C formal concept analysis library

Generated by Doxygen 1.7.6.1

Wed Aug 21 2013 15:20:55

Contents

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

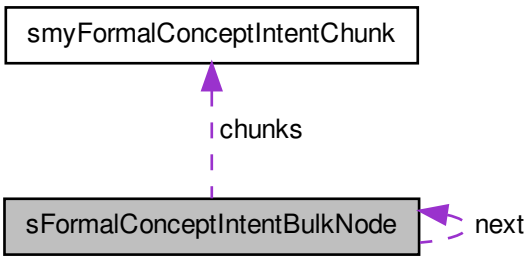
Data Structure Documentation

3.1 sFormalConceptIntentBulkNode Struct Reference

a node of a single linked list of concept chunks.

`#include <structs.h>`

Collaboration diagram for sFormalConceptIntentBulkNode:



Data Fields

- int attributes
  number of attributes of the concept intents
- int size
  number of chunks used
- myFormalConceptIntentChunk ** chunks
  array to at most BULKSIZE chunks

- struct sFormalConceptIntentBulkNode * next

  pointer to the next BulkNode, or 0

3.1.1 Detailed Description

a node of a single linked list of concept chunks.

bulk nodes are filled chunk wise, but a bulk node may have non-empty successor nodes even if it is not entire full.

Definition at line 89 of file structs.h.

3.1.2 Field Documentation

3.1.2.1 int sFormalConceptIntentBulkNode::attributes

number of attributes of the concept intents

Definition at line 94 of file structs.h.

Referenced by addConceptToBulk(), newConceptBulk(), and writeConceptsToFile().

3.1.2.2 myFormalConceptIntentChunk** sFormalConceptIntentBulkNode::chunks

array to at most BULKSIZE chunks

Definition at line 102 of file structs.h.

Referenced by addConceptToBulk(), countConceptsInBulk(), deleteConceptBulk(), newConceptBulk(), and writeConceptsToFile().

3.1.2.3 struct sFormalConceptIntentBulkNode* sFormalConceptIntentBulkNode::next

pointer to the next BulkNode, or 0

Definition at line 106 of file structs.h.

Referenced by addConceptToBulk(), countConceptsInBulk(), deleteConceptBulk(), newConceptBulk(), and writeConceptsToFile().

3.1.2.4 int sFormalConceptIntentBulkNode::size

number of chunks used

Definition at line 98 of file structs.h.

Referenced by addConceptToBulk(), countConceptsInBulk(), deleteConceptBulk(), newConceptBulk(), and writeConceptsToFile().

The documentation for this struct was generated from the following file:

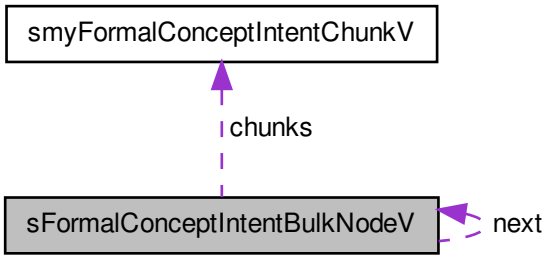
- src/fca/easy/structs.h

3.2 sFormalConceptIntentBulkNodeV Struct Reference

a node of a single linked list of concept chunks.

`#include <structs.h>`

Collaboration diagram for sFormalConceptIntentBulkNodeV:



Data Fields

- size_t attributes

  number of attributes of the concept intents
- size_t width

  width of each attribute vector
- size_t size

  number of chunks used
- myFormalConceptIntentChunkV ** chunks

  array to at most BULKSIZEV chunks
- struct sFormalConceptIntentBulkNodeV * next

  pointer to the next BulkNode, or 0

3.2.1 Detailed Description

a node of a single linked list of concept chunks.

bulk nodes are filled chunk wise, but a bulk node may have non-empty successor nodes even if it is not entire full.

Definition at line 93 of file structs.h.

3.2.2 Field Documentation

3.2.2.1 size_t sFormalConceptIntentBulkNodeV::attributes

number of attributes of the concept intents

Definition at line 98 of file structs.h.

Referenced by addConceptToBulkV(), newConceptBulkV(), and writeConceptsToFileV().

3.2.2.2 myFormalConceptIntentChunkV** sFormalConceptIntentBulkNodeV::chunks

array to at most BULKSIZEV chunks

Definition at line 110 of file structs.h.

Referenced by addConceptToBulkV(), countConceptsInBulkV(), deleteConceptBulkV(), newConceptBulkV(), and writeConceptsToFileV().

3.2.2.3 struct sFormalConceptIntentBulkNodeV* sFormalConceptIntentBulkNodeV::next

pointer to the next BulkNode, or 0

Definition at line 114 of file structs.h.

Referenced by addConceptToBulkV(), countConceptsInBulkV(), deleteConceptBulkV(), newConceptBulkV(), nextClosureVX(), and writeConceptsToFileV().

3.2.2.4 size_t sFormalConceptIntentBulkNodeV::size

number of chunks used

Definition at line 106 of file structs.h.

Referenced by addConceptToBulkV(), countConceptsInBulkV(), deleteConceptBulkV(), newConceptBulkV(), and writeConceptsToFileV().

3.2.2.5 size_t sFormalConceptIntentBulkNodeV::width

width of each attribute vector

Definition at line 102 of file structs.h.

Referenced by addConceptToBulkV(), and newConceptBulkV().

The documentation for this struct was generated from the following file:

- src/fca/vector/structs.h

3.3 sFormalIntent Struct Reference

intent structure of a formal concept

`#include <easy.h>`

Data Fields

- size_t attributes
- IncidenceCell * incidence

3.3.1 Detailed Description

intent structure of a formal concept

Definition at line 48 of file easy.h.

3.3.2 Field Documentation

3.3.2.1 size_t sFormalIntent::attributes

Definition at line 50 of file easy.h.

3.3.2.2 IncidenceCell* sFormalIntent::incidence

Definition at line 51 of file easy.h.

The documentation for this struct was generated from the following file:

- src/fca/easy.h

3.4 sFormalIntentV Struct Reference

intent structure of a formal concept

`#include <vector.h>`

Data Fields

- size_t attributes

  nbr of attributes in this vector
- size_t width

  the width, i.e.
- IncidenceVector incidence

  attribute vector

3.4.1 Detailed Description

intent structure of a formal concept

Definition at line 50 of file vector.h.

3.4.2 Field Documentation

3.4.2.1 size_t sFormalIntentV::attributes

nbr of attributes in this vector

Definition at line 55 of file vector.h.

3.4.2.2 IncidenceVector sFormalIntentV::incidence

attribute vector

Definition at line 63 of file vector.h.

3.4.2.3 size_t sFormalIntentV::width

the width, i.e.

floor of (attributes+63)/64

Definition at line 59 of file vector.h.

The documentation for this struct was generated from the following file:

- src/fca/vector.h

3.5 smyFormalConceptIntentChunk Struct Reference

A chunk of at most CHUNKSIZE formal concept intents.

```
#include <structs.h>
```

Data Fields

- int attributes

  number of attributes

- int size

  how many formal concepts are in this chunk

- IncidenceCell * incidence

  the intents of the concepts

3.5.1 Detailed Description

A chunk of at most CHUNKSIZE formal concept intents.

Definition at line 65 of file structs.h.

3.5.2 Field Documentation

3.5.2.1 int smyFormalConceptIntentChunk::attributes

number of attributes

Definition at line 70 of file structs.h.

Referenced by newConceptChunk().

3.5.2.2 IncidenceCell* smyFormalConceptIntentChunk::incidence

the intents of the concepts

Definition at line 78 of file structs.h.

Referenced by newConceptChunk().

3.5.2.3 int smyFormalConceptIntentChunk::size

how many formal concepts are in this chunk

Definition at line 74 of file structs.h.

Referenced by addConceptToBulk(), countConceptsInBulk(), newConceptChunk(), and writeConceptsToFile().

The documentation for this struct was generated from the following file:

- src/fca/easy/structs.h

3.6 smyFormalConceptIntentChunkV Struct Reference

A chunk of at most CHUNKSIZEV formal concept intent vectors.

`#include <structs.h>`

Data Fields

- size_t attributes
  number of attributes
- size_t width
  width of each attribute vector
- size_t size
  how many formal concepts are in this chunk
- IncidenceVector incidence
  the intents of the concepts

3.6.1 Detailed Description

A chunk of at most CHUNKSIZEV formal concept intent vectors.

Definition at line 66 of file structs.h.

3.6.2 Field Documentation

3.6.2.1 size_t smyFormalConceptIntentChunkV::attributes

number of attributes

Definition at line 71 of file structs.h.

Referenced by newConceptChunkV().

3.6.2.2 IncidenceVector smyFormalConceptIntentChunkV::incidence

the intents of the concepts

Definition at line 83 of file structs.h.

Referenced by newConceptChunkV().

3.6.2.3 size_t smyFormalConceptIntentChunkV::size

how many formal concepts are in this chunk

Definition at line 79 of file structs.h.

Referenced by addConceptToBulkV(), countConceptsInBulkV(), newConceptChunkV(), and writeConceptsToFileV().

3.6.2.4 size_t smyFormalConceptIntentChunkV::width

width of each attribute vector

Definition at line 75 of file structs.h.

Referenced by newConceptChunkV().

The documentation for this struct was generated from the following file:

- src/fca/vector/structs.h

3.7 smyFormalContext Struct Reference

each formal context has a finite number of objects and attributes, which may have names (though we do not require them to be unique or given), and an incidence relation which is represented by an objects×attributes-IncidenceCell matrix

```
#include <structs.h>
```

Data Fields

- int attributes
- int objects
- char ** attributeNames
- char ** objectNames
- IncidenceCell * incidence

3.7.1 Detailed Description

each formal context has a finite number of objects and attributes, which may have names (though we do not require them to be unique or given), and an incidence relation which is represented by an objects×attributes-IncidenceCell matrix

Definition at line 51 of file structs.h.

3.7.2 Field Documentation

3.7.2.1 char** smyFormalContext::attributeNames

Definition at line 54 of file structs.h.

Referenced by deleteFormalContext(), newFormalContext(), newFormalContextFromFile(), writeConceptsToFile(), and writeFormalContext().

3.7.2.2 int smyFormalContext::attributes

Definition at line 53 of file structs.h.

Referenced by closeIntent(), countContextConcepts(), countContextConcepts2(), deleteFormalContext(), newConceptBulkFromContext(), newFormalContext(), newFormalContextFromRandom(), writeConceptsToFile(), and writeFormalContext().

3.7.2.3 IncidenceCell* smyFormalContext::incidence

Definition at line 56 of file structs.h.

Referenced by deleteFormalContext(), and newFormalContext().

3.7.2.4 char** smyFormalContext::objectNames

Definition at line 55 of file structs.h.

Referenced by deleteFormalContext(), newFormalContext(), newFormalContextFromFile(), and writeFormalContext().

3.7.2.5 int smyFormalContext::objects

Definition at line 53 of file structs.h.

Referenced by closeIntent(), countContextConcepts2(), deleteFormalContext(), newFormalContext(), newFormalContextFromRandom(), and writeFormalContext().

The documentation for this struct was generated from the following file:

- src/fca/easy/structs.h

3.8 smyFormalContextV Struct Reference

each formal context has a finite number of objects and attributes, which may have names (though we do not require them to be unique or given), and an incidence relation which is represented by an objects×attributes-IncidenceCell matrix

`#include <structs.h>`

Data Fields

- size_t attributes
- size_t objects
- size_t width
- char ** attributeNames
- char ** objectNames
- IncidenceVector incidence

3.8.1 Detailed Description

each formal context has a finite number of objects and attributes, which may have names (though we do not require them to be unique or given), and an incidence relation which is represented by an objects×attributes-IncidenceCell matrix

for the vector implementation, we have the variable width which codes the width of each object's IncidenceVector

Definition at line 52 of file structs.h.

3.8.2 Field Documentation

3.8.2.1 char** smyFormalContextV::attributeNames

Definition at line 56 of file structs.h.

Referenced by deleteFormalContextV(), newFormalContextFromFileV(), newFormalContextV(), writeConceptsToFileV(), and writeFormalContextV().

3.8.2.2 size_t smyFormalContextV::attributes

Definition at line 54 of file structs.h.

Referenced by deleteFormalContextV(), newFormalContextV(), nextClosureVX(), writeConceptsToFileV(), and writeFormalContextV().

3.8.2.3 IncidenceVector smyFormalContextV::incidence

Definition at line 58 of file structs.h.

Referenced by deleteFormalContextV(), and newFormalContextV().

3.8.2.4 char** smyFormalContextV::objectNames

Definition at line 57 of file structs.h.

Referenced by deleteFormalContextV(), newFormalContextFromFileV(), newFormalContextV(), and writeFormalContextV().

3.8.2.5 size_t smyFormalContextV::objects

Definition at line 54 of file structs.h.

Referenced by deleteFormalContextV(), newFormalContextV(), and writeFormalContextV().

3.8.2.6 size_t smyFormalContextV::width

Definition at line 55 of file structs.h.

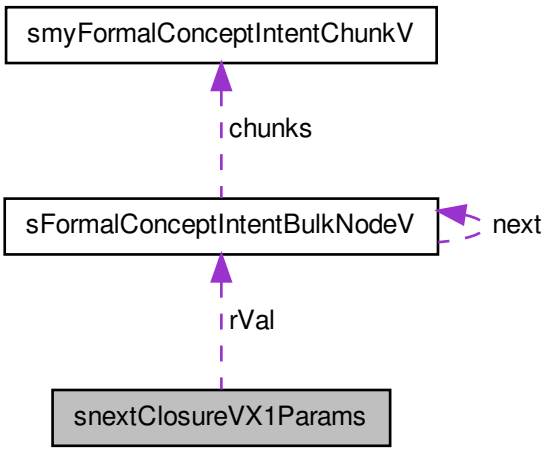
Referenced by newFormalContextV(), and nextClosureVX().

The documentation for this struct was generated from the following file:

- src/fca/vector/structs.h

3.9 snextClosureVX1Params Struct Reference

Collaboration diagram for snextClosureVX1Params:



Data Fields

- FormalConceptIntentBulkListV rVal
- FormalContextV ctx
- IncidenceVector start
- IncidenceVector stop

3.9.1 Detailed Description

Definition at line 172 of file fcaVnextClosureX.c.

3.9.2 Field Documentation

3.9.2.1 FormalContextV snextClosureVX1Params::ctx

Definition at line 175 of file fcaVnextClosureX.c.

Referenced by callNextClosureVX1(), and nextClosureVX().

3.9.2.2 FormalConceptIntentBulkListV snextClosureVX1Params::rVal

Definition at line 174 of file fcaVnextClosureX.c.

Referenced by callNextClosureVX1(), and nextClosureVX().

3.9.2.3 IncidenceVector snextClosureVX1Params::start

Definition at line 176 of file fcaVnextClosureX.c.

Referenced by callNextClosureVX1(), and nextClosureVX().

3.9.2.4 IncidenceVector snextClosureVX1Params::stop

Definition at line 177 of file fcaVnextClosureX.c.

Referenced by callNextClosureVX1(), and nextClosureVX().

The documentation for this struct was generated from the following file:

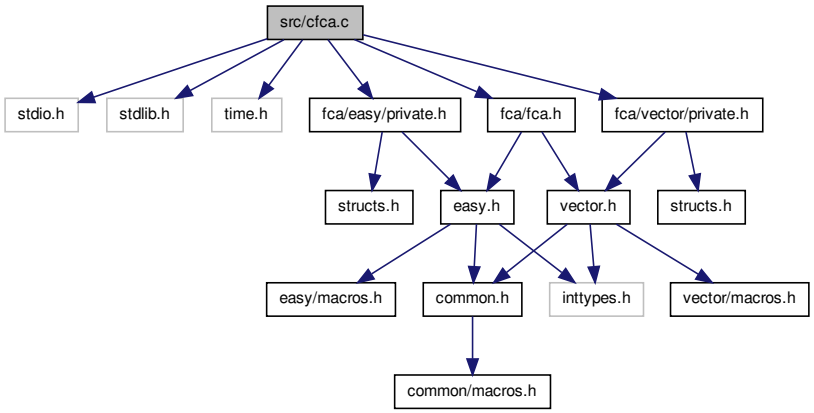
- src/fca/vector/fcaVnextClosureX.c

Chapter 4

File Documentation

4.1 src/cfca.c File Reference

`#include <stdio.h> #include <stdlib.h> #include <time.h> #include "fca/fca.h" #include "fca/easy/private.h" #include "fca/vector/private.h"` Include dependency graph for cfca.c:



Functions

- int main (void)

  cfca.c, (c) 2013, Immanuel Albrecht; Dresden University of Technology, Professur für die Psychologie des Lernen und Lehrens

4.1.1 Function Documentation

4.1.1.1 int main (void)

cfca.c, (c) 2013, Immanuel Albrecht; Dresden University of Technology, Professur für die Psychologie des Lernen und Lehrens

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>. this is the main testing routine for purposes of testing the formal concept analysis implementation for errors

Returns

initialize pseudo random number generator

start tests

Definition at line 34 of file cfca.c.

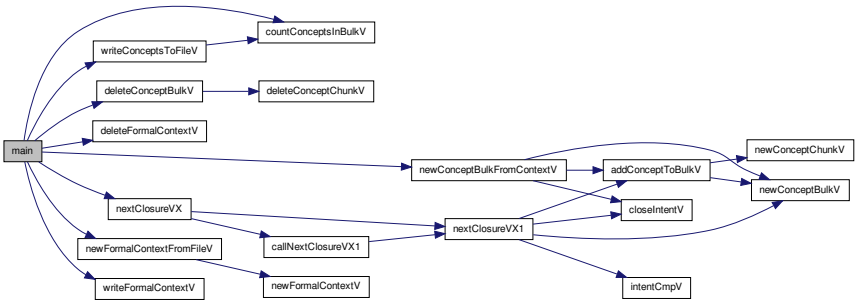
References countConceptsInBulk, countConceptsInBulkV(), deleteConceptBulk, deleteConceptBulkV(), deleteFormalContext, deleteFormalContextV(), newConceptBulkFromContext, newConceptBulkFromContextV(), newFormalContextFromFileV(), newFormalContextFromRandom, nextClosureVX(), writeConceptsToFile, writeConceptsToFileV(), writeFormalContext, and writeFormalContextV().

```
{
//  uint64_t test[2];
//  for (int i = 0; i < 65; ++i) {
//      test[0] = ~0ULL;
//
//      test[1] = ~0ULL;
//      MASKVECTOR(test,i+1);
//      printf("%2d: %16llx %16llx
       %16llx%16llx\n",BITNBR(i),BITVALUE(i),CRIMPVALUE(i),test[0]);
//  }

#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wconversion"

    srandom(time(0));

#pragma GCC diagnostic pop

    FormalContext ctx;

    ctx = newFormalContextFromRandom(80, 30, 0.3f);
```

```
    writeFormalContext(ctx, "/home/immo/tmp/test.cxt");

    puts("Cloning V...");

    FormalContextV ctxV;

    ctxV = newFormalContextFromFileV("/home/immo/tmp/test.cxt");

    writeFormalContextV(ctxV, "/home/immo/tmp/testV.cxt");

    puts("Performance testing....");

    clock_t start, end;
    time_t xstart, xend;

    FormalConceptIntentBulkListV conceptsV;

    time(&xstart);
    start = clock();

    conceptsV = newConceptBulkFromContextV(ctxV);

    end = clock();
    time(&xend);

    printf("Concepts: %zu\n", countConceptsInBulkV(conceptsV));

    printf("Time version V: %f sec in %f [%d-%d]\n",
            (float) (end - start) / CLOCKS_PER_SEC,
            (float) difftime(xend, xstart), start, end);

    FormalConceptIntentBulkListV conceptsVX;

    time(&xstart);
    start = clock();

    conceptsVX = nextClosureVX(ctxV);

    end = clock();
    time(&xend);

    printf("Concepts: %zu\n", countConceptsInBulkV(conceptsV));

    printf("Time version VX: %f sec in %f [%d-%d]\n",
            (float) (end - start) / CLOCKS_PER_SEC,
            (float) difftime(xend, xstart), start, end);

    FormalConceptIntentBulkList concepts;

    time(&xstart);
    start = clock();

    concepts = newConceptBulkFromContext(ctx);

    end = clock();
    time(&xend);

    printf("Concepts: %d\n", countConceptsInBulk(concepts));

    printf("Time version 1: %f sec in %f [%d-%d]\n",
            (float) (end - start) / CLOCKS_PER_SEC,
            (float) difftime(xend, xstart), start, end);
```

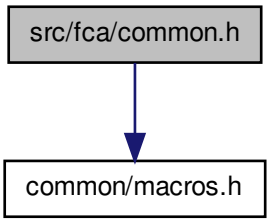
```
    writeConceptsToFile(ctx, concepts, "/home/immo/tmp/test1.cxt");
    writeConceptsToFileV(ctxV, conceptsV, "/home/immo/tmp/testV.cxt");
    writeConceptsToFileV(ctxV, conceptsVX, "/home/immo/tmp/testVX.cxt");

    /*puts("=====");
     FILE* status = fopen("/proc/self/status", "r");
     char line[1000];
     while (fgets(line, sizeof line, status) != NULL)
     {
     printf("%s", line);
     }
     fclose(status);
     puts("=====");*/

    puts("Clean up...");

    deleteConceptBulkV(&conceptsVX);
    deleteConceptBulkV(&conceptsV);
    deleteConceptBulk(&concepts);
    deleteFormalContext(&ctx);
    deleteFormalContextV(&ctxV);

    return EXIT_SUCCESS;
}
```

Here is the call graph for this function:

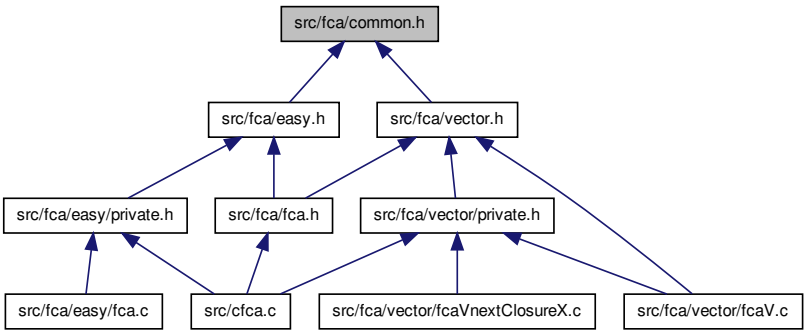


4.2 src/fca/common.h File Reference

`#include "common/macros.h"` Include dependency graph for common.h:

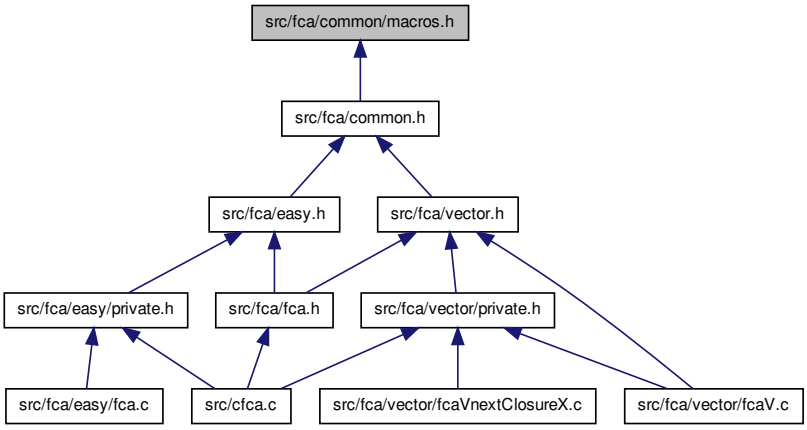


This graph shows which files directly or indirectly include this file:



4.3 src/fca/common/macros.h File Reference

This graph shows which files directly or indirectly include this file:



Defines

- #define RETURN_IF_ZERO(x) {if ((x == (void∗)0)) {fprintf(stderr, "WARNING: ZERO pointer %s in %s [%s:%u]\n", #x, __FUNCTION__, __FILE__,__LINE__); return;}}

  fca_macros.h, (c) 2013, Immanuel Albrecht; Dresden University of Technology, - Professur für die Psychologie des Lernen und Lehrens

- #define RETURN_ZERO_IF_ZERO(x) {if ((x == (void∗)0)) {fprintf(stderr, "WARNING: ZERO pointer %s in %s [%s:%u]\n", #x, __FUNCTION__, __FILE__,__LINE__); return 0;}}

  checks whether x == 0, and returns 0;

- #define WARN_IF_UNEQUAL_DO(x, y, d) {if (((x) != (y))) {fprintf(stderr, "WARNING: %s NOT EQUAL TO %s in %s [%s:%u]\n", #x, #y, __FUNCTION__, __FILE__,__LINE__); d;}}

  if x!=y, prints a warning and calls the statement d

- #define MIN(a, b) (((a)<(b))?(a):(b))

  gives minimum

- #define MAX(a, b) (((a)>(b))?(a):(b))

  gives maximum

4.3.1 Define Documentation

4.3.1.1 #define MAX(a, b) (((a)>(b))?(a):(b))

gives maximum

Definition at line 56 of file macros.h.

