, University of
 * Technology, Braunschweig, Germany.
 *
 * (C) 1995, 1998 Cord Hockemeyer (CHockemeyer@acm.org)
 */

#ifndef _SRBI_UTIL_H_INCLUDED_
#define _SRBI_UTIL_H_INCLUDED_


#ifdef __cplusplus
extern "C" {
#endif


/*
```

```
 * flag indicating verbosity mode
 * The verbose flag indicates a verbosity mode.
 */
extern int verbose;

/*
 * flag indicating debug mode
 * The verbose flag indicates a debugging mode.
 */
extern int debug;


/*
 * Generating random numbers with larger domain.
 * On many systems, the rand() function from stdlib
 * generates only 15 bit integer numbers. In such cases
 * random() should produce 30 bit integers.
 *
 * This function uses stdlib's rand() function and, therefore,
 * can be influenced through stdlib's srand() function.
 */
extern long random();


/*
 * Generating random numbers with larger domain.
 * On many systems, the rand() function from stdlib
 * generates only 15 bit integer numbers. In such cases
 * urandom() should produce 32 bit integers.
 *
 * This function uses stdlib's rand() function and, therefore,
 * can be influenced through stdlib's srand() function.
 */
extern unsigned int urandom();


#ifdef __cplusplus
}
#endif

#endif
```

D.4 filetype.h

```
/* filetype.h - Types of SRBI and SRBT files

 * This library was developed within the Project "Surmise
 * Relations Between Tests" (SRBT) at the Department of
 * Psychology, University of Graz, Austria. Partially it
 * is based on an older library developed at the Section
 * of Mathematical and Social Psychology, University of
 * Technology, Braunschweig, Germany.
 *
 * (C) 1998 Cord Hockemeyer (CHockemeyer@acm.org)
 */

#ifndef _INTERNAL_FILETYPE_H_INCLUDED_
#define _INTERNAL_FILETYPE_H_INCLUDED_

#include <stdio.h>
#include <srbi-set.h>

#ifdef __cplusplus
extern "C" {
#endif


typedef unsigned int structtype;

/* structtypes must be less than 256!          */
#define UNKNOWN      0
#define BASIS        1
#define SPACE        2
#define STRUCTURE    3
#define RELATION     4
#define DATA         5
#define DISJPARTITION 6
#define PARTITION    7
#define PATTERNS     8
#define TESTRELATION 9
#define I_HYPOTHESIS 10
```

```
/* Strings describing the various structure types. */
extern const char *structtype_names[];

/* Bitmask to extract structtype from filetype */
#define STRUCTTYPE      0x00ff

typedef unsigned int filetype;

/* Filetype attributes as bitmasks between 0x0100 and 0x8000
 * For each attribute, a bitmask and all possible values are
 * defined. Excluding attributes may share bits.
 */
#define FORMAT_MASK     0x0100
#define NEW_FORMAT      0x0000
#define OLD_FORMAT      0x0100

#define B_ORDER_MASK    0x0200
#define B_UNORDERED     0x0000
#define B_ORDERED       0x0200

#define STOR_MASK       0x0200
#define ASCII_FILE      0x0000
#define BINARY_FILE     0x0200

#define ENDIAN_MASK     0x0400
#define LITTLE_ENDIAN   0x0000
#define BIG_ENDIAN      0x0400


/*
 * Checks for the type of an SRBI/SRBT file.
 * type_of_file() opens a file and tries to find out which type
 * of SRBI/SRBT data are stored in the file. It is closed
 * afterwards, again.
 * Returns: Type of file including type of contents and
 *          additional attributes.
 *          Filetypes contain in the lower eight bits the
 *          structtype, in the next eight bits there are
```

```
 *           format specifications, and the next 16 bits
 *           contain the wordsize for binary files.
 *
 * Warning:
 * Old-style binary files are always identified as data files
 * since we can assume none of the stricter conditions to be
 * fulfilled for knowledge structures or knowledge spaces.
 */
extern filetype type_of_file(
  const char *filename    /* Name of file to be tested */
);

/* String containing SRBT mark and version.
 * Version 2.0 marks the new format header lines.
 * Version 1.0 is to be used for the ``old KST format.
 */
extern const char srbt_version[];

