Upper bounds on Weight Hierarchies of Extremal Non-Chain Codes

A Study in a Projective Geometry

Thesis for the degree of
Candidatus Scientiarum

Industrial and Applied Mathematics and Informatics
Applicable Algebra and Coding Theory

Hans Georg Schaathun

20th December 1998

Department of Informatics

Universitas Bergensis

BOOK II: THE BOOK OF SOURCE CODE

Contents

Chapter 1

Common header files

```
/* (C) 1998: Hans Georg Schaathun <georg@ii.uib.no> */
/* $Id: common.h,v 5.0 1998/12/20 00:23:58 georg Exp $ */

#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <strings.h>
#include <time.h>
#include <sys/times.h>
#include <limits.h>
#include <libgen.h>
#include <hgd.h>
#include <errno.h>

#include "gen.h"
#include "errors.h"
#include "aux.h"

#if SYSTEM == alpha
#  define cftime(s,p,t)    strcpy ( s, ctime ( t ) )
#endif

#define TRUE      1
#define FALSE     0
#define SUCCESS  0
#define TERM  1
#define ECHO  1<<1

#define min(a,b)  ( ( a<b ) ? a : b )
#define max(a,b)  ( ( a>b ) ? a : b )

/* End of common.h */
```

```
/* (C) 1998: Hans Georg Schaathun <georg@ii.uib.no> */
/* $Id: gen.h,v 5.0 1998/12/20 00:24:09 georg Exp $ */

/* Config options */
#define ALLOW_FIX
#define DEF_ALARM     0

```

```c
/* Version numbers and other identification of programmes */
#define GEN_VERSION  "4.1"
#define VSH_VERSION  "4.2"
#define GEN_NAME     "gen"
#define VDG_NAME     "vdg"
#define VSH_NAME     "vsh"
#define PGC_NAME     "pgc"
#define PID_NAME     "pidcheck"
#define PGT_NAME     "pgtest"
#define GEN_COMPILE  __DATE__ ", " __TIME__

/* External programmes */
#define mta          "/usr/lib/sendmail"

/* Array and string sizes. */
#define MAX_DIM        8
#define FILE_NAME_L    24
#define PATH_NAME_L    128
#define CALL_STRING_L  128
#define TIME_STRING_L  40
#define LINE_LENGTH    160

/* Backup Control */
#define MIN_BCINT    65536          /* = 2^16 */
#define DEF_BCINT    16777216       /* = 2^24 */
#define DEF_BCEXT    ".bc"
#define DEF_BCBCEXT  "~"

/* Default nice values */
#ifdef DEBUG
#  define DEF_NICE     0
#  define DEF_CNICE    0
#  define CC_DEBUG     "+DEBUG"
#else
#  define DEF_NICE     19
#  define DEF_CNICE    6
#  define CC_DEBUG     "-DEBUG"
#endif

#define CC_DEFINES     CC_DEBUG

/* File identification strings */
/* Keep support for VT_FILEID VT1vt */
#define VT_FILEID      "VT2vt"
#define PG_FILEID      "PG1.6pg"

/* Default File Names */
#ifndef PG_DIR
#  define PG_DIR       "/Home/stud/georg/inf/coding/gen/data/pg"
#endif
#define GEN_OUTFILE      "gen:output"
#define GEN_BOUTFILE     "gen:binary"
#define GEN_INFILE       "pgc:output"
#define GEN_ACTFILE      "active"
#define GEN_PASFILE      "passive"
#define GEN_RESTART      "restart"
#define DEF_SIGBFILE     "gen:signalexit"
#define VAL_STATUSFILE   "gen:status"
#define VSH_CONF         "vsh.config"

/* Special macroes */
#define StatusReport(a)  val_statusreport (a)
```

```
/* End of gen.h */
```

```
/* (C) 1998: Hans Georg Schaathun <georg@ii.uib.no> */
/* $Id: errors.h,v 5.0 1998/12/20 00:24:01 georg Exp $ */

#define ENO_OK        0
#define ERR_USE       "Usage Error",                  2
#define ENO_USE       2
#define ERR_FLAGS     "Usage Error: Incompatible flags.", 2
#define ERR_BCINT     "Minimum backup interval is 1024.", 2
#define ENO_SIG       4
#define ENO_INT       5
#define ERR_DATA      "Lacking mandatory data",   6
#define ERR_NOGF      "No support for Galois Fields larger than 5", 8

#define ERR_MEM       "Could not allocate memory", 16
#define ENO_MEM       16
#define ERR_OVERFLOW  "Number overflow",          17
#define ERR_BUF       "Buffer overflow",         18
#define ENO_BUF       18


#define ERR_FOPEN     "Could not open file: ",   32
#define ENO_FOPEN     32
#define ERR_FFORMAT   "Illegal file format",     33
#define ERR_LOCKFAIL  "Failed to creat lockfile", 34

#define ERR_IMP       "Feature not implemented", 128
#define ENO_IMP       128
#define ERR_PROG      "Fatal bug",             129

/* End of errors.h */
```

Chapter 2

Projective Geometry

2.1 The simple representation

```
/* (C) 1998: Hans Georg Schaathun <georg@ii.uib.no> */
/* $Id: pg.h,v 5.0 1998/12/20 00:24:43 georg Exp $ */

#define pg_maxel 65534
#define pg_noel  65535

typedef unsigned short t_ps;   /* Subspace by number */
typedef unsigned char  t_pdim; /* Parameter of a projective geometry */

/* Projective Geometry */
typedef struct s_pg {
  t_pdim k, q;  /* The dimension of the geometry */
  t_ps      ptno[MAX_DIM];
  t_ps      elno[MAX_DIM];
  t_ps      theta[MAX_DIM][MAX_DIM];
    /* Lookup tables for number of points and subspaces. **
    ** See the macroes below.                            */
  t_ps   ****ps;
    /* Incidence structure: ps[i][j][k][l] = m, means that     **
    ** (i+j+1)-space #m is the l'th superspace of i-space #k. */
} t_pg;

t_pg *pg_load  ( FILE *stream );
  /* Constructor:  Load geometry from file    */
void  pg_del   ( t_pg *pg );
  /* Destructor:   Deallocate memory          */
t_pg *pg_alloc ( t_pdim k, t_pdim q );
  /* Allocate memory (used by constructors, incl. pg_create in the **
  ** pginit module.                                                */
t_ps  pg_brack ( t_pdim q, t_ps r, t_ps s );
  /* Compute [r-s,r]_/[1,s]_ */
int pg_dimprint ( FILE *fs, t_pdim k, t_pdim q ) ;
  /* Print counts of subspaces and points in PG(k,q) */
unsigned long pg_igs ( int q, int p ) ;
  /* Compute a geomtric series of integer values without calling pow(3). **
  ** i.e. compute Q(p) = q^0+q^1+q^2+...+q^p.                           */

#define pg_q(q,i)        pg_igs (q,i)
  /* Compatibility alias for pg_igs() */
```

```c
/* Costly macros */
#define pg_q_pow (q, i)   ( ( pow ( q, i+1 ) - 1 ) / ( q - 1 ) )
  /* Compute Q(i) */
#define pg_elno(q,k,d)    pg_brack ( q, k+1, d+1 )
  /* Compute the number of d-spaces */
#define pg_theta(r,s,k,q) pg_brack ( q, k-(r), k-(s) )
  /* Compute \chi (r,s;k,q) */

/* Looking up the same values in tables */
#define pg_eln(pg,i)      ( pg->elno[i] )
  /* Return the number of d-spaces, i.e. get pg_elno() from a lookup table. */
#define pg_qpow(pg,i)     ( pg->ptno[i] )
  /* Return Q(i), i.e. the number of points in an i-space,  **
  ** from a lookup table.                                   */
#define pg_thet(pg,r,s)   ( pg->theta[r][s] )
  /* Look up the number of s-spaces through any particular r-space. */

/* End: pg.h */
```

```c
/* (C) 1998: Hans Georg Schaathun <georg@ii.uib.no> */
/* $Id: pg.c,v 5.0 1998/12/20 00:24:40 georg Exp $ */

/* The constructing functions are in the pginit module. */

#include "common.h"

#include "pg.h"
#include "aux.h"

/* Constructors */
t_pg *pg_load ( FILE *stream ) {
  t_ps g, gm, h, hm, i = 0, im, j, jm;
  t_pg *pg;
  t_pdim k, q;

  while ( 0 != PG_FILEID[i] ) {
    if ( PG_FILEID[i++] != getc ( stream ) ) ErrEx ( ERR_FFORMAT );
  };

  k = getc ( stream );
  q = getc ( stream );

  pg = pg_alloc ( k, q );

  for ( gm = pg->k-1, g = 0; g < gm; ++g ) {
    im = pg_eln ( pg, g );
    for ( hm = pg->k-g-1, h = 0; h < hm; ++h ) {
      jm = pg_thet ( pg, g, g+h+1 );
      for ( i = 0; i < im; ++i ) {
        for ( j = 0; j < jm; ++j ) pg->ps[g][h][i][j] = getshort ( stream );
      };
    };
  };

  return ( pg );
}

/* Destructor */
void  pg_del ( t_pg *pg ) {
  t_ps g, gm, h, hm, i, im;

```

```
  for ( gm = pg->k-1, g = 0; g < gm; ++g ) {
    im = pg_eln ( pg, g );
    for ( hm = pg->k-g-2, h = 0; h < hm; ++h ) {
      for ( i = 0; i < im; ++i ) free ( pg->ps[g][h][i] );
      free ( pg->ps[g][h] );
    };
    free ( pg->ps[g] );
  };

  free ( pg->ps );
  free ( pg );

}

/* Allocate memory (subroutine for the constructors). */
t_pg *pg_alloc ( t_pdim k, t_pdim q ) {
  t_pg *pg;       /* pointer to be returned */
  char h;         /* loop index */
  t_ps i, j, g, /* loop indices */
       jm, gm;  /* upper bounds for loop indices */

/* Allocate memory and insert the parameters for the size of the geometry. */
  pg = getmem ( sizeof ( t_pg ) );
  pg->k = k; pg->q = q;
  pg->ps = getmem ( (k-1)*sizeof (void*) );

/* Precompute the number of m-spaces for each m, and the theta function. */
  for ( gm = k, g = 0; g <= gm; g++ ) {
    pg->elno[g] = pg_elno (q,k,g);
    pg->ptno[g] = pg_q (q,g);
    for ( h = g; h++ < gm; ) {
      pg->theta[g][h] = pg_theta ( g, h, k, q );
    };
  };

/* Allocate the arrays. */
  for ( h = 0; h < k - 1; h++ ) {
    pg->ps[h] = getmem ( ( k - h - 1 )*sizeof (void*) );
    jm = pg_eln ( pg, h );
    for ( i = 0; i < k - h - 1; i++ ) {
      gm = pg_thet ( pg, h, h+i+1 ) ;
      pg->ps[h][i] = getmem ( jm*sizeof (void*) );
      for ( j = 0; j < jm; j++ ) {
        pg->ps[h][i][j] = getmem ( gm*sizeof ( t_ps ) );
        for ( g=0; g < gm; ++g ) pg->ps[h][i][j][g] = pg_noel;
      };
    };
  };

  return ( pg );
}

unsigned long pg_igs ( int q, int p ) {
  unsigned int i;
  unsigned long t = 1, s = 1;
  for ( i = 1; i <= p; ++i ) {
    t *= q; /* Get the next power of q. */
    s += t; /* Add a power of q to the sum. */
  }
  return ( s );
}
```

```
t_ps pg_brack ( t_pdim q, t_ps r, t_ps s ) {
  unsigned long a = 1;
  char i;
  for ( i = 0; i < s; ++i ) {
  /* We take the risk to compute pg_igs(q,i) twice for some values of i. **
  ** The overhead is probably not a great problem, anyway.                */
    a *= pg_igs ( q, r - s + i );
    a /= pg_igs ( q, i );
  };
  if ( a > pg_maxel ) ErrEx ( ERR_OVERFLOW );
  return ( a );
}

int pg_dimprint ( FILE *fs, t_pdim k, t_pdim q ) {
  int i;
  for ( i = 0; i < k; ++i ) {
    fprintf ( fs, "#%i: %i\n", i, pg_elno ( q, k, i ) );
  };
  return ( 0 );
}

/* End of pg.c */
```

2.2 The coordinate representation

```
/* (C) 1998: Hans Georg Schaathun <georg@ii.uib.no> */
/* $Id: gf.h,v 5.0 1998/12/20 00:24:16 georg Exp $ */

/* Arithmetric functions in small Galois Fields. */

char gf_mult ( char a, char b, char q ) ;
  /* Compute a*b in GF(q) */
char gf_add  ( char a, char b, char q ) ;
  /* Compute a+b in GF(q) */
char gf_inv  ( char a, char q ) ;
  /* Compute the multiplicative inverse of a in GF(q) */

/* End of gf.h */
```

```
/* (C) 1998: Hans Georg Schaathun <georg@ii.uib.no> */
/* $Id: gf.c,v 5.0 1998/12/20 00:24:13 georg Exp $ */

/* Arithmetric functions in small Galois Fields. */

#include "common.h"

char gf_mult ( char a, char b, char q ) {
  static char t[4][4] =
      { { 0, 0, 0, 0 },
        { 0, 1, 2, 3 },
        { 0, 2, 3, 1 },
        { 0, 3, 1, 2 } };
  if ( ( 6 == q) || ( q > 7 ) || ( q < 2 ) ) ErrExit ( ERR_NOGF );
  /* If q is prime, we use arithmetics modulo q. */
  return ( ( q == 4 ) ? t[a][b] : ( a * b % q ) );
}

char gf_add ( char a, char b, char q ) {
  static char t[4][4] =
```

```
    { { 0, 1, 2, 3 },
      { 1, 0, 3, 2 },
      { 2, 3, 0, 1 },
      { 3, 2, 1, 0 } };
  if ( ( 6 == q) || ( q > 7 ) || ( q < 2 ) ) ErrExit ( ERR_NOGF );
  /* If q is prime, we use arithmetics modulo q. */
  return ( ( q == 4 ) ? t[a][b] : ( ( a + b ) % q ) );
}

char gf_inv ( char a, char q ) {
  static char t[6][7] =
    { { 0, 1, -1, -1, -1, -1, -1 },
      { 0, 1,  2, -1, -1, -1, -1 },
      { 0, 1,  3,  2, -1, -1, -1 },
      { 0, 1,  3,  2,  4, -1, -1 },
      { -1, -1, -1, -1, -1, -1, -1 },
      { 0, 1,  4,  5,  2, 3, 6 } };
  if ( ( 6 == q) || ( q > 7 ) || ( q < 2 ) ) ErrExit ( ERR_NOGF );
  return ( t[q-2][a] );
}

/* End of gf.c */
```

```
/* (C) 1998: Hans Georg Schaathun <georg@ii.uib.no> */
/* $Id: points.h,v 5.0 1998/12/20 00:25:23 georg Exp $ */

typedef t_pdim *t_pt;    /* Coordinates of a point */

t_pt   pt_init    ( t_pdim k ) ;
  /* Constructor: allocate memory for a point in a k-dimensional geometry  */
#define pt_del(pt)  free(pt)
  /* Destructor: deallocate memory  */

int    pt_add     ( t_pt  a, t_pt  b, t_pdim k, t_pdim q ) ;
  /* Add the points a,b in GF(q)^{k+1} */
int    pt_mult    ( t_pt  a, int   c, t_pdim k, t_pdim q ) ;
  /* Multiply the point a by the scalar b in GF(q)^{k+1} */
t_ps   pt_getno   ( t_pt  a, t_pdim k, t_pdim q ) ;
  /* Get the number representation of the point a in PG(k,q) */
int    pt_fromno  ( t_pt  a, t_ps pt, t_pdim k, t_pdim q ) ;
  /* Put the coordinate representation of point pt in PG(k,q) in a */

#define pt_normalize(c,k,q)  ( 2==q ? 0 : pt_norm(c,k,q) )
int    pt_norm        ( t_pt c, t_pdim k, t_pdim q ) ;
  /* Normalize the coordinate representation, c, i.e. make **
  ** sure that the first non-zero coordinate is 1.        **
  ** The macro saves some time if q=2.                    */

long   pt_qpow    ( t_pdim q, int e );
  /* Return q^e, by looking up in a static array which is computed    **
  ** the first time a power of a particular value of q is requested. */

void   pt_print       ( t_pt pt, FILE *stream, t_pdim k, t_pdim q ) ;
  /* Print the point to stream, showing the coordinate representation. */
int    pt_fromstr     ( t_pt c, char *pt ) ;
  /* Convert the ASCII string pt to internal coordinate representation. **
  ** Each coordinate position correspond to one digit in the string.    */

/* End of points.h */
```

```
/* (C) 1998: Hans Georg Schaathun <georg@ii.uib.no> */
/* $Id: points.c,v 5.0 1998/12/20 00:25:20 georg Exp $ */

#include "common.h"
#include "pg.h"
#include "points.h"
#include "gf.h"
#include "aux.h"

static t_ps  pt_vno    ( t_pt c, t_pdim k, t_pdim q ) ;
  /* Auxiliary function for pt_getno(). */
static void  pt_nov    ( t_pt c, t_pdim k, t_pdim q, t_ps pt ) ;
  /* Auxiliary function for pt_fromno(). */

int   pt_add    ( t_pt  a, t_pt  b, t_pdim k, t_pdim q ) {
  register int i;
  for ( i=0; i <= k; ++i ) a[i] = gf_add ( a[i], b[i], q );
  return ( 0 );
}

int   pt_mult   ( t_pt  a, int   c, t_pdim k, t_pdim q ) {
  register int i;
  for ( i=0; i <= k; ++i ) a[i] = gf_mult ( a[i], c, q );
  return ( 0 );
}

t_ps  pt_getno  ( t_pt c, t_pdim k, t_pdim q ) {
  return ( ( k == 0 ) ? ( c[0] == 0 ? pg_noel : 0 ) : (
          ( c[0] == 0 ) ? pt_getno ( c+1, k-1, q )
                  : pt_vno ( c+1, k, q ) + pg_q ( q, k-1 ) + 1 ) );
}

t_ps   pt_vno ( t_pt c, t_pdim k, t_pdim q ) {
  register t_ps i = 0;
  while ( k-- > 0 ) {
    i += *(c++)*pt_qpow ( q, k );
  };
  return ( i - 1 );
}

int   pt_fromno ( t_pt c, t_ps pt, t_pdim k, t_pdim q ) {
  register int i;
  if ( k > 0 ) {
    if ( pt >= ( i = pg_q ( q, k-1 ) ) ) {
      *c = 1;
      /* A non-zero coordinate is placed,               **
      ** do the rest as if it were a Euclidean geometry. */
      pt_nov ( c+1, k, q, pt-i );
    } else {
      *c = 0;
      /* Only non-zero coordinates so far, recurse for the remaining. */
      pt_fromno ( c+1, pt, k-1, q );
    };
  } else {
    /* This is the last coordinate, that one has to be one. */
    *c = 1;
  };
}

void  pt_nov ( t_pt c, t_pdim k, t_pdim q, t_ps pt ) {
  register int i;
  if ( k > 0 )
```

```
  if ( pt >= ( i = pt_qpow ( q, k-1 ) ) ) {
    *c = pt/i;
    pt_nov ( c+1, k-1, q, pt-((*c)*i) );
  } else {
    *c = 0;
    pt_nov ( c+1, k-1, q, pt );
  };
}

t_pt  pt_init   ( t_pdim k ) {
  t_pt pt;
  register int i;
  pt = getmem ( (k+1)*sizeof (t_pdim) );
  for ( i=0; i <= k; ++i ) pt[i] = 0;
  return ( pt );
}

/* pt_norm normalizes the representation *c of a projective point, **
** so that the first nonzero coordinate is equal to 1.              */
int  pt_norm ( t_pt c, t_pdim k, t_pdim q ) {
  register unsigned char i = 0, b;

  while ( c[i] == 0 && i++ < k ) {};
  b = gf_inv(c[i], q);
  c[i] = 1;
  while ( i++ < k ) c[i] = gf_mult (c[i],b,q);
  return ( 0 );
}

void  pt_print     ( t_pt pt, FILE *stream, t_pdim k, t_pdim q ) {
  t_ps i;
  fprintf ( stream, "( " );
  for ( i = 0; i < k; ++ i ) fprintf ( stream, "%i, ", pt[i] );
  fprintf ( stream, "%i )", pt[k] );
}

void  pt_noprint   ( t_ps ps, FILE *stream, t_pdim k, t_pdim q ) {
  char pt[12];
  fprintf ( stream, "%i ", ps );
  pt_fromno ( (t_pt)pt, ps, k, q );
  pt_print ( (t_pt)pt, stream, k, q );
}

int   pt_fromstr ( t_pt c, char *pt ) {
  t_pdim i;
  for ( i = 0; pt[i] != 0; ++i ) c[i] = pt[i] - '0';
  return ( 0 );
}

long  pt_qpow   ( t_pdim b, int e ) {
  static int q[MAX_DIM] = { 1, 0 };
  /* The static array stores powers of q[1]. If powers of a new base b, **
  ** not equal to q[1], the array is recomputed;  but that should not   **
  ** occur very frequently.                                             */
  int i;
  if ( q[1] == b ) return ( q[e] );
  for ( i = 0; ++i < MAX_DIM; ) q[i] = q[i-1]*b;
  return ( q[e] );
}
```

```
/* (C) 1998: Hans Georg Schaathun <georg@ii.uib.no> */
```

```
/* $Id: base.h,v 5.0 1998/12/20 00:23:55 georg Exp $ */

typedef struct s_bas {
  t_pdim  m;   /* The dimension of the basis. */
  t_ps   *bas; /* Array of points for the basis. */
} t_bas;

t_ps   bas_comb ( t_bas *bas, t_ps d, t_pdim k, t_pdim q ) ;
  /* Form a linear combination of points. */

t_bas *bas_init ( t_pdim m ) ;
  /* Constructor: allocate memory, initialize each point with pg_noel */

t_bas *bas_copy ( t_pdim m, t_bas *bas ) ;
  /* Constructor, as many points as possible are copied from bas.   **
  ** The first points are used if bas has greater dimension than m. **
  ** The remaining points are initialized to pg_noel.               */

#define bas_del(b)  ( free ( b->bas ), free ( b ) )
  /* Destructor: deallocate memory */

/* End of base.h */
```

```
/* (C) 1998: Hans Georg Schaathun <georg@ii.uib.no>  */
/* $Id: base.c,v 5.0 1998/12/20 00:23:54 georg Exp $ */

#include "common.h"

#include "aux.h"
#include "pg.h"
#include "points.h"
#include "base.h"

t_ps bas_comb ( t_bas *bas, t_ps d, t_pdim k, t_pdim q ) {
  t_pt a,b;
  t_ps pt;
  int i, e;
  a = pt_init ( k );
  b = pt_init ( k );
  for ( i = 0; i <= bas->m; ++i ) {
    e = d % q;
    d = d / q;
    if ( e ) {
      pt_fromno ( b, bas->bas[i], k, q );
      pt_mult ( b, e, k, q );
      pt_add ( a, b, k, q );
    };
  };

  pt_normalize (a, k, q);
  pt = pt_getno(a, k, q);
  pt_del ( a ); pt_del ( b );
  return ( pt );
}

t_bas *bas_init ( t_pdim m ) {
  t_bas *bas;
  int i;
  bas = getmem (sizeof(t_bas));
  bas->m = m;
  bas->bas = getmem ( (m+1)*sizeof (t_ps) );
```

```
  for ( i=0; i <= m; ++i ) bas->bas[i] = pg_noel;
  return ( bas );
}

t_bas *bas_copy ( t_pdim m, t_bas *bas ) {
  t_bas *b;
  int i;
  b = bas_init ( m );
  m = min ( m, bas->m );
  for ( i=0; i <= m; ++ i ) b->bas[i] = bas->bas[i];
  return ( b );
}

/* End of base.c */
```