4.3.1.2 #define MIN(a, b) (((a)<(b))?(a):(b))

gives minimum

Definition at line 49 of file macros.h.

Referenced by newFormalContextFromFile(), and newFormalContextFromFileV().

4.3.1.3 #define RETURN_IF_ZERO(x) {if ((x == (void*)0)) {fprintf(stderr, "WARNING: ZERO pointer %s in %s [%s:%u]\n", #x, __FUNCTION__, __FILE__,__LINE__); return;}}

fca_macros.h, (c) 2013, Immanuel Albrecht; Dresden University of Technology, - Professur für die Psychologie des Lernen und Lehrens

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

checks whether x == 0, and returns

Definition at line 27 of file macros.h.

Referenced by deleteConceptBulk(), deleteConceptBulkV(), deleteConceptChunk(), deleteConceptChunkV(), deleteFormalContext(), deleteFormalContextV(), writeConceptsToFile(), writeConceptsToFileV(), writeFormalContext(), and writeFormalContextV().

4.3.1.4 #define RETURN_ZERO_IF_ZERO(x) {if ((x == (void*)0)) {fprintf(stderr, "WARNING: ZERO pointer %s in %s [%s:%u]\n", #x, __FUNCTION__, __FILE__,__LINE__); return 0;}}

checks whether x == 0, and returns 0;

Definition at line 34 of file macros.h.

Referenced by addConceptToBulk(), addConceptToBulkV(), countConceptsInBulk(), countConceptsInBulkV(), countContextConcepts(), countContextConcepts2(), countContextConceptsV(), newConceptBulkFromContext(), newConceptBulkFromContextV(), newFormalContextFromFile(), newFormalContextFromFileV(), nextClosureVX(), and nextClosureVX1().

4.3.1.5 #define WARN_IF_UNEQUAL_DO(x, y, d) {if (((x) != (y))) {fprintf(stderr, "WARNING: %s NOT EQUAL TO %s in %s [%s:%u]\n", #x, #y, __FUNCTION__, __FILE__,__LINE__); d;}}

if x!=y, prints a warning and calls the statement d

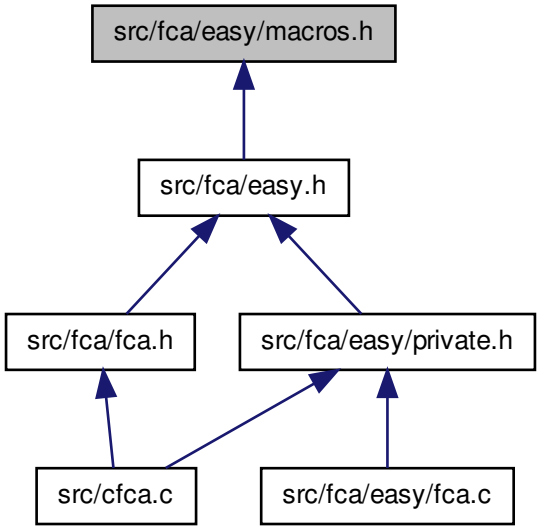
Definition at line 41 of file macros.h.

Referenced by writeConceptsToFile(), and writeConceptsToFileV().

4.4 src/fca/easy/macros.h File Reference

easy/macros.h, (c) 2013, Immanuel Albrecht; Dresden University of Technology, - Professur für die Psychologie des Lernen und Lehrens

This graph shows which files directly or indirectly include this file:



Defines

- #define INCIDES(x) (((x)&1))

  checks whether something incides by testing the 1-bit
- #define CLEAR(x) { (x) = 0; }

  clears the mark

- #define CROSS(x) { (x) = 1; }

  sets the mark

- #define CELL(g, I, m) ((I)->incidence[(I)->attributes * (g) + (m)])

  results in the cell that encodes whether g incides with m

- #define gIm(g, I, m) INCIDES(CELL((g) , (I) , (m)))

  test whether g and m incides

4.4.1 Detailed Description

easy/macros.h, (c) 2013, Immanuel Albrecht; Dresden University of Technology, - Professur für die Psychologie des Lernen und Lehrens This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public - License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>. These macros are used for IncidenceCell array implementations of formal contexts. Such implementations are easier to debug, but take up far too much memory for big scale contexts

Definition in file macros.h.

4.4.2 Define Documentation

4.4.2.1 #define CELL(*g, I, m*) ((I)->incidence[(I)->attributes * (g) + (m)])

results in the cell that encodes whether g incides with m

I may be a formal context, then g refers to the object number, or I may be a chunk of formal concepts, then g refers to the concept number.

Definition at line 61 of file macros.h.

Referenced by addConceptToBulk(), newFormalContextFromFile(), and newFormalContextFromRandom().

4.4.2.2 #define CLEAR(*x*) { (x) = 0; }

clears the mark

Definition at line 42 of file macros.h.

Referenced by closeIntent(), closeIntent2(), countContextConcepts(), countContextConcepts2(), and newConceptBulkFromContext().

4.4.2.3 #define CROSS(x) { (x) = 1; }

sets the mark

Definition at line 50 of file macros.h.

Referenced by closeIntent(), closeIntent2(), countContextConcepts(), countContextConcepts2(), newConceptBulkFromContext(), newFormalContextFromFile(), and newFormalContextFromRandom().

4.4.2.4 #define glm(g, l, m) INCIDES(CELL((g) , (l) , (m)))

test whether g and m incides

Definition at line 68 of file macros.h.

Referenced by closeIntent(), closeIntent2(), writeConceptsToFile(), and writeFormalContext().

4.4.2.5 #define INCIDES(x) (((x)&1))

checks whether something incides by testing the 1-bit

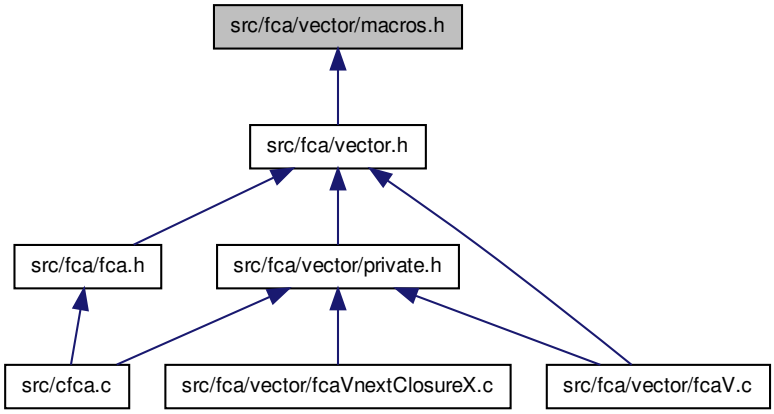
Definition at line 35 of file macros.h.

Referenced by closeIntent(), closeIntent2(), countContextConcepts(), countContextConcepts2(), intentCmp(), and newConceptBulkFromContext().

4.5 src/fca/vector/macros.h File Reference

vector/macros.h, (c) 2013, Immanuel Albrecht; Dresden University of Technology, - Professur für die Psychologie des Lernen und Lehrens

This graph shows which files directly or indirectly include this file:



Defines

- #define OFFSET(x) ((unsigned)(x)>>6)

  get the offset of the x-th bit in an 64-bit integer vector
- #define BITNBR(x) (((unsigned)(x))&(63))

  get the remainder of the x-th bit in an 64-bit integer vector, i.e.
- #define WIDTH(x) ((((unsigned)(x))&(63))?((unsigned)(x)/64)+1:((unsigned)(x)/64))

  determine the length of an 64-bit integer vector that can hold x bits
- #define BITVALUE(x) ((1ULL<<(63-BITNBR(x))))

  gives the bit-value of the x-th bit.
- #define CRIMPVALUE(x) ((∼(0ULL))>>(63-(BITNBR(x)))<<(63-BITNBR(x)))

  gives an 64-bit integer that has set the bits 0 through x.
- #define MASKVECTOR(v, x) {if (BITNBR((x))) { ∗((v)+OFFSET((x)-1)) = ((∗((v)+-OFFSET((x)-1))>>(63-BITNBR((x)-1)))) << (63-BITNBR((x)-1)); }}

  set the unused attribute bits to zero.
- #define CROSSV(v, x) { ∗((v)+OFFSET(x)) |= BITVALUE(x); }

  crosses the x-th attribute of an attribute vector
- #define CLEARV(v, x) { ∗((v)+OFFSET(x)) &= ∼ (BITVALUE(x)); }

  clears the x-th attribute of an attribute vector
- #define INCIDESV(v, x) ((∗((v)+OFFSET(x)) >> (63-BITNBR(x))) & 1)

  checks whether the x-th attribute of an attribute vector is crossed
- #define ROW(g, l) ((l)->incidence + ((l)->width ∗ (g)))

  gives the attribute vector for a given object

4.5.1 Detailed Description

vector/macros.h, (c) 2013, Immanuel Albrecht; Dresden University of Technology, - Professur für die Psychologie des Lernen und Lehrens This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public - License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>. These macros are used for uint64_t bit-stream arrays

Definition in file macros.h.

4.5.2 Define Documentation

4.5.2.1 #define BITNBR(*x*) (((unsigned)(x))&(63))

get the remainder of the x-th bit in an 64-bit integer vector, i.e.

65=64+ *1*

Definition at line 39 of file macros.h.

Referenced by intentCmpV().

4.5.2.2 #define BITVALUE(*x*) ((1ULL<<(63-BITNBR(x))))

gives the bit-value of the x-th bit.

(Note that bit 0 is the most, and bit 63 is the least significant bit)

Definition at line 55 of file macros.h.

4.5.2.3 #define CLEARV(*v, x*) { *((v)+OFFSET(x)) &= ~ (BITVALUE(x)); }

clears the x-th attribute of an attribute vector

Definition at line 108 of file macros.h.

Referenced by countContextConceptsV(), newConceptBulkFromContextV(), and nextClosureVX1().

4.5.2.4 #define CRIMPVALUE(*x*) ((~(0ULL))>>(63-(BITNBR(x)))<<(63-BITNBR(x)))

gives an 64-bit integer that has set the bits 0 through x.

CRIMPVALUE(0) == 0x8000000000000000 CRIMPVALUE(1) == 0xc000000000000000 CRIMPVALUE(2) == 0xe000000000000000 CRIMPVALUE(3) == 0xf000000000000000 CRIMPVALUE(4) == 0xf800000000000000 etc.

Definition at line 77 of file macros.h.

Referenced by countContextConceptsV(), intentCmpV(), newConceptBulkFromContextV(), and nextClosureVX1().

4.5.2.5 #define CROSSV(v, x) { *((v)+OFFSET(x)) |= BITVALUE(x); }

crosses the x-th attribute of an attribute vector

Definition at line 102 of file macros.h.

Referenced by countContextConceptsV(), newConceptBulkFromContextV(), newFormalContextFromFileV(), nextClosureVX(), and nextClosureVX1().

4.5.2.6 #define INCIDESV(v, x) ((*((v)+OFFSET(x)) >> (63-BITNBR(x))) & 1)

checks whether the x-th attribute of an attribute vector is crossed

Definition at line 114 of file macros.h.

Referenced by countContextConceptsV(), newConceptBulkFromContextV(), nextClosureVX1(), writeConceptsToFileV(), and writeFormalContextV().

4.5.2.7 #define MASKVECTOR(v, x) {if (BITNBR((x))) { *((v)+OFFSET((x)-1)) = ((*((v)+OFFSET((x)-1))>>(63-BITNBR((x)-1)))) << (63-BITNBR((x)-1)); }}

set the unused attribute bits to zero.

(i.e. attributes == 100 -> width == 2, BITNBR(99) == 35) where v is a 64-bit integer vector, and x is the number used bits.

Definition at line 91 of file macros.h.

Referenced by closeIntentV().

4.5.2.8 #define OFFSET(x) ((unsigned)(x)>>6)

get the offset of the x-th bit in an 64-bit integer vector

Definition at line 33 of file macros.h.

Referenced by countContextConceptsV(), intentCmpV(), newConceptBulkFromContextV(), and nextClosureVX1().

4.5.2.9 #define ROW(g, I) ((I)->incidence + ((I)->width * (g)))

gives the attribute vector for a given object

Definition at line 125 of file macros.h.

Referenced by addConceptToBulkV(), closeIntentV(), newFormalContextFromFileV(), writeConceptsToFileV(), and writeFormalContextV().

4.5.2.10 #define WIDTH(*x*) ((((unsigned)(x))&(63))?((unsigned)(x)/64)+1:((unsigned)(x)/64))

determine the length of an 64-bit integer vector that can hold x bits

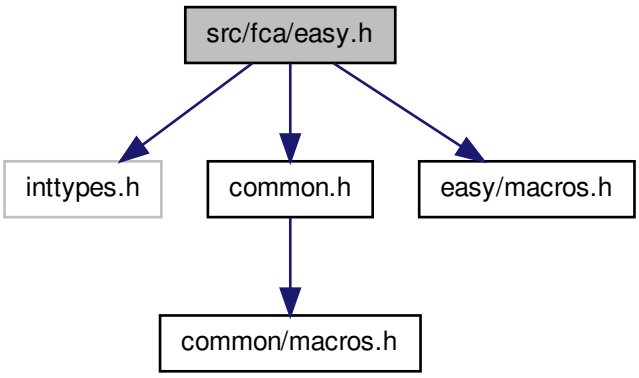
Definition at line 46 of file macros.h.

Referenced by newConceptBulkV(), newConceptChunkV(), and newFormalContextV().

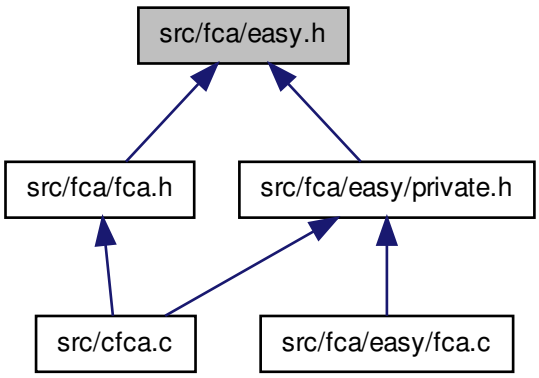
4.6 src/fca/easy.h File Reference

easy.h, (c) 2013, Immanuel Albrecht; Dresden University of Technology, Professur für die Psychologie des Lernen und Lehrens

`#include <inttypes.h>` `#include "common.h"` `#include "easy/macros.h"` Include dependency graph for easy.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct sFormalIntent

  intent structure of a formal concept

Typedefs

- typedef int8_t IncidenceCell

  type of the incidence relation matrix cells
- typedef struct sFormalContext * FormalContext
- typedef struct sFormalIntent FormalIntent

  intent structure of a formal concept

Functions

- FormalContext newFormalContext (int objects, int attributes)

  create a new formal context
- FormalContext newFormalContextFromRandom (int objects, int attributes, float p)

  create a new formal context with random incidence relation
- FormalContext newFormalContextFromFile (const char *filename)

  create a new formal context object from a .cxt file
- int countContextConcepts (FormalContext ctx)

counts the concepts in the concept lattice of ctx, using next closure algorithm

- void writeFormalContext (FormalContext ctx, const char *filename)

  save the context ctx at the given file location

- void deleteFormalContext (FormalContext *ctx)

  *deletes the formal context *ctx, and sets the pointer to zero*

4.6.1 Detailed Description

easy.h, (c) 2013, Immanuel Albrecht; Dresden University of Technology, Professur für die Psychologie des Lernen und Lehrens This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

This header file provides interfaces with the easy IncidenceCell* implementations

Definition in file easy.h.

4.6.2 Typedef Documentation

4.6.2.1 typedef struct sFormalContext* FormalContext

Definition at line 42 of file easy.h.

4.6.2.2 typedef struct sFormalIntent FormalIntent

intent structure of a formal concept

4.6.2.3 typedef int8_t IncidenceCell

type of the incidence relation matrix cells

Definition at line 35 of file easy.h.

4.6.3 Function Documentation

4.6.3.1 int countContextConcepts (FormalContext *ctx*)

counts the concepts in the concept lattice of ctx, using next closure algorithm

Parameters

ctx	formal context

Returns

number of concepts in context

Definition at line 926 of file fca.c.

References smyFormalContext::attributes, CLEAR, closeIntent, CROSS, INCIDES, and RETURN_ZERO_IF_ZERO.

```
{
    RETURN_ZERO_IF_ZERO(ctx);

    myFormalContext *c;
    c = (myFormalContext*) ctx;

    IncidenceCell *M;
    IncidenceCell *Y;

#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wsign-conversion"

    Y = calloc(c->attributes, sizeof(IncidenceCell));
    M = malloc(c->attributes * sizeof(IncidenceCell));

#pragma GCC diagnostic pop

    /*
     * calculate the bottom intent of the concept lattice, i.e. {}''
     */
    closeIntent(ctx, Y, M);

    int count;

    count = 1;

    /*
     * begin of nextClosure function iteration
     */
    nextClosure:

    for (int i = c->attributes - 1; i >= 0; --i)
    {

        if (!INCIDES(M[i]))
        {
            CROSS(M[i]);
            closeIntent(ctx, M, Y);

            int good;
            good = 1;

            for (int j = 0; j < i; ++j)
            {
                if (INCIDES(Y[j]))
                {
                    if (!INCIDES((M[j])))
```

```
                    {

                        good = 0;
                        break;
                    }
                }
            }
            if (good)
            {
                /*
                 * we found the next intent
                 */
                count++;

                /*
                 * continue with Y for M
                 */

                IncidenceCell *DELTA;
                DELTA = M;
                M = Y;
                Y = DELTA;
                /*
                 * do the nextClosure
                 */
                goto nextClosure;
            }
        }

        CLEAR(M[i]);
    }

    /*
     * free up memory
     */

    free(M);
    free(Y);

    return count;
}
```

4.6.3.2 void deleteFormalContext (FormalContext * *ctx*)

deletes the formal context *ctx, and sets the pointer to zero

Parameters

ctx	pointer to the formal context object to be deleted

Definition at line 264 of file fca.c.

References smyFormalContext::attributeNames, smyFormalContext::attributes, smyFormalContext::incidence, smyFormalContext::objectNames, smyFormalContext::objects, and RETURN_IF_ZERO.

```
{
```

```
    RETURN_IF_ZERO(ctx);
    RETURN_IF_ZERO(*ctx);

    myFormalContext *c;

    c = (myFormalContext*) *ctx;

    *ctx = 0;

    for (int var = 0; var < c->attributes; ++var)
    {
        free(c->attributeNames[var]);
    }

    for (int var = 0; var < c->objects; ++var)
    {
        free(c->objectNames[var]);
    }

    free(c->objectNames);
    free(c->attributeNames);
    free(c->incidence);
    free(c);
}
```

4.6.3.3 FormalContext newFormalContext (int *objects,* int *attributes*)

create a new formal context

Parameters

objects	object count
attributes	attribute count

Returns

a new FormalContext object

Definition at line 38 of file fca.c.

References smyFormalContext::attributeNames, smyFormalContext::attributes, smyFormalContext::incidence, smyFormalContext::objectNames, and smyFormalContext::objects.

```
{
    myFormalContext *ctx = malloc(sizeof(myFormalContext));

    ctx->attributes = attributes;
    ctx->objects = objects;

#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wsign-conversion"

    ctx->attributeNames = calloc(attributes, sizeof(char*));
    ctx->objectNames = calloc(objects, sizeof(char*));
```

```
#pragma GCC diagnostic pop

    for (int var = 0; var < attributes; ++var)
    {
        ctx->attributeNames[var] = calloc(1, sizeof(char));
    }

    for (int var = 0; var < objects; ++var)
    {
        ctx->objectNames[var] = calloc(1, sizeof(char));
    }

#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wsign-conversion"

    ctx->incidence = calloc(objects * attributes, sizeof(IncidenceCell));

#pragma GCC diagnostic pop

    return (FormalContext) ctx;
}
```

4.6.3.4 FormalContext newFormalContextFromFile (const char * *filename*)

create a new formal context object from a .cxt file

Parameters

filename	

Returns

the formal context that has been read from the given file

Definition at line 80 of file fca.c.