#ifdef __cplusplus
}
#endif

#endif
```

D.5 error-list.h

```
/**
*** ERRORS.H
***
*** Definition of error codes.
***
*** We define this here global to use identical
**/
#ifndef _ERRORLIST_H_INCLUDED_
#define _ERRORLIST_H_INCLUDED_

#ifdef __cplusplus
extern "C" {
#endif
```

```
#define NO_ERROR                0 /** Things work correctly      **/

#define MEMORY_ERROR            1 /** Not enough memory          **/

#define OPEN_ERROR             11 /** Unable to open file        **/
#define READ_ERROR             12 /** Unable to read             **/
#define WRITE_ERROR            13 /** Unable to write            **/

#define FORMAT_ERROR           21 /** File has wrong format      **/
#define INCOMP_ERROR           22 /** Using incompatible files   **/
#define UDEFD_PARAM_ERROR      23 /** Parameter not defined      **/
#define UDEFD_TOKEN_ERROR      24 /** Token not defined          **/
#define ILLEGAL_PARAM_ERROR    25 /** Illegal Param. specified   **/

#define PARAM_ERROR            31 /** Wrong cmd line param.      **/
#define NO_PARAM_ERROR         32 /** Not enough param. given    **/
#define TOO_PARAM_ERROR        33 /** Too many param. given      **/

#define PROB_ERROR            101 /** Probabilities don't sum    **/

#define NULL_ERROR            201 /** Access to NULL pointer     **/
#define NOPEN_ERROR           202 /** Access to non-open file    **/
#define SCAN_PARAM_ERROR      203 /** Unscanned parameter file   **/

#define NOT_IMPL_ERROR       1001 /** Not yet implemented        **/
#define INTERNAL_ERROR       1002 /** Internal program error     **/
#define SETSIZE_ERROR        1003 /** Exceeding set size         **/
#define SUBSCRIPT_ERROR      1004 /** Exceeding array range      **/

#ifdef __cplusplus
}
#endif

#endif
```

D.6 internal/srbi-space-io.h

```
/* srbi-space-io.h - Internal IO functions for knowledge spaces
```

```
***
*** Internal headerfile for the libsrbi.a library.
***
*** This headerfile declares IO functions for internal use only.
**/
#ifndef _SRBI_SPACE_IO_H_INCLUDED_
#define _SRBI_SPACE_IO_H_INCLUDED_

#include <stdio.h>
#include <srbi-set.h>
#include <srbi-space.h>
#include <filetype.h>

#ifdef __cplusplus
extern "C" {
#endif


/*
** Functions for opening files. The function names have the form
** XopenY_ZZZZ where X is an 'a' or a 'b' determining whether the
** file to be opened has ASCII or binary format, Y is an 'i' or
** an 'o' for input or output (i.e. whether to read or to write
** the data) and ZZZZ describes the kind of data, i.e. whether
** to read/write a knowledge space or a basis.
**
** The functions used to simply read states (i.e. sets) from
** these files are defined in the libset.a library.
**
** To close these files simply use the fclose() from standard
** library.
*/

/* Open a spacefile in ASCII format for reading */
extern FILE *aopeni_space(const char *filename,
                          int *q_size,
                          int *s_size);

/* Open a spacefile in binary format for reading */
extern FILE *bopeni_space(const char *filename,
                          int *q_size,
```

```
                                int *s_size);

/* Open a spacefile in ASCII format for writing */
extern FILE *aopeno_space(const char *filename,
                                int q_size,
                                int s_size);

/* Open a spacefile in binary format for writing */
extern FILE *bopeno_space(const char *filename,
                                int q_size,
                                int s_size);

/*
** Base files are always stored in ASCII format so far.
** Therefore we need no identifier for the file format.
*/

/* Open a basisfile for reading */
extern FILE *openi_basis(const char *filename,
                               int *q_size,
                               int *b_size);

/* Open a basisfile for writing */
extern FILE *openo_basis(const char *filename,
                               int q_size,
                               int b_size);

/*
** For the basis files we need topic specific reading routines,
** i.e. functions to determine whether a set is minimal for an
** item or not. The result is just an error code.
*/
extern int read_basis_element(FILE *in,
                                    bitset minimal,
                                    bitset nonminimal,
                                    int q_size);

/*
** Similar for writing
```

```
*/
extern int write_basis_element(FILE *out,
                               bitset minimal,
                               bitset nonminimal,
                               int q_size);



