

Gareth Evans | IPM 4097 Project | 2001-2002

Appendix B: Source Code

Table of Contents

Page	Description
B3	<i>frmon.h</i>
B6	<i>utils.h</i>
B7	<i>utils.c</i>
B13	<i>rwsa.h / rwse.h</i>
B14	<i>rwsa.c / rwse.c</i>
B30	<i>kba.c</i>
B33	<i>kbe.c</i>

Notes:

- **frmon.h** contains the definitions of the functions and variables that are available in the library;
- **utils.h** contains the definitions for the functions that are implemented in **utils.c**;
- **rwsa.h / rwse.h** contains the definitions for the functions that are implemented in **rwsa.c / rwse.c**.

frmon.h

```
1  #ifndef _FMON_H_
   #define _FMON_H_

   #include "basic.h"

5  typedef Pointer FMon;

   typedef struct
10  {
       FMon lft;
       FMon rt;
   }
   FMonPair;

15  typedef struct
   {
       FMon lft;
       FMon mid;
       FMon rt;
20  }
   FMonTriple;

   typedef struct fmonlst
   {
25  FMon first;
       struct fmonlst *rest;
   }
   *FMonList;

30  #ifdef OLD_CODE
   typedef struct fmonpairlst
   {
       FMonPair first;
       struct fmonpairlst *rest;
35  }
   *FMonPairList;
   #endif

   typedef struct fmonpairlst
40  {
       FMon lft;
       FMon rt;
       struct fmonpairlst *rest;
   }
45  *FMonPairList;

   typedef Bool (*OrdFun) (FMon, FMon);
   typedef String (*GenFun) (String);
   typedef ULong (*ExpFun) (ULong);

50  #define fMonListNul (FMonList) NULL
   #define fMonPairListNul (FMonPairList) NULL

   extern OrdFun theOrdFun;

55  extern FMon fMonGen (String);
   extern FMon fMonLCLF (FMon, FMon);
   extern FMon fMonLCRF (FMon, FMon);
   extern FMon fMonLDiv (FMon, FMon);
60  extern FMon fMonLDivIfCan (FMon, FMon, Short *);
   extern FMon fMonLMul (String, FMon);
   extern FMon fMonLeadPowFac (FMon);
   extern FMon fMonMapExp (ExpFun, FMon);
   extern FMon fMonMapGen (GenFun, FMon);
65  extern FMon fMonOne (void);
   extern FMon fMonPow (FMon, ULong);
   extern FMon fMonPrefix (FMon, ULong);
   extern FMon fMonRDiv (FMon, FMon);
   extern FMon fMonRDivIfCan (FMon, FMon, Short *);
70  extern FMon fMonRMul (FMon, String);
   extern FMon fMonRest (FMon);
   extern FMon fMonSingle (String, ULong);
   extern FMon fMonSubWord (FMon, ULong, ULong);
   extern FMon fMonSubWordLen (FMon, ULong, ULong);
75  extern FMon fMonSubst (FMon, FMon, FMon);
   extern FMon fMonSuffix (FMon, ULong);
   extern FMon fMonTailFac (FMon);
   extern FMon fMonTimes (FMon, FMon);
   extern FMon parseStrToFMon (String);
80  extern FMonPair fMonDivIfCan (FMon, FMon, Short *);
   extern FMonTriple fMonOverlap (FMon, FMon);
```

```

extern String fMonLeadVar (FMon);
extern String fMonToParseStr (FMon);
85 extern String fMonToStr (FMon);

extern Bool fMonEqual (FMon, FMon);
extern Bool fMonIsOne (FMon);

90 extern ULong fMonLeadExp (FMon);
extern ULong fMonLength (FMon);
extern ULong fMonNumVars (FMon);

extern Bool fMonListEqual (FMonList, FMonList);
95 extern FMonList fMonFXListRem (FMonList, FMon);
extern FMonList fMonListAppend (FMonList, FMonList);
extern FMonList fMonListCopy (FMonList);
extern FMonList fMonListFXRev (FMonList);
extern FMonList fMonListPush (FMon, FMonList);
100 extern FMonList fMonListRemDups (FMonList);
extern FMonList fMonListRev (FMonList);
extern FMonList fMonListSingle (FMon);
extern ULong fMonListLength (FMonList);

105 extern Bool fMonPairListEqual (FMonPairList, FMonPairList);
extern FMonPairList fMonFXPairListRem (FMonPairList, FMonPair);
extern FMonPairList fMonPairListAppend (FMonPairList, FMonPairList);
extern FMonPairList fMonPairListCopy (FMonPairList);
extern FMonPairList fMonPairListFXRev (FMonPairList);
110 extern FMonPairList fMonPairListPush (FMon, FMon, FMonPairList);
extern FMonPairList fMonPairListRemDups (FMonPairList);
extern FMonPairList fMonPairListRev (FMonPairList);
extern FMonPairList fMonPairListSingle (FMon, FMon);
extern ULong fMonPairListLength (FMonPairList);

115 extern Bool fMonTLex (FMon, FMon);

/* begin Bangor extensions */

120 typedef struct intfmonlst
{
    int i;
    FMon mon;
    struct intfmonlst *rest;
125 }
*IntFMonList;

#define intFMonListNul (IntFMonList) NULL

130 typedef struct loglst
{
    FMon lft;
    FMon rt;
    IntFMonList logger;
135 struct loglst *rest;
}
*LogList;

#define logListNul (LogList) NULL

140 typedef struct ifmp
{
    int ilft;
    FMon mlft;
145 int irt;
    FMon mrt;
    struct ifmp *rest;
}
*IntFMonPairList;

150 #define intFMonPairListNul (IntFMonPairList) NULL;

extern Bool intFMonPairListEqual (IntFMonPairList, IntFMonPairList);
extern IntFMonPairList intFMonMonPairListRev (IntFMonPairList);
155 extern IntFMonPairList intFMonFXPairListRem (IntFMonPairList, int, FMon, int, FMon);
extern IntFMonPairList intFMonPairListAppend (IntFMonPairList, IntFMonPairList);
extern IntFMonPairList intFMonPairListCopy (IntFMonPairList);
/* FX on 1st argument */
extern IntFMonPairList intFMonPairListFXAppend (IntFMonPairList, IntFMonPairList);
160 extern IntFMonPairList intFMonPairListFXRev (IntFMonPairList);
extern IntFMonPairList intFMonPairListFXSetLast (IntFMonPairList, int, FMon, int, FMon);
extern IntFMonPairList intFMonPairListPush (int, FMon, int, FMon, IntFMonPairList);
extern IntFMonPairList intFMonPairListRemDups (IntFMonPairList);
extern ULong intFMonPairListLength (IntFMonPairList);

165 extern Bool intFMonListEqual (IntFMonList, IntFMonList);
extern IntFMonList intMonListRev (IntFMonList);
extern IntFMonList intFMonFXListRem (IntFMonList, int, FMon);