References smyFormalContext::attributeNames, CELL, CROSS, INPUTBUFFERSIZE, MIN, newFormalContext, smyFormalContext::objectNames, and RETURN_ZERO_IF_-ZERO.

```
{
    char *line;
    size_t len;

    len = (INPUTBUFFERSIZE);
    line = malloc(sizeof(char) * len);

    FILE *file;

    if (strcmp(filename, "-") == 0)
    {
        file = stdin;
    }
    else
    {
        file = fopen(filename, "r");
```

```
    RETURN_ZERO_IF_ZERO(file);
}

ssize_t read;

int line_nbr;
line_nbr = 0;

int objects;
int attributes;

attributes = 0;
objects = 0;

myFormalContext *ctx;
ctx = 0;

while ((read = getline(&line, &len, file)) != -1)
{

    /*
     * this should never happen, right?
     */
    if (read == 0)
        break;
    line[read - 1] = 0;

    if (line_nbr == 0)
    {
        if (strcmp(line, "B"))
        {
            fprintf(stderr, "File '%s' is not a .cxt file\n", filename);
            goto grace;
        }
    }
    else if (line_nbr == 1)
    {
        //empty line
    }
    else if (line_nbr == 2)
    {
        objects = atoi(line);
    }
    else if (line_nbr == 3)
    {
        attributes = atoi(line);
        ctx = (myFormalContext *) newFormalContext(objects, attributes);
    }
    else if (line_nbr == 4)
    {
        //empty line
    }
    else if (line_nbr < objects + 5)
    {

        int i;
        i = line_nbr - 5;

        free(ctx->objectNames[i]);
        ctx->objectNames[i] = strdup(line);

    }
```

```
        else if (line_nbr < objects + attributes + 5)
        {
            int i;
            i = line_nbr - 5 - objects;

            free(ctx->attributeNames[i]);
            ctx->attributeNames[i] = strdup(line);
        }
        else if (line_nbr < objects * 2 + attributes + 5)
        {
            int i;
            i = line_nbr - 5 - objects - attributes;

            int width;
            width = MIN((signed)strlen(line),attributes);

            for (int var = 0; var < width; ++var)
            {
                if ((line[var] == 'x') || (line[var] == 'X')
                        || (line[var] == '1'))
                {
                    CROSS(CELL (i, ctx, var));
                }
            }

        }
        else
        {
            /*
             * we read all data
             */
            break;
        }

        line_nbr++;
    }

    /*
     * free memory and return
     */

    grace: if (file != stdin)
    {
        fclose(file);
    }

    free(line);

    return (FormalContext) ctx;
}
```

4.6.3.5 FormalContext newFormalContextFromRandom (int *objects*, int *attributes*, float *p*)

create a new formal context with random incidence relation

Parameters

objects	
attributes	
p	probability of a cross

Returns

context

Definition at line 798 of file fca.c.

References smyFormalContext::attributes, CELL, CROSS, newFormalContext, and smyFormalContext::objects.

```
{
    FormalContext ctx;
    ctx = newFormalContext(objects, attributes);

    myFormalContext *c;

    c = (myFormalContext*) ctx;

    for (int g = 0; g < c->objects; ++g)
    {
        for (int m = 0; m < c->attributes; ++m)
        {
            float x;
            x = (float) random() / (float) RAND_MAX;
            if (x >= p)
            {
                CROSS(CELL(g,c,m));
            }
        }
    }
    return ctx;
}
```

4.6.3.6 void writeFormalContext (FormalContext *ctx,* const char * *filename*)

save the context ctx at the given file location

Parameters

ctx	
filename	

Definition at line 216 of file fca.c.

References smyFormalContext::attributeNames, smyFormalContext::attributes, gIm, smyFormalContext::objectNames, smyFormalContext::objects, and RETURN_IF_ZERO.

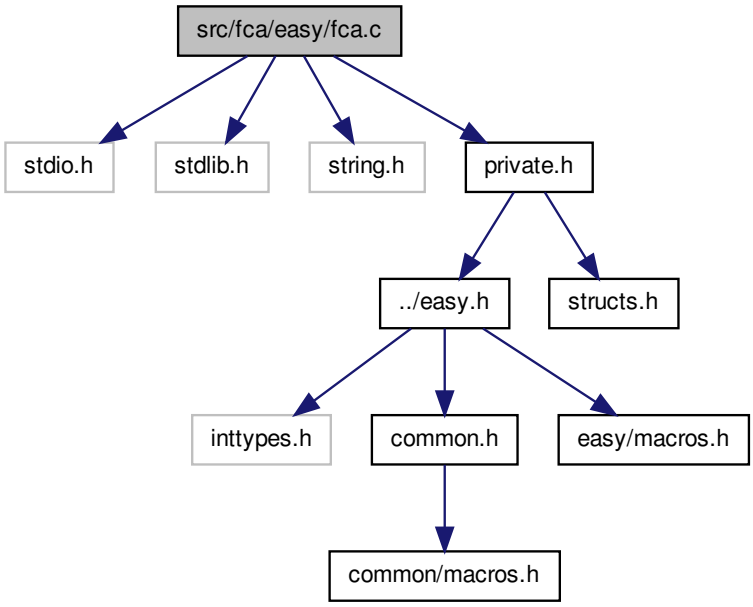
```
{
```

```
    RETURN_IF_ZERO(ctx);
    RETURN_IF_ZERO(filename);

    FILE* file;
    file = fopen(filename, "w");

    RETURN_IF_ZERO(file);

    myFormalContext *c;
    c = (myFormalContext*) ctx;

    fprintf(file, "B\n\n%d\n%d\n\n", c->objects, c->attributes);

    for (int var = 0; var < c->objects; ++var)
    {
        fputs(c->objectNames[var], file);
        fputs("\n", file);
    }

    for (int var = 0; var < c->attributes; ++var)
    {
        fputs(c->attributeNames[var], file);
        fputs("\n", file);
    }

    for (int g = 0; g < c->objects; ++g)
    {
        for (int m = 0; m < c->attributes; ++m)
        {
            if ( gIm(g, c, m))
                fputs("X", file);
            else
                fputs(".", file);
        }
        fputs("\n", file);
    }

    fclose(file);
}
```

4.7 src/fca/easy/fca.c File Reference

fca.c, (c) 2013, Immanuel Albrecht; Dresden University of Technology, Professur für die Psychologie des Lernen und Lehrens

`#include <stdio.h>` `#include <stdlib.h>` `#include <string.h>` `#include "private.h"` Include dependency graph for fca.c:



Functions

- FormalContext newFormalContext (int objects, int attributes)
  create a new formal context
- FormalContext newFormalContextFromFile (const char *filename)
  create a new formal context object from a .cxt file
- void writeFormalContext (FormalContext ctx, const char *filename)
  save the context ctx at the given file location
- void deleteFormalContext (FormalContext *ctx)
  *deletes the formal context *ctx, and sets the pointer to zero*
- myFormalConceptIntentChunk * newConceptChunk (int attributes)
  create a new formal concept chunk
- void deleteConceptChunk (myFormalConceptIntentChunk **c)
  deletes a concept chunk object and sets its pointer to zero
- FormalConceptIntentBulkList newConceptBulk (int attributes)
  creates a new formal concept intent bulk list
- void deleteConceptBulk (FormalConceptIntentBulkList *rootNode)

deletes the entire bulk list

- int countConceptsInBulk (FormalConceptIntentBulkList root)

  use this for bulks that are filled in order

- FormalConceptIntentBulkList addConceptToBulk (FormalConceptIntentBulkList root, const IncidenceCell *intent)

  copies the given intent to the bulk denoted by the root node.

- void closeIntent2 (FormalContext restrict ctx, const IncidenceCell *restrict input, IncidenceCell *restrict outputIntent, IncidenceCell *restrict outputExtent)

  close an attribute set, i.e.

- void closeIntent (FormalContext ctx, const IncidenceCell *restrict input, IncidenceCell *restrict output)

  close an attribute set, i.e.

- int intentCmp (int attributes, const IncidenceCell *minus, const IncidenceCell *plus)

  compare two intent vectors

- FormalConceptIntentBulkList newConceptBulkFromContext (FormalContext ctx)

  creates a new formal concept intent chunk and fills it with the intents of all formal concepts in the concept lattice of ctx, using next closure algorithm

- void writeConceptsToFile (FormalContext ctx, FormalConceptIntentBulkList root, const char *filename)

  write a list of concept intents into a .cxt file

- FormalContext newFormalContextFromRandom (int objects, int attributes, float p)

  create a new formal context with random incidence relation

- int countContextConcepts2 (FormalContext ctx)

  counts the concepts in the concept lattice of ctx, using next closure algorithm

- int countContextConcepts (FormalContext ctx)

  counts the concepts in the concept lattice of ctx, using next closure algorithm

4.7.1 Detailed Description

fca.c, (c) 2013, Immanuel Albrecht; Dresden University of Technology, Professur für die Psychologie des Lernen und Lehrens This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>. this file contains general formal context related operations and routines

Definition in file fca.c.

4.7.2 Function Documentation

4.7.2.1 FormalConceptIntentBulkList addConceptToBulk (FormalConceptIntentBulkList *root,* const IncidenceCell * *intent*)

copies the given intent to the bulk denoted by the root node.

Parameters

root	root node of the bulk
intent	read-only pointer to an array of IncidenceCell[root->attributes]

Returns

the node where the intent was added to the last chunk

Definition at line 423 of file fca.c.

References sFormalConceptIntentBulkNode::attributes, BULKSIZE, CELL, sFormalConceptIntentBulkNode::chunks, CHUNKSIZE, newConceptBulk, newConceptChunk, sFormalConceptIntentBulkNode::next, RETURN_ZERO_IF_ZERO, smyFormalConceptIntentChunk::size, and sFormalConceptIntentBulkNode::size.

```
{
    RETURN_ZERO_IF_ZERO(root);

    do
    {

        if (root->size == 0)
        {
            root->chunks[0] = newConceptChunk(root->attributes);
            root->size = 1;
        }

        int last_index;
        last_index = root->size - 1;

        if (root->chunks[last_index]->size == CHUNKSIZE)
        {
            if (root->size == BULKSIZE)
            {
                if (root->next == 0)
                {
                    root->next = newConceptBulk(root->attributes);
                }
                root = root->next;
                continue;
            }
            else
            {
                last_index = root->size++;
                root->chunks[last_index] = newConceptChunk(root->attributes);
            }
        }

#pragma GCC diagnostic push
```

```
#pragma GCC diagnostic ignored "-Wsign-conversion"

        memcpy(
                &(CELL(root->chunks[last_index]->size, root->chunks[last_index]
      ,0)),
                intent, sizeof(IncidenceCell) * root->attributes);

#pragma GCC diagnostic pop

        root->chunks[last_index]->size++;

        break;

    } while (1);

    return root;
}
```

4.7.2.2 void closeIntent (FormalContext *ctx,* const IncidenceCell ∗restrict *input,* IncidenceCell ∗restrict *output*)

close an attribute set, i.e.

add further attributes

Parameters

ctx	formal context
input	the intent set that is to be closed
output	the closure intent" wrt. ctx

Definition at line 534 of file fca.c.

References smyFormalContext::attributes, CLEAR, CROSS, gIm, INCIDES, and smyFormalContext::objects.

```
{

    myFormalContext* I;
    I = (myFormalContext*) ctx;
    for (int var = 0; var < I->attributes; ++var)
    {
        CROSS(output[var]);
    }

    for (int g = 0; g < I->objects; ++g)
    {
        int good;
        good = 1;

        for (int m = 0; m < I->attributes; ++m)
        {
            if (INCIDES(input[m]))
                if (!gIm(g,I,m))
                {
                    /*
                     * some attribute is not present for this object -> next
```

```
        object
                      */
                     good = 0;
                     break;
                 }
         }
         if (good)
             /*
              * remove attributes that are not common among all objects that
        have the input
              * attributes
              */
             for (int m = 0; m < I->attributes; ++m)
             {
                 if (!gIm(g,I,m))
                 {
                     CLEAR(output[m]);
                 }
             }
    }
}
```

4.7.2.3 void closeIntent2 (FormalContext restrict *ctx,* const IncidenceCell *restrict *input,* IncidenceCell *restrict *outputIntent,* IncidenceCell *restrict *outputExtent*)

close an attribute set, i.e.

add further attributes, 1.92 times slower than closeIntent

Parameters

ctx	formal context
input	the intent set that is to be closed
outputIntent	the closure intent" wrt. ctx
outputExtent	the corresponding objects, i.e. intent' wrt. ctx

Definition at line 487 of file fca.c.

References CLEAR, CROSS, gIm, and INCIDES.

```
{
    myFormalContext* restrict I;
    I = (myFormalContext*) ctx;

    for (int g = 0; g < I->objects; ++g)
    {
        CROSS(outputExtent[g]);
        for (int m = 0; m < I->attributes; ++m)
        {
            if (INCIDES(input[m]))
                if (!gIm(g,I,m))
                {
                    /*
                     * some attribute is not present for this object -> next
       object
                     */
                    CLEAR(outputExtent[g]);
```

```
                    break;
                }
            }
        }

        for (int m = 0; m < I->attributes; ++m)
        {
            CROSS(outputIntent[m]);
            for (int g = 0; g < I->objects; ++g)
            {
                if (INCIDES(outputExtent[g]))
                    if (!gIm(g,I,m))
                    {
                        CLEAR(outputIntent[m]);
                        break;
                    }
            }
        }

}
```

4.7.2.4 int countConceptsInBulk (FormalConceptIntentBulkList *root*)

use this for bulks that are filled in order

Parameters

root	

Returns

number of concepts in bulk

Definition at line 388 of file fca.c.

References sFormalConceptIntentBulkNode::chunks, CHUNKSIZE, sFormalConceptIntentBulkNode::next, RETURN_ZERO_IF_ZERO, smyFormalConceptIntentChunk::size, and sFormalConceptIntentBulkNode::size.

```
{
    RETURN_ZERO_IF_ZERO(root);

    int count = 0;

    while (root != 0)
    {
        if (root->size > 0)
        {
            /*
             * count the full chunks
             */
            count += CHUNKSIZE * (root->size - 1);
            /*
             * and the last chunk
             */
            count += root->chunks[root->size - 1]->size;
```

```
        }
        root = root->next;
    }

    return count;
}
```

4.7.2.5 int countContextConcepts (FormalContext *ctx*)

counts the concepts in the concept lattice of ctx, using next closure algorithm

Parameters

ctx	formal context

Returns

number of concepts in context

Definition at line 926 of file fca.c.

References smyFormalContext::attributes, CLEAR, closeIntent, CROSS, INCIDES, and RETURN_ZERO_IF_ZERO.

```
{
    RETURN_ZERO_IF_ZERO(ctx);

    myFormalContext *c;
    c = (myFormalContext*) ctx;

    IncidenceCell *M;
    IncidenceCell *Y;

#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wsign-conversion"

    Y = calloc(c->attributes, sizeof(IncidenceCell));
    M = malloc(c->attributes * sizeof(IncidenceCell));

#pragma GCC diagnostic pop

    /*
     * calculate the bottom intent of the concept lattice, i.e. {}''
     */
    closeIntent(ctx, Y, M);

    int count;

    count = 1;

    /*
     * begin of nextClosure function iteration
     */
    nextClosure:

    for (int i = c->attributes - 1; i >= 0; --i)
```

```
    {

        if (!INCIDES(M[i]))
        {
            CROSS(M[i]);
            closeIntent(ctx, M, Y);

            int good;
            good = 1;

            for (int j = 0; j < i; ++j)
            {
                if (INCIDES(Y[j]))
                {
                    if (!INCIDES((M[j])))
                    {

                        good = 0;
                        break;
                    }
                }
            }
            if (good)
            {
                /*
                 * we found the next intent
                 */
                count++;

                /*
                 * continue with Y for M
                 */

                IncidenceCell *DELTA;
                DELTA = M;
                M = Y;
                Y = DELTA;
                /*
                 * do the nextClosure
                 */
                goto nextClosure;
            }
        }

        CLEAR(M[i]);
    }

    /*
     * free up memory
     */

    free(M);
    free(Y);

    return count;
}
```

4.7.2.6 int countContextConcepts2 (FormalContext *ctx*)

counts the concepts in the concept lattice of ctx, using next closure algorithm

Parameters

ctx	formal context

Returns

number of concepts in context

Definition at line 828 of file fca.c.

References smyFormalContext::attributes, CLEAR, closeIntent2, CROSS, INCIDES, smyFormalContext::objects, and RETURN_ZERO_IF_ZERO.

```
{
    RETURN_ZERO_IF_ZERO(ctx);

    myFormalContext *c;
    c = (myFormalContext*) ctx;

    IncidenceCell *M;
    IncidenceCell *Y;
    IncidenceCell *extent;

#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wsign-conversion"

    Y = calloc(c->attributes, sizeof(IncidenceCell));
    M = malloc(c->attributes * sizeof(IncidenceCell));
    extent = malloc(c->objects * sizeof(IncidenceCell));

#pragma GCC diagnostic pop

    /*
     * calculate the bottom intent of the concept lattice, i.e. {}''
     */
    closeIntent2(ctx, Y, M, extent);

    int count;

    count = 1;

    /*
     * begin of nextClosure function iteration
     */
    nextClosure:

    for (int i = c->attributes - 1; i >= 0; --i)
    {

        if (!INCIDES(M[i]))
        {
            CROSS(M[i]);
            closeIntent2(ctx, M, Y, extent);

            int good;
            good = 1;

            for (int j = 0; j < i; ++j)
            {
                if (INCIDES(Y[j]))
```

```
                {
                    if (!INCIDES((M[j])))
                    {

                        good = 0;
                        break;
                    }
                }
            }
            if (good)
            {
                /*
                 * we found the next intent
                 */
                count++;

                /*
                 * continue with Y for M
                 */

                IncidenceCell *DELTA;
                DELTA = M;
                M = Y;
                Y = DELTA;
                /*
                 * do the nextClosure
                 */
                goto nextClosure;
            }
        }

        CLEAR(M[i]);
    }

    /*
     * free up memory
     */

    free(M);
    free(Y);
    free(extent);
    return count;
}
```

4.7.2.7 void deleteConceptBulk (FormalConceptIntentBulkList * *rootNode*)

deletes the entire bulk list

Parameters

rootNode	pointer to the first node

Definition at line 356 of file fca.c.

References sFormalConceptIntentBulkNode::chunks, deleteConceptChunk, sFormalConceptIntentBulkNode::next, RETURN_IF_ZERO, and sFormalConceptIntentBulkNode::size.

```
{
    RETURN_IF_ZERO(rootNode);
    RETURN_IF_ZERO(*rootNode);

    FormalConceptIntentBulkList l;
    l = *rootNode;
    *rootNode = 0;

    do
    {
        for (int var = 0; var < l->size; ++var)
        {
            deleteConceptChunk(&(l->chunks[var]));
        }

        FormalConceptIntentBulkList next;
        next = l->next;

        free(l->chunks);
        free(l);

        l = next;
    } while (l != 0);
}
```

4.7.2.8 void deleteConceptChunk (myFormalConceptIntentChunk ** *c*)

deletes a concept chunk object and sets its pointer to zero

Parameters

c	pointer to the concept chunk to be deleted

Definition at line 321 of file fca.c.

References RETURN_IF_ZERO.

```
{
    RETURN_IF_ZERO(c);
    RETURN_IF_ZERO(*c);

    free((*c)->incidence);

    free(*c);
    *c = 0;
}
```

4.7.2.9 void deleteFormalContext (FormalContext * *ctx*)

deletes the formal context *ctx, and sets the pointer to zero

Parameters

ctx	pointer to the formal context object to be deleted

Definition at line 264 of file fca.c.

References smyFormalContext::attributeNames, smyFormalContext::attributes, smyFormalContext::incidence, smyFormalContext::objectNames, smyFormalContext::objects, and RETURN_IF_ZERO.

```
{
    RETURN_IF_ZERO(ctx);
    RETURN_IF_ZERO(*ctx);

    myFormalContext *c;

    c = (myFormalContext*) *ctx;

    *ctx = 0;

    for (int var = 0; var < c->attributes; ++var)
    {
        free(c->attributeNames[var]);
    }

    for (int var = 0; var < c->objects; ++var)
    {
        free(c->objectNames[var]);
    }

    free(c->objectNames);
    free(c->attributeNames);
    free(c->incidence);
    free(c);
}
```

4.7.2.10 int intentCmp (int *attributes*, const IncidenceCell * *minus*, const IncidenceCell * *plus*)

compare two intent vectors

Parameters

attributes	attribute count
minus	"left" operand
plus	"right" operand

Returns

-1 if minus is bigger, 1 if plus is bigger, 0 if minus and plus is the same

Definition at line 587 of file fca.c.

References INCIDES.

```
{
    for (int var = 0; var < attributes; ++var)
    {
        if (INCIDES(minus[var]))
```

```
        {
            if (!INCIDES((plus[var])))
                return -1;
        }
        else if (INCIDES(plus[var]))
        {
            return 1;
        }
    }
    return 0;
}
```

4.7.2.11 FormalConceptIntentBulkList newConceptBulk (int *attributes*)

creates a new formal concept intent bulk list

Parameters

attributes	number of attributes of the concept intents

Returns

new formal concept intent bulk list's first node

Definition at line 339 of file fca.c.

References sFormalConceptIntentBulkNode::attributes, BULKSIZE, sFormalConceptIntentBulkNode::chunks, sFormalConceptIntentBulkNode::next, and sFormalConceptIntentBulkNode::size.

```
{
    FormalConceptIntentBulkList l;
    l = malloc(sizeof(struct sFormalConceptIntentBulkNode));

    l->attributes = attributes;
    l->size = 0;
    l->chunks = calloc(BULKSIZE, sizeof(myFormalConceptIntentChunk*));
    l->next = 0;
    return l;
}
```

4.7.2.12 FormalConceptIntentBulkList newConceptBulkFromContext (FormalContext *ctx*)

creates a new formal concept intent chunk and fills it with the intents of all formal concepts in the concept lattice of ctx, using next closure algorithm

Parameters

ctx	formal context

Returns

concept intents

Definition at line 612 of file fca.c.

References addConceptToBulk, smyFormalContext::attributes, CLEAR, closeIntent, CROSS, INCIDES, newConceptBulk, and RETURN_ZERO_IF_ZERO.

```
{
    RETURN_ZERO_IF_ZERO(ctx);

    myFormalContext *c;
    c = (myFormalContext*) ctx;

    IncidenceCell *M;
    IncidenceCell *Y;

#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wsign-conversion"

    Y = calloc(c->attributes, sizeof(IncidenceCell));
    M = malloc(c->attributes * sizeof(IncidenceCell));

#pragma GCC diagnostic pop

    /*
     * calculate the bottom intent of the concept lattice, i.e. {}''
     */
    closeIntent(ctx, Y, M);

    FormalConceptIntentBulkList root;
    FormalConceptIntentBulkList last;

    root = newConceptBulk(c->attributes);

    /*
     * add the bottom element of the concept lattice (a concept lattice is
       never empty)
     */

    last = addConceptToBulk(root, M);

    /*
     * begin of nextClosure function iteration
     */
    nextClosure:

    for (int i = c->attributes - 1; i >= 0; --i)
    {

        if (!INCIDES(M[i]))
        {
            CROSS(M[i]);
            closeIntent(ctx, M, Y);

            int good;
            good = 1;

            for (int j = 0; j < i; ++j)
            {
                if (INCIDES(Y[j]))
```

```
                {
                    if (!INCIDES((M[j])))
                    {

                        good = 0;
                        break;
                    }
                }
            }
            if (good)
            {
                /*
                 * we found the next intent
                 */
                last = addConceptToBulk(last, Y);

                /*
                 * continue with Y for M
                 */

                IncidenceCell *DELTA;
                DELTA = M;
                M = Y;
                Y = DELTA;
                /*
                 * do the nextClosure
                 */
                goto nextClosure;
            }
        }

        CLEAR(M[i]);
    }

    /*
     * free up memory
     */

    free(M);
    free(Y);
    return root;
}
```

4.7.2.13 myFormalConceptIntentChunk* newConceptChunk (int *attributes*)

create a new formal concept chunk

easy/private.h, (c) 2013, Immanuel Albrecht; Dresden University of Technology, - Professur für die Psychologie des Lernen und Lehrens

Parameters

attributes	number of attributes of the hosting formal context

Returns

a new concept chunk object

Definition at line 298 of file fca.c.