/**
*** Here some functions for the usage of knowledge spaces,
**/

/*
** Again we start with I/O functions:
*/

/* Read a complete knowledge space in ASCII format. */
extern space *ard_space(const char *filename);

/* Read a complete knowledge space in binary format. */
extern space *brd_space(const char *filename);

/* Write a complete knowledge space in ASCII format. */
extern int awr_space(
   const char *filename, /* Name of spacefile */
   space      *s  /* Pointer to space structure */
);

/* Write a complete knowledge space in binary format. */
extern int bwr_space(
   const char *filename, /* Name of spacefile */
   space      *s   /* Pointer to space structure */
);

/* Read a complete basisfile. */
extern basis *rd_basis(
   const char *filename  /* Name of basisfile */
```

```
);

/* Write a complete basisfile. */
extern int wr_basis(
    const char *filename, /* Name of basisfile */
    basis      *b  /* Pointer to basis structure */
);



/*
** Additionally, some basic IO functions
*/

/* Read an integer value from binary stream with endian corrections. */
int read_int(
    FILE *f,              /* File to read from */
    structtype endian           /* Do we have to change endian? */
);


/* Read a bitset_basis value from binary stream with endian corrections.
bitset_basis read_bb(
    FILE *f,              /* File to read from */
    structtype endian           /* Do we have to change endian? */
);


/*
 * Read a state from an ASCII file.
 * Returns: Error code.
 */
int a_read_state(
    FILE *f,              /* File to read from */
    bitset bs,            /* Already allocated bitset */
    int q_size            /* Size of set */
);
```

```
/*
 * Read a state from a binary file.
 * Returns: Error code.
 */
int b_read_state(
   FILE *f,            /* File to read from */
   bitset bs,          /* Already allocated bitset */
   int q_size,         /* Size of set */
   structtype endian          /* Do we have to change endian? */
);


/*
** For the files with patterns we need topic specific reading routines,
** i.e. functions to determine whether an item was not solved correctly or not
** answered. The result is just an error code.
*/
extern int read_patterns_element(FILE *in,
                                 bitset solved,
                                 bitset unanswered,
                                 int q_size);

/*
** Similar for writing
*/
extern int write_patterns_element(FILE *out,
                                  bitset solved,
                                  bitset unanswered,
                                  int q_size);



/*
** Functions for reading item- and line-infos from a file
*/

long read_item_info(FILE* f, int** info, int q, char* buf);

long read_line_info(FILE* f, int** info, int l, char* buf);
```

```
int** copy_info(int** orig, int number);

int write_item_info(FILE* f, int** info, int number);

int write_line_info(FILE* f, int** info, int number);

int set_info(int** copy, int** orig, int number);

#ifdef __cplusplus
}
#endif

#endif
```

Bibliography

[c2man] c2man. *C2man*. Online available at `ftp://ftp.research.canon.com.au/pub/c2man`.

[1] Dowling, C. E. & Hockemeyer, C. (1995). *Wissensdiagnose in der beruflichen Ausbildung* [Knowledge assessment in professional education]. (Technical report), Institut für Psychologie, Technische Universität Braunschweig, Germany.

[2] Knuth, D. E. (1982). *The WEB system of structured documentation—version 1*. Stanford, CA, USA: Stanford University.

[3] Knuth, D. E. (1984). Literate programming. *The Computer Journal*, *27*(2), 97–111.

[4] Knuth, D. E. (1986). *The TEXbook*, (Vol. A) of *Computers and Typesetting*. Reading, MA, USA: Addison-Wesley.

[5] Knuth, D. E. (1986). *TEX: The program*, (Vol. B) of *Computers and Typesetting*. Reading, MA, USA: Addison-Wesley.

[6] Knuth, D. E. (1992). *Literate programming*. CSLI Lecture Notes Number 27. Stanford, CA, USA: Stanford University Center for the Study of Language and Information.

[7] Knuth, D. E. & Levy, S. (1993). *The CWEB system of structured documentation, version 3.0*. Reading, MA, USA: Addison-Wesley.

[van Ammers and Kramer] van Ammers, E. W. & Kramer, M. R. *The CLiP style of literate programming*. Online available from CTAN: `/web/clip/clip_style.ps`. Version 4.074, 26-feb-93.

List of Manpages