2.3 The complete incidence structure

```
/* (C) 1998: Hans Georg Schaathun <georg@ii.uib.no> */
/* $Id: subspace.h,v 5.0 1998/12/20 00:25:54 georg Exp $ */

#include "base.h"

#define msp_in(p,b)        b->pt[p]

typedef struct s_msp {
  t_bas  *bas;
    /* The basis */
  char   *pt;
    /* A boolean array of all points, stating TRUE if the point */
    /* is in the space, FALSE if not.                            */
} t_msp;

t_msp  *msp_init   ( t_bas *bas, t_pdim k, t_pdim q );
  /* Constructor: create a new subspace by spanning the given basis bas  */
  /*   in PG(k,q).                                                       */
#define msp_del(sp)  (bas_del(sp->bas), pt_del(sp->pt), free ( sp ))
  /* Destructor:  deallocate memory.    */
char     msp_subset ( t_msp *a, t_msp *b );
  /* Return TRUE if a is a subspace of b. */
void     msp_print  ( t_msp *sp, FILE *stream, t_pdim k, t_pdim q ) ;
  /* List all points in sp by coordinate representation. */

/* End of subspace.h */
```

```
/* (C) 1998: Hans Georg Schaathun <georg@ii.uib.no> */
/* $Id: subspace.c,v 5.0 1998/12/20 00:25:51 georg Exp $ */

#include "common.h"


#include "pg.h"
#include "points.h"
#include "subspace.h"
#include "aux.h"

t_msp *msp_init   ( t_bas *bas, t_pdim k, t_pdim q ) {
  t_msp *sp;
  t_ps     ptno, i, imax;

```

```
  sp       =  getmem ( sizeof (t_msp) );
  ptno     =  pg_q (q,k);
  sp->bas =  bas;
  sp->pt   =  getmem ( ptno*sizeof (char) );

  for ( i=0; i < ptno; ++i ) sp->pt[i] = 0;

  imax = pt_qpow ( q, bas->m + 1 );
  for ( i=1; i < imax; ++i ) sp->pt[bas_comb( bas, i, k, q )] = 1;

  return ( sp );
}

char  msp_subset ( t_msp *a, t_msp *b ) {
  int i;
  for ( i = 0; i <= a->bas->m; i++ ) {
    /* If one point in the basis of a is not in b, a is not in b. */
    if ( ! msp_in ( a->bas->bas[i], b ) ) return ( FALSE );
  };
  /* The basis of a is in b, so a is in b. */
  return ( TRUE );
}

void    msp_print ( t_msp *sp, FILE *stream, t_pdim k, t_pdim q ) {
  t_ps     ptno, i, imax;
  t_pdim   c[12];

  ptno     =  pg_q (q,k);
  for ( i = 0; i < ptno; ++i )
    if ( msp_in(i,sp) ) {
      fprintf ( stream, "%i ", i );
      pt_fromno ( c, i, k, q );
      pt_print ( c, stream, k, q );
      putc ( '\n', stream );
    };
}

/* End of subspace.c */
```

```
/* (C) 1998: Hans Georg Schaathun <georg@ii.uib.no> */
/* $Id: pgmsp.h,v 5.0 1998/12/20 00:25:03 georg Exp $ */

#include "pg.h"
#include "subspace.h"

/* Collection of subspaces */
typedef struct s_s {
  t_msp   ***s;
    /* s[i][j] is a pointer to a t_msp for the j'th i-space. */
} t_s;

t_ps   pg_get     ( t_pg *pg, int arg[ ], int argno ) ;
  /* Get the subspace spanned by the points in arg */

/* Function to create ASCII prints. */
char  pg_print ( FILE *stream, t_pg *pg );
  /* Print the entire data structure. (Debug purposes) */
char  pg_sprint ( FILE *stream, t_pg *pg );
  /* Print a list of subspaces for each space. (Debug purposes) */
char  pg_pprint ( FILE *stream, t_pg *pg );
  /* Print a list of points for each space. (Debug purposes) */
```

```

char pg_ssprint ( FILE *stream, t_pg *pg,
                  t_pdim dim1, t_pdim dim2, t_ps ps ) ;
  /* Print all dim2-spaces being a sub-/superset of dim1-space #ps */
char  pg_space  ( FILE *stream, t_pg *pg, t_pdim dim, t_ps ps ) ;
  /* Print all points in a given subspace */

  /* Operations on t_s */
t_s  *s_init   ( t_pdim k, t_pdim q );
  /* Constructor: create the t_s struct, and initialize the array **
  ** with (t_msp*) NULL pointers.  The t_msp structs are created  **
  ** pg_create()                                                  */
int   s_del    ( t_s *s, t_pdim k, t_pdim q );
  /* Destructor: deallocate memory     */
int   s_print  ( t_s *s, FILE *stream, t_pdim k, t_pdim q ) ;
  /* Print the entire representation.  (For debug purposes.) */

/* End: pgmsp.h */
```

```
/* (C) 1998: Hans Georg Schaathun <georg@ii.uib.no> */
/* $Id: pgmsp.c,v 5.0 1998/12/20 00:25:01 georg Exp $ */

#include "common.h"
#include "pgmsp.h"
#include "points.h"

t_s   *sp = NULL;

/* Static functions */
static char  pg_supprint ( FILE *stream, t_pg *pg,
                  t_pdim dim, t_pdim supdim, t_ps ps ) ;
  /* Print all superspaces of a given dimension for a given subspace */
static char pg_subprint ( FILE *stream, t_pg *pg,
                  t_pdim dim, t_pdim subdim, t_ps ps ) ;
  /* Print all subspaces of a given dimension for a given subspace */
static t_ps pg_auxget ( t_pg *pg, t_pdim m, t_ps s, t_ps p ) ;
  /* Find an m+1-space through m-space #s and point #p */

/* t_s (subspaces) */

t_s *s_init ( t_pdim k, t_pdim q ) {
  t_s *s;
  t_ps elno, h, i;

  s     = getmem ( sizeof (t_s) );
  s->s  = getmem ( k*sizeof(void*) );

  for ( h = 0; h < k; ++h ) {
    elno = pg_elno ( q,k,h );
    s->s[h] = getmem ( elno*sizeof(void*) );
    for ( i = 0; i < elno; ++i ) s->s[h][i] = (t_msp*) NULL;
  };
  return ( s );
}

int s_del ( t_s *s, t_pdim k, t_pdim q ) {
  t_ps elno, h,i;

  for ( h = 0; h < k; ++h ) {
    elno = pg_elno ( q,k,h );
    for ( i = 0; i < elno; ++i ) {
```

```
      msp_del ( s->s[h][i] );
      free ( s->s[h][i] );
    };
    free ( s->s[h] );
  };

  free ( s->s );
  free ( s );
  return ( 0 );
}

int s_print ( t_s *s, FILE *stream, t_pdim k, t_pdim q ) {
  t_ps i, elno; t_pdim m;
  fprintf ( stream, "Points:\n" );
  elno = pg_elno ( q, k, 0 );
  for ( i=0; i < elno; ) {
    pt_noprint ( i++, stream, k, q );
    putc ( '\n', stream );
  };
  for ( m=0; m++ < k-1; ) {
    fprintf ( stream, "%i-spaces:\n", m );
    elno = pg_elno ( q, k, m );
    for ( i=0; i < elno; ++i ) {
      fprintf ( stream, "%i-space #%i:\n", m, i );
      msp_print ( s->s[m][i], stream, k, q );
    };
  };
  return ( 0 );
}

/* functions on t_pg */
t_ps pg_auxget ( t_pg *pg, t_pdim m, t_ps s, t_ps p ) {
  /* Optimise using a t_s struct. */
  int i, im, j, jm;
  im = pg_theta ( m, m+1, pg->k, pg->q );
  jm = pg_theta ( 0, m+1, pg->k, pg->q );
  /* Consider the im m+1-spaces through m-space #s */
  for ( i = 0; i < im; ++i ) {
    /* Consider the jm m+1-spaces through point #p */
    for ( j = 0; j < jm; ++j ) {
      if ( pg->ps[m][0][s][i] == pg->ps[0][m][p][j] )
        return ( pg->ps[m][0][s][i] );
    }
  }
  return ( pg_noel ); /* No proper space found. */
}

/* Span a space by points */
t_ps pg_get ( t_pg *pg, int arg[], int argno ) {
  int i, j, im, jm, m, tmp, p;
  t_ps s;

  s = arg[--argno];
  /* argno is now the (assumed) dimension of the subspace */

  for ( m = 0; m < argno; ++m ) {
    if ( pg_noel == ( s = pg_auxget( pg, m, s, arg[m] ) ) )
      return ( pg_noel );
  };
  return ( s );
}
```

```
/* File printing */
char pg_print ( FILE *stream, t_pg *pg ) {
  t_ps g, gm, h, hm, i, im;

  for ( gm = pg->k-1, g = 0; g < gm; ++g ) {
    im = pg_elno ( pg->q, pg->k, g );
    for ( hm = pg->k-g-1, h = 0; h < hm; ++h ) {
      for ( i = 0; i < im; ++i ) pg_supprint ( stream, pg, g, g+h+1, i );
    };
  };
  return ( 0 );
}

char pg_supprint ( FILE *stream, t_pg *pg,
                   t_pdim dim, t_pdim supdim, t_ps ps ) {
 t_ps j, jm;
 jm = pg_theta ( dim, supdim, pg->k, pg->q );
 fprintf ( stream, "%i-spaces including %i-space #%i:\n", supdim, dim, ps );
 supdim -= dim + 1;
 for ( j = 0; j < jm; ++j ) {
   fprintf ( stream, "%i", pg->ps[dim][supdim][ps][j] );
   if ( 9 == j % 10 ) putc ( '\n', stream ) ;
   else putc ( '\t', stream );
 };
 putc ( '\n', stream );
}

char pg_subprint ( FILE *stream, t_pg *pg,
                   t_pdim dim, t_pdim subdim, t_ps ps ) {
  t_ps j, jm; char c;
  extern t_s *sp;
  t_msp *ss;

  ss = sp->s[dim][ps];
  jm = pg_elno ( pg->q, pg->k, subdim );
  fprintf ( stream, "%i-spaces in %i-space #%i:\n", subdim, dim, ps );
  for ( c = j = 0; j < jm; ++j ) {
    if ( msp_subset ( sp->s[subdim][j], ss ) ) {
      /* Print the subspace number and separating white space. */
      fprintf ( stream, "%i%c", j, (9==c++?(c=0,'\n'):'\t') );
    };
  };
  if ( c != 0 ) putc ( '\n', stream );
}

char pg_ssprint ( FILE *stream, t_pg *pg,
                  t_pdim dim1, t_pdim dim2, t_ps ps ) {
  return (
    ( dim1 < dim2 ? pg_subprint : pg_supprint )( stream, pg, dim2, dim1, ps )
  );
}

char pg_pprint ( FILE *stream, t_pg *pg ) {
  t_ps g, gm, i, im;

  for ( gm = pg->k-1, g = 0; g < gm; ++g ) {
    im = pg_elno ( pg->q, pg->k, g+1 );
    for ( i = 0; i < im; ++i ) {
      pg_space ( stream, pg, g+1, i );
    };
  };
  return ( 0 );
```

```
}

char pg_space ( FILE *stream, t_pg *pg, t_pdim dim, t_ps ps ) {
  t_ps j, jm, h, hm, c;
  hm = pg_theta ( 0, dim, pg->k, pg->q );
  jm = pg_elno ( pg->q, pg->k, 0 );
  if ( 0 == dim ) {
    fprintf ( stream, "Point #%i.\n", ps );
    return ( FALSE );
  };
  fprintf ( stream, "Points in %i-space #%i:\n", dim--, ps );
  for ( c = j = 0; j < jm; ++j ) {
    for ( h = 0; h < hm; ++h ) {
      if ( pg->ps[0][dim][j][h] == ps ) {
        fprintf ( stream, "%i", j );
        if ( 9 == c++ ) {
          c = 0;
          putc ( '\n', stream );
        } else {
          putc ( '\t', stream );
        };
      };
    };
  };
  putc ( '\n', stream );
  return ( TRUE );
}

char pg_sprint ( FILE *stream, t_pg *pg ) {
  t_ps g, gm, i, im;
  for ( gm = pg->k-1, g = 0; g < gm; ++g ) {
    im = pg_elno ( pg->q, pg->k, g+1 );
    for ( i = 0; i < im; ++i ) pg_subprint ( stream, pg, g+1, g, i );
  };
  return ( 0 );
}

/* End of pgmsp.c */
```

```
/* (C) 1998: Hans Georg Schaathun <georg@ii.uib.no> */
/* $Id: pginit.h,v 5.0 1998/12/20 00:24:57 georg Exp $ */

#include "pgmsp.h"

/* Point Queue */
typedef struct s_ptq {
  t_ps    size, next;
  char   *pt;
} t_ptq;

    /* Operations on t_pg */
#define pg_init(k,q)    pg_create(k,q,NULL)
t_pg    *pg_create ( t_pdim k, t_pdim q, t_s *s );
  /* Constructor: allocate memory and build the geometry.       **
  ** If s is NULL pg_create builds its own t_s struct, which    **
  ** is deleted after use.  Otherwise s is used and it remains  **
  ** intact.                                                   */
char   pg_dump  ( FILE *stream, t_pg *pg );
  /* Store the geometry on a binary file */

  /* Operations on t_ptq */
```

```
t_ptq *ptq_alloc   ( t_ps size );
  /* Constructor: allocate memory */
#define ptq_init(s)  ptq_reinit ( ptq_alloc ( s ) )
  /* Constructor: create a queue with all points in it  */
t_ptq *ptq_reinit  ( t_ptq *ptq );
  /* Reinitialize: put all points in the queue */
#define ptq_del(ptq)  (free ( ptq->pt ), free ( ptq ))
  /* Destructor: deallocate memory */
void   ptq_rmset   ( t_ptq *ptq, t_msp *sp );
  /* remove all points in sp from the queue */
t_ps   ptq_extract ( t_ptq *ptq );
  /* return the next point which is removed from the queue */
#define ptq_empty(ptq)       ( ptq->next == ptq->size )
  /* Return TRUE if the queue is empty */
#define ptq_remove(ptq,p)  ( ptq->pt[p] = FALSE )
  /* Remove a single point from the queue */

/* End: pginit.h */
```

```
/* (C) 1998: Hans Georg Schaathun <georg@ii.uib.no> */
/* $Id: pginit.c,v 5.0 1998/12/20 00:24:52 georg Exp $ */

#include "common.h"

#include "pginit.h"
#include "points.h"

/* Constructor */
t_pg *pg_create ( t_pdim k, t_pdim q, t_s *ps ) {
  t_pg     *pg;  /* Return value */
  t_ptq    *ptq;
  t_s      *s; /**/
  t_pdim    m;
  t_ps      e, f, g, gm, h, i, j, ptno, elno, pt;
  t_ps      melno, helno;
  t_bas    *bas;

  pg = pg_alloc ( k, q );          /* Allocate memory */
  ptno = pg_elno ( q, k, 0 );      /* No. of points */

  /* Allocate auxilliary storage */
  s = ( NULL == ps ) ? s_init ( k, q ) : ps;
  ptq = ptq_alloc ( ptno );

  /* Make the Q(i) function table */
  for ( gm = pg->k, g = 0; g <= gm; ) {
    pg->ptno[g++] = pg_q ( pg->q, g );
  };

  /* Creating points */
  for ( i = 0; i < ptno; ++i ) {
    bas = bas_init ( 0 );
    *(bas->bas) = i;
    s->s[0][i] = msp_init ( bas, k, q );
  };

  /* Creating (m+1)-spaces, m >= 0 */
  for ( m = 0; m < k-1; ++m ) {
    elno = pg_elno ( q, k, m );              /* No. of m-spaces */
    j = 0;
    gm = pg_theta ( m, m+1, k, q );
```

```
      /* No. of m+1-dimensional superspaces for a given m-space */

    /* Creating superspaces of m-space #i */
    for ( i = 0; i < elno; ++i ) {
      ptq_reinit (ptq);
        /* ptq represents the set of all points */
      ptq_rmset ( ptq, s->s[m][i] );
        /* All points in m-space #i are removed from ptq */

      for ( g = 0; g < gm && pg->ps[m][0][i][g] != pg_noel; ) {
        ptq_rmset ( ptq, s->s[m+1][pg->ps[m][0][i][g++]] );
      }; /* Any point is removed if it is in a m+1-space already created */
         /* as a superspace of m-space #1; also g superspaces are        */
         /* already found, we will find the g^th one.                    */

      while ( pg_noel != ( pt = ptq_extract ( ptq ) ) ) {
        bas = bas_copy ( m+1, s->s[m][i]->bas );
        bas->bas[m+1] = pt;
        s->s[m+1][j] = msp_init ( bas, k, q );
          /* span an m+1-space by taking m+1 points spanning m-space #i */
          /* and one point from ptq.                                    */

        ptq_rmset ( ptq, s->s[m+1][j] );
          /* Remove from ptq the points in the newly created m+1-space. */

        for ( e=i+1; e < elno; ++e ) {
          if ( msp_subset ( s->s[m][e], s->s[m+1][j] ) ) {
            /* m-space #e is also in m+1-space #j */
            for ( f=0; pg->ps[m][0][e][f] != pg_noel &&
                       pg->ps[m][0][e][f] != j ; ++f ) {};
            /* f superspaces of m-space #e was found before m+1-space #j. */
            pg->ps[m][0][e][f] = j;
            /* This is the f^th superspace of m-space #e. */
          };
        };
        pg->ps[m][0][i][g++] = j++;
          /* Of course, m+1-space #j is a superspace of m-space #i; */
          /* by definition.                                          */
      }; /* [while] ptq is now the empty set */
      Invariant ( g == gm ) ;
        /* We are supposed to have found g superspaces of dimension */
        /* m+1 of m-space #i.                                        */
    };
    Invariant ( j == pg_elno ( q, k, m + 1 ) ) ;
      /* We have found j m+1-spaces —        */
      /* hopefully, that's the correct number. */
  };

  /* Find for each m-space, s, the l-spaces, l > m+1, having s as a subspace */
  for ( m = 0; m < k-2; ++m ) {
    melno = pg_elno ( q, k, m );
    for ( h = 0; h++ < k-2-m; ) {
      helno = pg_elno ( q, k, m+h+1 );
      gm = pg_theta ( m, m+h+1, k, q );
      for ( i = 0; i < melno; ++i ) {
        for ( g = j = 0; j < helno; ++j ) {
          if ( msp_subset ( s->s[m][i], s->s[m+h+1][j] ) ) {
            Invariant ( g < gm );
            pg->ps[m][h][i][g++] = j;
          };
        };
      };
```

```
    };
  };

  /* Deallocate auxilliary storage */
  if ( NULL == ps ) s_del ( s, k, q );
  ptq_del ( ptq );

  return ( pg );
}

/* File dumping */
char pg_dump ( FILE *stream, t_pg *pg ) {
  t_ps g, gm, h, hm, i, im, j, jm;
  fprintf ( stream, PG_FILEID );
  putc ( pg->k, stream );
  putc ( pg->q, stream );

  for ( gm = pg->k-1, g = 0; g < gm; ++g ) {
    im = pg_elno ( pg->q, pg->k, g );
    for ( hm = pg->k-g-1, h = 0; h < hm; ++h ) {
      jm = pg_theta ( g, g+h+1, pg->k, pg->q );
      for ( i = 0; i < im; ++i ) {
        for ( j = 0; j < jm; ++j ) putshort ( stream, pg->ps[g][h][i][j] );
      };
    };
  };
  return ( 0 );
}

/* End of pginit.c */
```

```
/* (C) 1998: Hans Georg Schaathun <georg@ii.uib.no> */
/* $Id: pgaux.c,v 5.0 1998/12/20 00:24:47 georg Exp $ */

#include "common.h"

#include "pginit.h"
#include "points.h"

/* Point queue */
t_ptq *ptq_alloc ( t_ps size ) {
  t_ptq *ptq;
  ptq = getmem ( sizeof ( t_ptq ) );
  ptq->size = size;
  ptq->pt   = getmem ( size + 1 );
  return ( ptq );
}

t_ptq *ptq_reinit ( t_ptq *ptq ) {
  static t_ps i;
  ptq->pt[ptq->size] = 0;
  ptq->next = 0;
  for ( i = 0; i < ptq->size; ++i ) ptq->pt[i] = TRUE;
  return ( ptq );
}

void   ptq_rmset   ( t_ptq *ptq, t_msp *sp ) {
  static int i ;
  Invariant ( sp != NULL );
  for ( i=0; i < ptq->size; ++i ) {
    if ( msp_in (i,sp) )
```

```
      ptq_remove ( ptq, i );
  };
}

t_ps ptq_extract ( t_ptq *ptq ) {
  while ( ( ! ptq_empty (ptq) ) && ( ptq->pt[ptq->next] == FALSE ) )
    ++(ptq->next);
  return ( ptq_empty(ptq) ? pg_noel : (ptq->next)++ );
}

/* End: pgaux.c */
```