References smyFormalConceptIntentChunk::attributes, CHUNKSIZE, smyFormalConceptIntentChunk::incidence, and smyFormalConceptIntentChunk::size.

```
{
    myFormalConceptIntentChunk *c;
    c = malloc(sizeof(myFormalConceptIntentChunk));
    c->attributes = attributes;
    c->size = 0;

#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wsign-conversion"

    c->incidence = calloc(attributes * CHUNKSIZE, sizeof(IncidenceCell));

#pragma GCC diagnostic pop

    return c;
}
```

4.7.2.14 FormalContext newFormalContext (int *objects,* int *attributes*)

create a new formal context

Parameters

objects	object count
attributes	attribute count

Returns

a new FormalContext object

Definition at line 38 of file fca.c.

References smyFormalContext::attributeNames, smyFormalContext::attributes, smyFormalContext::incidence, smyFormalContext::objectNames, and smyFormalContext::objects.

```
{
    myFormalContext *ctx = malloc(sizeof(myFormalContext));

    ctx->attributes = attributes;
    ctx->objects = objects;

#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wsign-conversion"

    ctx->attributeNames = calloc(attributes, sizeof(char*));
    ctx->objectNames = calloc(objects, sizeof(char*));
```

```
#pragma GCC diagnostic pop

    for (int var = 0; var < attributes; ++var)
    {
        ctx->attributeNames[var] = calloc(1, sizeof(char));
    }

    for (int var = 0; var < objects; ++var)
    {
        ctx->objectNames[var] = calloc(1, sizeof(char));
    }

#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wsign-conversion"

    ctx->incidence = calloc(objects * attributes, sizeof(IncidenceCell));

#pragma GCC diagnostic pop

    return (FormalContext) ctx;
}
```

4.7.2.15 FormalContext newFormalContextFromFile (const char * *filename*)

create a new formal context object from a .cxt file

Parameters

filename	

Returns

the formal context that has been read from the given file

Definition at line 80 of file fca.c.

References smyFormalContext::attributeNames, CELL, CROSS, INPUTBUFFERSIZE, MIN, newFormalContext, smyFormalContext::objectNames, and RETURN_ZERO_IF_-ZERO.

```
{
    char *line;
    size_t len;

    len = (INPUTBUFFERSIZE);
    line = malloc(sizeof(char) * len);

    FILE *file;

    if (strcmp(filename, "-") == 0)
    {
        file = stdin;
    }
    else
    {
```

```
    file = fopen(filename, "r");
    RETURN_ZERO_IF_ZERO(file);
}

ssize_t read;

int line_nbr;
line_nbr = 0;

int objects;
int attributes;

attributes = 0;
objects = 0;

myFormalContext *ctx;
ctx = 0;

while ((read = getline(&line, &len, file)) != -1)
{

    /*
     * this should never happen, right?
     */
    if (read == 0)
        break;
    line[read - 1] = 0;

    if (line_nbr == 0)
    {
        if (strcmp(line, "B"))
        {
            fprintf(stderr, "File '%s' is not a .cxt file\n", filename);
            goto grace;
        }
    }
    else if (line_nbr == 1)
    {
        //empty line
    }
    else if (line_nbr == 2)
    {
        objects = atoi(line);
    }
    else if (line_nbr == 3)
    {
        attributes = atoi(line);
        ctx = (myFormalContext *) newFormalContext(objects, attributes);
    }
    else if (line_nbr == 4)
    {
        //empty line
    }
    else if (line_nbr < objects + 5)
    {

        int i;
        i = line_nbr - 5;

        free(ctx->objectNames[i]);
        ctx->objectNames[i] = strdup(line);
```

```
        }
        else if (line_nbr < objects + attributes + 5)
        {
            int i;
            i = line_nbr - 5 - objects;

            free(ctx->attributeNames[i]);
            ctx->attributeNames[i] = strdup(line);
        }
        else if (line_nbr < objects * 2 + attributes + 5)
        {
            int i;
            i = line_nbr - 5 - objects - attributes;

            int width;
            width = MIN((signed)strlen(line),attributes);

            for (int var = 0; var < width; ++var)
            {
                if ((line[var] == 'x') || (line[var] == 'X')
                        || (line[var] == '1'))
                {
                    CROSS(CELL (i, ctx, var));
                }
            }

        }
        else
        {
            /*
             * we read all data
             */
            break;
        }

        line_nbr++;
    }

    /*
     * free memory and return
     */

    grace: if (file != stdin)
    {
        fclose(file);
    }

    free(line);

    return (FormalContext) ctx;
}
```

4.7.2.16 FormalContext newFormalContextFromRandom (int *objects*, int *attributes*, float *p*)

create a new formal context with random incidence relation

Parameters

objects	
attributes	
p	probability of a cross

Returns

context

Definition at line 798 of file fca.c.

References smyFormalContext::attributes, CELL, CROSS, newFormalContext, and smyFormalContext::objects.

```
{
    FormalContext ctx;
    ctx = newFormalContext(objects, attributes);

    myFormalContext *c;

    c = (myFormalContext*) ctx;

    for (int g = 0; g < c->objects; ++g)
    {
        for (int m = 0; m < c->attributes; ++m)
        {
            float x;
            x = (float) random() / (float) RAND_MAX;
            if (x >= p)
            {
                CROSS(CELL(g,c,m));
            }
        }
    }
    return ctx;
}
```

4.7.2.17 void writeConceptsToFile (FormalContext *ctx,* FormalConceptIntentBulkList *root,* const char * *filename*)

write a list of concept intents into a .cxt file

Parameters

ctx	formal context (or 0, is used for attribute names)
root	the first node of the formal concept intent bulk
filename	output file name (.cxt)

Definition at line 716 of file fca.c.

References smyFormalContext::attributeNames, smyFormalContext::attributes, sFormalConceptIntentBulkNode::attributes, sFormalConceptIntentBulkNode::chunks, countConceptsInBulk, gIm, sFormalConceptIntentBulkNode::next, RETURN_IF_ZE-

RO, smyFormalConceptIntentChunk::size, sFormalConceptIntentBulkNode::size, and WARN_IF_UNEQUAL_DO.

```
{
    RETURN_IF_ZERO(root);

    myFormalContext* c;

    if (ctx != 0)
    {
        c = (myFormalContext*) ctx;

        WARN_IF_UNEQUAL_DO(c->attributes, root->attributes, c = 0);
    }
    else
    {
        c = 0;
    }

    RETURN_IF_ZERO(filename);

    FILE* file;
    file = fopen(filename, "w");

    RETURN_IF_ZERO(file);

    int objects;
    objects = countConceptsInBulk(root);

    fprintf(file, "B\n\n%d\n%d\n\n", objects, root->attributes);

    for (int var = 0; var < objects; ++var)
    {
        fprintf(file, "C%8d\n", (var + 1));
    }

    if (c != 0)
    {
        for (int var = 0; var < c->attributes; ++var)
        {
            fputs(c->attributeNames[var], file);
            fputs("\n", file);
        }
    }
    else
    {
        for (int var = 0; var < root->attributes; ++var)
        {
            fprintf(file, "m%8d\n", (var + 1));
        }
    }

    for (; root != 0; root = root->next)
    {
        for (int chunk = 0; chunk < root->size; ++chunk)
        {
            for (int g = 0; g < root->chunks[chunk]->size; ++g)
            {
                for (int m = 0; m < root->attributes; ++m)
                {
                    if ( gIm(g, root->chunks[chunk], m))
                        fputs("X", file);
```

```
                    else
                        fputs(".", file);
                }
                fputs("\n", file);
            }
        }
    }

    fclose(file);

}
```

4.7.2.18 void writeFormalContext (FormalContext *ctx*, const char * *filename*)

save the context ctx at the given file location

Parameters

ctx	
filename	

Definition at line 216 of file fca.c.

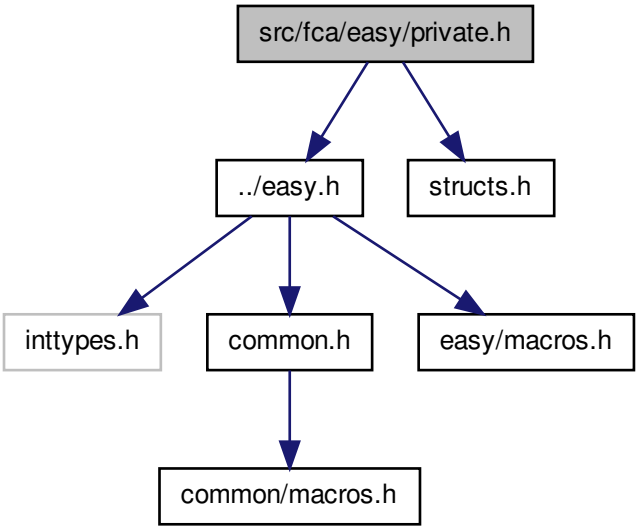
References smyFormalContext::attributeNames, smyFormalContext::attributes, gIm, smyFormalContext::objectNames, smyFormalContext::objects, and RETURN_IF_ZERO.

```
{
    RETURN_IF_ZERO(ctx);
    RETURN_IF_ZERO(filename);

    FILE* file;
    file = fopen(filename, "w");

    RETURN_IF_ZERO(file);

    myFormalContext *c;
    c = (myFormalContext*) ctx;

    fprintf(file, "B\n\n%d\n%d\n\n", c->objects, c->attributes);

    for (int var = 0; var < c->objects; ++var)
    {
        fputs(c->objectNames[var], file);
        fputs("\n", file);
    }

    for (int var = 0; var < c->attributes; ++var)
    {
        fputs(c->attributeNames[var], file);
        fputs("\n", file);
    }

    for (int g = 0; g < c->objects; ++g)
    {
        for (int m = 0; m < c->attributes; ++m)
        {
```

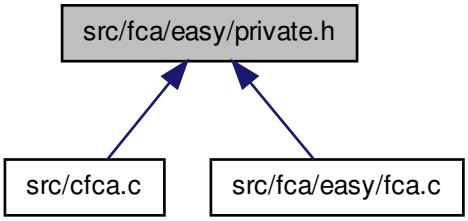
```
            if ( gIm(g, c, m))
                fputs("X", file);
            else
                fputs(".", file);
        }
        fputs("\n", file);
    }

    fclose(file);
}
```

4.8 src/fca/easy/private.h File Reference

`#include "../easy.h"` `#include "structs.h"` Include dependency graph for private.h:



This graph shows which files directly or indirectly include this file:



Functions

- myFormalConceptIntentChunk ∗ newConceptChunk (int attributes)

  easy/private.h, (c) 2013, Immanuel Albrecht; Dresden University of Technology, - Professur für die Psychologie des Lernen und Lehrens
- void deleteConceptChunk (myFormalConceptIntentChunk ∗∗c)

  deletes a concept chunk object and sets its pointer to zero
- FormalConceptIntentBulkList newConceptBulk (int attributes)

  creates a new formal concept intent bulk list
- FormalConceptIntentBulkList newConceptBulkFromContext (FormalContext ctx)

  creates a new formal concept intent chunk and fills it with the intents of all formal concepts in the concept lattice of ctx, using next closure algorithm
- void writeConceptsToFile (FormalContext ctx, FormalConceptIntentBulkList root, const char ∗filename)

  write a list of concept intents into a .cxt file
- void deleteConceptBulk (FormalConceptIntentBulkList ∗rootNode)

  deletes the entire bulk list
- int countConceptsInBulk (FormalConceptIntentBulkList root)

  use this for bulks that are filled in order
- FormalConceptIntentBulkList addConceptToBulk (FormalConceptIntentBulkList root, const IncidenceCell ∗intent)

  copies the given intent to the bulk denoted by the root node.
- void closeIntent (FormalContext restrict ctx, const IncidenceCell ∗restrict input, IncidenceCell ∗restrict output)
- void closeIntent2 (FormalContext restrict ctx, const IncidenceCell ∗restrict input, IncidenceCell ∗restrict outputIntent, IncidenceCell ∗restrict outputExtent)

  close an attribute set, i.e.

- int intentCmp (int attributes, const IncidenceCell *minus, const IncidenceCell *plus)

  compare two intent vectors

4.8.1 Function Documentation

4.8.1.1 FormalConceptIntentBulkList addConceptToBulk (FormalConceptIntentBulkList *root,* const IncidenceCell * *intent*)

copies the given intent to the bulk denoted by the root node.

Parameters

root	root node of the bulk
intent	read-only pointer to an array of IncidenceCell[root->attributes]

Returns

the node where the intent was added to the last chunk

Definition at line 423 of file fca.c.

References sFormalConceptIntentBulkNode::attributes, BULKSIZE, CELL, sFormalConceptIntentBulkNode::chunks, CHUNKSIZE, newConceptBulk, newConceptChunk, sFormalConceptIntentBulkNode::next, RETURN_ZERO_IF_ZERO, smyFormalConceptIntentChunk::size, and sFormalConceptIntentBulkNode::size.

```
{
    RETURN_ZERO_IF_ZERO(root);

    do
    {

        if (root->size == 0)
        {
            root->chunks[0] = newConceptChunk(root->attributes);
            root->size = 1;
        }

        int last_index;
        last_index = root->size - 1;

        if (root->chunks[last_index]->size == CHUNKSIZE)
        {
            if (root->size == BULKSIZE)
            {
                if (root->next == 0)
                {
                    root->next = newConceptBulk(root->attributes);
                }
                root = root->next;
                continue;
            }
            else
            {
```

```
                last_index = root->size++;
                root->chunks[last_index] = newConceptChunk(root->attributes);
            }
        }

#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wsign-conversion"

        memcpy(
                &(CELL(root->chunks[last_index]->size, root->chunks[last_index]
      ,0)),
                intent, sizeof(IncidenceCell) * root->attributes);

#pragma GCC diagnostic pop

        root->chunks[last_index]->size++;

        break;

    } while (1);

    return root;
}
```

4.8.1.2 void closeIntent (FormalContext restrict *ctx,* const IncidenceCell *restrict *input,* IncidenceCell *restrict *output*)

4.8.1.3 void closeIntent2 (FormalContext restrict *ctx,* const IncidenceCell *restrict *input,* IncidenceCell *restrict *outputIntent,* IncidenceCell *restrict *outputExtent*)

close an attribute set, i.e.

add further attributes, 1.92 times slower than closeIntent

Parameters

ctx	formal context
input	the intent set that is to be closed
outputIntent	the closure intent" wrt. ctx
outputExtent	the corresponding objects, i.e. intent' wrt. ctx

Definition at line 487 of file fca.c.

References CLEAR, CROSS, gIm, and INCIDES.

```
{
    myFormalContext* restrict I;
    I = (myFormalContext*) ctx;

    for (int g = 0; g < I->objects; ++g)
    {
        CROSS(outputExtent[g]);
        for (int m = 0; m < I->attributes; ++m)
        {
            if (INCIDES(input[m]))
                if (!gIm(g,I,m))
```

```
                {
                    /*
                     * some attribute is not present for this object -> next
       object
                     */
                    CLEAR(outputExtent[g]);
                    break;
                }
        }
    }

    for (int m = 0; m < I->attributes; ++m)
    {
        CROSS(outputIntent[m]);
        for (int g = 0; g < I->objects; ++g)
        {
            if (INCIDES(outputExtent[g]))
                if (!gIm(g,I,m))
                {
                    CLEAR(outputIntent[m]);
                    break;
                }
        }
    }

}
```

4.8.1.4 int countConceptsInBulk (FormalConceptIntentBulkList *root*)

use this for bulks that are filled in order

Parameters

root	

Returns

number of concepts in bulk

Definition at line 388 of file fca.c.

References sFormalConceptIntentBulkNode::chunks, CHUNKSIZE, sFormalConceptIntentBulkNode::next, RETURN_ZERO_IF_ZERO, smyFormalConceptIntentChunk::size, and sFormalConceptIntentBulkNode::size.

```
{
    RETURN_ZERO_IF_ZERO(root);

    int count = 0;

    while (root != 0)
    {
        if (root->size > 0)
        {
            /*
             * count the full chunks
```

```
             */
            count += CHUNKSIZE * (root->size - 1);
            /*
             * and the last chunk
             */
            count += root->chunks[root->size - 1]->size;
        }
        root = root->next;
    }

    return count;
}
```

4.8.1.5 void deleteConceptBulk (FormalConceptIntentBulkList * *rootNode*)

deletes the entire bulk list

Parameters

rootNode	pointer to the first node

Definition at line 356 of file fca.c.

References sFormalConceptIntentBulkNode::chunks, deleteConceptChunk, sFormalConceptIntentBulkNode::next, RETURN_IF_ZERO, and sFormalConceptIntentBulkNode::size.

```
{
    RETURN_IF_ZERO(rootNode);
    RETURN_IF_ZERO(*rootNode);

    FormalConceptIntentBulkList l;
    l = *rootNode;
    *rootNode = 0;

    do
    {
        for (int var = 0; var < l->size; ++var)
        {
            deleteConceptChunk(&(l->chunks[var]));
        }

        FormalConceptIntentBulkList next;
        next = l->next;

        free(l->chunks);
        free(l);

        l = next;
    } while (l != 0);
}
```

4.8.1.6 void deleteConceptChunk (myFormalConceptIntentChunk ** *c*)

deletes a concept chunk object and sets its pointer to zero

Parameters

c	pointer to the concept chunk to be deleted

Definition at line 321 of file fca.c.

References RETURN_IF_ZERO.

```
{
    RETURN_IF_ZERO(c);
    RETURN_IF_ZERO(*c);

    free((*c)->incidence);

    free(*c);
    *c = 0;
}
```

4.8.1.7 int intentCmp (int *attributes*, const IncidenceCell * *minus*, const IncidenceCell * *plus*)

compare two intent vectors

Parameters

attributes	attribute count
minus	"left" operand
plus	"right" operand

Returns

-1 if minus is bigger, 1 if plus is bigger, 0 if minus and plus is the same

Definition at line 587 of file fca.c.

References INCIDES.

```
{
    for (int var = 0; var < attributes; ++var)
    {
        if (INCIDES(minus[var]))
        {
            if (!INCIDES((plus[var])))
                return -1;
        }
        else if (INCIDES(plus[var]))
        {
            return 1;
        }
    }
    return 0;
}
```

4.8.1.8 FormalConceptIntentBulkList newConceptBulk (int *attributes*)

creates a new formal concept intent bulk list

Parameters

attributes	number of attributes of the concept intents

Returns

new formal concept intent bulk list's first node

Definition at line 339 of file fca.c.

References sFormalConceptIntentBulkNode::attributes, BULKSIZE, sFormalConceptIntentBulkNode::chunks, sFormalConceptIntentBulkNode::next, and sFormalConceptIntentBulkNode::size.

```
{
    FormalConceptIntentBulkList l;
    l = malloc(sizeof(struct sFormalConceptIntentBulkNode));

    l->attributes = attributes;
    l->size = 0;
    l->chunks = calloc(BULKSIZE, sizeof(myFormalConceptIntentChunk*));
    l->next = 0;
    return l;
}
```

4.8.1.9 FormalConceptIntentBulkList newConceptBulkFromContext (FormalContext *ctx*)

creates a new formal concept intent chunk and fills it with the intents of all formal concepts in the concept lattice of ctx, using next closure algorithm

Parameters

ctx	formal context

Returns

concept intents

Definition at line 612 of file fca.c.

References addConceptToBulk, smyFormalContext::attributes, CLEAR, closeIntent, CROSS, INCIDES, newConceptBulk, and RETURN_ZERO_IF_ZERO.

```
{
    RETURN_ZERO_IF_ZERO(ctx);
```

```
    myFormalContext *c;
    c = (myFormalContext*) ctx;

    IncidenceCell *M;
    IncidenceCell *Y;

#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wsign-conversion"

    Y = calloc(c->attributes, sizeof(IncidenceCell));
    M = malloc(c->attributes * sizeof(IncidenceCell));

#pragma GCC diagnostic pop

    /*
     * calculate the bottom intent of the concept lattice, i.e. {}''
     */
    closeIntent(ctx, Y, M);

    FormalConceptIntentBulkList root;
    FormalConceptIntentBulkList last;

    root = newConceptBulk(c->attributes);

    /*
     * add the bottom element of the concept lattice (a concept lattice is
       never empty)
     */

    last = addConceptToBulk(root, M);

    /*
     * begin of nextClosure function iteration
     */
    nextClosure:

    for (int i = c->attributes - 1; i >= 0; --i)
    {

        if (!INCIDES(M[i]))
        {
            CROSS(M[i]);
            closeIntent(ctx, M, Y);

            int good;
            good = 1;

            for (int j = 0; j < i; ++j)
            {
                if (INCIDES(Y[j]))
                {
                    if (!INCIDES((M[j])))
                    {

                        good = 0;
                        break;
                    }
                }
            }
            if (good)
            {
                /*
```

```
                 * we found the next intent
                 */
                last = addConceptToBulk(last, Y);

                /*
                 * continue with Y for M
                 */

                IncidenceCell *DELTA;
                DELTA = M;
                M = Y;
                Y = DELTA;
                /*
                 * do the nextClosure
                 */
                goto nextClosure;
            }
        }

        CLEAR(M[i]);
    }

    /*
     * free up memory
     */

    free(M);
    free(Y);
    return root;
}
```

4.8.1.10 myFormalConceptIntentChunk* newConceptChunk (int *attributes*)

easy/private.h, (c) 2013, Immanuel Albrecht; Dresden University of Technology, - Professur für die Psychologie des Lernen und Lehrens

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

easy/private.h, (c) 2013, Immanuel Albrecht; Dresden University of Technology, - Professur für die Psychologie des Lernen und Lehrens

Parameters

attributes	number of attributes of the hosting formal context

Returns

a new concept chunk object

Definition at line 298 of file fca.c.

References smyFormalConceptIntentChunk::attributes, CHUNKSIZE, smyFormalConceptIntentChunk::incidence, and smyFormalConceptIntentChunk::size.

```
{
    myFormalConceptIntentChunk *c;
    c = malloc(sizeof(myFormalConceptIntentChunk));
    c->attributes = attributes;
    c->size = 0;

#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wsign-conversion"

    c->incidence = calloc(attributes * CHUNKSIZE, sizeof(IncidenceCell));

#pragma GCC diagnostic pop

    return c;
}
```

4.8.1.11 void writeConceptsToFile (FormalContext *ctx*, FormalConceptIntentBulkList *root*, const char * *filename*)

write a list of concept intents into a .cxt file

Parameters

ctx	formal context (or 0, is used for attribute names)
root	the first node of the formal concept intent bulk
filename	output file name (.cxt)

Definition at line 716 of file fca.c.