```

```
extern IntFMonList intFMonListAppend (IntFMonList, IntFMonList);
170 extern IntFMonList intFMonListCopy (IntFMonList);
    /* FX on 1st argument */
extern IntFMonList intFMonListFXAppend (IntFMonList, IntFMonList);
extern IntFMonList intFMonListFXRev (IntFMonList);
extern IntFMonList intFMonListFXSetLast (IntFMonList, int, FMon);
175 extern IntFMonList intFMonListPush (int, FMon, IntFMonList);
extern IntFMonList intFMonListRemDups (IntFMonList);
extern ULong intFMonListLength (IntFMonList);

extern Bool logListEqual (LogList, LogList);
180 extern LogList logListRev (LogList);
extern LogList logFXListRem (LogList, FMon, FMon, IntFMonList);
extern LogList logListAppend (LogList, LogList);
extern LogList logListCopy (LogList);
    /* FX on 1st argument */
185 extern LogList logListFXAppend (LogList, LogList);
extern LogList logListFXRev (LogList);
extern LogList logListFXSetLast (LogList, FMon, FMon, IntFMonList);
extern LogList logListPush (FMon, FMon, IntFMonList, LogList);
extern LogList logListRemDups (LogList);
190 extern ULong logListLength (LogList);

/* end Bangor extensions */

#endif /* _FMON_H_ */
```