Chapter 3

Value Assignments

3.1 The basic modules

```
/* (C) 1998: Hans Georg Schaathun <georg@ii.uib.no> */
/* $Id: val.h,v 5.0 1998/12/20 00:26:00 georg Exp $ */

#include "valaux.h"
#define NO_MORE_VAL     0
#define RET_VAL         1
#define val_noclass     255
#define val_novalue     255
#define V_ALLOCN        64*1024

/* t_v is a value assignment with a classification. */
typedef struct s_v {
  t_pg     *pg;    /* The projective geometry used */
  t_class  class;  /* The class number */
  t_value  *val;   /* The values for each point */
  char     ok;     /* true if the assignment is correctly classified */
  t_ds     ds;     /* The difference sequence */
} t_v;

/* t_val stores the value assigment of all subspaces. */
typedef struct s_val {
  t_pg      *pg;     /* The projective space */
  t_value   **val;   /* The values */
#ifdef ALLOW_FIX
  char      *fix;    /* Flag for fixed values */
#endif
  t_mval    **m;     /* The maximum values */
  t_v       *v;      /* Pointer to a classification of the value assignment */
  t_value   total;   /* The total value summed over all points */
  t_class   nclass;  /* The number of possible classes. */
} t_val;

/* Functions on t_val */

#define val_valprint(s,val) (fprintf( s, ";\n"), v_valprint( s,val->v ), 1)
  /* Print value assignment and classification */

t_val  *val_allocfor  ( t_pg  *pg, t_v *v );
  /* Make a t_val instance using the instance of t_v */

t_val  *val_clear    ( t_val *val, t_value initvalue );
```

```
  /* Initialize the value assignment with a specified initvalue. */

#define val_reinit(v)  ( val_clear ( v, val_novalue ) )
  /* Initialize: no points have any value yet */

t_val  *val_stepinit  ( t_val *val, t_value del ) ;
  /* Initialize the assignment for val_stepval(). */

void    val_del       ( t_val *v );
  /* Dispose the memory */

#define val_class(v)  val_getclass ( v, NULL )
t_class val_getclass ( const t_val *v, FILE *fs );
  /* Get the class number of the assignment. */

t_ds   *val_ds        ( t_val *v );
  /* Get the difference sequence from the assignment. */

char    val_stepval   ( t_val *v );
  /* Step to the next value assignment permitted. */

char    val_stepclass ( t_val *v, t_class c );
  /* Step to the next permitted value assignment of the given class. */

t_val  *val_load      ( FILE *stream, t_pg *pg ) ;
  /* Load from binary file, this is only used by vt_load(), and it **
  ** does not load the fix array member.                           */

int     val_dump      ( FILE *stream, t_val *v ) ;
  /* Write to binary file, this is only used by vt_dump(), and it **
  ** does not write the fix array member.                         */

t_v    *val_freev     ( t_val *val ) ;
  /* Make a copy of the t_v struct field */

char    val_assign    ( t_val *val, FILE *in ) ;
  /* Assign values read from file */

void    val_sumval    ( t_val *v );
  /* Compute the subspace values */

void    val_statusreport ( t_val *v );
  /* Write a status report */

void    val_fixdump ( FILE *stream, t_val *val );
void    val_fixload ( FILE *stream, t_val *val );
  /* Dump and load the fix array member, used in vt_load()/vt_dump(). */

#define val_classprint(val,f)     v_classprint(f,val->v)
  /* Print the class number and difference sequence */

#define val_alloc(p)     val_allocfor(p,NULL )
  /* Allocate memory for a t_val */

#define val_init(pg)     (val_reinit(val_alloc(pg)))
  /* Allocate memory and initialise a t_val */

#define val_nextv(v)     ( ( NO_MORE_VAL == val_stepval ( v ) ) ? NULL : v->v )
  /* Step to the next value assignment and return the t_v representation */

#define val_vprint(s,a) ( v_print ( s, a->v ) )
  /* Print the value assignment */
```

```
#define val_classify(p)  (void)( val_class(p), val_ds(p), v_cert( p->v ) )
  /* Compute the class number and difference sequence */

/* Macroes to set and fix values of points. */
#ifdef ALLOW_FIX
#  define val_set(vv,p,a)  ( vv->v->val[p] = a, vv->fix[p] = TRUE )
#  define val_setval(vv,p,a)  ( vv->v->val[p] = a )
#  define val_fix(v,c)      ( v->fix[c] )
#  define val_fixit(v,c)     ( v->fix[c] = TRUE )
#else
#  define mode_f           TRUE
#  define val_fix(v,c)    FALSE
#endif

/* Functions on t_v */
t_v     *v_alloc    ( t_pg *pg, t_value *val ) ;
  /* Constructor: allocate memory */
char     v_dump     ( FILE *stream, t_v *v ) ;
  /* Store the struct on a binary file */
t_v     *v_load     ( FILE *stream, t_pg *pg ) ;
  /* Load the struct from a binary file */
int      v_compare ( const void *a, const void *b ) ;
  /* Compare the to value assignments by their classification. **
  ** To be used with tsearch(3).                                 */
char     v_valprint ( FILE *stream, t_v *v ) ;
  /* Print value assignment in ASCII */
#define v_print(f,v)  ( v_classprint(f,v) && v_valprint(f,v) )
  /* Print class number, difference sequence and values in ASCII */
char     v_classprint ( FILE *out, t_v *v ) ;
  /* Print the classification (class number and DS) in ASCII */
#define v_ok(v)       ( v->ok )
  /* Return true if the classification is up to date. */
#define v_reinit(v)  ( v->ok = 0 )
  /* Claim that the classification is not up to date. */
#define v_cert(v)     ( v->ok = 1 )
  /* Claim that the classification is up to date. */

/* End of val.h */
```

```
/* (C) 1998: Hans Georg Schaathun <georg@ii.uib.no> */
/* $Id: val.c,v 5.0 1998/12/20 00:25:56 georg Exp $ */

#include "common.h"

#include "pg.h"
#include "val.h"
#include "aux.h"

static char val_auxchain ( const t_val *v, t_pdim m, t_ps ps );
  /* Recursive validation of the chain condition */

static char val_chained  ( const t_val *v );
  /* Check the chain condition */

static void val_nullsums ( t_val *v );
  /* Nullify subspace values */

/* Constructor */
t_val *val_allocfor ( t_pg *pg, t_v *v ) {
  t_pdim m;
```

```
  t_ps    elno;
  t_class i;
  t_val *val;

  val = getmem ( sizeof(t_val) );

  val->v       = ( v == NULL ? ( v_alloc( pg, NULL ) ) : v );
  val->pg      = pg;
  val->val     = getmem  ( (pg->k)*sizeof (void*) );
  val->m       = getmem  ( (pg->k)*sizeof (void*) );
  *(val->val) = val->v->val;
  val->m[0]    = mval_init ( pg_noel );
#ifdef ALLOW_FIX
  val->fix     = getmem ( pg_eln ( pg, 0 ) );
#endif
  for ( m=1; m < pg->k; ++m ) {
    val->m[m]    = mval_init ( pg_noel );
    elno         = pg_eln ( pg, m );
    val->val[m] = getmem ( elno*sizeof (t_value) );
  };

  for ( val->nclass = i = 0; i < pg->k; ++i ) val->nclass += i;
  val->nclass = 1<<val->nclass;

  return ( val );
} /* val_allocfor() */

/* Store on binary file. */
int val_dump ( FILE *stream, t_val *v ) {
  putc ( v->pg->k, stream );
  putc ( v->pg->q, stream );
  v_cert ( v->v );  /* Cheating: escape invariant test in v_dump (). */
  v_dump ( stream, v->v );
  return ( 1 );
} /* val_dump() */

/* Constructor: Load from binary file. */
t_val *val_load ( FILE *stream, t_pg *pg ) {
  t_val *val;
  t_pdim k, q;
  char InFile[PATH_NAME_L];
  FILE *InStream;
  k = getc ( stream ); q = getc ( stream );
  if ( NULL == pg ) {
    sprintf ( InFile, "%s/pg(%i,%i)", PG_DIR, k, q );
    InStream = fileopen ( InFile, "rb" );
    pg = pg_load ( InStream );
    fclose ( InStream );
  };
  val = val_allocfor ( pg, v_load ( stream, pg ) );
  v_reinit ( val->v );
  val_sumval ( val );      /* Sum the values of subspaces. */
  val_classify ( val );   /* Classify the initial value assignment */
  return ( val );
} /* val_load() */

/* Release the value assignment from auxilliary data structure (by copying). */
t_v *val_freev ( t_val *val ) {
  t_v *v;
  t_ps ptno, i;

  v = val->v;  /* Get a pointer to the t_v struct */
```

```
  ptno = pg_qpow ( val->pg, val->pg->k );

  /* Make a new t_v struct for val */
  v_reinit ( ( val->v = v_alloc ( val->pg, NULL ) ) );
  /* Copy data from v to val->v */
  for ( i=0; i<ptno; ++i ) {
    val->v->val[i] = v->val[i];
  };
  /* Incorporate the value assignment in the val array of t_val */
  val->val[0] = val->v->val;

  return ( v );
} /* val_freev() */

/* Reinitialize */
t_val *val_clear ( t_val *val, t_value initvalue ) {
  t_pdim m;
  t_ps   i, elno;
  val->total = 0;
  for ( m=0; m < val->pg->k-1; ++m ) {
    elno = pg_eln ( val->pg, m );
#ifdef ALLOW_FIX
    if ( 0 == m ) for ( i=0; i < elno; ++i ) val->fix[i] = FALSE;
#endif
    for ( i=0; i < elno; ++i ) val->val[m][i] = initvalue;
    mval_reinit ( val->m[m] );
  };
  v_reinit ( val->v );
  val_nullsums ( val );

  return ( val );
} /* val_clear */

t_val *val_stepinit ( t_val *val, t_value del ) {
  t_pdim m;
  t_ps   i, elno;

  val_clear ( val, 0 );

  *(val->v->val) = del;
  for ( m=0; m < val->pg->k; ) val->v->val[pg_qpow(val->pg,m++)] = 1;

  val_sumval ( val );      /* Sum the values of subspaces. */
  val_classify ( val );   /* Classify the initial value assignment */

  return ( val );
} /* val_stepinit() */

/* Destructor */
void   val_del  ( t_val *v ) {
  t_pdim m;
  if ( NULL == v ) return ;
  for ( m=0; m < v->pg->k-1; ++m ) {
    mval_del ( v->m[m] );
    free ( v->val[m] );
  };
  free ( v->val );
  free ( v->m );
  free ( v );
} /* val_del */

/* Classify (part 1) */
```

```c
t_class val_getclass ( const t_val *v, FILE *fs ) {
  t_cond cond = 0;
  t_pdim m,l;
  t_ps i, j, jmax;
  if ( v_ok ( v->v ) ) return ( v->v->class );
  v->v->class = 0;
  for ( m=0; m < v->pg->k - 1; ++m ) {
    for ( l = m+1; l < v->pg->k; ++l ) {
      jmax = pg_thet(v->pg,m,l);
      mval_reset(v->m[m]);
      while ( pg_noel != ( i = mval_next(v->m[m]) ) ) {
        for ( j = 0; j < jmax; ++j ) {
          if ( v->val[l][v->pg->ps[m][l-m-1][i][j]] == v->m[l]->v ) {
            if ( NULL != fs ) {
              fprintf ( fs, "%i-space #%i is in %i-space #%i.\n",
                        m, i, l, v->pg->ps[m][l-m-1][i][j] );
            } else {
              j = jmax;
            };
            mval_nomore(v->m[m]);
            v->v->class = ( v->v->class | ( 1 << cond ) );
          };
        };
      };
      ++cond;
    };
  };

  if ( ( v->v->class == v->nclass - 1 ) && val_chained ( v ) )
    v->v->class = v->nclass;
  return ( v->v->class );
} /* val_getclass */

/* Classify (part 2) */
t_ds *val_ds ( t_val *val ) {
  register t_pdim i;
  register t_value s = 0;

  if ( v_ok ( val->v ) ) return ( &val->v->ds );
  for ( i=0; i < val->pg->k; ++i ) {
    s += ( val->v->ds.delta[i] = ( val->m[i]->v - s ) );
  };
  val->v->ds.delta[val->pg->k] = val->total - s;
  return ( &val->v->ds );
} /* val_ds */

/* Step to the next possible value assignment. */
char val_stepval ( t_val *v ) {
  t_ps  maxc, lst;
  register t_ps c;
  t_value max;
  t_pdim  m;
  t_ps i, elno;

  c = lst = pg_qpow(v->pg,v->pg->k) - 1;
  max = v->v->val[maxc = pg_qpow(v->pg, m = v->pg->k-1)];

  while ( val_fix ( v, c ) || ( v->v->val[c]++ >= max ) ) {
    if ( --c == 0 ) return ( NO_MORE_VAL );
    if ( m == 0 ) {
      max = v->v->val[maxc = c - 1];
    /* Assume at least one point with lower value. */
```

```
      if ( ( c == v->pg->q ) && ( max == v->v->val[0] ) ) --max;
    } else if ( c == maxc ) {
      max = v->v->val[maxc = pg_qpow(v->pg, --m)];
    };
  };

  if ( c < lst ) {
  /* Recompute all subspace values. */
    maxc = pg_qpow(v->pg, ++m);

    while ( c++ < lst ) {
      if ( ! val_fix (v,c) ) {
        if ( c == maxc ) {
          maxc = pg_qpow(v->pg, ++m);
          v->v->val[c] = 1;
        } else {
          v->v->val[c] = 0;
        };
      };
    };

    v_reinit ( v->v );
    val_sumval ( v );       /* Sum values for subspaces */
  } else {
  /* Only one point has increased value; increase value of subspaces **
  ** including that point.                                           */
    for ( m=0; m < v->pg->k; ++m ) mval_reinit ( v->m[m] );

    ++v->total;
    for ( m = 0; m++ < v->pg->k-1; )
      for ( elno = pg_thet(v->pg,0,m), i=0; i < elno; ++i )
        ++v->val[m][v->pg->ps[0][m-1][c][i]];

    for ( m = 0; m < v->pg->k; ++m )
      for ( elno = pg_eln( v->pg, m ), i = 0; i < elno; ++i )
        mval_add ( v->m[m], i, v->val[m][i] );
  };
  val_classify ( v );

  return ( RET_VAL );
} /* val_stepval() */

/* Reset the value of each m-space, m > 0, to zero. */
void val_nullsums ( t_val *v ) {
  register t_pdim m = 0;
  register t_ps i, elno;

  v->total = 0;
  for ( m=0; m < v->pg->k; ++m ) mval_reinit ( v->m[m] );
  for ( m=0; m++ < v->pg->k-1; ) {
    elno = pg_eln ( v->pg, m );
    for ( i=0; i < elno; ) v->val[m][i++] = 0;
  };
  return ;
} /* val_nullsums */

/* Evaluate the weight of each m-space, m > 0 */
void val_sumval ( t_val *v ) {
  t_pdim m;
  t_ps   p, i, ptno, elno;

  val_nullsums ( v );  /* Reinitialize */
```

```
  for ( ptno = pg_qpow ( v->pg, v->pg->k ), p = 0; p < ptno; ++p ) {
    if ( 0 != v->val[0][p] ) {
      v->total += v->val[0][p];
      for ( m = 0; m++ < v->pg->k-1; ) {
        for ( elno = pg_thet(v->pg,0,m), i=0; i < elno; ++i ) {
          v->val[m][v->pg->ps[0][m-1][p][i]] += v->val[0][p];
        };
      };
    };
  };

  for ( m = 0; m < v->pg->k; ++m ) {
    for ( elno = pg_eln( v->pg, m ), i = 0; i < elno; ++i ) {
      mval_add ( v->m[m], i, v->val[m][i] );
    };
  };

  return ;
} /* val_sumval */

/* Check for the chain condition. */
char val_chained ( const t_val *v ) {
  t_pdim m=0;
  t_ps i;
  mval_reset( v->m[m] );
  while ( pg_noel != ( i = mval_next(v->m[0]) ) ) {
    if ( val_auxchain ( v, 0, i ) ) return ( TRUE );
  };
  return ( FALSE );
} /* val_chained() */

/* Auxilliary function for val_chained(). */
char val_auxchain ( const t_val *v, t_pdim m, t_ps ps ) {
  t_ps j, jmax;
  if ( m == v->pg->k - 1 ) return ( TRUE );
  jmax = pg_thet ( v->pg, m, m+1 );
  for ( j = 0; j < jmax; ++j ) {
    if ( ( v->val[m+1][v->pg->ps[m][0][ps][j]] == v->m[m+1]->v ) &&
         val_auxchain( v, m+1, v->pg->ps[m][0][ps][j] ) ) return ( TRUE );
  };
  return ( FALSE );
} /* val_auxchain() */

/* Step to next value assignment in the specified class. */
char val_stepclass ( t_val *v, t_class c ) {
  char r;
  r = val_stepval ( v );
  while ( ( RET_VAL == r ) && ( val_class ( v ) != c ) ) r = val_stepval ( v );
  return ( r );
} /* val_stepclass() */

void     val_statusreport ( t_val *v ) {
  static long val_repno = 0; /* Counter for the reports. */
  FILE *stream;
  extern char *StatusFile;
  /* We use a global variable for the file name, so that this can  **
  ** be used from a signal handler(SIG_USR1).                       */

  stream = fileopen ( StatusFile, (0 == val_repno++ ? "w" : "a") );
  fprintf ( stream, "Status Report no.: %i\n", val_repno );
  val_vprint ( stream, v );
```

```
  fclose ( stream );
} /* val_statusreport() */

void    val_fixdump ( FILE *stream, t_val *val ) {
  t_ps ptno, i;
  ptno = pg_qpow ( val->pg, val->pg->k );
  for ( i=0; i < ptno; ++i ) {
#ifdef ALLOW_FIX
    val->fix[i] = getc ( stream );
#endif
  };
  return ;
} /* val_fixdump() */

void    val_fixload ( FILE *stream, t_val *val ) {
  t_ps ptno, i;
  ptno = pg_qpow ( val->pg, val->pg->k );
  for ( i=0; i < ptno; ++i ) {
#ifdef ALLOW_FIX
    putc ( val->fix[i], stream );
#else
    putc ( 0, stream );
#endif
  };
  return ;
} /* val_fixload() */

/* End of val.c */
```

```
/* (C) 1998: Hans Georg Schaathun <georg@ii.uib.no> */
/* $Id: valv.c,v 5.0 1998/12/20 00:26:20 georg Exp $ */

#include "common.h"

#include "pg.h"
#include "val.h"
#include "aux.h"

unsigned long v_allocn = V_ALLOCN;

t_v *v_alloc ( t_pg *pg, t_value *val ) {
  /* Warning: This function does not enable deallocation of memory. */
  static void *v_first, *v_next, *v_last;
  static int v_size = -1;
  t_v *v;

  /* If called with pg == NULL, we reinitialize */