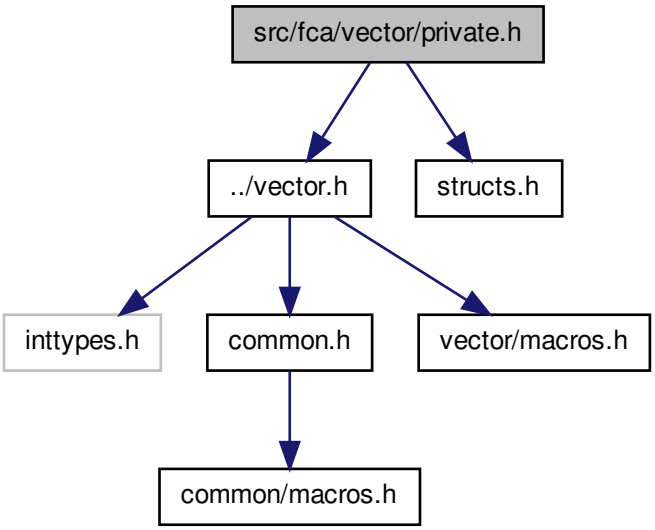
References smyFormalContext::attributeNames, smyFormalContext::attributes, sFormalConceptIntentBulkNode::attributes, sFormalConceptIntentBulkNode::chunks, countConceptsInBulk, gIm, sFormalConceptIntentBulkNode::next, RETURN_IF_ZERO, smyFormalConceptIntentChunk::size, sFormalConceptIntentBulkNode::size, and WARN_IF_UNEQUAL_DO.

```
{
    RETURN_IF_ZERO(root);

    myFormalContext* c;

    if (ctx != 0)
    {
        c = (myFormalContext*) ctx;

        WARN_IF_UNEQUAL_DO(c->attributes, root->attributes, c = 0);
    }
```

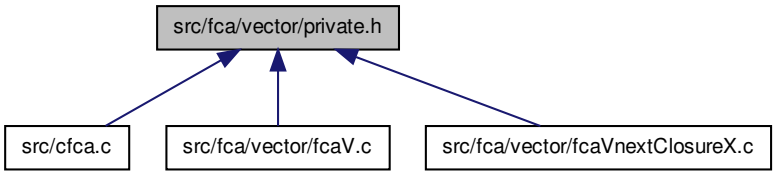
```
    else
    {
        c = 0;
    }

    RETURN_IF_ZERO(filename);

    FILE* file;
    file = fopen(filename, "w");

    RETURN_IF_ZERO(file);

    int objects;
    objects = countConceptsInBulk(root);

    fprintf(file, "B\n\n%d\n%d\n\n", objects, root->attributes);

    for (int var = 0; var < objects; ++var)
    {
        fprintf(file, "C%8d\n", (var + 1));
    }

    if (c != 0)
    {
        for (int var = 0; var < c->attributes; ++var)
        {
            fputs(c->attributeNames[var], file);
            fputs("\n", file);
        }
    }
    else
    {
        for (int var = 0; var < root->attributes; ++var)
        {
            fprintf(file, "m%8d\n", (var + 1));
        }
    }

    for (; root != 0; root = root->next)
    {
        for (int chunk = 0; chunk < root->size; ++chunk)
        {
            for (int g = 0; g < root->chunks[chunk]->size; ++g)
            {
                for (int m = 0; m < root->attributes; ++m)
                {
                    if ( gIm(g, root->chunks[chunk], m))
                        fputs("X", file);
                    else
                        fputs(".", file);
                }
                fputs("\n", file);
            }
        }
    }

    fclose(file);

}
```

4.9 src/fca/vector/private.h File Reference

`#include "../vector.h" #include "structs.h"` Include dependency graph for private.h:



This graph shows which files directly or indirectly include this file:



Functions

- void closeIntentV (FormalContextV restrict ctx, const IncidenceVector restrict input, IncidenceVector restrict output)

vector/private.h, (c) 2013, Immanuel Albrecht; Dresden University of Technology, - Professur für die Psychologie des Lernen und Lehrens

- int intentCmpV (size_t attributes, const IncidenceVector minus, const IncidenceVector plus)

  compare two intent vectors
- myFormalConceptIntentChunkV * newConceptChunkV (size_t attributes)

  create a new formal concept chunk
- void deleteConceptChunkV (myFormalConceptIntentChunkV **c)

  deletes a concept chunk object and sets its pointer to zero
- FormalConceptIntentBulkListV newConceptBulkV (size_t attributes)

  creates a new formal concept intent bulk list
- FormalConceptIntentBulkListV newConceptBulkFromContextV (FormalContextV ctx)

  creates a new formal concept intent chunk and fills it with the intents of all formal concepts in the concept lattice of ctx, using next closure algorithm
- void writeConceptsToFileV (FormalContextV ctx, FormalConceptIntentBulkListV root, const char *filename)

  write a list of concept intents into a .cxt file
- void deleteConceptBulkV (FormalConceptIntentBulkListV *rootNode)

  deletes the entire bulk list
- size_t countConceptsInBulkV (FormalConceptIntentBulkListV root)

  use this for bulks that are filled in order
- FormalConceptIntentBulkListV addConceptToBulkV (FormalConceptIntentBulkListV root, const IncidenceVector intent)

  copies the given intent to the bulk denoted by the root node.
- FormalConceptIntentBulkListV nextClosureVX1 (FormalContextV ctx, const IncidenceVector restrict start, const IncidenceVector restrict stop)

  creates a new formal concept intent chunk and fills it with the intents of all formal concepts in the concept lattice of ctx, using next closure algorithm, that are in a given lexicographic interval of the powerset
- FormalConceptIntentBulkListV nextClosureVX (FormalContextV ctx)

  creates a new formal concept intent chunk and fills it with the intents of all formal concepts in the concept lattice of ctx, using a parallel next closure algorithm with up to 8 threads

4.9.1 Function Documentation

4.9.1.1 FormalConceptIntentBulkListV addConceptToBulkV (FormalConceptIntentBulkListV *root*, const IncidenceVector *intent*)

copies the given intent to the bulk denoted by the root node.

Parameters

root	root node of the bulk
intent	read-only pointer to an array of IncidenceCell[root->attributes]

Returns

the node where the intent was added to the last chunk

Definition at line 830 of file fcaV.c.

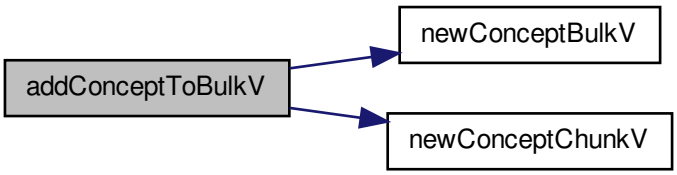
References sFormalConceptIntentBulkNodeV::attributes, BULKSIZEV, sFormalConceptIntentBulkNodeV::chunks, CHUNKSIZEV, newConceptBulkV(), newConceptChunkV(), sFormalConceptIntentBulkNodeV::next, RETURN_ZERO_IF_ZERO, ROW, smyFormalConceptIntentChunkV::size, sFormalConceptIntentBulkNodeV::size, and sFormalConceptIntentBulkNodeV::width.

Referenced by newConceptBulkFromContextV(), and nextClosureVX1().

```
{
    RETURN_ZERO_IF_ZERO(root);

    do
    {

        if (root->size == 0)
        {
            root->chunks[0] = newConceptChunkV(root->attributes);
            root->size = 1;
        }

        size_t last_index;
        last_index = root->size - 1;

        if (root->chunks[last_index]->size == CHUNKSIZEV)
        {
            if (root->size == BULKSIZEV)
            {
                if (root->next == 0)
                {
                    root->next = newConceptBulkV(root->attributes);
                }
                root = root->next;
                continue;
            }
            else
            {
                last_index = root->size++;
                root->chunks[last_index] = newConceptChunkV(root->attributes);
            }
        }

#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wsign-conversion"

        memcpy( ROW(root->chunks[last_index]->size, root->chunks[last_index]),
                intent, sizeof(uint64_t) * root->width);

#pragma GCC diagnostic pop

        root->chunks[last_index]->size++;

        break;

    } while (1);
```

```
    return root;
}
```

Here is the call graph for this function:



4.9.1.2 void closeIntentV (FormalContextV restrict *ctx,* const IncidenceVector restrict *input,* IncidenceVector restrict *output*)

vector/private.h, (c) 2013, Immanuel Albrecht; Dresden University of Technology, - Professur für die Psychologie des Lernen und Lehrens

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

vector/private.h, (c) 2013, Immanuel Albrecht; Dresden University of Technology, - Professur für die Psychologie des Lernen und Lehrens

add further attributes

Parameters

ctx	formal context
input	the intent set that is to be closed
output	the closure intent" wrt. ctx

some attribute is not present for this object -> next object

remove attributes that are not common among all objects that have the input attributes

Definition at line 411 of file fcaV.c.

References MASKVECTOR, and ROW.

Referenced by countContextConceptsV(), newConceptBulkFromContextV(), and nextClosureVX1().

```
{

    myFormalContextV* restrict I;
    I = (myFormalContextV*) ctx;
    for (size_t var = 0; var < I->width; ++var)
    {
        output[var] = ~0ULL;
    }

    MASKVECTOR(output, I->attributes);

    for (size_t g = 0; g < I->objects; ++g)
    {
        int good;
        good = 1;

        for (size_t i = 0; i < I->width; ++i)
        {
            if ((input[i]) & (~(ROW(g,I)[i])))
            {
                good = 0;
                break;
            }

        }

        if (good)
        {
            for (size_t i = 0; i < I->width; ++i)
            {
                output[i] &= ROW(g,I)[i];
            }
        }
    }
}
```

4.9.1.3 size_t countConceptsInBulkV (FormalConceptIntentBulkListV *root*)

use this for bulks that are filled in order

Parameters

root	

Returns

number of concepts in bulk

count the full chunks

and the last chunk

Definition at line 795 of file fcaV.c.

References sFormalConceptIntentBulkNodeV::chunks, CHUNKSIZEV, sFormalConceptIntentBulkNodeV::next, RETURN_ZERO_IF_ZERO, smyFormalConceptIntentChunkV::size, and sFormalConceptIntentBulkNodeV::size.

Referenced by main(), and writeConceptsToFileV().

```
{
    RETURN_ZERO_IF_ZERO(root);

    size_t count = 0;

    while (root != 0)
    {
        if (root->size > 0)
        {
            count += CHUNKSIZEV * (root->size - 1);
            count += root->chunks[root->size - 1]->size;
        }
        root = root->next;
    }

    return count;
}
```

4.9.1.4 void deleteConceptBulkV (FormalConceptIntentBulkListV * *rootNode*)

deletes the entire bulk list

Parameters

rootNode	pointer to the first node

Definition at line 763 of file fcaV.c.

References sFormalConceptIntentBulkNodeV::chunks, deleteConceptChunkV(), sFormalConceptIntentBulkNodeV::next, RETURN_IF_ZERO, and sFormalConceptIntentBulkNodeV::size.

Referenced by main().

```
{
    RETURN_IF_ZERO(rootNode);
    RETURN_IF_ZERO(*rootNode);

    FormalConceptIntentBulkListV l;
    l = *rootNode;
    *rootNode = 0;

    do
    {
        for (size_t var = 0; var < l->size; ++var)
        {
            deleteConceptChunkV(&(l->chunks[var]));
        }
```

```
        FormalConceptIntentBulkListV next;
        next = l->next;

        free(l->chunks);
        free(l);

        l = next;
    } while (l != 0);
}
```

Here is the call graph for this function:

4.9.1.5 void deleteConceptChunkV (myFormalConceptIntentChunkV ** c)

deletes a concept chunk object and sets its pointer to zero

Parameters

c	pointer to the concept chunk to be deleted

Definition at line 536 of file fcaV.c.

References RETURN_IF_ZERO.

Referenced by deleteConceptBulkV().

```
{
    RETURN_IF_ZERO(c);
    RETURN_IF_ZERO(*c);

    free((*c)->incidence);

    free(*c);
    *c = 0;
}
```

4.9.1.6 int intentCmpV (size_t *attributes,* const IncidenceVector *minus,* const IncidenceVector *plus*)

compare two intent vectors

Parameters

attributes	attribute count
minus	"left" operand
plus	"right" operand

Returns

-1 if minus is bigger, 1 if plus is bigger, 0 if minus and plus is the same

in this case, OFFSET(attributes) == OFFSET(attributes-1)

we only check the lower bits 0 through (attributes-1)

ELSE: attributes has 64 as factor, so we have done all necessary comparisons in the first loop.

Definition at line 466 of file fcaV.c.

References BITNBR, CRIMPVALUE, and OFFSET.

Referenced by nextClosureVX1().

```
{
    for (size_t var = 0; var < OFFSET(attributes); ++var)
    {
        if (minus[var] > plus[var])
            return -1;
        if (plus[var] > minus[var])
            return 1;
    }

    if (BITNBR(attributes))
    {
        uint64_t l, r;

        l = minus[OFFSET(attributes)] & CRIMPVALUE(attributes-1);
        r = plus[OFFSET(attributes)] & CRIMPVALUE(attributes-1);

        if (l > r)
            return -1;

        if (r > l)
            return 1;
    }
    return 0;
}
```

4.9.1.7 FormalConceptIntentBulkListV newConceptBulkFromContextV (FormalContextV *ctx*)

creates a new formal concept intent chunk and fills it with the intents of all formal concepts in the concept lattice of ctx, using next closure algorithm

Parameters

ctx	formal context

Returns

concept intents

calculate the bottom intent of the concept lattice, i.e. {}”

add the bottom element of the concept lattice (a concept lattice is never empty)

begin of nextClosure function iteration

we found the next intent

continue with Y for M

do the nextClosure

free up memory

Definition at line 575 of file fcaV.c.

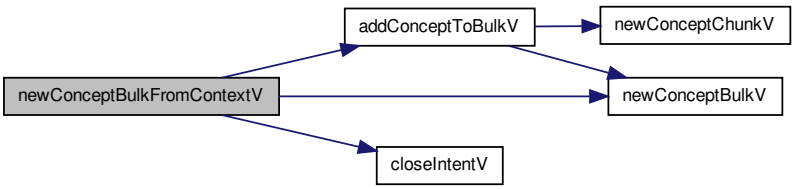
References addConceptToBulkV(), CLEARV, closeIntentV(), CRIMPVALUE, CROSSV, INCIDESV, newConceptBulkV(), OFFSET, and RETURN_ZERO_IF_ZERO.

Referenced by main().

```
{
    RETURN_ZERO_IF_ZERO(ctx);

    myFormalContextV * restrict c;
    c = (myFormalContextV*) ctx;

    IncidenceVector restrict M;
    IncidenceVector restrict Y;

#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wsign-conversion"

    Y = calloc(c->width, sizeof(uint64_t));
    M = malloc(c->width * sizeof(uint64_t));

#pragma GCC diagnostic pop

    closeIntentV(ctx, Y, M);

    FormalConceptIntentBulkListV root;
    FormalConceptIntentBulkListV last;

    root = newConceptBulkV(c->attributes);

    last = addConceptToBulkV(root, M);

    nextClosure:

    for (size_t i = c->attributes; i > 0;)
    {
        --i;

        if (!INCIDESV(M,i))
        {
            CROSSV(M, i);
            closeIntentV(ctx, M, Y);

            int good;
```

```
            good = 1;

            for (unsigned int j = 0; j < OFFSET(i); ++j)
            {
                if (Y[j] & (~(M[j])))
                {
                    good = 0;
                    break;
                }
            }
            if (good)
            {
                if (Y[OFFSET(i)] & (~M[OFFSET(i)]) & CRIMPVALUE(i))
                {
                    good = 0;
                }
            }

            if (good)
            {

                last = addConceptToBulkV(last, Y);

                IncidenceVector DELTA;
                DELTA = M;
                M = Y;
                Y = DELTA;
                goto nextClosure;
            }
        }

        CLEARV(M, i);
    }

    free(M);
    free(Y);

    return root;
}
```

Here is the call graph for this function:



4.9.1.8 FormalConceptIntentBulkListV newConceptBulkV (size_t *attributes*)

creates a new formal concept intent bulk list

Parameters

attributes	number of attributes of the concept intents

Returns

new formal concept intent bulk list's first node

Definition at line 554 of file fcaV.c.

References sFormalConceptIntentBulkNodeV::attributes, BULKSIZEV, sFormalConceptIntentBulkNodeV::chunks, sFormalConceptIntentBulkNodeV::next, sFormalConceptIntentBulkNodeV::size, WIDTH, and sFormalConceptIntentBulkNodeV::width.

Referenced by addConceptToBulkV(), newConceptBulkFromContextV(), and nextClosureVX1().

```
{
    FormalConceptIntentBulkListV l;
    l = malloc(sizeof(struct sFormalConceptIntentBulkNodeV));

    l->attributes = attributes;
    l->width = WIDTH(attributes);
    l->size = 0;
    l->chunks = calloc(BULKSIZEV, sizeof(myFormalConceptIntentChunkV*));
    l->next = 0;
    return l;
}
```

4.9.1.9 myFormalConceptIntentChunkV* newConceptChunkV (size_t *attributes*)

create a new formal concept chunk

Parameters

attributes	number of attributes of the hosting formal context

Returns

a new concept chunk object

Definition at line 509 of file fcaV.c.

References smyFormalConceptIntentChunkV::attributes, CHUNKSIZEV, smyFormalConceptIntentChunkV::incidence, smyFormalConceptIntentChunkV::size, WIDTH, and smyFormalConceptIntentChunkV::width.

Referenced by addConceptToBulkV().

```
{

    myFormalConceptIntentChunkV *c;

    c = malloc(sizeof(myFormalConceptIntentChunkV));

    c->attributes = attributes;
    c->width = WIDTH(attributes);
    c->size = 0;

#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wsign-conversion"

    c->incidence = calloc(c->width * CHUNKSIZEV, sizeof(uint64_t));

#pragma GCC diagnostic pop

    return c;
}
```

4.9.1.10 FormalConceptIntentBulkListV nextClosureVX (FormalContextV *ctx*)

creates a new formal concept intent chunk and fills it with the intents of all formal concepts in the concept lattice of ctx, using a parallel next closure algorithm with up to 8 threads

Parameters

ctx	formal context

Returns

concept intents

Definition at line 199 of file fcaVnextClosureX.c.

References smyFormalContextV::attributes, callNextClosureVX1(), CROSSV, snextClosureVX1Params::ctx, sFormalConceptIntentBulkNodeV::next, nextClosureVX1(), RETURN_ZERO_IF_ZERO, snextClosureVX1Params::rVal, snextClosureVX1Params::start, snextClosureVX1Params::stop, and smyFormalContextV::width.

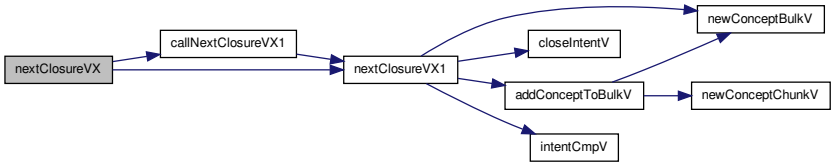
Referenced by main().

```
{
    RETURN_ZERO_IF_ZERO(ctx);

    myFormalContextV *c;
    c = (myFormalContextV*) ctx;

    size_t N;

    N = 1;

    if (c->attributes >= 3)
        N = 8;
    else if (c->attributes >= 2)
```

```
        N = 4;
    else if (c->attributes >= 1)
        N = 2;

    if (N < 2)
        return nextClosureVX1(ctx, 0, 0);

    IncidenceVector bounds;
    bounds = calloc(c->width * (N - 1), sizeof(uint64_t));

    if (N == 2)
    {
        CROSSV(bounds, 0);
    }
    else if (N == 4)
    {
        CROSSV(bounds, 1); //01

        CROSSV(bounds + c->width, 0); //10

        CROSSV(bounds + c->width * 2, 1); //11
        CROSSV(bounds + c->width * 2, 0);
    }
    else if (N == 8)
    {
        CROSSV(bounds, 2); // 001

        CROSSV(bounds + c->width, 1); //010

        CROSSV(bounds + c->width * 2, 2); //011
        CROSSV(bounds + c->width * 2, 1);

        CROSSV(bounds + c->width * 3, 0); //100

        CROSSV(bounds + c->width * 4, 0); //101
        CROSSV(bounds + c->width * 4, 2);

        CROSSV(bounds + c->width * 5, 1); //110
        CROSSV(bounds + c->width * 5, 0);

        CROSSV(bounds + c->width * 6, 0); //111
        CROSSV(bounds + c->width * 6, 1);
        CROSSV(bounds + c->width * 6, 2);
    }

//  for (int i = 0; i < N - 1; ++i)
//  {
//      printf("BOUND %16llx\n", *(bounds + i *
       c->width)&CRIMPVALUE(c->attributes-1));
//      if (i > 0)
//          printf("CMP %d\n",
//                  intentCmpV(c->attributes, bounds + (i - 1) * c->width,
//                          bounds + i * c->width));
//  }


    nextClosureVX1Params chunks;
    chunks = malloc(N * sizeof(struct snextClosureVX1Params));

    pthread_t *threads;
    threads = malloc(N * sizeof(pthread_t));
```

```
    for (size_t i = 0; i < N; ++i)
    {
        chunks[i].ctx = ctx;
        if (i > 0)
            chunks[i].start = (bounds + c->width * (i - 1));
        else
            chunks[i].start = 0;

        if (i < N - 1)
            chunks[i].stop = (bounds + c->width * (i));
        else
            chunks[i].stop = 0;
    }

    for (size_t i = 0; i < N; ++i)
    {
        pthread_create(&threads[i], 0, callNextClosureVX1, (void*) &chunks[i]);
    }

    for (size_t i = 0; i < N; ++i)
    {
        pthread_join(threads[i], 0);
    }

//  for (size_t i = 0; i < N; ++i)
//  {
//      printf("%zu thread: counts %zu\n", i,
//              countConceptsInBulkV(chunks[i].rVal));
//  }

    FormalConceptIntentBulkListV root;
    FormalConceptIntentBulkListV last;

    root = chunks[0].rVal;
    last = root;

    for (size_t var = 1; var < N; ++var)
    {

        while (last->next)
            last = last->next;

        last->next = chunks[var].rVal;
    }

    free(bounds);
    free(chunks);

    return root;
}
```

Here is the call graph for this function:



4.9.1.11 FormalConceptIntentBulkListV nextClosureVX1 (FormalContextV *ctx,* const IncidenceVector restrict *start,* const IncidenceVector restrict *stop*)

creates a new formal concept intent chunk and fills it with the intents of all formal concepts in the concept lattice of ctx, using next closure algorithm, that are in a given lexicographic interval of the powerset

Parameters

ctx	formal context
start	first attribute vector (not included if it is a concept intent) if this is zero, start with M={}, in this case, we add the bottom concept in case of -M"=={}
stop	last attribute vector, or zero to continue until the end

Returns

concept intents between (start, stop]

calculate the bottom intent of the concept lattice, i.e. {}"

add the bottom element of the concept lattice (a concept lattice is never empty)

begin of nextClosure function iteration

check whether we are still in range

the (pseudo)intent Y is bigger than stop

we found the next intent

continue with Y for M

do the nextClosure

free up memory

Definition at line 47 of file fcaVnextClosureX.c.

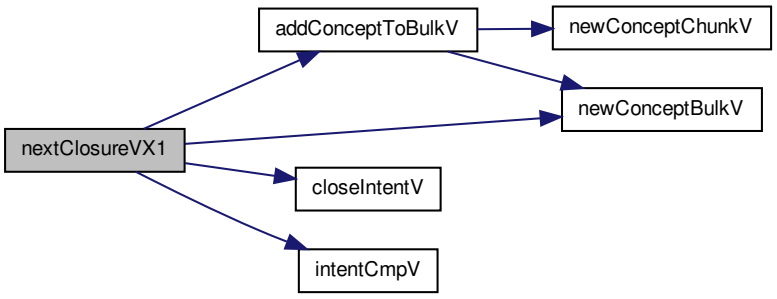
References addConceptToBulkV(), CLEARV, closeIntentV(), CRIMPVALUE, CROSSV, INCIDESV, intentCmpV(), newConceptBulkV(), OFFSET, and RETURN_ZERO_IF_ZERO.