utils.h

```
1  /*
   * File: utils.h
   * Author: Gareth Evans (modified from original work by CDW)
   * Last Modified: 15th April 2002
5  */

   int fMonListDisplay();
   int fMonPairListDisplay();
   int intFMonListDisplay();
10  int logListDisplay();
   int intFMonPairListDisplay();

   Bool fMonListIsMember();
   Bool fMonPairListIsMember();
15  Bool intFMonListIsMember();

   int fMonListPosition();

   int strlen2();
20  int getline();
   FMon fMonFromStr();
   int intFromStr();

   int firstLineInt();
25  FMonList secondLineFMons();
   IntFMonList remainingLines();

   FMonList fMonListFromFile();
   FMonPairList fMonPairListFromFile();
30  IntFMonList intFMonListFromFile();
   LogList logListFromFile();
   IntFMonPairList intFMonPairListFromFile();
```

utils.c

```
1  /*
   * File: utils.c
   * Author: Gareth Evans (modified from original work by CDW)
   * Last Modified: 16th April 2002
5  */

#define MAXLINE 100

/* ===== list display functions ===== */
10 int
fMonListDisplay( L ) // print out the words in an FMonList
FMonList L;
{ FMon w;
15  FMonList K = fMonListCopy( L );
  while( K )
  { w = K -> first;
    printf( "%s\n", fMonToStr(w) );
    K = K -> rest;
20  }
  return 0;
}

int
25 fMonPairListDisplay( L ) // print out the word pairs in an FMonPairList
FMonPairList L;
{ FMonPairList K = fMonPairListCopy( L );
  while( K )
  {
30  printf( "(%s -> %s)\n", fMonToStr(K -> lft), fMonToStr(K -> rt) );
    K = K -> rest;
  }
  return 0;
}
35 }

int
intFMonListDisplay( L ) // print out integers and words in an IntFMonList
// (equivalence class version)
IntFMonList L;
40 { IntFMonList K = intFMonListCopy( L );
  while( K )
  { printf( "[%i, %s]\n", K -> i, fMonToStr(K -> mon) );
    K = K -> rest;
  }
45  return 0;
}

int
logListDisplay( L ) // print out logged rewrite rules in a LogList
50 LogList L;
{ LogList K = logListCopy( L );
  while( K )
  {
55  printf( "%s -> %s by ", fMonToStr(K -> lft), fMonToStr(K -> rt) );
    intFMonListDisplay(K -> logger);
    K = K -> rest;
  }
  return 0;
}
60 }

int
intFMonPairListDisplay( L ) // print pairs of integers and words
IntFMonPairList L;
{ IntFMonPairList K = intFMonPairListCopy( L );
65  while( K )
  { printf( "(%i, %s) -> (%i, %s)\n", K -> ilft, fMonToStr(K -> mlft),
    K -> irt, fMonToStr(K -> mrt) );
    K = K -> rest;
70  }
  return 0;
}

/* ===== list membership functions ===== */
75 Bool
fMonListIsMember( w, L ) // test whether w in L
FMonList L;
FMon w;
80 { Short flag = 0;
  FMonList K = fMonListCopy( L );
  while( ( K ) && ( flag == 0 ) )
```

```

    {
85     if ( fMonEqual( w, K -> first ) == 1 )
        { flag = 1;
          }
        K = K -> rest;
    }
90 return flag;
}

Bool
fMonPairListIsMember( l, r, L ) // test whether (l, r) in L
FMonPairList L;
95 FMon l, r;
{ Short flag = 0;
  FMonPairList K = fMonPairListCopy( L );
  while( ( K ) && ( flag == 0 ) )
    {
100     if ( ( fMonEqual( l, K -> lft ) == 1 ) && ( fMonEqual( r, K -> rt ) == 1 ) )
        { flag = 1;
          }
        K = K -> rest;
    }
105 return flag;
}

Bool
intFMonListIsMember( i, m, L ) // test whether [i, m] in L
110 intFMonList L;
int i;
FMon m;
{ Short flag = 0;
  intFMonList K = intFMonListCopy( L );
115 while( ( K ) && ( flag == 0 ) )
    {
        if ( ( K -> i == i ) && ( fMonEqual( m, K -> mon ) == 1 ) )
            { flag = 1;
              }
    }
120 K = K -> rest;
}
return flag;
}

125 /* ===== list position functions ===== */

int
fMonListPosition( w, L ) // position of w in L
FMonList L;
130 FMon w;
{ Short flag = 0;
  int pos = 0;
  FMonList K = fMonListCopy( L );
  if ( fMonListLength( L ) == 0 )
    { return 0; };
135 while( ( K ) && ( flag == 0 ) )
    { pos++;
      if ( fMonEqual( w, K -> first ) == 1 )
        { flag = 1;
          }
    }
140 K = K -> rest;
}
if ( flag == 0 )
  return 0;
145 else
  return pos;
}

/* ===== functions for input from file =====*/
150
int
strlen2(s)
char s[];
{ int i = 0;
155 while (s[i] != '\0') i++;
  return(i);
}

int
160 getline( infil, s, lim ) /* get line from file into string s, return length */
FILE *infil;
char s[];
int lim;
{ int c, i, k;
165 for (i = 0; (i < lim-1) && ((c = fgetc(infil)) != -1) && (c != '\n'); ++i)
    { s[i] = c; }
  if (c == '\n')
    { s[i] = c; ++i; }
}