  if ( NULL == pg ) { v_size = -1; return ( NULL ); };

  /* If v_size hasn't been set we have to set it now. */
  if ( v_size < 0 ) {
    int tmp;
    v_size = sizeof ( t_v ) + pg->ptno[pg->k]*sizeof ( t_value );
    if ( tmp = ( v_size % sizeof ( void* ) ) ) v_size += sizeof (void*) - tmp;
    v_first = NULL;
  };

  if ( ( NULL == v_first ) ||
       (long)( v_next = (char*) v_next + v_size ) >= (long)v_last ) {
    v_first = v_next = getmem ( v_size*v_allocn );
    v_last = (char*) v_first + v_size*v_allocn;
```

```
  };
  v = (t_v*) v_next;
  v->pg = pg;
  v->val = ( ( NULL == val ) ?
             (t_value*) ( (char*) v_next + sizeof (t_v) ) : val );
  return ( v );
}

char v_classprint ( FILE *out, t_v *v ) {
  Invariant ( v_ok ( v ) );
  fprintf ( out, "#%i ", v->class );
  ds_print ( out, &v->ds, v->pg->k );
  putc ( '\n', out );

  return ( v_ok ( v ) );
}

char    v_valprint   ( FILE *stream, t_v *v ) {
  t_ps i=0, elno;

  elno = pg_qpow ( v->pg, v->pg->k );
  while ( i < elno ) {
    fprintf ( stream, "%i", v->val[i++] );
    putc ( ( ( 0 == ( i % 10 ) ) ? '\n' : '\t' ), stream );
  };
  if ( 0 != ( i % 10 ) ) putc ( '\n', stream );

  return ( v_ok ( v ) );
}

char    v_dump   ( FILE *stream, t_v *v ) {
  int i, ptno;
  Invariant ( v_ok ( v ) );
  putc ( v->class, stream );
  for ( i=0; i <= v->pg->k; ++i ) putshort ( stream, v->ds.delta[i] );
  ptno = pg_qpow ( v->pg, v->pg->k );
  for ( i=0; i < ptno; ++i ) putc ( v->val[i], stream );
  return ( 1 );
}

t_v *v_load ( FILE *stream, t_pg *pg ) {
  t_v *v;
  t_ps ptno;
  int i;

  v = v_alloc ( pg, NULL );
  if ( (t_class)EOF == ( v->class = getc ( stream ) ) ) return ( NULL );

  for ( i=0; i <= pg->k; ++i ) v->ds.delta[i] = getshort ( stream );
  ptno = pg_qpow ( pg, pg->k );
  for ( i=0; i < ptno; ++i ) v->val[i] = getc ( stream );
  v_cert ( v );

  return ( v );
}

int v_compare ( const void *a, const void *b ) {
  t_pdim i;
  t_v *v, *w;

  v = ((t_v*) a); w = ((t_v*) b);
```

```
  Invariant ( v_ok ( v ) && v_ok ( w ) );
  Invariant ( v->pg->k == w->pg->k );

  if ( v->class < w->class ) return ( -1 );
  if ( v->class > w->class ) return (  1 );
  for ( i=0; i <= v->pg->k; ++i ) {
    if ( v->ds.delta[i] < w->ds.delta[i] ) return ( -1 );
    if ( v->ds.delta[i] > w->ds.delta[i] ) return (  1 );
  };
  return ( 0 );
}

/* End of valv.c */
```

```
/* (C) 1998: Hans Georg Schaathun <georg@ii.uib.no> */
/* $Id: valaux.h,v 5.0 1998/12/20 00:26:11 georg Exp $ */

#include "stack.h"

/* Type defs */
typedef unsigned short t_value;
  /* Warning: t_value is frequently handled as char. */
typedef unsigned short t_class;
  /* Warning: t_class may be handled as char. */
typedef unsigned char t_cond;

/* Structs */
typedef struct s_mval {
  t_stack *s; /* A set of points having this value. */
  t_value  v;  /* The value.                        */
} t_mval;

/* t_ds is a difference sequence. */
typedef struct s_ds {
  t_value  delta[MAX_DIM];
} t_ds;

/* Functions on t_mval */
t_mval *mval_alloc ( t_ps size );
  /* Allocate memory for a t_mval. */
void    mval_print ( FILE *stream, t_mval *mv ) ;
  /* Print the value, and the list of points on the given stream. */
#define  mval_del(mv)      ( stk_del(mv->s), free(mv) )
  /* Destructor: dispose the memory */
#define  mval_reinit(mv)  ( stk_reinit ( mv->s ), mv->v = 0, mv )
  /* Reinitialize: the value is zero, with no points. */
#define  mval_add(mv,pt,val) \
        ( ( val > mv->v ) ? \
          ( mv->v = val, stk_replace ( mv->s, pt ) ) : \
          ( ( val == mv->v ) ? stk_push ( mv->s, pt ) : 0 ) )
  /* If the point has greater value, let it replace all existing points,    **
  ** and update the value, otherwise add the point if it has the same value */
#define mval_init(s)    ( mval_reinit ( mval_alloc ( s ) ) )
  /* Constructor: Create and initialize the t_mval */
#define mval_reset(v)   ( stk_reset(v->s) )
  /* Reset the stack.  Next time we inspect an point, we get the top-most. */
#define mval_next(v)    ( stk_next(v->s) )
  /* Return the next point, without changing the stack. */
#define mval_nomore(v)  ( v->s->current = 0 )
  /* TRUE if there are no more points to inspect with mval_next */
```

```
/* Functions on t_ds */
#define ds_alloc(k)    ( (t_ds*) getmem ( sizeof (t_ds) ) )
  /* Allocate memory for the t_ds. */
char     ds_print   ( FILE *stream, t_ds *ds, t_pdim k );
  /* Print the difference sequence on the given stream. */

/* End of valaux.h */
```

```
/* (C) 1998: Hans Georg Schaathun <georg@ii.uib.no> */
/* $Id: valaux.c,v 5.0 1998/12/20 00:26:06 georg Exp $ */

#include "common.h"
#include "pg.h"
#include "valaux.h"

/* mval struct */
t_mval *mval_alloc  ( t_ps   size ) {
  t_mval *mv;
  mv = getmem ( sizeof ( t_mval ) ) ;
  mv->s = stk_alloc ( size );
  return ( mv );
}

void mval_print ( FILE *stream, t_mval *mv ) {
  char c = 0;
  stk_reset ( mv->s );
  fprintf ( stream, "[%i]:", mv->v );
  while ( stk_more ( mv->s ) ) {
    fprintf ( stream, ( 8 == c++ ? "\n\t%i" : "\t%i" ), stk_next ( mv->s ) );
  };
  putc ( '\n', stream );
  return ;
}

/* ds struct */
char ds_print ( FILE *stream, t_ds *ds, t_pdim k ) {
  t_pdim i;
  Invariant ( k > 0 );
  fprintf ( stream, "( %i", ds->delta[i=0] );
  while ( i++ < k ) fprintf ( stream, ", %i", ds->delta[i] );
  fprintf ( stream, " )" );
  return ( 1 );
}

/* End of valaux.c */
```

```
/* (C) 1998: Hans Georg Schaathun <georg@ii.uib.no> */
/* $Id: vtree.h,v 5.0 1998/12/20 00:26:34 georg Exp $ */

extern FILE *vt_stream; /* Temporary pointer for macroes. */

typedef struct s_vt {
  void  *root;  /* The root of the binary search tree */
  t_val *val;   /* Related data structure for the current value assignment. */
} t_vt;

t_vt *vt_finfo ( FILE *stream ) ;
  /* Print the dimension of the Projectiv Space used in the file stream. */

void  vt_insert ( t_val *val, t_vt *vt ) ;
```

```
  /* Insert a value assignment in the tree. */

#define vt_backup(f,t) ( vt_dump(vt_stream = fileopen (f,"wb"), t ), \
                         fclose (vt_stream) )
void  vt_dump ( FILE *stream, t_vt *vt ) ;
  /* Store the valueassignments in binary format. */

void  vt_printbyclass ( char *FileName, t_vt *vt ) ;
  /* Print the valueassignments in ascii format sorted by class. */

t_vt *vt_readbackup ( const char *FileName, const t_pg *pg ) ;
t_vt *vt_load ( FILE *stream, t_vt *vt, t_pg *pg ) ;
  /* Load a tree of value assignmnents from the given file. */

void  vt_del ( t_vt *vt ) ;
  /* Elliminate the tree. */

t_vt *vt_init ( t_val *val ) ;
  /* Create a new and empty tree for value assignments. */

/* End of vtree.h */
```

```
/* (C) 1998: Hans Georg Schaathun <georg@ii.uib.no> */
/* $Id: vtree.c,v 5.0 1998/12/20 00:26:31 georg Exp $ */

/* Object file for storing different value assignments up to equivalence **
** in a binary search tree.                                              */

#include "common.h"

#include <search.h>

#include "pg.h"
#include "val.h"
#include "vtree.h"
#include "aux.h"

/* Global variables */
FILE *vt_stream;
static char *vt_file;
static t_class cc;

/* Static function declarations */
static void vt_walkbackup        ( const void *val, VISIT v, int n ) ;
  /* Make an binary dump of the node, used by vt_dump(). */
static void vt_walkprintbyclass ( const void *val, VISIT v, int n ) ;
  /* Make an ASCII print of the node, used by vt_printbyclass(). */

/* Functions */
void vt_insert ( t_val *val, t_vt *vt ) {
  if ( ( NULL != vt->root ) &&
       ( NULL != tfind ( val->v, &(vt->root), &v_compare ) ) ) return ;
  tsearch ( val->v, &(vt->root), &v_compare );
  Invariant ( v_ok ( val->v ) );
  val_freev ( val );
  return ;
}

t_vt *vt_init ( t_val *val ) {
  t_vt *vt;
  vt = getmem ( sizeof (t_vt) );
```

```
  vt->root = NULL;
  vt->val  = val;
  return ( vt );
}

void  vt_printbyclass ( char *FileName, t_vt *vt ) {
  extern FILE *vt_stream;
  extern char *vt_file;
  vt_file = FileName;
  twalk ( vt->root, &vt_walkprintbyclass );
  fclose ( vt_stream );
  return ;
}

void  vt_walkprintbyclass ( const void *val, VISIT v, int n ) {
/* Auxiliary function for vt_printbyclass(). */
  extern FILE *vt_stream;
  extern char *vt_file;
  extern t_class cc;
  static char  FileName[FILE_NAME_L];
  t_v *vv;

  vv = *(t_v**) val;
  Invariant ( v_ok ( vv ) ) ;

  if ( ( v == preorder ) || ( v == endorder ) ) return ;
  if ( ( vv->class > cc ) || ( vt_stream == NULL ) ) {
    cc = vv->class;
    fclose ( vt_stream );
    sprintf ( FileName, "%s:%i", vt_file, cc );
    vt_stream = fileopen ( FileName, "w" );
    fprintf ( vt_stream, "PG(%i,%i): d = %i, class %i:\n",
              vv->pg->k, vv->pg->q, vv->ds.delta[0], cc );
  };
  v_print ( vt_stream, vv );
  return ;
}

t_vt *vt_finfo ( FILE *stream ) {
  int i = 0;
  t_pdim k, q;
  while ( 0 != VT_FILEID[i] ) {
    if ( VT_FILEID[i++] != getc ( stream ) ) ErrEx ( ERR_FFORMAT );
  };
  k = getc ( stream ); q = getc ( stream );
  printf ( "k = %i, q = %i\n", k, q );
}

t_vt *vt_load ( FILE *stream, t_vt *vt, t_pg *pg ) {
  t_v *v;
  int i = 0;
  char fver = 2, c;
  if ( vt == NULL ) vt = vt_init ( NULL );
  while ( 0 != VT_FILEID[i] ) {
    if ( VT_FILEID[i++] != ( c = getc ( stream ) ) )
      if ( isdigit ( c ) ) fver = c - '0';
      else ErrEx ( ERR_FFORMAT );
  };
  if ( ( fver < 1 ) || ( fver > 2 ) ) ErrEx ( ERR_FFORMAT );
  vt->val = val_load ( stream, pg );
  if ( 2 == fver ) val_fixload ( stream, vt->val );
  while ( 0 == feof ( stream ) &&
```

```
        NULL != ( v = v_load ( stream, vt->val->pg ) ) ) {
    tsearch ( v, &(vt->root), &v_compare );
  };
  return ( vt );
}

void vt_dump ( FILE *stream, t_vt *vt ) {
  extern FILE *vt_stream;
  vt_stream = stream;
  fprintf ( vt_stream, VT_FILEID );
  val_dump ( stream, vt->val );
  val_fixdump ( stream, vt->val );
  twalk ( vt->root, &vt_walkbackup );
  return ;
}

t_vt *vt_readbackup ( const char *FileName, const t_pg *pg ) {
  extern FILE *vt_stream;
  t_vt *vt;
  vt_stream = fileopen ( FileName, "w" );
  ErrEx ( ERR_IMP );
  fclose ( vt_stream );
  return ( vt );
}

void  vt_walkbackup ( const void *val, VISIT v, int n ) {
/* Auxiliary function for vt_backup. */
  extern char *vt_file;
  v_dump ( vt_stream, *(t_v**) val );
  return ;
}

void vt_del ( t_vt *vt ) {
  /* Only one tree is allocated, so we don't bother to deallocate it. */
  return ;
}

/* End: vtree.c */
```

```
/* (C) 1998: Hans Georg Schaathun <georg@ii.uib.no> */
/* $Id: stack.h,v 5.0 1998/12/20 00:25:48 georg Exp $ */

/* This defines a quasi-stack to store a set of points. */
/* In addition to ordinary stack operations, one can travers the  **
** stack from top to bottom, without popping elements.           */
/* One can also discard all elements and insert a single new one. */

typedef struct s_stack {
  unsigned char size,    /* Current size */
                msize,   /* Maximum size */
                current; /* Current element with respect to traversing */
  t_ps *ps;              /* Array for the elements */
} t_stack;

/* Allocation and deallocation. */
t_stack *stk_init   ( unsigned char size  );
  /* Constructor: allocate memory, for a stack of maximum size size. */
void     stk_del    ( t_stack      *stack );
  /* Destructor: deallocate memory */
#define stk_alloc(s)    stk_init(s)
  /* allocate memory */
```

```
#define stk_reinit(s)    ( s->size = 0 )
  /* empty the stacke */

/* Basic stack operations: pop and push. */
#define stk_pop(stk)     ( stk_empty(stk) ? pg_noel : stk->ps[--(stk->size)] )
#define stk_push(stk,c)  ( stk_full(stk)? 0: ( stk->ps[(stk->size)++] = c, 1 ) )

/* Observators */
#define stk_empty(stk)   ( 0 == stk->size )
  /* True if the stack is empty. */
#define stk_full(stk)    ( stk->size >= stk->msize )
  /* True if the stack is full. */
#define stk_more(stk)    ( 0 != stk->current )
  /* True if stk_next can return an element. */

/* Traversing */
#define stk_reset(s)     ( s->current = s->size )
  /* Reset the stack.  Next time we inspect an element, we get the top-most. */
#define stk_next(stk)    ( stk_more(stk) ? stk->ps[--(stk->current)] : pg_noel )
  /* Return the next element, without changing the stack. */

/**/
#define stk_replace(stk,c) ( stk->size = 1, stk->ps[0] = c, 1 )
  /* Replace all elements by a single new element. */

/* end of stack.h */
```

```
/* (C) 1998: Hans Georg Schaathun <georg@ii.uib.no> */
/* $Id: stack.c,v 5.0 1998/12/20 00:25:43 georg Exp $ */

#include "common.h"
#include "pg.h"
#include "stack.h"
#include "aux.h"

t_stack *stk_init    ( unsigned char size   ) {
  t_stack *stk;
  stk = getmem ( sizeof (t_stack) );
  stk->size = 0;
  stk->msize = size;
  stk->ps = getmem ( size*sizeof (t_ps) );
  return ( stk );
}

void     stk_del     ( t_stack      *stack ) {
  free ( stack->ps );
  free ( stack );
}

/* End of stack.c */
```

```
/* (C) 1998: Hans Georg Schaathun <georg@ii.uib.no> */
/* $Id: valex.l,v 5.0 1998/12/20 00:26:13 georg Exp $ */

/* Parser for value assignments */

%option prefix="yyval_"
%option noyywrap
%option stack
%option 7bit
```

```
%{

/* Include */
#include "common.h"
#include "pg.h"
#include "val.h"

/* Internal variables */
static t_val *val;

%}

  /* Character classes */

COMMENT #
HEAD    ;
CHAR    [a-zA-Z0-9_]
NAME    [a-zA-Z]
LETTER  [a-zA-Z_]
WORD    {LETTER}{CHAR}*
DIGIT   [0-9]
NUMBER  [0-9]+
WHITE   [ \t\n]
FIX     "*"

  /* Keywords */

C_FIX      @fix
C_VERBOSE  @verbose
C_END      @end
C_ENDFILE  @endfile

%x assign
%s ex_val

%%

%{
  static t_ps pt;
  static char verbose = TRUE, fixall, *asp;
%}

{COMMENT}.*\n        { /* Ignore comments */ }
{WHITE}+             { /* Ignore whitespace */ }

  /* The header marks the start of the assignment. */
<INITIAL>{HEAD}.*\n {
  if ( verbose ) printf ( "New value assignment:\n%s", yytext );
  BEGIN ( ex_val ) ;
  fixall = FALSE; pt = 0; }

{C_VERBOSE}=          { asp = &verbose; yy_push_state ( assign ); }
<ex_val>{C_FIX}=      { asp = &fixall;  yy_push_state ( assign ); }
<assign>{NUMBER}      { *asp = atoi ( yytext ); yy_pop_state (); }

  /* Setting the point number for the next assignment */
<ex_val>{NUMBER}/:  { pt = atoi ( yytext ); input (); }

  /* Assigning value */
<ex_val>{NUMBER}/({WHITE}|{FIX}) {
  if ( fixall ) { val_set ( val, pt, atoi ( yytext ) ); ++pt; }
```

```
  else val_setval ( val, pt++, atoi ( yytext ) ); }

  /* Assigning no value */
<ex_val>-/({WHITE}|{FIX}) { val_setval ( val, pt++, val_novalue ); }

  /* Fix value */
<ex_val>{FIX}        { val_fixit ( val, pt - 1 ); }

  /* A new head indicates the end of the current set of assignments. */
<ex_val>{HEAD}       {
  if ( verbose ) printf ( "End of assignment.\n" );
  unput ( ';' ); BEGIN ( INITIAL ); return ( 0 ); }
  /* Explicit end of current value assignment. */
<ex_val>{C_END}      {
  if ( verbose ) printf ( "End of assignment.\n" );
  BEGIN ( INITIAL ); return ( 0 ); }

  /* End of input */
<ex_val><<EOF>>      { BEGIN ( INITIAL ); return ( 0 ); }
{C_ENDFILE}          { BEGIN ( INITIAL ); return ( 0 ); }
<INITIAL><<EOF>>     { return ( EOF ); }
<INITIAL>{C_ENDFILE} { return ( EOF ); }

  /* Don't accept garble within value assignments. */
<ex_val>.            { fprintf ( stderr,
                     "Syntax error reading value assignments:\n%s\n",
                     yytext ); }

<INITIAL>.           { /* Do ignore garble outside value assignments. */ }

%%

  /* val_assign() takes parameters to initialize global variables before **
  ** running yy_lex().                                                   */
char val_assign ( t_val *v, FILE *fs ) {
  yyin = fs;
  val = v;
  return ( yylex () );
}

/* End of lex.l */
```