Referenced by callNextClosureVX1(), and nextClosureVX().

```
{
    RETURN_ZERO_IF_ZERO(ctx);

    myFormalContextV * restrict c;
    c = (myFormalContextV*) ctx;

    IncidenceVector restrict M;
    IncidenceVector restrict Y;

    Y = calloc(c->width, sizeof(uint64_t));
    M = malloc(c->width * sizeof(uint64_t));

    FormalConceptIntentBulkListV root;
    FormalConceptIntentBulkListV last;

    root = newConceptBulkV(c->attributes);

    if (start)
    {
        memcpy(M, start, c->width * sizeof(uint64_t));

        last = root;

    }
    else
    {

        closeIntentV(ctx, Y, M);

        last = addConceptToBulkV(root, M);
    }

    nextClosure:

    for (size_t i = c->attributes; i > 0;)
    {
        --i;

        if (!INCIDESV(M,i))
        {
            CROSSV(M, i);
            closeIntentV(ctx, M, Y);

            int good;
            good = 1;

            for (unsigned int j = 0; j < OFFSET(i); ++j)
            {
                if (Y[j] & (~(M[j])))
                {
                    good = 0;
                    break;
                }
            }
            if (good)
            {
                if (Y[OFFSET(i)] & (~M[OFFSET(i)]) & CRIMPVALUE(i))
                {
                    good = 0;
                }
```

```
            }

            if (good)
            {
                if (stop)
                {
                    if (intentCmpV(c->attributes, Y, stop) < 0)
                    {
                        goto gracefulReturn;
                    }
                }

                last = addConceptToBulkV(last, Y);

                IncidenceVector DELTA;
                DELTA = M;
                M = Y;
                Y = DELTA;
                goto nextClosure;
            }
        }

        CLEARV(M, i);
    }

    gracefulReturn:
    free(M);
    free(Y);

    return root;
}
```

Here is the call graph for this function:



4.9.1.12 void writeConceptsToFileV (FormalContextV *ctx*, FormalConceptIntentBulkListV *root*, const char * *filename*)

write a list of concept intents into a .cxt file

Parameters

ctx	formal context (or 0, is used for attribute names)
root	the first node of the formal concept intent bulk
filename	output file name (.cxt)

Definition at line 686 of file fcaV.c.

References smyFormalContextV::attributeNames, smyFormalContextV::attributes, sFormalConceptIntentBulkNodeV::attributes, sFormalConceptIntentBulkNodeV::chunks, countConceptsInBulkV(), INCIDESV, sFormalConceptIntentBulkNodeV::next, RETURN_IF_ZERO, ROW, smyFormalConceptIntentChunkV::size, sFormalConceptIntentBulkNodeV::size, and WARN_IF_UNEQUAL_DO.

Referenced by main().

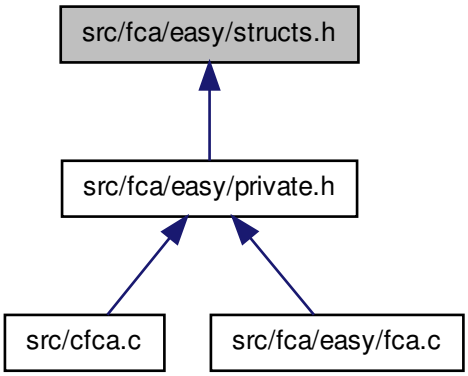
```
{
    RETURN_IF_ZERO(root);

    myFormalContextV* c;

    if (ctx != 0)
    {
        c = (myFormalContextV*) ctx;

        WARN_IF_UNEQUAL_DO(c->attributes, root->attributes, c = 0);
    }
    else
    {
        c = 0;
    }

    RETURN_IF_ZERO(filename);

    FILE* file;
    file = fopen(filename, "w");

    RETURN_IF_ZERO(file);

    size_t objects;
    objects = countConceptsInBulkV(root);

    fprintf(file, "B\n\n%zu\n%zu\n\n", objects, root->attributes);

    for (size_t var = 0; var < objects; ++var)
    {
        fprintf(file, "C%8zu\n", (var + 1));
    }

    if (c != 0)
    {
        for (size_t var = 0; var < c->attributes; ++var)
        {
            fputs(c->attributeNames[var], file);
            fputs("\n", file);
        }
    }
    else
    {
        for (size_t var = 0; var < root->attributes; ++var)
        {
```

```
            fprintf(file, "m%8zu\n", (var + 1));
        }
    }

    for (; root != 0; root = root->next)
    {
        for (size_t chunk = 0; chunk < root->size; ++chunk)
        {
            for (size_t g = 0; g < root->chunks[chunk]->size; ++g)
            {
                for (size_t m = 0; m < root->attributes; ++m)
                {
                    if (INCIDESV(ROW(g, root->chunks[chunk]), m))
                        fputs("X", file);
                    else
                        fputs(".", file);
                }
                fputs("\n", file);
            }
        }
    }

    fclose(file);
}
```

Here is the call graph for this function:

4.10 src/fca/easy/structs.h File Reference

This graph shows which files directly or indirectly include this file:



Data Structures

- struct smyFormalContext

  each formal context has a finite number of objects and attributes, which may have names (though we do not require them to be unique or given), and an incidence relation which is represented by an objects×attributes-IncidenceCell matrix
- struct smyFormalConceptIntentChunk

  A chunk of at most CHUNKSIZE formal concept intents.
- struct sFormalConceptIntentBulkNode

  a node of a single linked list of concept chunks.

Defines

- #define CHUNKSIZE (64)

  easy/structs.h, (c) 2013, Immanuel Albrecht; Dresden University of Technology, - Professur für die Psychologie des Lernen und Lehrens
- #define BULKSIZE (1024)

  size of chunks per bulk
- #define INPUTBUFFERSIZE (1024)

  maximal (initial) size of a line (getline will resize buffers if necessary) (including delimiter)

Typedefs

- typedef struct smyFormalContext myFormalContext

  each formal context has a finite number of objects and attributes, which may have names (though we do not require them to be unique or given), and an incidence relation which is represented by an objects×attributes-IncidenceCell matrix

- typedef struct smyFormalConceptIntentChunk myFormalConceptIntentChunk

  A chunk of at most CHUNKSIZE formal concept intents.

- typedef struct sFormalConceptIntentBulkNode * FormalConceptIntentBulkList

  a node of a single linked list of concept chunks.

4.10.1 Define Documentation

4.10.1.1 #define BULKSIZE (1024)

size of chunks per bulk

Definition at line 35 of file structs.h.

Referenced by addConceptToBulk(), and newConceptBulk().

4.10.1.2 #define CHUNKSIZE (64)

easy/structs.h, (c) 2013, Immanuel Albrecht; Dresden University of Technology, - Professur für die Psychologie des Lernen und Lehrens

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

size of a chunk of concepts

Definition at line 27 of file structs.h.

Referenced by addConceptToBulk(), countConceptsInBulk(), and newConceptChunk().

4.10.1.3 #define INPUTBUFFERSIZE (1024)

maximal (initial) size of a line (getline will resize buffers if necessary) (including delimiter)

Definition at line 42 of file structs.h.

Referenced by newFormalContextFromFile(), and newFormalContextFromFileV().

4.10.2 Typedef Documentation

4.10.2.1 typedef struct sFormalConceptIntentBulkNode* FormalConceptIntentBulkList

a node of a single linked list of concept chunks.

bulk nodes are filled chunk wise, but a bulk node may have non-empty successor nodes even if it is not entire full.

4.10.2.2 typedef struct smyFormalConceptIntentChunk myFormalConceptIntentChunk

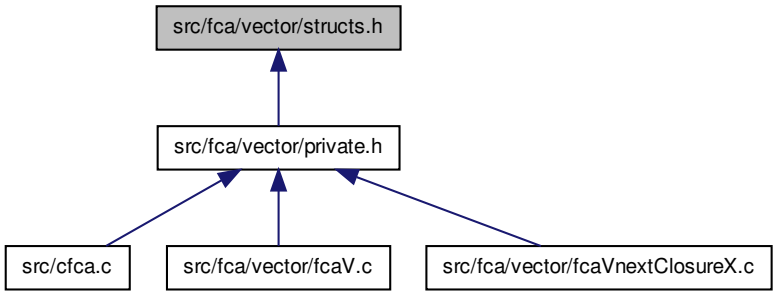
A chunk of at most CHUNKSIZE formal concept intents.

4.10.2.3 typedef struct smyFormalContext myFormalContext

each formal context has a finite number of objects and attributes, which may have names (though we do not require them to be unique or given), and an incidence relation which is represented by an objects×attributes-IncidenceCell matrix

4.11 src/fca/vector/structs.h File Reference

This graph shows which files directly or indirectly include this file:



Data Structures

- struct smyFormalContextV

each formal context has a finite number of objects and attributes, which may have names (though we do not require them to be unique or given), and an incidence relation which is represented by an objects×attributes-IncidenceCell matrix

- struct smyFormalConceptIntentChunkV

  A chunk of at most CHUNKSIZEV formal concept intent vectors.

- struct sFormalConceptIntentBulkNodeV

  a node of a single linked list of concept chunks.

Defines

- #define CHUNKSIZEV (1024)

  vector/structs.h, (c) 2013, Immanuel Albrecht; Dresden University of Technology, - Professur für die Psychologie des Lernen und Lehrens

- #define BULKSIZEV (1024)

  size of chunks per bulk (vector version)

- #define INPUTBUFFERSIZE (1024)

  maximal (initial) size of a line (getline will resize buffers if necessary) (including delimiter)

Typedefs

- typedef struct smyFormalContextV myFormalContextV

  each formal context has a finite number of objects and attributes, which may have names (though we do not require them to be unique or given), and an incidence relation which is represented by an objects×attributes-IncidenceCell matrix

- typedef struct smyFormalConceptIntentChunkV myFormalConceptIntentChunkV

  A chunk of at most CHUNKSIZEV formal concept intent vectors.

- typedef struct sFormalConceptIntentBulkNodeV * FormalConceptIntentBulkListV

  a node of a single linked list of concept chunks.

4.11.1 Define Documentation

4.11.1.1 #define BULKSIZEV (1024)

size of chunks per bulk (vector version)

Definition at line 32 of file structs.h.

Referenced by addConceptToBulkV(), and newConceptBulkV().

4.11.1.2 #define CHUNKSIZEV (1024)

vector/structs.h, (c) 2013, Immanuel Albrecht; Dresden University of Technology, - Professur für die Psychologie des Lernen und Lehrens

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

size of a chunk of vector concepts

Definition at line 26 of file structs.h.

Referenced by addConceptToBulkV(), countConceptsInBulkV(), and newConceptChunkV().

4.11.1.3 #define INPUTBUFFERSIZE (1024)

maximal (initial) size of a line (getline will resize buffers if necessary) (including delimiter)

Definition at line 40 of file structs.h.

4.11.2 Typedef Documentation

4.11.2.1 typedef struct sFormalConceptIntentBulkNodeV* FormalConceptIntentBulkListV

a node of a single linked list of concept chunks.

bulk nodes are filled chunk wise, but a bulk node may have non-empty successor nodes even if it is not entire full.

4.11.2.2 typedef struct smyFormalConceptIntentChunkV myFormalConceptIntentChunkV

A chunk of at most CHUNKSIZEV formal concept intent vectors.

4.11.2.3 typedef struct smyFormalContextV myFormalContextV

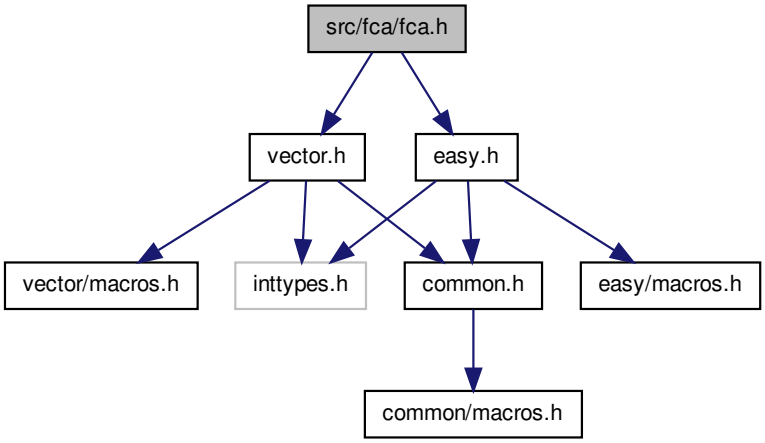
each formal context has a finite number of objects and attributes, which may have names (though we do not require them to be unique or given), and an incidence relation which is represented by an objects×attributes-IncidenceCell matrix

for the vector implementation, we have the variable width which codes the width of each object's IncidenceVector

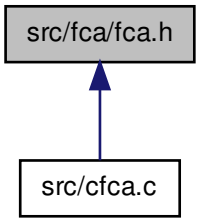
4.12 src/fca/fca.h File Reference

fca.h, (c) 2013, Immanuel Albrecht; Dresden University of Technology, Professur für die Psychologie des Lernen und Lehrens

`#include "easy.h"` `#include "vector.h"` Include dependency graph for fca.h:



This graph shows which files directly or indirectly include this file:



4.12.1 Detailed Description

fca.h, (c) 2013, Immanuel Albrecht; Dresden University of Technology, Professur für die Psychologie des Lernen und Lehrens This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

this file contains the public interface to the formal context analysis code

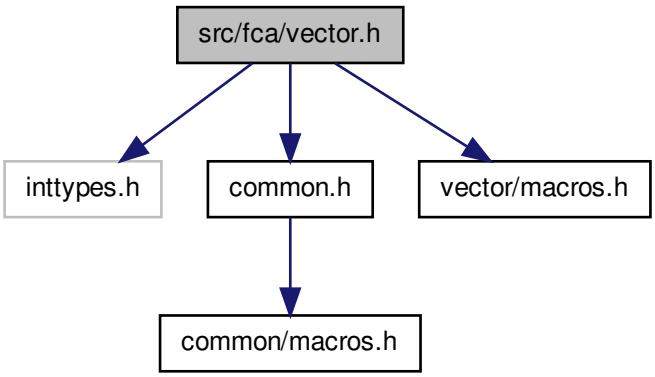
this file provides both easy and vector routines

Definition in file fca.h.

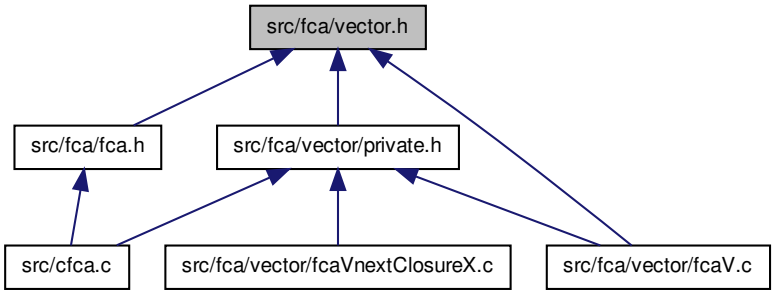
4.13 src/fca/vector.h File Reference

vector.h, (c) 2013, Immanuel Albrecht; Dresden University of Technology, Professur für die Psychologie des Lernen und Lehrens

`#include <inttypes.h>` `#include "common.h"` `#include "vector/macros.h"` Include dependency graph for vector.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct sFormalIntentV

  intent structure of a formal concept

Typedefs

- typedef uint64_t * IncidenceVector

  type of compressed attribute vectors
- typedef struct sFormalContextV * FormalContextV
- typedef struct sFormalIntentV FormalIntentV

  intent structure of a formal concept

Functions

- FormalContextV newFormalContextV (unsigned int objects, unsigned int attributes)

  create a new formal context
- void deleteFormalContextV (FormalContextV *ctx)

  *deletes the formal context *ctx, and sets the pointer to zero*
- void writeFormalContextV (FormalContextV ctx, const char *filename)

  save the context ctx at the given file location
- FormalContextV newFormalContextFromFileV (const char *filename)

  create a new formal context object from a .cxt file

4.13.1 Detailed Description

vector.h, (c) 2013, Immanuel Albrecht; Dresden University of Technology, Professur für die Psychologie des Lernen und Lehrens This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

This header file provides interfaces with the faster IncidenceVector implementations

Definition in file vector.h.

4.13.2 Typedef Documentation

4.13.2.1 typedef struct sFormalContextV* FormalContextV

Definition at line 44 of file vector.h.

4.13.2.2 typedef struct sFormalIntentV FormalIntentV

intent structure of a formal concept

4.13.2.3 typedef uint64_t* IncidenceVector

type of compressed attribute vectors

Definition at line 36 of file vector.h.

4.13.3 Function Documentation

4.13.3.1 void deleteFormalContextV (FormalContextV * *ctx*)

deletes the formal context *ctx, and sets the pointer to zero

Parameters

ctx	pointer to the formal context object to be deleted

Definition at line 84 of file fcaV.c.

References smyFormalContextV::attributeNames, smyFormalContextV::attributes, smyFormalContextV::incidence, smyFormalContextV::objectNames, smyFormal-

ContextV::objects, and RETURN_IF_ZERO.

Referenced by main().

```
{
    RETURN_IF_ZERO(ctx);
    RETURN_IF_ZERO(*ctx);

    myFormalContextV *c;

    c = (myFormalContextV*) *ctx;

    *ctx = 0;

    for (unsigned int var = 0; var < c->attributes; ++var)
    {
        free(c->attributeNames[var]);
    }

    for (unsigned int var = 0; var < c->objects; ++var)
    {
        free(c->objectNames[var]);
    }

    free(c->objectNames);
    free(c->attributeNames);
    free(c->incidence);
    free(c);
}
```

4.13.3.2 FormalContextV newFormalContextFromFileV (const char * *filename*)

create a new formal context object from a .cxt file

Parameters

filename	

Returns

the formal context that has been read from the given file

this should never happen, right?

we read all data

free memory and return

Definition at line 118 of file fcaV.c.

References smyFormalContextV::attributeNames, CROSSV, INPUTBUFFERSIZE, MIN, newFormalContextV(), smyFormalContextV::objectNames, RETURN_ZERO_IF_ZERO, and ROW.

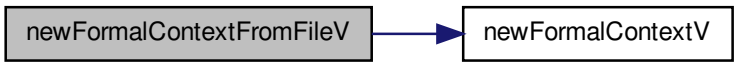
Referenced by main().

```
{
```

```
    char *line;
    size_t len;

    len = (INPUTBUFFERSIZE);
    line = malloc(sizeof(char) * len);

    FILE *file;

    if (strcmp(filename, "-") == 0)
    {
        file = stdin;
    }
    else
    {
        file = fopen(filename, "r");
        RETURN_ZERO_IF_ZERO(file);
    }

    ssize_t read;

    unsigned int line_nbr;
    line_nbr = 0;

    unsigned int objects;
    unsigned int attributes;

    attributes = 0;
    objects = 0;

    myFormalContextV *ctx;
    ctx = 0;

    while ((read = getline(&line, &len, file)) != -1)
    {

        if (read == 0)
            break;
        line[read - 1] = 0;

        if (line_nbr == 0)
        {
            if (strcmp(line, "B"))
            {
                fprintf(stderr, "File '%s' is not a .cxt file\n", filename);
                goto grace;
            }
        }
        else if (line_nbr == 1)
        {
            //empty line
        }
        else if (line_nbr == 2)
        {
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wsign-conversion"
            objects = atoi(line);
#pragma GCC diagnostic pop
        }
        else if (line_nbr == 3)
        {
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wsign-conversion"
```

```
            attributes = atoi(line);
            ctx = (myFormalContextV *) newFormalContextV(objects, attributes);
#pragma GCC diagnostic pop
        }
        else if (line_nbr == 4)
        {
            //empty line
        }
        else if (line_nbr < objects + 5)
        {

            unsigned int i;
            i = line_nbr - 5;

            free(ctx->objectNames[i]);
            ctx->objectNames[i] = strdup(line);

        }
        else if (line_nbr < objects + attributes + 5)
        {
            unsigned int i;
            i = line_nbr - 5 - objects;

            free(ctx->attributeNames[i]);
            ctx->attributeNames[i] = strdup(line);
        }
        else if (line_nbr < objects * 2 + attributes + 5)
        {
            unsigned int i;
            i = line_nbr - 5 - objects - attributes;

            unsigned int width;
            width = MIN((unsigned int)strlen(line),attributes);

            for (unsigned int var = 0; var < width; ++var)
            {
                if ((line[var] == 'x') || (line[var] == 'X')
                        || (line[var] == '1'))
                {
                    CROSSV(ROW(i, ctx), var);
                }
            }

        }
        else
        {
            break;
        }

        line_nbr++;
    }

    grace: if (file != stdin)
    {
        fclose(file);
    }

    free(line);

    return (FormalContextV) ctx;
}
```

Here is the call graph for this function:



4.13.3.3 FormalContextV newFormalContextV (unsigned int *objects*, unsigned int *attributes*)

create a new formal context

Parameters

objects	object count
attributes	attribute count

Returns

a new FormalContext object

Definition at line 42 of file fcaV.c.

References smyFormalContextV::attributeNames, smyFormalContextV::attributes, smyFormalContextV::incidence, smyFormalContextV::objectNames, smyFormalContextV::objects, WIDTH, and smyFormalContextV::width.

Referenced by newFormalContextFromFileV().

```
{
    myFormalContextV *ctx = malloc(sizeof(myFormalContextV));

    ctx->attributes = attributes;
    ctx->objects = objects;
    ctx->width = WIDTH(attributes);

#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wsign-conversion"

    ctx->attributeNames = calloc(attributes, sizeof(char*));
    ctx->objectNames = calloc(objects, sizeof(char*));

#pragma GCC diagnostic pop

    for (unsigned int var = 0; var < attributes; ++var)
    {
        ctx->attributeNames[var] = calloc(1, sizeof(char));
    }
```

```
    for (unsigned int var = 0; var < objects; ++var)
    {
        ctx->objectNames[var] = calloc(1, sizeof(char));
    }

#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wsign-conversion"

    ctx->incidence = calloc(objects * ctx->width, sizeof(uint64_t));

#pragma GCC diagnostic pop

    return (FormalContextV) ctx;
}
```

4.13.3.4 void writeFormalContextV (FormalContextV *ctx,* const char * *filename*)

save the context ctx at the given file location

Parameters

ctx	
filename	

Definition at line 260 of file fcaV.c.

References smyFormalContextV::attributeNames, smyFormalContextV::attributes, INCIDESV, smyFormalContextV::objectNames, smyFormalContextV::objects, RETURN_IF_ZERO, and ROW.

Referenced by main().