```

```

    s[i] = '\0';
170     return i;
    }

FMon
fMonFromStr( s, j, pk ) /* routine to pick an FMon from a character string */
175     char s[];
    int j, *pk;
    { char c=' ', a[MAXLINE];
      int i = 0, k = j;

180     while (c != ';')
        { c = s[k];
          if (c == ';')
            { a[i] = '\0';
            }
185         else
            { a[i] = c;
              i++;
            }
          k++;
190     }
    *pk = k;
    // printf("k = %i, a = [%s]\n",k,a);
    return parseStrToFMon( a );
195 }

int
intFromStr( s, j, pk ) /* routine to pick an integer from a string */
    char s[];
    int j, *pk;
200     { char c = ' ';
      int i = 0, n = 0, sign = 1, k = j;

      c = s[k];
      while (c == ' ')
205         { k++;
          c = s[k];
        }
      if (c == '+')
        { k++; c = s[k]; }
210     else if (c == '-')
        { sign = -1; k++; c=s[k]; }
      while (c != ',')
        {
215         if ( (c >= '0') && (c <= '9') )
            { n = 10*n + c - '0'; k++; c = s[k]; }
        }
      *pk = k;
      return sign*n;
220     }

/*
 * Gareth Evans' Extensions
 */

225     int
firstLineInt( infil ) // Routine to read the first line of the file
    FILE *infil;
    {
230     char s[1024], c;
      int a = -1, j = 0, k = 0;

      // Get the first line from the file and find its length
      getline( infil, s, MAXLINE );

235     // Get the integer from the first line of the file and return it
      a = intFromStr(s, j, &k);
      return a;
    }

240     FMonList
secondLineFMons( infil, count ) // Routine to read images of M generators
    FILE *infil;
    int count;
    {
245     FMonList back;
      FMon push;
      char s[1024], c;
      int i = 0, j = 0, k = 0, len = 0;

250     // Initialise our list
      back = fMonListNul;
      // printf("Now displaying the empty fMonList: ");
      // fMonListDisplay(back); printf("\n");