3.2 Complex modules

```
/* (C) 1998: Hans Georg Schaathun <georg@ii.uib.no> */
/* $Id: pgval.h,v 5.0 1998/12/20 00:25:13 georg Exp $ */

#include "pginit.h"

#define FIX_NOTE         " *"
#define DIGIT            "0123456789-"
#define pv_pprint(fs,p)  fprintf ( fs, "%4d:%c%c", p, \
          DIGIT[val_novalue == val->v->val[p] ? 10 : val->v->val[p]], \
          FIX_NOTE[val->fix[p]] )
  /* Print a string representing a point and its value. */

void pv_init ( t_pdim k, t_pdim q ) ;
  /* Reinitilise global variables. */

void pv_sprint ( FILE *stream, t_pdim dim, t_ps ps ) ;
```

```
  /* Print a subspace and the values for each point */
void pv_printall ( FILE *s, t_pdim d ) ;
  /* Print all d-spaces and the values for each point */
void pv_print ( FILE *stream ) ;
  /* Print values for each point (valex format). */
int pv_fixcount ( ) ;
  /* Count fixed values. */
char pv_ssprint ( FILE *stream, t_pdim dim1, t_pdim dim2, t_ps ps ) ;
  /* Print all dim1-spaces being sub- or superset of dim2-space ps,
  ** with values.                                                  */

/* end of pgval.h */
```

```
/* (C) 1998: Hans Georg Schaathun <georg@ii.uib.no> */
/* $Id: pgval.c,v 5.0 1998/12/20 00:25:11 georg Exp $ */

#include "common.h"
#include "pgval.h"
#include "val.h"

static char pv_supprint ( FILE *stream, t_pdim dim, t_pdim subdim, t_ps ps ) ;
static char pv_subprint ( FILE *stream, t_pdim dim, t_pdim subdim, t_ps ps ) ;

t_val *val = NULL;
t_pg  *pg = NULL;
t_pdim  pg_dim=4, pg_order=2;     /* PG parameters */

void pv_printall ( FILE *s, t_pdim d ) {
  int elno, i;
  extern t_pg  *pg;
  elno = pg_eln ( pg, d );
  for ( i = 0; i < elno; ++i ) pv_sprint ( s, d, i );
}

void pv_sprint ( FILE *stream, t_pdim dim, t_ps ps ) {
  int elno, i, tab;
  extern t_val *val;
  extern t_pg  *pg;
  extern t_s   *sp;
  extern t_pdim pg_dim;
  char *msp;

  if ( dim >= pg_dim ) { return; }; /* Sanity check */

  /* Print a header with number and value for the space. */
  fprintf ( stream, "%i-space[%4i] (%i):", dim, ps, val->val[dim][ps] );
  tab = 2; /* This is the current tab position. */

  /* The boolean array indicating the points in the subspace */
  msp = sp->s[dim][ps]->pt;
  /* The number of points in the subspace. */
  elno = pg_qpow ( pg, pg->k );
  /* For each point */
  for ( i = 0; i < elno; ++i ) {
    /* If it is in the subspace */
    if ( msp[i] == TRUE ) {
      putc ( '\t', stream );
      /* Print its number and value */
      pv_pprint ( stream, i );
      /* On the last tab position, put a line feed. */
      if ( 0 == ( ++tab % 9 ) ) putc ( '\n', stream );
```

```
    }
  };
  /* If we're in the midle of a line, put a line feed. */
  if ( ( tab % 9 ) != 0 ) putc ( '\n', stream );
  return ;
}

void pv_print ( FILE *stream ) {
  extern t_pg *pg;
  extern t_val *val;
  t_ps i=0, elno;

  fprintf ( stream, ";\n" ); /* Print header */
  elno = pg_qpow ( pg, pg->k );
  while ( i < elno ) {
    pv_pprint ( stream, i ); ++i;
    putc ( ( ( 0 == ( i % 10 ) ) ? '\n' : '\t' ), stream );
  };
  if ( 0 != ( i % 10 ) ) putc ( '\n', stream );
  return ;
}

void pv_init ( t_pdim k, t_pdim q ) {
  extern t_val *val;
  extern t_pg  *pg;
  extern t_s   *sp;
  extern t_pdim pg_dim, pg_order;
  if ( NULL != sp ) s_del ( sp, pg_dim, pg_order ) ;
  if ( NULL != pg ) pg_del ( pg ) ;
  if ( NULL != val ) val_del ( val ) ;
  v_alloc ( NULL, 0 ); /* Reinitalize for a new geometry. */
  pg_dim = k;
  pg_order = q;
  sp = s_init ( pg_dim, pg_order );
  pg = pg_create ( pg_dim, pg_order, sp );
  val = val_init ( pg );
}

int pv_fixcount ( ) {
  extern t_val *val;
  extern t_pg  *pg;
  int elno, i = 0, r = 0;
  elno = pg_qpow ( pg, pg->k );
  while ( i < elno ) {
    if ( val->fix[i++] ) ++r;
  };
  return ( r );
}

char pv_ssprint ( FILE *stream, t_pdim dim1, t_pdim dim2, t_ps ps ) {
  return (
    ( dim1 < dim2 ? pv_subprint : pv_supprint )( stream, dim2, dim1, ps )
  );
}

char pv_supprint ( FILE *stream, t_pdim dim, t_pdim subdim, t_ps ps ) {
  return ( ENO_IMP ) ;
}

char pv_subprint ( FILE *stream, t_pdim dim, t_pdim subdim, t_ps ps ) {
  int h, hm, j, jm, c;
  jm = pg_elno ( pg->q, pg->k, dim );
```

```
  hm = pg_theta ( subdim, dim, pg->k, pg->q );
  fprintf ( stream, "%i-spaces in %i-space #%i:\n", subdim, dim, ps );
  for ( c = j = 0; j < jm; ++j ) {
    for ( h = 0; h < hm; ++h ) {
      if ( pg->ps[subdim][dim-subdim-1][j][h] == ps ) {
        pv_sprint ( stream, subdim, j );
      };
    };
  };
  putc ( '\n', stream );
}

/* end of pgval.c */
```

```
/* (C) 1998: Hans Georg Schaathun <georg@ii.uib.no> */
/* $Id: valmsp.h,v 5.0 1998/12/20 00:26:18 georg Exp $ */

#include "val.h"

#define NO_FIXCHANGE   -1
#define NO_CHANGE       0
#define IGN_FIX         1
#define IGN_VAL         2
#define DO_CHANGE       (IGN_FIX|IGN_VAL)

char val_spaceval ( t_pdim dim, t_ps ps, signed short v,
                    char change, char fix ) ;
  /* Assign the same value to all points in a dim-space.  If change is  */
  /* NO_CHANGE only points with current value val_novalue is changed.   */
  /* If v is negative, the value isn't changed, but values may still be */
  /* fixed.                                                              */
#define val_spacefix(d,p)     val_spaceval ( d, p, -1, DO_CHANGE, TRUE );
#define val_spaceunfix(d,p)   val_spaceval ( d, p, -1, DO_CHANGE, FALSE );

t_ps val_incprobe ( FILE *fs, char inc ) ;
  /* Try and increase the value of all non-fixed points, without **
  ** changing the class.                                          */

/* End of valmsp.h */
```

```
/* (C) 1998: Hans Georg Schaathun <georg@ii.uib.no> */
/* $Id: valmsp.c,v 5.0 1998/12/20 00:26:15 georg Exp $ */

#include "common.h"
#include "pginit.h"
#include "valmsp.h"

char val_spaceval ( t_pdim dim, t_ps ps, signed short v,
                    char change, char fix ) {
  int elno, i;
  extern t_val *val;
  extern t_pg   *pg;
  extern t_s    *sp;
  extern t_pdim pg_dim;
  char *msp;

  if ( pg_dim != dim ) msp = sp->s[dim][ps]->pt;
  elno = pg_qpow ( pg, pg_dim );
  /* We loop to check all possible points. */
  for ( i = 0; i < elno; ++i ) {
```

```
    if ( ( ( pg_dim == dim ) || ( msp[i] == TRUE ) ) &&
         ( ( val->v->val[i] == val_novalue ) || (change&IGN_VAL) ) &&
         ( ( val->fix[i] == FALSE ) || (change&IGN_FIX) ) ) {
      /* If expression:
      ** line 1) if the point is in the given space
      ** line 2) if it has a value, or we may change an existing values
      ** line 3) if the value isn't fixed or we are allowed to ignore it */
      if ( v >= 0 ) val->v->val[i] = v;
      if ( NO_FIXCHANGE != fix ) val->fix[i] = fix;
    }
  }
}

/* Try to increase values without changing class */
t_ps val_incprobe ( FILE *fs, char inc ) {
  int class, fixit = 0;
  t_ps ptno, i, r = 0;
  extern t_val *val;
  extern t_pg   *pg;
  extern t_pdim pg_dim;

  /* If inc is 0, we use an increment of 1, and fix values we cannot
  ** increase. */
  if ( 0 == inc ) { ++fixit; ++inc; };

  /* Classify the assignment. */
  val_sumval ( val );
  v_reinit ( val->v );
  class = val_class ( val );

  ptno = pg_qpow ( pg, pg_dim );
  /* For each point */
  for ( i = 0; i < ptno; ++i ) {
    /* If the value isn't fixed */
    if ( val->fix[i] == FALSE ) {
      val->v->val[i] += inc;  /* Try an increament */
      val_sumval ( val );
      if ( class != val_class ( val ) ) {
        /* If the class has changed, undo the change in the value */
        val->v->val[i] -= inc;
        if ( fixit ) val->fix[i] = TRUE;
      } else {
        ++r;
        if ( NULL != fs )
          fprintf ( fs, "Increasing value of point #%i.\n", i );
      };
    };
  };
  return ( r ); /* Return the number of increased values. */
}


/* End of valmsp.c */
```

```
/* (C) 1998: Hans Georg Schaathun <georg@ii.uib.no> */
/* $Id: nametab.h,v 5.0 1998/12/20 00:24:33 georg Exp $ */

#include "common.h"
#include "pginit.h"

#define POINT_NO 26
```

```
#define SPACE_NO 26

typedef struct s_ntab {
  t_ps    point[POINT_NO];
  t_pdim sdim[SPACE_NO];
  t_ps    space[SPACE_NO];
} t_ntab;

void nt_init () ;
  /* Initialise name tables. */
int nt_lookup ( char name, int out[] ) ;
  /* Resolve the symbolic name, and insert the result in out. **
  ** Return the number of integers inserted.                  **
  ** out[] is usually a argument vector in the lex module      */
char nt_sdef ( char name, t_pdim m, t_ps p ) ;
  /* Define a symbolic name for a subspace. */
char nt_pdef ( char name, t_ps p ) ;
  /* Define a symbolic name for a point. */
char nt_line ( char name, t_ps a, t_ps b ) ;
  /* Define a symbolic names for a points on a line. */

/* End of nametab.h */
```

```
/* (C) 1998: Hans Georg Schaathun <georg@ii.uib.no> */
/* $Id: nametab.c,v 5.0 1998/12/20 00:24:32 georg Exp $ */

#include "nametab.h"

static t_ntab ntab;

void nt_init () {
  int i;
  extern t_pg *pg;
  extern t_pdim pg_dim;
  for ( i = 0; i < POINT_NO; ) ntab.point[i++] = pg_noel;
  for ( i = 0; i < SPACE_NO; ) ntab.space[i++] = pg_noel;
  nt_pdef ( 'A', 0 );
  for ( i = 0; i < pg_dim; ++i ) nt_pdef ( 'B' + i, pg_qpow ( pg, i ) );
}

char nt_line ( char name, t_ps a, t_ps b ) {
  t_ps s;
  char *msp;
  int i, imax, arg[2];
  extern t_s *sp;
  extern t_pg *pg;
  extern t_pdim pg_dim;

  arg[0] = a;
  arg[1] = b;
  s = pg_get ( pg, arg, 2 );
  msp = sp->s[1][s]->pt;

  if ( pg_noel == s ) {
    fprintf ( stderr, "Error: no such subspace.\n" );
    return ( FALSE );
  };

  imax = pg_qpow ( pg, pg_dim );
  for ( i = 0; i < imax; ++i ) {
    if ( ( a != i ) && ( b != i ) && ( msp[i] == TRUE ) ) {
```

```
      if ( ! nt_pdef ( name++, i ) ) return ( FALSE );
    }
  }
  return ( TRUE );
}

int nt_lookup ( char name, int out[] ) {
  if ( isupper ( name ) ) {
    name -= 'A' - 1;
    if ( pg_noel == ntab.point[name] ) return ( 0 );
    out[0] = ntab.point[name];
    return ( 1 );
  } else if ( islower ( name ) ) {
    name -= 'a' - 1;
    if ( pg_noel == ntab.space[name] ) return ( 0 );
    out[0] = ntab.sdim[name];
    out[1] = ntab.space[name];
    return ( 2 );
  };
  return ( 0 );
}

char nt_pdef ( char name, t_ps p ) {
  if ( isupper ( name ) ) {
    name -= 'A' - 1;
    ntab.point[name] = p;
    return ( 1 );
  };
  return ( 0 );
}

char nt_sdef ( char name, t_pdim m, t_ps p ) {
  if ( islower ( name ) ) {
    name -= 'a' - 1;
    ntab.sdim[name] = m;
    ntab.space[name] = p;
    return ( 1 );
  };
  return ( 0 );
}

/* End of nametab.c */
```

Chapter 4

Other common modules

```
/* (C) 1998: Hans Georg Schaathun <georg@ii.uib.no> */
/* $Id: aux.h,v 5.0 1998/12/20 00:23:52 georg Exp $ */

/* Auxilliary file functions */
char   freadline ( FILE *stream, char *b ) ;
  /* Read a line of characters from stream into b. */
short  getshort ( FILE *stream ) ;
  /* Read a short variable from stream */
#define putshort(s,n)   ( putc ( (n)%256, s ), putc ( (n) >> 8, s ), 0 )
  /* Write a short variable to stream */

void PrintHelp ();
  /* Print a help message.  (Not very useful implementation.) */
void PrintVersion ();
  /* Print version and compile information. */
void ErrorUsage ();
  /* Print an error message and make a clean exit. */

char fid_verify ( FILE *stream, const char *id ) ;
  /* Read the file ID from stream and check that it matches id. **
  ** Exit on failure.                                           */

int MailNote ( char *msg ) ;
  /* Mail diagnostics to the operating user. */

extern void *aux_p;  /* Aux. variable for macros. */
extern FILE *aux_s;  /* Aux. variable for macros. */

#define getmem(s) ( ( ( aux_p = malloc ( s ) ) == NULL ) ? \
                    (void *) ErrEx ( ERR_MEM ) : aux_p )
  /* Allocate memory, exit on failure */
#define fileopen(f,m) ( ( aux_s = fopen (f,m) ) == NULL ? \
                    (FILE*) ErrExDbl (ERR_FOPEN,f) : aux_s )
  /* Open file, exit on failure */

#define CMD_TEE   "/usr/bin/tee"
  /* Name of the tee programme */
void tee_func ( const char *teeFile, char *opt ) ;
  /* Fork out a tee process. teeFile is the name of the file given **
  ** to tee(1), and opt are options passed to tee.                 */
#define tee(a)          tee_func ( a, NULL )
  /* Use tee to write stdout to file a, in adition to the existing stdout. */
#define tee_append(a)   tee_func ( a, "-a" )
  /* Use tee to append stdout to file a, in adition to the existing stdout. */
```

```c

/* End of aux.h */
```

```c
/* (C) 1998: Hans Georg Schaathun <georg@ii.uib.no> */
/* $Id: aux.c,v 5.0 1998/12/20 00:23:51 georg Exp $ */

#include "common.h"
#include "aux.h"

void *aux_p;
FILE *aux_s;

/* Auxilliary file functions */
short getshort ( FILE *stream ) {
  register short c;
  c = getc ( stream );
  c += getc ( stream ) << 8;
  if ( feof ( stream ) ) c = EOF;
  return ( c );
}

char freadline ( FILE *stream, char *b ) {
  char c;
  while ( ( EOF != ( c = getc ( stream ) ) ) ) {
    if ( '\n' == c ) { *b = 0; return ( c ); };
    *(b++) = c;
  };
  return ( *b = EOF );
}

void PrintHelp () {
  printf ( "RTFM\n" );
}

void PrintVersion () {
  extern char *RealName, *MyVersion, *CompileDate, *MyPathName;
  printf ( "%s %s\n", RealName, MyVersion );
  printf ( "Compiled %s.\n", CompileDate );
  printf ( "Running from %s.\n", MyPathName );
  printf ( "Compile options: %s.\n", CC_DEFINES );
}

int MailNote ( char *msg ) {
  char mtext[2048], timeString[TIME_STRING_L], timeUsed[16];
  extern char *MyName, *UserName, *HostName,
            *MyVersion, *CompileDate,
            *MailTo, CallString[], timeStart[];
  time_t theTime;
  unsigned long hour, min, sec;
  int mta_in[2], mta_out[2], i;

  /* Calculate time in hours, minutes and seconds. */
  sec  = (unsigned long)(clock()/CLOCKS_PER_SEC);
  min  = (unsigned long)(sec/60);
  sec  = sec % 60;
  hour = (unsigned long)(min/60);
  min  = min % 60;

  /* Make a string showing the time used. */
  sprintf ( timeUsed, "%i:%2i:%2i", hour, min, sec );
  for ( i = 0; timeUsed[++i] != 0; ) {
```

```
    if ( timeUsed[i] == ' ' ) timeUsed[i] = '0';
  };

  /* Fork and call sendmail */
  pipe ( mta_in );
  if ( 0 == fork ()  ) {
    close ( 0 );
    dup ( mta_in[0] );
    close ( mta_in[0] );
    close ( mta_in[1] );
    execlp ( mta, mta, MailTo, NULL );
  } else {
    close ( mta_in[0] );
    time ( &theTime );
    cftime ( timeString, NULL, &theTime );
    sprintf ( mtext,
      "From: %s (%s@%s)\nTo: %s\nDate: %sSubject: Output from %s@%s\n\n"
      "The following command has just finished on %s.\n%s\n\n%s\n\n"
      "Start: %sEnd: %s\nThe CPU time used is: %s\n\n"
      "The executable was version %s, compiled %s.\n",
      UserName, MyName, HostName, UserName, timeString, MyName,
      HostName, HostName, CallString, msg, timeStart, timeString, timeUsed,
      MyVersion, CompileDate
    );
    write ( mta_in[1], mtext, strlen ( mtext ) );
    close ( mta_in[1] );
  };
}

char fid_verify ( FILE *stream, const char *id ) {
  int i = 0;
  while ( 0 != id[i] ) {
    if ( id[i++] != getc ( stream ) ) return ( FALSE );
  }
  return ( TRUE );
}

void tee_func ( const char *teeFile, char *opt ) {
  int mta_in[2];
  pipe ( mta_in );
  if ( 0 == fork () ) {
    close ( 0 );
    dup ( mta_in[0] );
    close ( mta_in[0] );
    close ( mta_in[1] );
    if ( NULL == teeFile ) {
      execlp ( CMD_TEE, CMD_TEE, NULL );
    } else if ( NULL == opt ) {
      execlp ( CMD_TEE, CMD_TEE, teeFile, NULL );
    } else {
      execlp ( CMD_TEE, CMD_TEE, opt, teeFile, NULL );
    }
  } else {
    close ( 1 );
    dup ( mta_in[1] );
    close ( mta_in[0] );
    close ( mta_in[1] );
  };
}

void ErrorUsage () {
  extern char *MyPathName;
```

```
  fprintf ( stderr, "%s -h for help.\n", MyPathName );
  exit ( 2 );
}

/* End of aux.c */
```