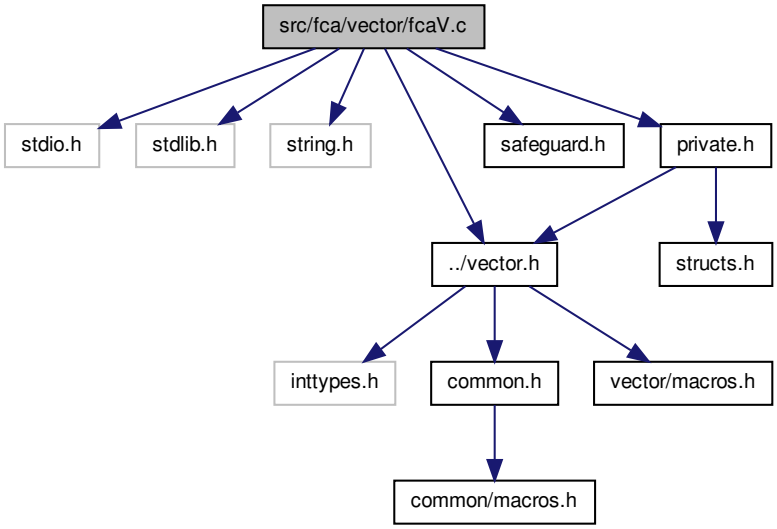
```
{
    RETURN_IF_ZERO(ctx);
    RETURN_IF_ZERO(filename);

    FILE* file;
    file = fopen(filename, "w");

    RETURN_IF_ZERO(file);

    myFormalContextV *c;
    c = (myFormalContextV*) ctx;

    fprintf(file, "B\n\n%zu\n%zu\n\n", c->objects, c->attributes);

    for (unsigned int var = 0; var < c->objects; ++var)
    {
        fputs(c->objectNames[var], file);
        fputs("\n", file);
    }

    for (unsigned int var = 0; var < c->attributes; ++var)
    {
        fputs(c->attributeNames[var], file);
        fputs("\n", file);
```

```
    }

    for (unsigned int g = 0; g < c->objects; ++g)
    {
        for (unsigned int m = 0; m < c->attributes; ++m)
        {
            if ( INCIDESV(ROW(g, c), m))
                fputs("X", file);
            else
                fputs(".", file);
        }
        fputs("\n", file);
    }

    fclose(file);
}
```

4.14 src/fca/vector/fcaV.c File Reference

fcaV.c, (c) 2013, Immanuel Albrecht; Dresden University of Technology, Professur für die Psychologie des Lernen und Lehrens

`#include <stdio.h> #include <stdlib.h> #include <string.h> #include "../vector.h" #include "safeguard.h" #include "private.h"` Include dependency graph for fcaV.c:



Functions

- FormalContextV newFormalContextV (unsigned int objects, unsigned int attributes)

  create a new formal context

- void deleteFormalContextV (FormalContextV ∗ctx)

  deletes the formal context ∗ctx, and sets the pointer to zero

- FormalContextV newFormalContextFromFileV (const char ∗filename)

  create a new formal context object from a .cxt file

- void writeFormalContextV (FormalContextV ctx, const char ∗filename)

  save the context ctx at the given file location

- int countContextConceptsV (FormalContextV ctx)

  counts the concepts in the concept lattice of ctx, using next closure algorithm

- void closeIntentV (FormalContextV restrict ctx, const IncidenceVector restrict input, IncidenceVector restrict output)

  close an attribute set, i.e.

- int intentCmpV (size_t attributes, const IncidenceVector minus, const IncidenceVector plus)

  compare two intent vectors

- myFormalConceptIntentChunkV ∗ newConceptChunkV (size_t attributes)

  create a new formal concept chunk

- void deleteConceptChunkV (myFormalConceptIntentChunkV ∗∗c)

  deletes a concept chunk object and sets its pointer to zero

- FormalConceptIntentBulkListV newConceptBulkV (size_t attributes)

  creates a new formal concept intent bulk list

- FormalConceptIntentBulkListV newConceptBulkFromContextV (FormalContextV ctx)

  creates a new formal concept intent chunk and fills it with the intents of all formal concepts in the concept lattice of ctx, using next closure algorithm

- void writeConceptsToFileV (FormalContextV ctx, FormalConceptIntentBulkListV root, const char ∗filename)

  write a list of concept intents into a .cxt file

- void deleteConceptBulkV (FormalConceptIntentBulkListV ∗rootNode)

  deletes the entire bulk list

- size_t countConceptsInBulkV (FormalConceptIntentBulkListV root)

  use this for bulks that are filled in order

- FormalConceptIntentBulkListV addConceptToBulkV (FormalConceptIntentBulkListV root, const IncidenceVector intent)

  copies the given intent to the bulk denoted by the root node.

4.14.1 Detailed Description

fcaV.c, (c) 2013, Immanuel Albrecht; Dresden University of Technology, Professur für die Psychologie des Lernen und Lehrens This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>. this file contains general formal context related operations and routines with IncidenceVector implementation

Definition in file fcaV.c.

4.14.2 Function Documentation

4.14.2.1 FormalConceptIntentBulkListV addConceptToBulkV (FormalConceptIntentBulkListV *root,* const IncidenceVector *intent*)

copies the given intent to the bulk denoted by the root node.

Parameters

root	root node of the bulk
intent	read-only pointer to an array of IncidenceCell[root->attributes]

Returns

the node where the intent was added to the last chunk

Definition at line 830 of file fcaV.c.

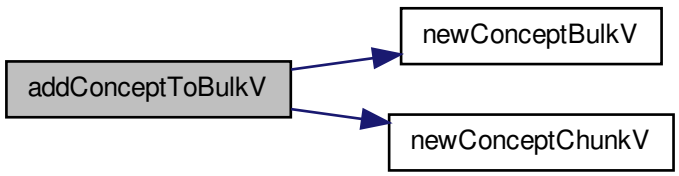
References sFormalConceptIntentBulkNodeV::attributes, BULKSIZEV, sFormalConceptIntentBulkNodeV::chunks, CHUNKSIZEV, newConceptBulkV(), newConceptChunkV(), sFormalConceptIntentBulkNodeV::next, RETURN_ZERO_IF_ZERO, ROW, smyFormalConceptIntentChunkV::size, sFormalConceptIntentBulkNodeV::size, and sFormalConceptIntentBulkNodeV::width.

Referenced by newConceptBulkFromContextV(), and nextClosureVX1().

```
{
    RETURN_ZERO_IF_ZERO(root);

    do
    {

        if (root->size == 0)
        {
```

```
            root->chunks[0] = newConceptChunkV(root->attributes);
            root->size = 1;
        }

        size_t last_index;
        last_index = root->size - 1;

        if (root->chunks[last_index]->size == CHUNKSIZEV)
        {
            if (root->size == BULKSIZEV)
            {
                if (root->next == 0)
                {
                    root->next = newConceptBulkV(root->attributes);
                }
                root = root->next;
                continue;
            }
            else
            {
                last_index = root->size++;
                root->chunks[last_index] = newConceptChunkV(root->attributes);
            }
        }

#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wsign-conversion"

        memcpy( ROW(root->chunks[last_index]->size, root->chunks[last_index]),
                intent, sizeof(uint64_t) * root->width);

#pragma GCC diagnostic pop

        root->chunks[last_index]->size++;

        break;

    } while (1);

    return root;
}
```

Here is the call graph for this function:



4.14.2.2 void closeIntentV (FormalContextV restrict *ctx,* const IncidenceVector restrict *input,* IncidenceVector restrict *output*)

close an attribute set, i.e.

vector/private.h, (c) 2013, Immanuel Albrecht; Dresden University of Technology, - Professur für die Psychologie des Lernen und Lehrens

add further attributes

Parameters

ctx	formal context
input	the intent set that is to be closed
output	the closure intent" wrt. ctx

some attribute is not present for this object -> next object

remove attributes that are not common among all objects that have the input attributes

Definition at line 411 of file fcaV.c.

References MASKVECTOR, and ROW.

Referenced by countContextConceptsV(), newConceptBulkFromContextV(), and nextClosureVX1().

```
{

    myFormalContextV* restrict I;
    I = (myFormalContextV*) ctx;
    for (size_t var = 0; var < I->width; ++var)
    {
        output[var] = ~0ULL;
    }

    MASKVECTOR(output, I->attributes);

    for (size_t g = 0; g < I->objects; ++g)
    {
        int good;
        good = 1;

        for (size_t i = 0; i < I->width; ++i)
        {
            if ((input[i]) & (~(ROW(g,I)[i])))
            {
                good = 0;
                break;
            }

        }

        if (good)
        {
            for (size_t i = 0; i < I->width; ++i)
            {
                output[i] &= ROW(g,I)[i];
            }
        }
```

```
    }
}
```

4.14.2.3 size_t countConceptsInBulkV (FormalConceptIntentBulkListV *root*)

use this for bulks that are filled in order

Parameters

root	

Returns

number of concepts in bulk

count the full chunks

and the last chunk

Definition at line 795 of file fcaV.c.

References sFormalConceptIntentBulkNodeV::chunks, CHUNKSIZEV, sFormalConceptIntentBulkNodeV::next, RETURN_ZERO_IF_ZERO, smyFormalConceptIntentChunkV::size, and sFormalConceptIntentBulkNodeV::size.

Referenced by main(), and writeConceptsToFileV().

```
{
    RETURN_ZERO_IF_ZERO(root);

    size_t count = 0;

    while (root != 0)
    {
        if (root->size > 0)
        {
            count += CHUNKSIZEV * (root->size - 1);
            count += root->chunks[root->size - 1]->size;
        }
        root = root->next;
    }

    return count;
}
```

4.14.2.4 int countContextConceptsV (FormalContextV *ctx*)

counts the concepts in the concept lattice of ctx, using next closure algorithm

Parameters

ctx	formal context

Returns

number of concepts in context

calculate the bottom intent of the concept lattice, i.e. {}”

begin of nextClosure function iteration

we found the next intent

continue with Y for M

do the nextClosure

free up memory

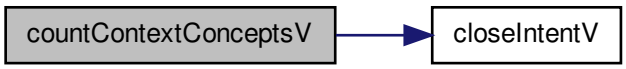
Definition at line 308 of file fcaV.c.

References CLEARV, closeIntentV(), CRIMPVALUE, CROSSV, INCIDESV, OFFSET, and RETURN_ZERO_IF_ZERO.

```
{
    RETURN_ZERO_IF_ZERO(ctx);

    myFormalContextV *restrict c;
    c = (myFormalContextV*) ctx;

    IncidenceVector restrict M;
    IncidenceVector restrict Y;

#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wsign-conversion"

    Y = calloc(c->width, sizeof(uint64_t));
    M = malloc(c->width * sizeof(uint64_t));

#pragma GCC diagnostic pop

    closeIntentV(ctx, Y, M);

    int count;

    count = 1;

    nextClosure:

    for (size_t i = c->attributes; i > 0;)
    {
        --i;

        if (!INCIDESV(M,i))
        {
            CROSSV(M, i);
            closeIntentV(ctx, M, Y);

            int good;
            good = 1;

            for (unsigned int j = 0; j < OFFSET(i); ++j)
            {
                if (Y[j] & (~(M[j])))
                {
                    good = 0;
```

```
                    break;
                }
            }
            if (good)
            {
                if (Y[OFFSET(i)] & (~M[OFFSET(i)]) & CRIMPVALUE(i))
                {
                    good = 0;
                }
            }

            if (good)
            {
                count++;

                IncidenceVector DELTA;
                DELTA = M;
                M = Y;
                Y = DELTA;
                goto nextClosure;
            }
        }

        CLEARV(M, i);
    }

    free(M);
    free(Y);

    return count;
}
```

Here is the call graph for this function:



4.14.2.5 void deleteConceptBulkV (FormalConceptIntentBulkListV * *rootNode*)

deletes the entire bulk list

Parameters

rootNode	pointer to the first node

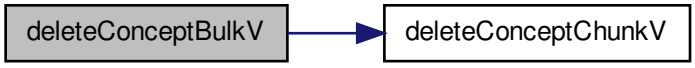
Definition at line 763 of file fcaV.c.

References sFormalConceptIntentBulkNodeV::chunks, deleteConceptChunkV(), sFormalConceptIntentBulkNodeV::next, RETURN_IF_ZERO, and sFormalConceptIntentBulkNodeV::size.

Referenced by main().

```
{
    RETURN_IF_ZERO(rootNode);
    RETURN_IF_ZERO(*rootNode);

    FormalConceptIntentBulkListV l;
    l = *rootNode;
    *rootNode = 0;

    do
    {
        for (size_t var = 0; var < l->size; ++var)
        {
            deleteConceptChunkV(&(l->chunks[var]));
        }

        FormalConceptIntentBulkListV next;
        next = l->next;

        free(l->chunks);
        free(l);

        l = next;
    } while (l != 0);
}
```

Here is the call graph for this function:



4.14.2.6 void deleteConceptChunkV (myFormalConceptIntentChunkV ** c)

deletes a concept chunk object and sets its pointer to zero

Parameters

c	pointer to the concept chunk to be deleted

Definition at line 536 of file fcaV.c.

References RETURN_IF_ZERO.

Referenced by deleteConceptBulkV().

```
{
    RETURN_IF_ZERO(c);
    RETURN_IF_ZERO(*c);

    free((*c)->incidence);

    free(*c);
    *c = 0;
}
```

4.14.2.7 void deleteFormalContextV (FormalContextV * *ctx*)

deletes the formal context *ctx, and sets the pointer to zero

Parameters

ctx	pointer to the formal context object to be deleted

Definition at line 84 of file fcaV.c.

References smyFormalContextV::attributeNames, smyFormalContextV::attributes, smyFormalContextV::incidence, smyFormalContextV::objectNames, smyFormalContextV::objects, and RETURN_IF_ZERO.

Referenced by main().

```
{
    RETURN_IF_ZERO(ctx);
    RETURN_IF_ZERO(*ctx);

    myFormalContextV *c;

    c = (myFormalContextV*) *ctx;

    *ctx = 0;

    for (unsigned int var = 0; var < c->attributes; ++var)
    {
        free(c->attributeNames[var]);
    }

    for (unsigned int var = 0; var < c->objects; ++var)
    {
        free(c->objectNames[var]);
    }

    free(c->objectNames);
    free(c->attributeNames);
    free(c->incidence);
    free(c);
}
```

4.14.2.8 int intentCmpV (size_t *attributes*, const IncidenceVector *minus*, const IncidenceVector *plus*)

compare two intent vectors

Parameters

attributes	attribute count
minus	"left" operand
plus	"right" operand

Returns

-1 if minus is bigger, 1 if plus is bigger, 0 if minus and plus is the same

in this case, OFFSET(attributes) == OFFSET(attributes-1)

we only check the lower bits 0 through (attributes-1)

ELSE: attributes has 64 as factor, so we have done all necessary comparisons in the first loop.

Definition at line 466 of file fcaV.c.

References BITNBR, CRIMPVALUE, and OFFSET.

Referenced by nextClosureVX1().

```
{
    for (size_t var = 0; var < OFFSET(attributes); ++var)
    {
        if (minus[var] > plus[var])
            return -1;
        if (plus[var] > minus[var])
            return 1;
    }

    if (BITNBR(attributes))
    {
        uint64_t l, r;

        l = minus[OFFSET(attributes)] & CRIMPVALUE(attributes-1);
        r = plus[OFFSET(attributes)] & CRIMPVALUE(attributes-1);

        if (l > r)
            return -1;

        if (r > l)
            return 1;
    }
    return 0;
}
```

4.14.2.9 FormalConceptIntentBulkListV newConceptBulkFromContextV (FormalContextV *ctx*)

creates a new formal concept intent chunk and fills it with the intents of all formal concepts in the concept lattice of ctx, using next closure algorithm

Parameters

ctx	formal context

Returns

concept intents

calculate the bottom intent of the concept lattice, i.e. {}"

add the bottom element of the concept lattice (a concept lattice is never empty)

begin of nextClosure function iteration

we found the next intent

continue with Y for M

do the nextClosure

free up memory

Definition at line 575 of file fcaV.c.

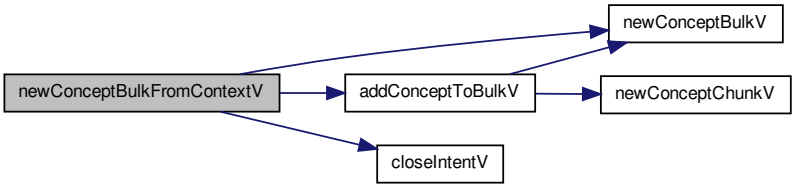
References addConceptToBulkV(), CLEARV, closeIntentV(), CRIMPVALUE, CROSSV, INCIDESV, newConceptBulkV(), OFFSET, and RETURN_ZERO_IF_ZERO.

Referenced by main().

```
{
    RETURN_ZERO_IF_ZERO(ctx);

    myFormalContextV * restrict c;
    c = (myFormalContextV*) ctx;

    IncidenceVector restrict M;
    IncidenceVector restrict Y;

#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wsign-conversion"

    Y = calloc(c->width, sizeof(uint64_t));
    M = malloc(c->width * sizeof(uint64_t));

#pragma GCC diagnostic pop

    closeIntentV(ctx, Y, M);

    FormalConceptIntentBulkListV root;
    FormalConceptIntentBulkListV last;

    root = newConceptBulkV(c->attributes);
```

```
    last = addConceptToBulkV(root, M);

    nextClosure:

    for (size_t i = c->attributes; i > 0;)
    {
        --i;

        if (!INCIDESV(M,i))
        {
            CROSSV(M, i);
            closeIntentV(ctx, M, Y);

            int good;
            good = 1;

            for (unsigned int j = 0; j < OFFSET(i); ++j)
            {
                if (Y[j] & (~(M[j])))
                {
                    good = 0;
                    break;
                }
            }
            if (good)
            {
                if (Y[OFFSET(i)] & (~M[OFFSET(i)]) & CRIMPVALUE(i))
                {
                    good = 0;
                }
            }

            if (good)
            {

                last = addConceptToBulkV(last, Y);

                IncidenceVector DELTA;
                DELTA = M;
                M = Y;
                Y = DELTA;
                goto nextClosure;
            }
        }

        CLEARV(M, i);
    }

    free(M);
    free(Y);

    return root;
}
```

Here is the call graph for this function:



4.14.2.10 FormalConceptIntentBulkListV newConceptBulkV (size_t *attributes*)

creates a new formal concept intent bulk list

Parameters

attributes	number of attributes of the concept intents

Returns

new formal concept intent bulk list's first node

Definition at line 554 of file fcaV.c.

References sFormalConceptIntentBulkNodeV::attributes, BULKSIZEV, sFormalConceptIntentBulkNodeV::chunks, sFormalConceptIntentBulkNodeV::next, sFormalConceptIntentBulkNodeV::size, WIDTH, and sFormalConceptIntentBulkNodeV::width.

Referenced by addConceptToBulkV(), newConceptBulkFromContextV(), and nextClosureVX1().

```
{
    FormalConceptIntentBulkListV l;
    l = malloc(sizeof(struct sFormalConceptIntentBulkNodeV));

    l->attributes = attributes;
    l->width = WIDTH(attributes);
    l->size = 0;
    l->chunks = calloc(BULKSIZEV, sizeof(myFormalConceptIntentChunkV*));
    l->next = 0;
    return l;
}
```

4.14.2.11 myFormalConceptIntentChunkV* newConceptChunkV (size_t *attributes*)

create a new formal concept chunk

Parameters

attributes	number of attributes of the hosting formal context

Returns

a new concept chunk object

Definition at line 509 of file fcaV.c.

References smyFormalConceptIntentChunkV::attributes, CHUNKSIZEV, smyFormalConceptIntentChunkV::incidence, smyFormalConceptIntentChunkV::size, WIDTH, and smyFormalConceptIntentChunkV::width.

Referenced by addConceptToBulkV().

```
{

    myFormalConceptIntentChunkV *c;

    c = malloc(sizeof(myFormalConceptIntentChunkV));

    c->attributes = attributes;
    c->width = WIDTH(attributes);
    c->size = 0;

#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wsign-conversion"

    c->incidence = calloc(c->width * CHUNKSIZEV, sizeof(uint64_t));

#pragma GCC diagnostic pop

    return c;
}
```

4.14.2.12 FormalContextV newFormalContextFromFileV (const char * *filename*)

create a new formal context object from a .cxt file

Parameters

filename	

Returns

the formal context that has been read from the given file

this should never happen, right?

we read all data

free memory and return

Definition at line 118 of file fcaV.c.

References smyFormalContextV::attributeNames, CROSSV, INPUTBUFFERSIZE, MIN, newFormalContextV(), smyFormalContextV::objectNames, RETURN_ZERO_IF_ZERO, and ROW.

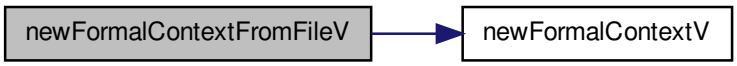
Referenced by main().

```
{
    char *line;
    size_t len;

    len = (INPUTBUFFERSIZE);
    line = malloc(sizeof(char) * len);

    FILE *file;

    if (strcmp(filename, "-") == 0)
    {
        file = stdin;
    }
    else
    {
        file = fopen(filename, "r");
        RETURN_ZERO_IF_ZERO(file);
    }

    ssize_t read;

    unsigned int line_nbr;
    line_nbr = 0;

    unsigned int objects;
    unsigned int attributes;

    attributes = 0;
    objects = 0;

    myFormalContextV *ctx;
    ctx = 0;

    while ((read = getline(&line, &len, file)) != -1)
    {

        if (read == 0)
            break;
        line[read - 1] = 0;

        if (line_nbr == 0)
        {
            if (strcmp(line, "B"))
            {
                fprintf(stderr, "File '%s' is not a .cxt file\n", filename);
                goto grace;
            }
        }
        else if (line_nbr == 1)
        {
            //empty line
        }
        else if (line_nbr == 2)
        {
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wsign-conversion"
```

```
            objects = atoi(line);
#pragma GCC diagnostic pop
        }
        else if (line_nbr == 3)
        {
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wsign-conversion"
            attributes = atoi(line);
            ctx = (myFormalContextV *) newFormalContextV(objects, attributes);
#pragma GCC diagnostic pop
        }
        else if (line_nbr == 4)
        {
            //empty line
        }
        else if (line_nbr < objects + 5)
        {

            unsigned int i;
            i = line_nbr - 5;

            free(ctx->objectNames[i]);
            ctx->objectNames[i] = strdup(line);

        }
        else if (line_nbr < objects + attributes + 5)
        {
            unsigned int i;
            i = line_nbr - 5 - objects;

            free(ctx->attributeNames[i]);
            ctx->attributeNames[i] = strdup(line);
        }
        else if (line_nbr < objects * 2 + attributes + 5)
        {
            unsigned int i;
            i = line_nbr - 5 - objects - attributes;

            unsigned int width;
            width = MIN((unsigned int)strlen(line),attributes);

            for (unsigned int var = 0; var < width; ++var)
            {
                if ((line[var] == 'x') || (line[var] == 'X')
                        || (line[var] == '1'))
                {
                    CROSSV(ROW(i, ctx), var);
                }
            }

        }
        else
        {
            break;
        }

        line_nbr++;
    }

    grace: if (file != stdin)
    {
        fclose(file);
```

```
    }

    free(line);

    return (FormalContextV) ctx;
}
```

Here is the call graph for this function:



4.14.2.13 FormalContextV newFormalContextV (unsigned int *objects*, unsigned int *attributes*)

create a new formal context

Parameters

objects	object count
attributes	attribute count

Returns

a new FormalContext object

Definition at line 42 of file fcaV.c.

References smyFormalContextV::attributeNames, smyFormalContextV::attributes, smyFormalContextV::incidence, smyFormalContextV::objectNames, smyFormalContextV::objects, WIDTH, and smyFormalContextV::width.