```

```

255 // Read in the second line (assuming that the first has been read in)
getline( infil, s, MAXLINE );
len = strlen2(s);
// printf("length = %i, string = %s", len, s);

260 // Add images to list
for(i = 1; i <= count; i++)
{
    push = fMonFromStr(s, j, &k);
    // printf("\nFMon added = %s", fMonToStr(push));
265    back = fMonListPush(push, back);
    j = k+1;
    // printf("Now displaying the fMonList: ");
    // fMonListDisplay(back); printf("\n");
}
270 return fMonListFXRev( back );
}

IntFMonList
remainingLines( infil, count ) // Routine to read integers
275 FILE *infil;
int count;
{
    IntFMonList back, link;
    FMon empty;
280 char s[1024], c;
    int i = 0, j = 0, k = 0, len = 0, olen = 0, process = 0;

    // Initialisation
    back = intFMonListNul;
285 empty = parseStrToFMon("z");
    // printf("Now displaying the empty fMonList: ");
    // intFMonListDisplay(back); printf("\n");

    // Add images to list
290 for(i = 1; i <= count; i++)
    {
        link = intFMonListNul;
        j = 0; k = 0; len = 0;
        len = getline( infil, s, MAXLINE ) - 1;
295 // len = strlen2(s);
        olen = len;
        // printf("length of line %i = %i, string = %s", i, len, s);

        // Get elements
300 while (len > 0)
        {
            process = intFromStr( s, j, &k );
            j = k+1;
            // printf("Original length = %i, ", len);
305 len = olen-j-1;
            // printf("New length = %i\n", len);

            // Push element onto intFMonList
            link = intFMonListPush( process, empty, link );
310 // printf("New List = \n");
            // intFMonListDisplay( link );
        }

        // Now reverse our link list and add it onto the return list
315 link = intFMonListFXRev( link );
        back = intFMonListAppend( back, link );
    }
    return back;
}
320
/*
* End of Gareth Evans' Extensions
*/

325 FMonList
fMonListFromFile( infil ) /* routine to read words from one line in file */
FILE *infil;
{
    FMon w;
330 FMonList words;
    char s[1024], c;
    int i = 0, j = 0, k = 0, len = 0;

    words = fMonListNul;
    getline( infil, s, MAXLINE );
    len = strlen2(s);
    // printf("in fMonListFromFile: len = %i, s = %s",len,s);
    while ( j < len )
    {
340 w = fMonFromStr( s, j, &k ); j = k+1;