Chapter 5

gen

```
/* (C) 1998: Hans Georg Schaathun <georg@ii.uib.no> */
/* $Id: gen.c,v 5.0 1998/12/20 00:24:03 georg Exp $ */

/* main function for gen(1)                      **
** gen creates and classifies value assigments   **
** defining error correcting codes.              */

#include "common.h"

#include <signal.h>

#include "aux.h"
#include "pg.h"
#include "val.h"
#include "vtree.h"
#include "signal.h"

#define ARGS        "a:B:b:c:d:efhk:m:n:O:p:q:sS:tT:vw:x"

/* Global Variables */
char *MyName, *MyPathName,       /* The programme name from argument no. 0 */
     *UserName, *HostName,       /* Environment. */
     *RealName = GEN_NAME,       /* The real name of the programme */
      CallString[CALL_STRING_L], /* The entire command call with arguments */
     *MyVersion = GEN_VERSION,   /* Programme version number. */
     *CompileDate = GEN_COMPILE, /* Date of compilation. */
     *MailTo, *StatusFile = VAL_STATUSFILE,
     *SignalBackUpFile = DEF_SIGBFILE,
      timeStart[TIME_STRING_L];
t_pdim  pg_dim=4, pg_order=2;   /* PG parameters */
t_vt  *vt = NULL;
pid_t child, pid;
t_val *val = NULL;
t_pg  *pg = NULL;

int main ( int argc, char *argv[] ) {

/* External variable declarations. */
  extern t_pdim pg_dim, pg_order;            /* PG parameters */
  extern char *optarg;                       /* for getopt(3) */
  extern int optind, opterr, optopt;         /* for getopt(3) */
  extern t_vt  *vt;                          /* Data structure */
  extern t_val *val;                         /* Data structure */
```

```
  extern t_pg  *pg;                          /* Data structure */

/* Internal variable declarations. */
#ifdef ALLOW_FIX
  char mode_f = FALSE;
#endif
  char f, n = DEF_NICE, w = DEF_CNICE, /* Option flag */
       mode_e = TRUE, mode_x = TRUE, mode_s = FALSE,
       moreval = RET_VAL;
  t_value d=2;                       /* \delta_0 */
  t_class c=val_noclass;             /* Class number */
  int i;                             /* Index counter */
  unsigned long bc = 0, b = DEF_BCINT, /* Backup control. */
              a = DEF_ALARM;            /* Alarm */
  char *OutFile = GEN_OUTFILE, *InFile = NULL, FileName[FILE_NAME_L],
       *TreeFile = NULL, *InitFile = NULL,
       *BOutFile = GEN_BOUTFILE, *BackUpFile = NULL,
        CBackUpFile[FILE_NAME_L], BBackUpFile[FILE_NAME_L], *cpt;
  char rflags[24] = { 0 };
  FILE *InStream, *actfile;
  time_t theTime;

/* General signal handling */
  signal ( SIGCHLD, SIG_IGN );  /* Do not create zombie children. */
  signal ( SIGHUP,  SIG_IGN );  /* Ignore Hangup */

/* Clock initialization. */
  clock ();
  time ( &theTime );
  cftime ( timeStart, NULL, &theTime );

/* Parsing argument 0. */
  MyPathName = *argv;
  MyName = basename ( MyPathName );

/* Environment parsing. */
  MailTo = UserName = getenv ( "USER" );
  HostName = getenv ( "HOST" );
  pid = getpid ();

/* Option parsing. */
  while ( -1 != ( f = getopt( argc, argv, ARGS ) ) )
    switch (f) {
    case 'a': a = atoi(optarg); break;
    case 'b': b = atoi(optarg); break;
    case 'c': c = atoi(optarg);
              strcat ( rflags, "-c " );
              strcat ( rflags, optarg );
              strcat ( rflags, " " );
              break;
    case 'd': d = atoi(optarg); break;
    case 'e': mode_e   = FALSE;
              strcat ( rflags, "-e " );
              break;
    case 'k': pg_dim = atoi(optarg); break;
    case 'm': MailTo   = optarg; break;
    case 'n': n = atoi(optarg); break;
#ifdef ALLOW_FIX
    case 'f': mode_f   = TRUE; break;
#endif
    case 'p': InFile   = optarg; break;
    case 'q': pg_order = atoi(optarg); break;
```

```
    case 'w': w = atoi(optarg); break;
    case 'x': mode_x   = FALSE; break;
    case 's': mode_s   = TRUE; break;
    case 'S': mode_s   = TRUE; StatusFile = optarg; break;
    case 'T': TreeFile = optarg; break;
    case 'B': BackUpFile = optarg; break;
    case 'O': BOutFile = optarg; break;
    case 't': break;
    case 'h': PrintHelp();    exit (1); break;
    case 'v': PrintVersion(); exit (1); break;
    default:  ErrorUsage();   exit (1);
    } /* switch */

/* Nice and signals */
  nice ( n );
  alarm ( 60*a );
  SignalHandling ();

/* Complete the argument parsing. */
  if ( ( ! mode_f ) && ( optind != argc ) ) ErrorUsage();
  if ( b < MIN_BCINT ) ErrEx ( ERR_BCINT ); /* Sanity check. */
  cpt = CallString;
  for ( i=0; i < argc; ++i ) {
    sprintf ( cpt, "%s ", argv[i] );
    cpt += 1 + strlen ( argv[i] );
  };

  GetData ( TreeFile, InFile );

/* Create backup files and file names. */
  if ( BackUpFile == NULL ) {
    sprintf ( CBackUpFile, "%s%s", OutFile, DEF_BCEXT );
    BackUpFile = CBackUpFile;
  };
  sprintf ( BBackUpFile, "%s%s", BackUpFile, DEF_BCBCEXT );
  unlink ( BBackUpFile );  /* Remove secondary backupfile. */
  /* creat ( BackUpFile, 0600 ); */

/* Create act file in current directory. */
  actfile = fileopen ( GEN_ACTFILE, "w" );
  strcat ( rflags, "-T" );
  fprintf ( actfile, "GEN_HOST=%s\n", HostName );
  fprintf ( actfile, "GEN_USER=%s\n", UserName );
  fprintf ( actfile, "GEN_PID=%i\n", pid );
  fprintf ( actfile, "GEN_CMD=%s\n", CallString );
  fprintf ( actfile, "GEN_GEN=%s\n", MyPathName );
  fprintf ( actfile, "GEN_FLAGS=%s\n", rflags );
  fprintf ( actfile, "GEN_BC=%s\n", BackUpFile );
  fclose ( actfile );

/* Generate the value assignments. */
  if ( NULL == TreeFile ) val_stepinit ( val, d );

  if ( mode_f ) {
  /* Assign values given on the command line. */
    while ( optind < argc ) {
      char *p, *a;
      a = p = argv[optind++];
      while ( *(++a) != ':' ) {};
      *(a++) = 0;
      val_set ( val, atoi (p), atoi (a) );
    };
```

```
    val_sumval (val);
    val_classify (val);
  };

  if ( mode_s ) StatusReport(val);

/* If we search only for a certain class, we step forward to an **
** assignment of the correct class.                             */
  if ( ( c != val_noclass ) && ( c != val_class ( val ) ) )
    moreval = val_stepclass (val,c);

/* The search loop. */
  while ( NO_MORE_VAL != moreval ) {
    /* Insert the value assignment in the tree. */
    if ( mode_e || ( val_class( val ) < val->nclass ) ) vt_insert( val, vt );

    /* Make regular backups. */
    if ( bc++ > b ) {
      if ( ( 0 == link ( BackUpFile, BBackUpFile ) ) &&
                                        /* Make secondary backup. */
           ( 0 == ( child = fork() ) ) ) {  /* Fork to backup. */
        DefaultSignals ();
        nice ( w );                         /* Renice child. */
        unlink ( BackUpFile );              /* Create primary backup file. */
        vt_backup ( BackUpFile, vt );       /* Make backup. */
        unlink ( BBackUpFile );             /* Remove secondary backup. */
        if ( mode_s ) StatusReport ( val ); /* Report the level of progress. */
        exit ( 0 );                         /* Exit backup child. */
      };
      bc = 0;
    }; /* if ( bc++ > b ) */
    /* Step to the next value assignment */
    moreval = ( val_noclass == c ? val_stepval (val) :
                val_stepclass (val,c) );
  }; /* while */
  vt_backup ( BOutFile, vt );       /* binary output */

/* Notification by mail. */
  if ( mode_x ) MailNote ( "The programme exited normally.\n" );

/* Deallocate memory and exit */
  vt_del ( vt );
  pg_del ( pg );
  val_del ( val );
  rename ( GEN_ACTFILE, GEN_PASFILE );
  exit ( 0 );
}

/* end of gen.c */
```

```
/* (C) 1998: Hans Georg Schaathun <georg@ii.uib.no> */
/* $Id: prep.h,v 5.0 1998/12/20 00:25:31 georg Exp $ */

void GetData ( char *TreeFile, char *InFile ) ;
  /* Allocate memory and load data from file */

/* End of prep.h */
```

```
/* (C) 1998: Hans Georg Schaathun <georg@ii.uib.no> */
/* $Id: prep.c,v 5.0 1998/12/20 00:25:27 georg Exp $ */
```

```
#include "common.h"
#include "pg.h"
#include "val.h"
#include "vtree.h"
#include "aux.h"
#include "prep.h"

void GetData ( char *TreeFile, char *InFile ) {
  extern t_pdim pg_order, pg_dim;
  extern t_val *val;
  extern t_pg *pg;
  extern t_vt *vt;
  FILE *InStream;
  if ( NULL == TreeFile ) {
  /* Load the Geometry */
    if ( NULL == InFile ) {
      InFile = getmem ( PATH_NAME_L );
      sprintf ( InFile, "%s/pg(%i,%i)", PG_DIR, pg_dim, pg_order );
    };
    InStream = fileopen ( InFile, "rb" );
    pg = pg_load ( InStream );
    fclose ( InStream );
  /* Allocate memory */
    val = val_init ( pg );
    vt = vt_init ( val );
  } else {
  /* Load the Tree. */
    InStream = fileopen ( TreeFile, "rb" );
    vt = vt_load ( InStream, NULL, pg );
    fclose ( InStream );
    val = vt->val;
    pg = val->pg;
  };

  pg_dim = pg->k; pg_order = pg->q;
}

/* End of prep.h */
```

```
/* (C) 1998: Hans Georg Schaathun <georg@ii.uib.no> */
/* $Id: signal.h,v 5.0 1998/12/20 00:25:40 georg Exp $ */

void CleanExit ( int sig ) ;
  /* Dump the data and exit cleanly (for SIGINT, SIGTERM, et c.). */
void CoreExit ( int sig ) ;
  /* Dump the data and exit using abort(3) to create a core dump. **
  ** (for SIGBUS, SIGSEGV, et c.). */
void SignalHandling ( void ) ;
  /* Set up the standard set of signal handlers. */
void DefaultSignals ( void ) ;
  /* Set up normal signal handlers, i.e. handle with SIG_DFL. */

/* End of signal.h */
```

```
/* (C) 1998: Hans Georg Schaathun <georg@ii.uib.no> */
/* $Id: signal.c,v 5.0 1998/12/20 00:25:34 georg Exp $ */

#include "common.h"
#include "aux.h"
```

```
#include "pg.h"
#include "val.h"
#include "vtree.h"
#include <sys/types.h>
#include <signal.h>

static void sig_statusreport ( int sig ) ;
  /* Write a status report using the function StatusReport(). */

void CleanExit ( int sig ) {
  extern t_vt *vt;
  extern char *SignalBackUpFile;
  char msg[128];
  sprintf ( msg, "The programme caught signal %i.\n", sig );
  MailNote ( msg );
  vt_backup ( SignalBackUpFile, vt );
  exit ( ENO_INT ) ;
}

void CoreExit ( int sig ) {
  extern t_vt *vt;
  extern char *SignalBackUpFile;
  char msg[128];
  sprintf ( msg, "The programme caught signal %i.\n", sig );
  MailNote ( msg );
  vt_backup ( SignalBackUpFile, vt );
  abort ();
}

void sig_statusreport ( int sig ) {
  extern t_val *val;
  StatusReport ( val );
  signal ( sig, &sig_statusreport );
  return ;
}

void DefaultSignals ( void ) {
  signal ( SIGINT,  SIG_DFL );
  signal ( SIGTERM, SIG_DFL );
  signal ( SIGQUIT, SIG_DFL );
  signal ( SIGALRM, SIG_DFL );

  signal ( SIGUSR1, SIG_DFL );

  signal ( SIGSEGV, SIG_DFL );
  signal ( SIGBUS, SIG_DFL );
  signal ( SIGILL, SIG_DFL );
  signal ( SIGFPE, SIG_DFL );
  signal ( SIGSYS, SIG_DFL );
}

void SignalHandling ( void ) {
  signal ( SIGINT,  &CleanExit );
  signal ( SIGTERM, &CleanExit );
  signal ( SIGQUIT, &CleanExit );
  signal ( SIGALRM, &CleanExit );

  signal ( SIGUSR1, &sig_statusreport );

  signal ( SIGSEGV, &CoreExit );
  signal ( SIGBUS, &CoreExit );
  signal ( SIGILL, &CoreExit );
```

```
  signal ( SIGFPE, &CoreExit );
  signal ( SIGSYS, &CoreExit );
}

/* End of signal.c */
```