Referenced by newFormalContextFromFileV().

```
{
    myFormalContextV *ctx = malloc(sizeof(myFormalContextV));

    ctx->attributes = attributes;
    ctx->objects = objects;
    ctx->width = WIDTH(attributes);

#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wsign-conversion"

    ctx->attributeNames = calloc(attributes, sizeof(char*));
```

```
    ctx->objectNames = calloc(objects, sizeof(char*));

#pragma GCC diagnostic pop

    for (unsigned int var = 0; var < attributes; ++var)
    {
        ctx->attributeNames[var] = calloc(1, sizeof(char));
    }

    for (unsigned int var = 0; var < objects; ++var)
    {
        ctx->objectNames[var] = calloc(1, sizeof(char));
    }

#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wsign-conversion"

    ctx->incidence = calloc(objects * ctx->width, sizeof(uint64_t));

#pragma GCC diagnostic pop

    return (FormalContextV) ctx;
}
```

4.14.2.14 void writeConceptsToFileV (FormalContextV *ctx,* FormalConceptIntentBulkListV *root,* const char * *filename*)

write a list of concept intents into a .cxt file

Parameters

ctx	formal context (or 0, is used for attribute names)
root	the first node of the formal concept intent bulk
filename	output file name (.cxt)

Definition at line 686 of file fcaV.c.

References smyFormalContextV::attributeNames, smyFormalContextV::attributes, sFormalConceptIntentBulkNodeV::attributes, sFormalConceptIntentBulkNodeV::chunks, countConceptsInBulkV(), INCIDESV, sFormalConceptIntentBulkNodeV::next, RETURN_IF_ZERO, ROW, smyFormalConceptIntentChunkV::size, sFormalConceptIntentBulkNodeV::size, and WARN_IF_UNEQUAL_DO.

Referenced by main().

```
{
    RETURN_IF_ZERO(root);

    myFormalContextV* c;

    if (ctx != 0)
    {
        c = (myFormalContextV*) ctx;

        WARN_IF_UNEQUAL_DO(c->attributes, root->attributes, c = 0);
    }
```

```
    else
    {
        c = 0;
    }

    RETURN_IF_ZERO(filename);

    FILE* file;
    file = fopen(filename, "w");

    RETURN_IF_ZERO(file);

    size_t objects;
    objects = countConceptsInBulkV(root);

    fprintf(file, "B\n\n%zu\n%zu\n\n", objects, root->attributes);

    for (size_t var = 0; var < objects; ++var)
    {
        fprintf(file, "C%8zu\n", (var + 1));
    }

    if (c != 0)
    {
        for (size_t var = 0; var < c->attributes; ++var)
        {
            fputs(c->attributeNames[var], file);
            fputs("\n", file);
        }
    }
    else
    {
        for (size_t var = 0; var < root->attributes; ++var)
        {
            fprintf(file, "m%8zu\n", (var + 1));
        }
    }

    for (; root != 0; root = root->next)
    {
        for (size_t chunk = 0; chunk < root->size; ++chunk)
        {
            for (size_t g = 0; g < root->chunks[chunk]->size; ++g)
            {
                for (size_t m = 0; m < root->attributes; ++m)
                {
                    if (INCIDESV(ROW(g, root->chunks[chunk]), m))
                        fputs("X", file);
                    else
                        fputs(".", file);
                }
                fputs("\n", file);
            }
        }
    }

    fclose(file);
}
```

Here is the call graph for this function:

4.14.2.15 void writeFormalContextV (FormalContextV *ctx*, const char * *filename*)

save the context ctx at the given file location

Parameters

ctx	
filename	

Definition at line 260 of file fcaV.c.

References smyFormalContextV::attributeNames, smyFormalContextV::attributes, INCIDESV, smyFormalContextV::objectNames, smyFormalContextV::objects, RETURN_IF_ZERO, and ROW.

Referenced by main().

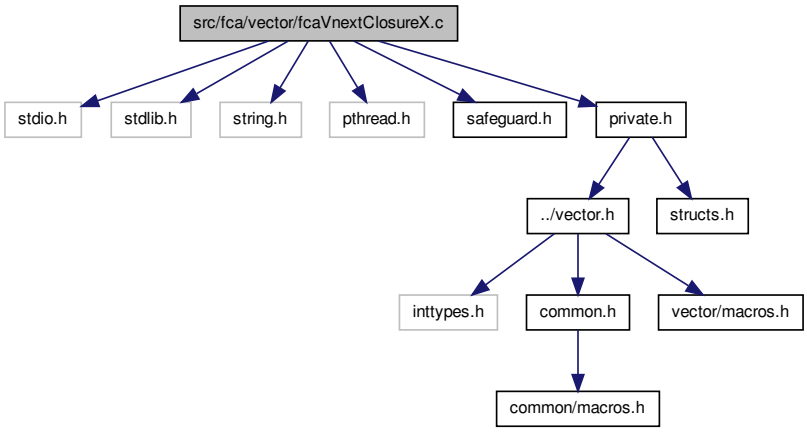
```
{
    RETURN_IF_ZERO(ctx);
    RETURN_IF_ZERO(filename);

    FILE* file;
    file = fopen(filename, "w");

    RETURN_IF_ZERO(file);

    myFormalContextV *c;
    c = (myFormalContextV*) ctx;

    fprintf(file, "B\n\n%zu\n%zu\n\n", c->objects, c->attributes);

    for (unsigned int var = 0; var < c->objects; ++var)
    {
        fputs(c->objectNames[var], file);
        fputs("\n", file);
    }

    for (unsigned int var = 0; var < c->attributes; ++var)
    {
        fputs(c->attributeNames[var], file);
        fputs("\n", file);
    }
```

```
    for (unsigned int g = 0; g < c->objects; ++g)
    {
        for (unsigned int m = 0; m < c->attributes; ++m)
        {
            if ( INCIDESV(ROW(g, c), m))
                fputs("X", file);
            else
                fputs(".", file);
        }
        fputs("\n", file);
    }

    fclose(file);
}
```

4.15 src/fca/vector/fcaVnextClosureX.c File Reference

fcaVnextClosureX.c, (c) 2013, Immanuel Albrecht; Dresden University of Technology, Professur für die Psychologie des Lernen und Lehrens

`#include <stdio.h> #include <stdlib.h> #include <string.h> #include <pthread.h> #include "safeguard.h" #include "private.h"` Include dependency graph for fcaVnextClosureX.c:



Data Structures

- struct snextClosureVX1Params

Typedefs

- typedef struct snextClosureVX1Params * nextClosureVX1Params

Functions

- FormalConceptIntentBulkListV nextClosureVX1 (FormalContextV ctx, const IncidenceVector restrict start, const IncidenceVector restrict stop)

  creates a new formal concept intent chunk and fills it with the intents of all formal concepts in the concept lattice of ctx, using next closure algorithm, that are in a given lexicographic interval of the powerset
- void * callNextClosureVX1 (void *params)
- FormalConceptIntentBulkListV nextClosureVX (FormalContextV ctx)

  creates a new formal concept intent chunk and fills it with the intents of all formal concepts in the concept lattice of ctx, using a parallel next closure algorithm with up to 8 threads

4.15.1 Detailed Description

fcaVnextClosureX.c, (c) 2013, Immanuel Albrecht; Dresden University of Technology, Professur für die Psychologie des Lernen und Lehrens This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>. this file contains a multi-threading nextClosure implementation using the IncidenceVector implementation

Definition in file fcaVnextClosureX.c.

4.15.2 Typedef Documentation

4.15.2.1 typedef struct snextClosureVX1Params* nextClosureVX1Params

4.15.3 Function Documentation

4.15.3.1 void* callNextClosureVX1 (void * *params*)

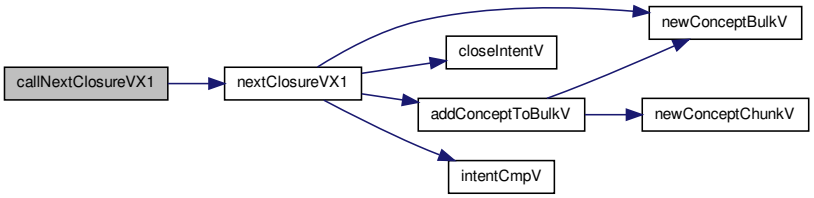
Definition at line 180 of file fcaVnextClosureX.c.

References snextClosureVX1Params::ctx, nextClosureVX1(), snextClosureVX1Params::rVal, snextClosureVX1Params::start, and snextClosureVX1Params::stop.

Referenced by nextClosureVX().

```
{
    nextClosureVX1Params p;
    p = (nextClosureVX1Params) params;

    p->rVal = nextClosureVX1(p->ctx, p->start, p->stop);

    return 0;
}
```

Here is the call graph for this function:



4.15.3.2 FormalConceptIntentBulkListV nextClosureVX (FormalContextV *ctx*)

creates a new formal concept intent chunk and fills it with the intents of all formal concepts in the concept lattice of ctx, using a parallel next closure algorithm with up to 8 threads

Parameters

ctx	formal context

Returns

concept intents

Definition at line 199 of file fcaVnextClosureX.c.

References smyFormalContextV::attributes, callNextClosureVX1(), CROSSV, snextClosureVX1Params::ctx, sFormalConceptIntentBulkNodeV::next, nextClosureVX1(), RETURN_ZERO_IF_ZERO, snextClosureVX1Params::rVal, snextClosureVX1Params::start, snextClosureVX1Params::stop, and smyFormalContextV::width.

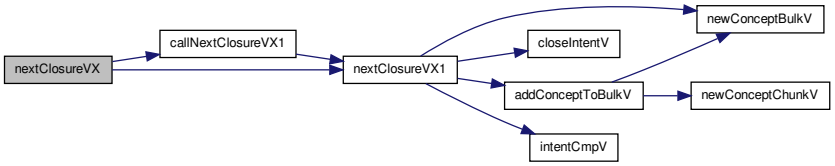
Referenced by main().

```
{
    RETURN_ZERO_IF_ZERO(ctx);

    myFormalContextV *c;
    c = (myFormalContextV*) ctx;
```

```
    size_t N;

    N = 1;

    if (c->attributes >= 3)
        N = 8;
    else if (c->attributes >= 2)
        N = 4;
    else if (c->attributes >= 1)
        N = 2;

    if (N < 2)
        return nextClosureVX1(ctx, 0, 0);

    IncidenceVector bounds;
    bounds = calloc(c->width * (N - 1), sizeof(uint64_t));

    if (N == 2)
    {
        CROSSV(bounds, 0);
    }
    else if (N == 4)
    {
        CROSSV(bounds, 1); //01

        CROSSV(bounds + c->width, 0); //10

        CROSSV(bounds + c->width * 2, 1); //11
        CROSSV(bounds + c->width * 2, 0);
    }
    else if (N == 8)
    {
        CROSSV(bounds, 2); // 001

        CROSSV(bounds + c->width, 1); //010

        CROSSV(bounds + c->width * 2, 2); //011
        CROSSV(bounds + c->width * 2, 1);

        CROSSV(bounds + c->width * 3, 0); //100

        CROSSV(bounds + c->width * 4, 0); //101
        CROSSV(bounds + c->width * 4, 2);

        CROSSV(bounds + c->width * 5, 1); //110
        CROSSV(bounds + c->width * 5, 0);

        CROSSV(bounds + c->width * 6, 0); //111
        CROSSV(bounds + c->width * 6, 1);
        CROSSV(bounds + c->width * 6, 2);
    }

//  for (int i = 0; i < N - 1; ++i)
//  {
//      printf("BOUND %16llx\n", *(bounds + i *
        c->width)&CRIMPVALUE(c->attributes-1));
//      if (i > 0)
//          printf("CMP %d\n",
//                  intentCmpV(c->attributes, bounds + (i - 1) * c->width,
//                          bounds + i * c->width));
//  }
```

```
    nextClosureVX1Params chunks;
    chunks = malloc(N * sizeof(struct snextClosureVX1Params));

    pthread_t *threads;
    threads = malloc(N * sizeof(pthread_t));

    for (size_t i = 0; i < N; ++i)
    {
        chunks[i].ctx = ctx;
        if (i > 0)
            chunks[i].start = (bounds + c->width * (i - 1));
        else
            chunks[i].start = 0;

        if (i < N - 1)
            chunks[i].stop = (bounds + c->width * (i));
        else
            chunks[i].stop = 0;
    }

    for (size_t i = 0; i < N; ++i)
    {
        pthread_create(&threads[i], 0, callNextClosureVX1, (void*) &chunks[i]);
    }

    for (size_t i = 0; i < N; ++i)
    {
        pthread_join(threads[i], 0);
    }

//  for (size_t i = 0; i < N; ++i)
//  {
//      printf("%zu thread: counts %zu\n", i,
//              countConceptsInBulkV(chunks[i].rVal));
//  }

    FormalConceptIntentBulkListV root;
    FormalConceptIntentBulkListV last;

    root = chunks[0].rVal;
    last = root;

    for (size_t var = 1; var < N; ++var)
    {

        while (last->next)
            last = last->next;

        last->next = chunks[var].rVal;
    }

    free(bounds);
    free(chunks);

    return root;
}
```

Here is the call graph for this function:



4.15.3.3 FormalConceptIntentBulkListV nextClosureVX1 (FormalContextV *ctx,* const IncidenceVector restrict *start,* const IncidenceVector restrict *stop*)

creates a new formal concept intent chunk and fills it with the intents of all formal concepts in the concept lattice of ctx, using next closure algorithm, that are in a given lexicographic interval of the powerset

Parameters

ctx	formal context
start	first attribute vector (not included if it is a concept intent) if this is zero, start with M={}, in this case, we add the bottom concept in case of -M"=={}
stop	last attribute vector, or zero to continue until the end

Returns

concept intents between (start, stop]

calculate the bottom intent of the concept lattice, i.e. {}"

add the bottom element of the concept lattice (a concept lattice is never empty)

begin of nextClosure function iteration

check whether we are still in range

the (pseudo)intent Y is bigger than stop

we found the next intent

continue with Y for M

do the nextClosure

free up memory

Definition at line 47 of file fcaVnextClosureX.c.

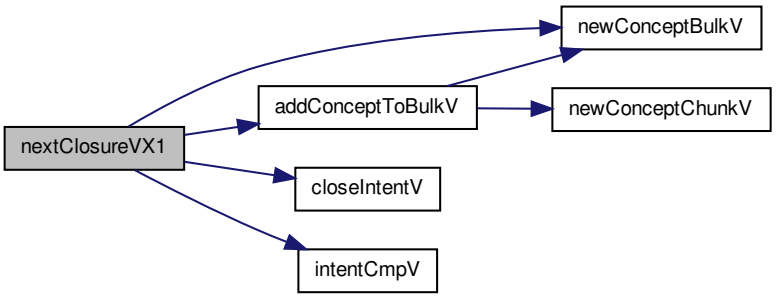
References addConceptToBulkV(), CLEARV, closeIntentV(), CRIMPVALUE, CROSSV, INCIDESV, intentCmpV(), newConceptBulkV(), OFFSET, and RETURN_ZERO_IF_ZERO.

Referenced by callNextClosureVX1(), and nextClosureVX().

```
{
    RETURN_ZERO_IF_ZERO(ctx);

    myFormalContextV * restrict c;
    c = (myFormalContextV*) ctx;

    IncidenceVector restrict M;
    IncidenceVector restrict Y;

    Y = calloc(c->width, sizeof(uint64_t));
    M = malloc(c->width * sizeof(uint64_t));

    FormalConceptIntentBulkListV root;
    FormalConceptIntentBulkListV last;

    root = newConceptBulkV(c->attributes);

    if (start)
    {
        memcpy(M, start, c->width * sizeof(uint64_t));

        last = root;

    }
    else
    {

        closeIntentV(ctx, Y, M);

        last = addConceptToBulkV(root, M);
    }

    nextClosure:

    for (size_t i = c->attributes; i > 0;)
    {
        --i;

        if (!INCIDESV(M,i))
        {
            CROSSV(M, i);
            closeIntentV(ctx, M, Y);

            int good;
            good = 1;

            for (unsigned int j = 0; j < OFFSET(i); ++j)
            {
                if (Y[j] & (~(M[j])))
                {
                    good = 0;
                    break;
                }
            }
            if (good)
            {
                if (Y[OFFSET(i)] & (~M[OFFSET(i)]) & CRIMPVALUE(i))
                {
                    good = 0;
                }
```

```
            }

            if (good)
            {
                if (stop)
                {
                    if (intentCmpV(c->attributes, Y, stop) < 0)
                    {
                        goto gracefulReturn;
                    }
                }

                last = addConceptToBulkV(last, Y);

                IncidenceVector DELTA;
                DELTA = M;
                M = Y;
                Y = DELTA;
                goto nextClosure;
            }
        }

        CLEARV(M, i);
    }

    gracefulReturn:
    free(M);
    free(Y);

    return root;
}
```

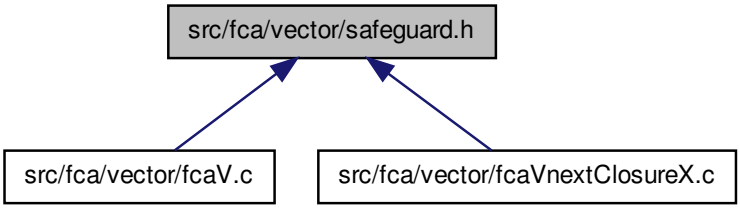
Here is the call graph for this function:



4.16 src/fca/vector/safeguard.h File Reference

vector/safeguard.h, (c) 2013, Immanuel Albrecht; Dresden University of Technology, - Professur für die Psychologie des Lernen und Lehrens

This graph shows which files directly or indirectly include this file:



Defines

- #define ERROR_TOKEN EASY_VERSION_CODE_WITH_VECTOR_FUNCTIONS
- #define CHUNKSIZE ERROR_TOKEN
- #define BULKSIZE ERROR_TOKEN
- #define smyFormalContext ERROR_TOKEN
- #define myFormalContext ERROR_TOKEN
- #define smyFormalConceptIntentChunk ERROR_TOKEN
- #define myFormalConceptIntentChunk ERROR_TOKEN
- #define newConceptChunk ERROR_TOKEN
- #define deleteConceptChunk ERROR_TOKEN
- #define newConceptBulk ERROR_TOKEN
- #define newConceptBulkFromContext ERROR_TOKEN
- #define writeConceptsToFile ERROR_TOKEN
- #define deleteConceptBulk ERROR_TOKEN
- #define countConceptsInBulk ERROR_TOKEN
- #define addConceptToBulk ERROR_TOKEN
- #define closeIntent ERROR_TOKEN
- #define closeIntent2 ERROR_TOKEN
- #define intentCmp ERROR_TOKEN
- #define IncidenceCell ERROR_TOKEN
- #define sFormalContext ERROR_TOKEN
- #define FormalContext ERROR_TOKEN
- #define sFormalIntent ERROR_TOKEN
- #define FormalIntent ERROR_TOKEN
- #define newFormalContext ERROR_TOKEN
- #define newFormalContextFromRandom ERROR_TOKEN
- #define newFormalContextFromFile ERROR_TOKEN
- #define countContextConcepts ERROR_TOKEN
- #define writeFormalContext ERROR_TOKEN
- #define deleteFormalContext ERROR_TOKEN

4.16.1 Detailed Description

vector/safeguard.h, (c) 2013, Immanuel Albrecht; Dresden University of Technology, - Professur für die Psychologie des Lernen und Lehrens This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public - License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>. this file contains safe-guard definitions that prevent using fca/easy functions with vector code

this file helps a lot when porting easy-fca code to vector-fca code by creating compiler errors when using the wrong functions

Definition in file safeguard.h.

4.16.2 Define Documentation

4.16.2.1 #define addConceptToBulk ERROR_TOKEN

Definition at line 47 of file safeguard.h.

Referenced by newConceptBulkFromContext().

4.16.2.2 #define BULKSIZE ERROR_TOKEN

Definition at line 33 of file safeguard.h.

4.16.2.3 #define CHUNKSIZE ERROR_TOKEN

Definition at line 32 of file safeguard.h.

4.16.2.4 #define closeIntent ERROR_TOKEN

Definition at line 48 of file safeguard.h.

Referenced by countContextConcepts(), and newConceptBulkFromContext().

4.16.2.5 #define closeIntent2 ERROR_TOKEN

Definition at line 49 of file safeguard.h.

Referenced by countContextConcepts2().

4.16.2.6 #define countConceptsInBulk ERROR_TOKEN

Definition at line 46 of file safeguard.h.

Referenced by main(), and writeConceptsToFile().

4.16.2.7 #define countContextConcepts ERROR_TOKEN

Definition at line 60 of file safeguard.h.

4.16.2.8 #define deleteConceptBulk ERROR_TOKEN

Definition at line 45 of file safeguard.h.

Referenced by main().

4.16.2.9 #define deleteConceptChunk ERROR_TOKEN

Definition at line 41 of file safeguard.h.

Referenced by deleteConceptBulk().

4.16.2.10 #define deleteFormalContext ERROR_TOKEN

Definition at line 62 of file safeguard.h.

Referenced by main().

4.16.2.11 #define ERROR_TOKEN EASY_VERSION_CODE_WITH_VECTOR_FUNCTIONS

Definition at line 30 of file safeguard.h.

4.16.2.12 #define FormalContext ERROR_TOKEN

Definition at line 54 of file safeguard.h.

4.16.2.13 #define FormalIntent ERROR_TOKEN

Definition at line 56 of file safeguard.h.

4.16.2.14 #define IncidenceCell ERROR_TOKEN

Definition at line 52 of file safeguard.h.

4.16.2.15 #define intentCmp ERROR_TOKEN

Definition at line 50 of file safeguard.h.

4.16.2.16 #define myFormalConceptIntentChunk ERROR_TOKEN

Definition at line 38 of file safeguard.h.

4.16.2.17 #define myFormalContext ERROR_TOKEN

Definition at line 36 of file safeguard.h.

4.16.2.18 #define newConceptBulk ERROR_TOKEN

Definition at line 42 of file safeguard.h.

Referenced by addConceptToBulk(), and newConceptBulkFromContext().

4.16.2.19 #define newConceptBulkFromContext ERROR_TOKEN

Definition at line 43 of file safeguard.h.

Referenced by main().

4.16.2.20 #define newConceptChunk ERROR_TOKEN

Definition at line 40 of file safeguard.h.

Referenced by addConceptToBulk().

4.16.2.21 #define newFormalContext ERROR_TOKEN

Definition at line 57 of file safeguard.h.

Referenced by newFormalContextFromFile(), and newFormalContextFromRandom().

4.16.2.22 #define newFormalContextFromFile ERROR_TOKEN

Definition at line 59 of file safeguard.h.

4.16.2.23 #define newFormalContextFromRandom ERROR_TOKEN

Definition at line 58 of file safeguard.h.

Referenced by main().

4.16.2.24 #define sFormalContext ERROR_TOKEN

Definition at line 53 of file safeguard.h.

4.16.2.25 #define sFormalIntent ERROR_TOKEN

Definition at line 55 of file safeguard.h.

4.16.2.26 #define smyFormalConceptIntentChunk ERROR_TOKEN

Definition at line 37 of file safeguard.h.

4.16.2.27 #define smyFormalContext ERROR_TOKEN

Definition at line 35 of file safeguard.h.

4.16.2.28 #define writeConceptsToFile ERROR_TOKEN

Definition at line 44 of file safeguard.h.

Referenced by main().

4.16.2.29 #define writeFormalContext ERROR_TOKEN

Definition at line 61 of file safeguard.h.

Referenced by main().