Chapter 6

vsh

```c
/* (C) 1998: Hans Georg Schaathun <georg@ii.uib.no> */
/* $Id: vsh.h,v 5.0 1998/12/20 00:26:29 georg Exp $ */

int parse ( FILE *in, char term ) ;

/* end of vsh.h */
```

```c
/* (C) 1998: Hans Georg Schaathun <georg@ii.uib.no> */
/* $Id: vsh.c,v 5.0 1998/12/20 00:26:25 georg Exp $ */

/* main function for vsh(1), the value shell.              **
** Interactive programme to classify value assignments     */

#include "common.h"

#include <signal.h>

#include "pginit.h"
#include "val.h"
#include "vtree.h"
#include "vsh.h"

#define VSH_ARGS "bek:nq:t:vhi"

/* Global Variables */
char *MyName, *MyPathName,          /* The programme name from argument no. 0 */
     *RealName = VSH_NAME,          /* The real name of the programme */
      CallString[CALL_STRING_L],    /* The entire command call with arguments */
     *UserName, *HostName,          /* Environment. */
     *MyVersion = VSH_VERSION,      /* Programme version number. */
     *CompileDate = GEN_COMPILE,    /* Date of compilation. */
     *MailTo, *StatusFile = VAL_STATUSFILE,
      timeStart[TIME_STRING_L];
t_vt  *vt = NULL;
pid_t child;

int main ( int argc, char *argv[] ) {

/* External variable declarations. */
  extern t_pdim pg_dim, pg_order;              /* PG parameters */
  extern char *optarg;                         /* for getopt(3) */
  extern int optind, opterr, optopt;           /* for getopt(3) */
  extern t_vt  *vt;                            /* Data structure */
```

```
  extern t_val *val;                     /* Data structure */
  extern t_pg  *pg;                      /* Data structure */
  extern t_s   *sp;                      /* Data structure */
  extern unsigned long v_allocn;

/* Internal variable declarations. */
  int i;                        /* Index counter */
  char *SFile = NULL, *InFile = NULL, FileName[FILE_NAME_L],
       *TreeFile = NULL, *InitFile = NULL,
       *BOutFile = GEN_BOUTFILE, *BackUpFile = NULL,
        CBackUpFile[FILE_NAME_L], BBackUpFile[FILE_NAME_L], *cpt;
  char f, mode_e = FALSE, mode_i = TERM;
  FILE *fs;
  time_t theTime;

/* Parsing argument 0. */
  MyPathName = *argv;
  MyName = basename ( MyPathName );

/* Variable initialisation. */
  v_allocn = 1; /* Allocate memory for one assignment at a time. */
  pg_dim = 0;
  mode_i = isatty ( 0 ); /* Initialise the flag indicating interactivity */

/* Option parsing. */
  while ( -1 != ( f = getopt( argc, argv, VSH_ARGS ) ) )
    switch (f) {
    case 'b': mode_i = 0; break;
    case 'k': pg_dim = atoi ( optarg ); break;
    case 'q': pg_order = atoi ( optarg ); break;
    case 'h': PrintHelp();    exit (1); break;
    case 'e': mode_e = ECHO; break;
    case 'i': mode_i = TERM; break;
    case 'n': mode_e = 0; break;
    case 'v': PrintVersion(); exit (1); break;
    default:  ErrorUsage();   exit (1);
    } /* switch */

  /* Initialise the global variables (pg, val, sp) */
  if ( pg_dim > 0 ) pv_init ( pg_dim, pg_order );

  if ( argc == optind ) {  /* No file name is given. */
    /* Read a config file */
    if ( NULL != ( fs = fopen ( VSH_CONF, "r" ) ) ) {
      parse ( fs, 0 );
      fclose ( fs );
    };
    parse ( stdin, mode_i|mode_e );
  } else { /* Read commands from file */
    fs = fileopen ( argv[optind], "r" );
    parse ( fs, mode_e );
    fclose ( fs );
  };

  exit ( 0 );
}

/* end of vsh.c */
```

```
/* (C) 1998: Hans Georg Schaathun <georg@ii.uib.no> */
/* $Id: lex.l,v 5.0 1998/12/20 00:24:18 georg Exp $ */
```

```
/* Lexical parser for vsh(1) */

%{

/* Configurable section */
#define MAX_STR_L  64
#define MAX_ARG    6

/* Include */
#include "common.h"
#include "aux.h"
#include "pgval.h"
#include "vsh.h"
#include "valmsp.h"
#include <string.h>

/* Internal macros */
#define prompt(n)   ( terminal ? fprintf ( stdout, "vsh:%i> ", n ) : 0 ),\
                    BEGIN (INITIAL)
#define NO_IMP      ( fprintf ( stderr, \
                     "Sorry, I haven't learnt how to do that, yet.\n" ) )

/* Enumerate all commands */
enum { C_LOG, C_OPEN, C_UNLOG, C_CLOSE, C_MAX, C_EVAL, C_VEVAL,
      C_CLEAR, C_SUBVAL, C_PVP, C_PALL, C_PRINT, C_VER, C_FIX, C_COUNT,
      C_SPFIX,
      C_HELP, C_CLASS, C_NEXT, C_SPACE, C_SUBP, C_SUB, C_SET, C_SPAN,
      C_SPDEF, C_SDEF, C_PDEF, C_ECHO, C_NOECHO, C_SETFIX,
      C_LINE,
      C__NECHO, C__ECHO, C__SP,
      C_QPROB, C_MPROB, C_PROBE,
      C_PG, C_SOURCE, C_CSET, C_SPSET, C_VAL, C_GET };

/* Static C identifiers. */
static void execute ( int c );
  /* Handle commands without arguments. */
static void strcmd ( int c, const char *s );
  /* Handle commands with string argument. */
static void intcmd ( int c, int arg[], int argno );
  /* Handle commands with integer arguments. */
static int  source ( const char *f ) ;
  /* Read commands from file. */
static char arglim ( int no, int mx, int mn ) ;
  /* Check that the number of argument no correct.  Print an error **
  ** message otherwise.                                            */

static YY_BUFFER_STATE mbuf;    /* Main buffer */
static FILE *valfile,           /* The file used by open/next/close */
            *outfile;           /* Usually equal to stdout */
static char terminal;           /* True if reading from stdin. */
static char name;               /* The identifier argument for pdef etc. */
static char echo;               /* True if commands should be echoed. */

%}

  /* Character classes */

COMMENT #
VCOM    :
QUO     ["]
CHAR    [a-zA-Z0-9_.]
```

```
NAME    [a-zA-Z]
LETTER  [a-zA-Z_]
WORD    {LETTER}{CHAR}*
DIGIT   [0-9]
NUMBER  [0-9]+
WHITE   [ \t]

  /* Command names and aliases. Aliases are not documented,
  ** only the main command name is.  Names starting with an
  ** underscore are reserved for testing, and not documented. */

C_CLASS   class|\.
C_CLEAR   clear|cv
C_CLOSE   (close|unopen)
C_COUNT   count
C_CSET    cset
C_ECHO    echo|"+"
C_EVAL    eval(uate)?
C_FIX     fix
C_SPFIX   spfix
C_GET     get
C_HELP    help|\?
C_LINE    line
C_LOG     log|>
C_MAX     max(val)?
C_MPROB   mp(robe)?
C_NEXT    n(ext)?|readval
C_NOECHO  noecho|"-"
C_OPEN    open
C_PALL    p(rint)?all
C_PDEF    pd(ef)?
C_PG      pg
C_PRINT   print
C_PROBE   probe
C_PVP     p(vp)?
C_QPROB   qp(robe)?
C_QUIT    (quit|q|exit|x)
C_SDEF    sd(ef)?
C_SET     set
C_SETFIX  sf|setfix
C_SHELL   sh(ell)?|!
C_SOURCE  source|<
C_SPACE   space
C_SPAN    span
C_SPDEF   spd(ef)?
C_SPSET   spset
C_SUB     (sub|subspace|sup|superspace)
C_SUBP    subp(rint)?
C_SUBVAL  subval|sp
C_UNLOG   unlog
C_VAL     val|"="
C_VER     ver(sion)?
C_VEVAL   veval(uate)?|classify

  /* Undocumented commands for testing */

C__NECHO  _necho
C__ECHO   _echo
C__SP     _sp

  /* Start conditions */

```

```
%x quote shell
%s unquote ex_string cmd_str S_QUIT no_arg IGNORE ex_int def

%%

%{
  static int  cmdno = 0;                      /* Counter for the prompt */
  static int  arg[MAX_ARG], cmd, argno = 0; /* Command and arguments */
  static char str[MAX_STR_L];               /* String arguments for command */

  prompt ( ++cmdno );  /* Print the initial prompt */
%}

^.+\n            { if ( echo ) fprintf (outfile, "%s", yytext ); REJECT; }

{WHITE}+           { /* Ignore white space */ }

<INITIAL>\n        { if ( echo ) putc ( '\n', outfile ); prompt ( cmdno );
                  /* ignore empty lines */ }

<S_QUIT>\n         { return ( 0 ); /* exit */ }

<INITIAL><<EOF>>   { putc ( '\n', stderr ); return ( 0 ) ; /* exit */ }

<<EOF>>            { fprintf ( stderr, "Error: Unexpected end of file.\n" );
                  return ( 1 ); }

<INITIAL>{C_QUIT}  { BEGIN ( S_QUIT ); }

  /* Handling define commands */

<INITIAL>{C_LINE}   { cmd = C_LINE; BEGIN ( def ); }
<INITIAL>{C_PDEF}   { cmd = C_PDEF; BEGIN ( def ); }
<INITIAL>{C_SPDEF}  { cmd = C_SPDEF; BEGIN ( def ); }
<INITIAL>{C_SDEF}   { cmd = C_SDEF; BEGIN ( def ); }
<def>{NAME}         { name = yytext[0]; BEGIN ( ex_int ); }
<def>.              {
    fprintf ( stderr, "Error: expecting name to define.\n" );
  }

  /* Handling commands with no argument */

<INITIAL>{C_MPROB}  { cmd = C_MPROB; BEGIN ( no_arg ); }
<INITIAL>{C_UNLOG}  { cmd = C_UNLOG; BEGIN ( no_arg ); }
<INITIAL>{C_COUNT}  { cmd = C_COUNT; BEGIN ( no_arg ); }
<INITIAL>{C_PRINT}  { cmd = C_PRINT; BEGIN ( no_arg ); }
<INITIAL>{C_PVP}    { cmd = C_PVP; BEGIN ( no_arg ); }
<INITIAL>{C_VER}    { cmd = C_VER; BEGIN ( no_arg ); }
<INITIAL>{C_HELP}   { cmd = C_HELP; BEGIN ( no_arg ); }
<INITIAL>{C_CLOSE}  { cmd = C_CLOSE; BEGIN ( no_arg ); }
<INITIAL>{C_EVAL}   { cmd = C_EVAL; BEGIN ( no_arg ); }
<INITIAL>{C_VEVAL}  { cmd = C_VEVAL; BEGIN ( no_arg ); }
<INITIAL>{C_CLASS}  { cmd = C_CLASS; BEGIN ( no_arg ); }
<INITIAL>{C_NEXT}   { cmd = C_NEXT; BEGIN ( no_arg ); }
<INITIAL>{C_CLEAR}  { cmd = C_CLEAR; BEGIN ( no_arg ); }
<INITIAL>{C_ECHO}   { cmd = C_ECHO; BEGIN ( no_arg ); }
<INITIAL>{C_NOECHO} { cmd = C_NOECHO; BEGIN ( no_arg ); }

<no_arg>\n       { execute ( cmd ); prompt ( ++cmdno ); }

<S_QUIT,no_arg,cmd_str>. {
    fprintf ( stderr, "Error: Trailing characters.\n" );
```

```
    prompt ( cmdno );
  }

  /* Handling commands with string argument */

<INITIAL>{C_LOG}      { cmd = C_LOG; BEGIN ( ex_string ); }
<INITIAL>{C_OPEN}     { cmd = C_OPEN; BEGIN ( ex_string ); }
<INITIAL>{C__ECHO}    { cmd = C__ECHO; BEGIN ( ex_string ); }
<INITIAL>{C_SOURCE}   { cmd = C_SOURCE; BEGIN ( ex_string ); }
<INITIAL>{C__SP}      { cmd = C__SP; BEGIN ( ex_string ); }

<ex_string>\n       {
   fprintf ( stderr, "Error: Expecting string argument.\n" );
   prompt ( cmdno );
 }

<ex_string>{QUO}    { BEGIN ( quote ); }

<ex_string>{WORD}   { strcpy ( str, yytext ); BEGIN ( cmd_str ); }

<quote>\n           { fprintf ( stderr, "Error: Unmatched quote.\n" ); }

<quote>.*/{QUO}     { strcpy ( str, yytext ); BEGIN ( unquote ); }

<unquote>{QUO}      { BEGIN ( cmd_str ); }

<cmd_str>\n         { strcmd ( cmd, str ); prompt ( ++cmdno ); }

<INITIAL>{C_SHELL}  {  BEGIN ( shell ); }
<shell>.*\n         { system ( yytext ); prompt ( ++cmdno ); }

  /* Handling commands with integer arguments */

<INITIAL>{C_PALL}   { cmd = C_PALL; BEGIN ( ex_int ); }
<INITIAL>{C_SPACE}  { cmd = C_SPACE; BEGIN ( ex_int ); }
<INITIAL>{C_SUB}    { cmd = C_SUB; BEGIN ( ex_int ); }
<INITIAL>{C_SUBP}   { cmd = C_SUBP; BEGIN ( ex_int ); }
<INITIAL>{C_SET}    { cmd = C_SET; BEGIN ( ex_int ); }
<INITIAL>{C_SETFIX} { cmd = C_SETFIX; BEGIN ( ex_int ); }
<INITIAL>{C_SPSET}  { cmd = C_SPSET; BEGIN ( ex_int ); }
<INITIAL>{C_SPFIX}  { cmd = C_SPFIX; BEGIN ( ex_int ); }
<INITIAL>{C_CSET}   { cmd = C_CSET; BEGIN ( ex_int ); }
<INITIAL>{C_GET}    { cmd = C_GET; BEGIN ( ex_int ); }
<INITIAL>{C_SPAN}   { cmd = C_SPAN; BEGIN ( ex_int ); }
<INITIAL>{C_VAL}    { cmd = C_VAL; BEGIN ( ex_int ); }
<INITIAL>{C_SUBVAL} { cmd = C_SUBVAL; BEGIN ( ex_int ); }
<INITIAL>{C_MAX}    { cmd = C_MAX; BEGIN ( ex_int ); }
<INITIAL>{C_PG}     { cmd = C_PG; BEGIN ( ex_int ); }
<INITIAL>{C_FIX}    { cmd = C_FIX; BEGIN ( ex_int ); }
<INITIAL>{C_PROBE}  { cmd = C_PROBE; BEGIN ( ex_int ); }
<INITIAL>{C_QPROB}  { cmd = C_QPROB; BEGIN ( ex_int ); }
<INITIAL>{C__NECHO} { cmd = C__NECHO; BEGIN ( ex_int ); }

<ex_int>{NUMBER}    { arg[argno++] = atoi ( yytext ); }
<ex_int>{NAME}      { argno += nt_lookup ( yytext[0], arg + argno ); }

<ex_int>\n          { intcmd ( cmd, arg, argno );
                    argno = 0; prompt ( ++cmdno ); }

<ex_int>.           {
    fprintf ( stderr, "Error: expecting integer.\n" );
    prompt ( cmdno );
```

```
  }

  /* Ignoring comments and white space */

{COMMENT}.*\n      { prompt ( cmdno ); }
{VCOM}.*\n         { if ( ! echo ) fprintf ( outfile, "%s", yytext );
                   prompt ( cmdno ); }

<INITIAL>{WORD}    {
  fprintf ( stderr, "Error: Unknown command: %s.\n", yytext );
  BEGIN ( IGNORE ); }

<IGNORE>.*\n       { prompt ( cmdno ); }

%%

int parse ( FILE *in, char term ) {
  extern char terminal;
  int r;
  terminal = term&TERM;
  echo = term&ECHO;
  yy_switch_to_buffer ( mbuf = yy_create_buffer ( in, YY_BUF_SIZE ) );
  outfile = stdout;
  r = yylex ();
  yy_delete_buffer ( mbuf );
  return ( r ) ;
}

static int source ( const char *f ) {
  FILE *s;
  YY_BUFFER_STATE b, ob;
  char t;
  extern char terminal;

  /* Open the file */
  if ( NULL == ( s = fopen ( f, "r" ) ) ) {
    fprintf ( stderr, "File %s does not exist.\n", f );
    return ( ENO_FOPEN );
  }

  /* Don't print the prompt while sourcing files. */
  t = terminal; terminal = 0;

  ob = YY_CURRENT_BUFFER;  /* Store a pointer to the old buffer */
  b = yy_create_buffer ( s, YY_BUF_SIZE );  /* Create buffer */
  yy_switch_to_buffer ( b );                /* Switch to the new buffer */
  yylex ();                  /* Parse the buffer */
  yy_delete_buffer ( b );  /* Deleta the buffer */
  fclose ( s );              /* Close the file */
  terminal = t;              /* Reset the flag to indicate if we want a prompt */
  yy_switch_to_buffer ( ob );  /* Return to the old buffer */
  return ( 0 );
}

void execute ( int c ) {
  extern t_val *val;
  switch ( c ) {
    case C_ECHO: echo = TRUE; break;
    case C_NOECHO: echo = FALSE; break;
    case C_MPROB:
      while ( 0 < val_incprobe ( NULL, 0 ) ) {};
    break;
```

```
  case C_CLEAR:
    nt_init ();
    val_clear ( val, val_novalue );
  break;
  case C_PVP:
    pv_print ( outfile );
  break;
  case C_PRINT:
    val_valprint ( outfile, val );
  break;
  case C_VEVAL:
    val_sumval ( val );
    v_reinit ( val->v );
    val_getclass ( val, outfile );
    val_ds ( val );
    val_classprint ( val, outfile );
  break;
  case C_EVAL:
    val_sumval ( val );
    v_reinit ( val->v );
    val_class ( val );
    val_ds ( val );
    val_classprint ( val, outfile );
  break;
  case C_CLOSE:
    if ( NULL != valfile ) fclose ( valfile );
  break;
  case C_CLASS:
    val_classprint ( val, outfile );
  break;
  case C_NEXT:
    if ( NULL == valfile ) {
      fprintf ( stderr, "Sorry, I have no file to read from.\n" );
      return;
    };
    if ( EOF == val_assign ( val, ( NULL == valfile ? stdin : valfile ) ) ) {
      fclose ( valfile );
      fprintf ( stderr, "EOF, closing file.\n" );
    };
    v_reinit ( val->v );
    val_sumval ( val );
    val_classify ( val );
  break;
  case C_COUNT:
    fprintf ( outfile, "%i fixed values.\n", pv_fixcount() );
  break;
  case C_UNLOG:
    if ( outfile != stdout ) fclose ( outfile );
    outfile = stdout;
  break;
  case C_VER:
    PrintVersion();
  break;
  case C_HELP:
    fprintf ( stderr, "Sorry, you have to read the man page.\n" );
  break;
  default:
    NO_IMP ;
  break;
 }
}
```

```
void strcmd ( int c, const char *s ) {
  FILE *fs;
  extern t_s *sp;
  extern t_pdim pg_dim, pg_order;
  switch ( c ) {
    case C_SOURCE:
      source ( s );
    break;
    case C_OPEN:
      if ( NULL != valfile ) fclose ( valfile );
      valfile = fileopen ( s, "r" );
    break;
    case C_LOG:
      outfile = fileopen ( s, "a" );
    break;
    case C__ECHO:
      fprintf ( outfile, "%s\n", s );
    break;
    case C__SP:
      if ( NULL == ( fs = fopen ( s, "w" ) ) ) {
        fprintf ( stderr, "Could not open file %s.\n", s );
        return ;
      }
      s_print ( sp, fs, pg_dim, pg_order );
      fclose ( fs );
    break;
    default:
      NO_IMP ;
    break;
  }
}

char arglim ( int no, int mx, int mn ) {
  if ( no < mn ) {
    fprintf ( stderr, "Error: Too few arguments.\n" );
    return ( -1 );
  }
  if ( no > mx ) {
    fprintf ( stderr, "Error: Too many arguments.\n" );
    return ( 1 );
  }
  return ( 0 );
}

void intcmd ( int c, int arg[], int argno ) {
  extern t_pg *pg;
  extern t_val *val;
  extern t_pdim pg_dim;
  static int i;
  t_ps subspace;
  switch ( c ) {
    case C_PALL:
      if ( arglim ( argno, 1, 0 ) ) return ;
      if ( argno == 0 ) arg[0] = 1;
      pv_printall ( outfile, arg[0] );
    break;
    case C_QPROB:
      if ( arglim ( argno, 1, 0 ) ) return ;
      if ( argno == 0 ) arg[0] = 0;
      val_incprobe ( NULL, arg[0] );
    break;
    case C_PROBE:
```

```
    if ( arglim ( argno, 1, 0 ) ) return ;
    if ( argno == 0 ) arg[0] = 0;
    val_incprobe ( outfile, arg[0] );
  break;
  case C_LINE:
    if ( arglim ( argno, 2, 2 ) ) return ;
    if ( ! nt_line ( name, arg[0], arg[1] ) )
      fprintf ( stderr, "Error defining points.\n" );
  break;
  case C_PDEF:
    if ( arglim ( argno, 1, 1 ) ) return ;
    if ( ! nt_pdef ( name, arg[0] ) )
      fprintf ( stderr, "Error defining point.\n" );
  break;
  case C_SDEF:
    if ( arglim ( argno, 2, 2 ) ) return ;
    if ( ! nt_sdef ( name, arg[0], arg[1] ) )
      fprintf ( stderr, "Error defining subspace.\n" );
  break;
  case C_SUBVAL:
    if ( arglim ( argno, 2, 2 ) ) return ;
    pv_sprint ( outfile, arg[0], arg[1] );
  break;
  case C_VAL:
    for ( i = 0; i < argno; ) {
      pv_pprint ( outfile, arg[i] );
      putc ( ++i % 10 == 0 ? '\n' : '\t', outfile );
    };
    if ( i % 10 != 0 ) putc ( '\n', outfile );
  break;
  case C_SPDEF:
    subspace = pg_get ( pg, arg, argno );
    if ( pg_noel == subspace ) {
      fprintf ( stderr, "Error: no such subspace.\n" );
    } else {
      if ( ! nt_sdef ( name, argno-1, subspace ) )
        fprintf ( stderr, "Error defining subspace.\n" );
    };
  break;
  case C_SPAN:
    subspace = pg_get ( pg, arg, argno );
    if ( pg_noel == subspace ) {
      fprintf ( stderr, "Error: no such subspace.\n" );
    } else {
      pv_sprint ( outfile, argno - 1, subspace );
    };
  break;
  case C_GET:
    subspace = pg_get ( pg, arg, argno );
    if ( pg_noel == subspace ) {
      fprintf ( stderr, "Error: no such subspace.\n" );
    } else {
      pg_space ( outfile, pg, argno - 1, subspace );
    };
  break;
  case C_SUBP:
    if ( arglim ( argno, 3, 3 ) ) return ;
    if ( arg[0] == arg[1] ) {
      fprintf ( stderr,
             "Error: Arg 1 and Arg 2 are supposed to be different.\n" );
      return ;
    };
```

```
  if ( ENO_IMP == pv_ssprint ( outfile, arg[0], arg[1], arg[2] ) ) NO_IMP ;
break;
case C_SUB:
  if ( arglim ( argno, 3, 3 ) ) return ;
  if ( arg[0] == arg[1] ) {
    fprintf ( stderr,
              "Error: Arg 1 and Arg 2 are supposed to be different.\n" );
    return ;
  };
  pg_ssprint ( outfile, pg, arg[0], arg[1], arg[2] );
break;
case C_SPFIX:
  if ( arglim ( argno, 3, 2 ) ) return ;
  if ( ( argno > 2 ) && ( arg[2] == 0 ) )
    val_spaceunfix ( arg[0], arg[1] )
  else
    val_spacefix ( arg[0], arg[1] );
break;
case C_FIX:
  if ( arglim ( argno, 1, 1 ) ) return ;
  val_fixit ( val, arg[0] );
break;
case C_MAX:
  if ( arglim ( argno, 1, 1 ) ) return ;
  if ( arg[0] >= pg_dim ) {
    fprintf ( stderr,
              "Error: k = %i, no subspaces of dimension %i exists.\n",
              pg_dim, arg[0] );
    return ;
  };
  mval_print ( outfile, val->m[arg[0]] );
break;
case C_SPACE:
  if ( arglim ( argno, 2, 2 ) ) return ;
  pg_space ( outfile, pg, arg[0], arg[1] );
break;
case C_SETFIX:
  if ( arglim ( argno, 2, 2 ) ) return ;
  val_set ( val, arg[0], arg[1] );
break;
case C_SET:
  if ( arglim ( argno, 2, 2 ) ) return ;
  val_setval ( val, arg[0], arg[1] );
break;
case C_SPSET:
  if ( arglim ( argno, 5, 3 ) ) return ;
  if ( argno < 4 ) arg[3] = NO_CHANGE;
  if ( argno < 5 ) arg[4] = TRUE;
  val_spaceval ( arg[0], arg[1], arg[2], arg[3], arg[4] );
break;
case C_CSET:
  if ( arglim ( argno, 5, 3 ) ) return ;
  if ( argno < 4 ) arg[3] = DO_CHANGE;
  if ( argno < 5 ) arg[4] = NO_FIXCHANGE;
  val_spaceval ( arg[0], arg[1], arg[2], arg[3], arg[4] );
break;
case C_PG:
  if ( arglim ( argno, 2, 2 ) ) return ;
  pv_init ( arg[0], arg[1] );
break;
case C__NECHO:
  fprintf ( outfile, "I have %i arguments.\n", argno );
```

```
      while ( argno-- > 0 ) {
        fprintf ( outfile, "Arg. #%i is %i.\n", argno, arg[argno] );
      };
    break;
    default:
      NO_IMP ;
    break;
  }
  return ;
}

/* End of lex.l */
```

Chapter 7

Auxiliary programmes

```
/* (C) 1998: Hans Georg Schaathun <georg@ii.uib.no> */
/* $Id: pgc.c,v 5.0 1998/12/20 00:24:50 georg Exp $ */

/* main function for pgc(1)                        **
** pgc creates a projective geometry and stores all **
** incidence relations.                            */

#include "common.h"


#include "pginit.h"
#include "points.h"
#include "aux.h"
#include "prep.h"

#define ARGS        "dhk:m:n:o:q:v"
#define PGC_OUTFILE "pgc:output"
#ifdef DEBUG
#  define PGC_DEBUG_FILE  "pgc:debugfile"
#endif

/* Global Variables */
char *MyName, *MyPathName,       /* The programme name from argument no. 0 */
     *UserName, *HostName,       /* Environment. */
     *RealName = PGC_NAME,       /* The real name of the programme */
      CallString[CALL_STRING_L], /* The entire command call with arguments */
     *MyVersion = GEN_VERSION,   /* Programme version number. */
     *CompileDate = GEN_COMPILE, /* Date of compilation. */
     *MailTo,
     *SignalBackUpFile = DEF_SIGBFILE,
      timeStart[TIME_STRING_L];
t_pdim  k=4, q=2;                /* PG parameters */
int   lfd;

int main ( int argc, char *argv[] ) {
  extern char *optarg;
  extern int optind, opterr, optopt;
  signed char m=0, f, i, c = 0, *arg;
  t_pdim k=4, q=2;
  signed char mode_d = 0;
  char  *OutFile = PGC_OUTFILE, coord[12], *S_File = NULL;
  char   filename[FILE_NAME_L];
  FILE  *OutStream;
```

```
  t_pg   *pg;
  t_msp *sp;
  t_s    *s;
  t_bas *bas;

  MyName = *argv;

  while ( −1 != ( f = getopt( argc, argv, ARGS ) ) )
    switch (f) {
    case 'q': q = atoi(optarg); break;
    case 'k': k = atoi(optarg); break;
    case 'o': OutFile = optarg; break;
    case 'd': ++mode_d; break;
    case 'h': PrintHelp();     exit (1); break;
    case 'v': PrintVersion(); exit (1); break;
    default:  ErrorUsage();   exit (1);
    }

  if ( mode_d ) exit ( pg_dimprint ( stdout, k, q ) );

  pg = pg_init ( k, q );
  OutStream = fileopen ( OutFile, "w" );
  pg_dump ( OutStream, pg );
  fclose ( OutStream );
  pg_del ( pg );
  exit ( 0 );
}

/* End of pgc.c */
```

```
/* (C) 1998: Hans Georg Schaathun <georg@ii.uib.no> */
/* $Id: vdg.c,v 5.0 1998/12/20 00:26:23 georg Exp $ */

#include "common.h"

#include "aux.h"
#include "pg.h"
#include "val.h"
#include "vtree.h"

#define ARGS          "CghPv"

/* Global Variables */
char *MyName, *MyPathName,        /* The programme name from argument no. 0 */
     *UserName, *HostName,        /* Environment. */
     *RealName = GEN_NAME,        /* The real name of the programme */
      CallString[CALL_STRING_L], /* The entire command call with arguments */
     *MyVersion = GEN_VERSION,    /* Programme version number. */
     *CompileDate = GEN_COMPILE, /* Date of compilation. */
     *MailTo, *StatusFile = VAL_STATUSFILE,
      timeStart[TIME_STRING_L];
t_vt   *vt = NULL;
t_pdim  pg_dim=4, pg_order=2;    /* PG parameters */
t_pg   *pg = NULL;
t_val *val = NULL;

int main ( int argc, char *argv[] ) {

/* External variable declarations. */
  extern char *optarg;                     /* for getopt(3) */
  extern int optind, opterr, optopt;       /* for getopt(3) */
```

```
  extern t_vt   *vt;                    /* Data structure */
  extern t_pg   *pg;                    /* Data structure */
  extern t_val *val;
  extern unsigned long v_allocn;

/* Internal variable declarations. */
  char f, mode = 0;
  char *OutFile = GEN_OUTFILE, *InFile = NULL, FileName[FILE_NAME_L];
  FILE *InStream;

  v_allocn = 1; /* Allocate memory for one assignment at a time. */

/* Parsing argument 0. */
  MyPathName = *argv;
  MyName = basename ( MyPathName );

/* Option parsing. */
  while ( -1 != ( f = getopt( argc, argv, ARGS ) ) )
    switch (f) {
    case 'C': case 'P': case 'g':
      if ( mode != 0 ) ErrEx ( ERR_FLAGS );
      mode = f;
      break;
    case 'h': PrintHelp();    exit (1); break;
    case 'k': pg_dim = atoi(optarg); break;
    case 'q': pg_order = atoi(optarg); break;
    case 'v': PrintVersion(); exit (1); break;
    default:  ErrorUsage();   exit (1);
    } /* switch */

/* Argument parsing. */
  if ( argc > optind ) InFile = argv[optind++];
  if ( argc > optind ) OutFile = argv[optind++];
  if ( argc > optind ) ErrorUsage () ;

/* Working */
  switch ( mode ) {
  case 'P': /* Print info about the file */
    if ( NULL == InFile ) ErrorUsage() ;
    InStream = fileopen ( InFile, "rb" );
    vt_finfo ( InStream );
    fclose ( InStream );
    exit ( 0 );
    break;
  case 'g':  /* Read a valex file and classify the assignments. */
    GetData ( NULL, NULL );
    InStream = ( NULL == InFile ? stdin : fileopen ( InFile, "r" ) );
    while ( EOF != val_assign ( val, InStream ) )
      val_classprint ( val, stdout );
    StatusReport ( val );
    fclose ( InStream );
    exit ( 0 );
    break;
  case 'C':  /* Translate the output from gen(1). */
    if ( NULL == InFile ) ErrorUsage () ;
    GetData ( InFile, NULL );
    vt_printbyclass ( OutFile, vt );  /* ASCII output */
    vt_del ( vt );
    exit ( 0 );
    break;
  };
```

```
  ErrorUsage ();
  exit ( 255 );
}

/* end of vdg.c */
```

```
/* (C) 1998: Hans Georg Schaathun <georg@ii.uib.no> */
/* $Id: pgtest.c,v 5.0 1998/12/20 00:25:05 georg Exp $ */

#include "common.h"
#include "pginit.h"
#include "points.h"
#include "aux.h"
#include "gf.h"

#define ARGS            "b:c:C:ghk:n:q:stvwxyz"

/* Global Variables */
char *MyName, *MyPathName,       /* The programme name from argument no. 0 */
     *UserName, *HostName,       /* Environment. */
     *RealName = PGT_NAME,       /* The real name of the programme */
     CallString[CALL_STRING_L], /* The entire command call with arguments */
     *MyVersion = GEN_VERSION,   /* Programme version number. */
     *CompileDate = GEN_COMPILE, /* Date of compilation. */
     *MailTo,
     *SignalBackUpFile = DEF_SIGBFILE,
     timeStart[TIME_STRING_L];
unsigned char k=4, q=2, m=0, n=0;

int main ( int argc, char *argv[] ) {
  extern char *optarg;
  extern int optind, opterr, optopt;
  signed char m=0, f, i, gea, geb, r, s, modeval;
  char *modestr, coord[12], mode = 0, *infile, *outfile;
  FILE *stream;
  t_ps  pt;
  t_pg *pg;
  t_bas *bas;
  t_msp *sp;

  MyName = *argv;

/* Parsing options */
  optarg = 0;
  while ( -1 != ( f = getopt( argc, argv, ARGS ) ) )
    switch (f) {
    case 'q': q = atoi(optarg); break;
    case 'k': k = atoi(optarg); break;
    case 'c':
      if ( mode ) ErrEx ( ERR_FLAGS ) ;
      mode = f;
      modestr = optarg;
      break;
    case 'n': case 'b': case 'C':
      if ( mode ) ErrEx ( ERR_FLAGS ) ;
      mode = f;
      modeval = atoi(optarg);
      break;
    case 'h': PrintHelp(); exit (1); break;
    case 'v': PrintVersion(); exit (1); break;
    case 'g': case 's': case 't': case 'x': case 'y': case 'z': case 'w':
```

```
    if ( mode ) ErrEx ( ERR_FLAGS ) ;
    mode = f;
    break;
  default: ErrorUsage(); exit ( 1 ) ;
  }

/* Executing */
 switch ( mode ) {
 case 'x': case 'y': case 'z': case 'w':
  /* Translate a file from pgc to readable output (different formats) */
  infile = argv[optind++];
  outfile = argv[optind++];
  if ( stream = fopen ( infile, "rb" ) ) {
    pg = pg_load ( stream );
    fclose ( stream );
  } else {
    ErrEx ( ERR_FOPEN );
  };
  if ( stream = fopen ( outfile, "wb" ) ) {
    switch ( mode ) {
      case 'w': pg_pprint (stream,pg); break;
      case 'x': pg_print (stream,pg); break;
      case 'y': pg_dump (stream,pg); break;
      case 'z': pg_sprint (stream,pg); break;
    }
    fclose ( stream );
  } else {
    ErrEx ( ERR_FOPEN );
  };
  exit ( 0 );
 case 'g': /* Test Galois Field arithmetrics. */
  gea = atoi ( argv[optind++] );
  geb = atoi ( argv[optind++] );
  printf ( "%i*%i=%i; %i, %i\n", gea, geb, gf_mult ( gea, geb, q ),
          gf_inv ( gea, q ), gf_inv ( geb, q ) );
  exit ( 0 );
 case 't': /* Test the pg_theta() function */
  r = atoi ( argv[optind++] );
  s = atoi ( argv[optind++] );
  printf ( "\\theta ( %i, %i; %i, %i ) = %i\n", r,s,k,q, pg_theta(r,s,k,q) );
  exit ( 0 );
 case 'c': /* Test the mapping between representations of points. */
  pt_fromstr ( (t_pt)coord, modestr );
  printf ( "Coordinates: ( " );
  for ( i = 0; i < k; ++ i ) printf ( "%i, ", coord[i] );
  printf ( "%i )\n", coord[k] );
  printf ( "Point no. %i\n", pt_getno ( (t_pt)coord, k, q ) );
  exit ( 0 );
 case 'n': /* Test the pg_elno() functions. */
  printf ( "\\phi ( %i; %i, %i ) = %i\n", modeval,k,q, pg_elno(q,k,modeval) );
  exit ( 0 );
 case 's': case 'b': /* Test bas_comb() and msp_init() */
  m = argc - optind - 1 ;
  bas = bas_init ( m );

  while ( optind < argc ) {
    pt_fromstr ( (t_pt)coord, argv[optind++] );
    bas->bas[m--] = pt_getno ( (t_pt)coord, k, q );
  };

  if ( mode == 's' ) {
    sp = msp_init ( bas, k, q );
```

```
      msp_print ( sp, stdout, k, q );
    } else {
      pt = bas_comb ( bas, modeval, k, q ) ;
      pt_fromno ( (t_pt)coord, pt, k, q );
      printf ( "%i ", pt );
      pt_print ( (t_pt)coord, stdout, k, q );
      putchar ( '\n' );
    };

    exit ( 0 );
  case 'C': /* Test the mapping between representations of points. */
    pt_fromno ( (t_pt)coord, modeval, k, q );
    printf ( "Coordinates: ( " );
    for ( i = 0; i < k; ++ i ) printf ( "%i, ", coord[i] );
    printf ( "%i )\n", coord[k] );
    exit ( 0 );
  };

  exit ( 1 );  /* We are not supposed to reach this line. */
}

/* End of pgtest.c */
```

Chapter 8

PID check

```
/* (C) 1998: Hans Georg Schaathun <georg@ii.uib.no> */
/* $Id: pidcheck.c,v 5.0 1998/12/20 00:25:17 georg Exp $ */

/* The pidcheck(1) programme reads an active file, like that left  **
** by gen(1), and checks whether the programme is still running.   **
** If it is not, it restarts it.                                    */

#include <sys/types.h>
#include <signal.h>
#include <sys/procset.h>
#include <stdio.h>
#include "common.h"
#include "gen.h"
#include "logenv.h"

#define MAX_ARG 12

int main ( int argc, char *argv[] ) {
  char CallString[CALL_STRING_L],
       *BackUpFile,
        BUFbuf[FILE_NAME_L], BBackUpFile[FILE_NAME_L];
  char *genargv[MAX_ARG], *actfile, *dir;
  char verbose = 0, i;
  FILE *f;
  extern char *gen_host, *gen_arg, *gen_gen, *gen_flags, *gen_bc;
  extern pid_t gen_pid;

  /* Argument parsing */
  if ( 0 == strcmp ( "-v", argv[1] ) ) { ++verbose; --argc; ++argv; }
  if ( 2 != argc ) {
    fprintf ( stderr, "Usage: %s [-v] directory.\n", argv[0] );
    exit ( ENO_USE );
  }

  /* Finding the directory and the active file. */
  actfile = GEN_ACTFILE;
  dir = argv[1];
  chdir ( dir );

  /* Read the active file. */
  switch ( envfile ( actfile ) ) {
    case ENO_MEM: ErrEx ( ERR_MEM ); break;
    case ENO_FOPEN: ErrEx ( ERR_FOPEN ); break;
  };
```

```
  if ( NULL == gen_gen ) { ErrEx ( ERR_DATA ); }; /* Sanity Check */

  /* Check if the process is running, by sending SIGNAL 0. */
  if ( 0 == sigsend ( P_PID, gen_pid, 0 ) ) {
    /* Diagnostic messages and exit. */
    if ( verbose ) printf ( "%s was still running.\n", gen_gen );
    exit ( ENO_OK );
  }

  /* Create the argument vector. */
  genargv[0] = gen_gen;
  if ( verbose ) printf ( "%s\n", gen_flags );
  if ( NULL != gen_flags ) {
    i = 1;
    if ( *gen_flags != 0 ) genargv[i++] = gen_flags;
    while ( *gen_flags != 0 ) {
      if ( *gen_flags == ' ' ) {
        *(gen_flags++) = 0;
        genargv[i++] = gen_flags;
      };
      ++gen_flags;
    };
  };
  genargv[i++] = GEN_RESTART;
  genargv[i] = NULL;

  /* Verbose messages */
  if ( verbose ) printf ( "Restarting %s\n", gen_gen );

  /* Find the backup file for restart. */
  if ( gen_bc == NULL )
    sprintf ( BackUpFile = BUFbuf, "%s%s", GEN_OUTFILE, DEF_BCEXT );
  else BackUpFile = gen_bc;
  sprintf ( BBackUpFile, "%s%s", BackUpFile, DEF_BCBCEXT );
  if ( NULL != ( f = fopen ( BBackUpFile, "r" ) ) ) {
    fclose ( f );
    unlink ( BackUpFile );
    rename ( BBackUpFile, GEN_RESTART );
  } else {
    rename ( BackUpFile, GEN_RESTART );
  };

  /* Diagnostic messages */
  printf ( "%s, starting:\n", argv[0] );
  for ( i = 0; genargv[i] != NULL; )
    printf ( "%s ", genargv[i++] );
  putchar ( '\n' );

  execv ( gen_gen, genargv );  /* Restart */
}

/* End of pidcheck.c */
```

```
/* (C) 1998: Hans Georg Schaathun <georg@ii.uib.no> */
/* $Id: logenv.h,v 5.0 1998/12/20 00:24:26 georg Exp $ */

/* The logenv module is primarily used by pidcheck(1). */

char envenv ( ) ;
  /* Parse the environment to get the information usually found in the **
```

```
  ** active file of gen (1).                                    */
char envfile ( char *f ) ;
  /* Parse the active file left by gen(1). */

/* End of logenv.h */
```

```
/* (C) 1998: Hans Georg Schaathun <georg@ii.uib.no> */
/* $Id: logenv.c,v 5.0 1998/12/20 00:24:24 georg Exp $ */

#include <sys/types.h>
#include <stdio.h>
#include <stdlib.h>
#include <hgd.h>
#include "errors.h"
#include "gen.h"
#include "logenv.h"

#define FIVE     "     "
#define ENV_SPACE "\0" FIVE FIVE FIVE FIVE
#define TAB      '\t'
#define SPACE    ' '
#define NEWLINE '\n'
#define ENV_L    CALL_STRING_L

char *gen_host, *gen_arg, *gen_gen, *gen_cmd, *gen_user, *gen_bc, *gen_flags,
     *UserName;
pid_t gen_pid;

char envfile ( char *f ) {
  FILE *s;
  static char *e, c;
  short i;

  if ( NULL == ( s = fopen ( f, "r" ) ) ) return ( ENO_FOPEN );
  i = 0;
  while ( EOF != ( c = getc ( s ) ) ) {
    if ( 0 == i ) e = malloc ( ENV_L );
    switch ( c ) {
      case NEWLINE:
        e[i] = 0;
        if ( 0 != putenv ( e ) ) return ( ENO_MEM ) ;
        i = 0;
      break;
      default:
        e[i++] = c;
    }
  }
  envenv ();
}

char envenv ( ) {
  char *e;
  UserName = getenv ( "USER" );
  gen_host = getenv ( "GEN_HOST" );
  gen_user = getenv ( "GEN_USER" );
  gen_cmd = getenv ( "GEN_CMD" );
  gen_gen = getenv ( "GEN_GEN" );
  gen_bc = getenv ( "GEN_BC" );
  gen_flags = getenv ( "GEN_FLAGS" );
  if ( NULL == UserName ) {
    char s[32];
```

```
    sprintf ( s, "USER=%s", gen_host );
    putenv ( s );
  };
  gen_pid = ( NULL == ( e = getenv ( "GEN_PID" ) ) ? -1 : atoi ( e ) ) ;
}

/* End of logenv.c */
```

Index