```

```

        // printf( "j = %i, w = %s\n", j, fMonToStr(w));
        words = fMonListPush( w, words );
    }
345 }

FMonPairList
fMonPairListFromFile( infil ) /* routine to read string pairs from disk */
FILE *infil;
350 {
    FMon wlft, wrt;
    FMonPairList pairs;
    char s[1024], c;
    int i = 0, j = 0, k = 0, len = 1;
355
    pairs = fMonPairListNul;
    while (len > 0)
    {
        getline( infil, s, MAXLINE );
360 len = strlen2(s);
        // printf("len = %i, s = %s",len,s);
        if ( len > 0 )
        {
            j = 0;
            wlft = fMonFromStr( s, j, &k ); j = k+1;
365 // printf( "j = %i, wlft = %s\n", j, fMonToStr(wlft));
            wrt = fMonFromStr( s, j, &k ); j = k+1;
            // printf( "j = %i, wrt = %s\n", j, fMonToStr(wrt));
            pairs = fMonPairListPush( wlft, wrt, pairs );
        }
370 }
    return fMonPairListFXRev( pairs );
}

IntFMonList
intFMonListFromFile( infil ) /* routine to read integer and string from disk */
FILE *infil;
{
    FMon w;
    IntFMonList pairs;
380 char s[1024], c;
    int z, i = 0, j = 0, k = 0, len = 1;

    pairs = intFMonListNul;
    while (len > 0)
385 {
        getline( infil, s, MAXLINE );
        len = strlen2(s);
        // printf("len = %i, s = %s", len, s);
        if ( len > 0 )
390 {
            j = 0;
            z = intFromStr( s, j, &k ); j = k+1;
            // printf( "j = %i, z = %i\n", j, z);
            w = fMonFromStr( s, j, &k ); j = k+1;
            // printf( "j = %i, w = %s\n", j, fMonToStr(w));
395 pairs = intFMonListPush( z, w, pairs );
        }
    }
    return intFMonListFXRev( pairs );
400 }

LogList
logListFromFile( infil ) /* routine to read logged string pairs from disk */
FILE *infil;
{
405 FMon w, wlft, wrt;
    IntFMonList wlog;
    LogList lrws = logListNul;
    int h = 0, i = 0, j = 0, len = 1, z = 0, k = 0;
    char s[MAXLINE], c;
410
    while (len > 0)
    {
        getline( infil, s, MAXLINE );
        len = strlen2(s);
        // printf("len = %i, s = %s", len, s);
415 if ( len > 0 )
        {
            j = 0;
            wlft = fMonFromStr( s, j, &k ); j = k+1;
            wrt = fMonFromStr( s, j, &k ); j = k;
            c = s[j];
420 // printf("%s -> %s\n", fMonToStr(wlft), fMonToStr(wrt));
            wlog = intFMonListNul;
            while ( c != '[' )
            {
                j++; c = s[j];
            }
425 j++;
            while ( c != ']' )

```

```

        { z = intFromStr( s, j, &k ); j = k+1;
          w = fMonFromStr( s, j, &k ); j = k;
          // printf("z, w = %i,%s\n", z, fMonToStr(w));
430      wlog = intFMonListPush( z, w, wlog );
          c = s[j];
        }
        lrws = logListPush( wlft, wrt, intFMonListFXRev(wlog), lrws );
    }
435 }
    return logListFXRev( lrws );
}

IntFMonPairList
intFMonPairListFromFile( infil ) /* routine to read int,fmon pairs from disk */
FILE *infil;
{
    FMon w, wl, wr;
    int il, ir, h = 0, i = 0, j = 0, len = 1, z = 0, k = 0;
445 char s[MAXLINE], c;
    IntFMonPairList L = intFMonPairListNul;

    while ( len > 0 )
    { getline( infil, s, MAXLINE );
450     len = strlen2(s);
        if ( len > 0 )
        { j = 0;
          il = intFromStr( s, j, &k ); j = k+1;
          wl = fMonFromStr( s, j, &k ); j = k+1;
455     ir = intFromStr( s, j, &k ); j = k+1;
          wr = fMonFromStr( s, j, &k ); j = k;
          c = s[j];
          L = intFMonPairListPush( il, wl, ir, wr, L );
        }
    }
460 }
    return intFMonPairListFXRev( L );
}

```

rwsa.h / rwse.h

```
1  /*
   * File: rwsa.h / rwse.h
   * Author: Gareth Evans (modified from original work by CDW)
   * Last Modified: 15th April 2002
5  */

#include "utils.h"

FMon fMonWordReduce();
10 FMonPairList fMonRulesReduce();
FMonPairList fMonKnuthBendix();
IntFMonList intFMonListWordReduce();
IntFMonPairList intFMonScout();
IntFMonPairList shortCuts();
15 IntFMonPairList intFMonRulesReduce();
IntFMonPairList intFMonKnuthBendix();
FMonList fMonMonoidElements();
IntFMonList intFMonMonoidElements();
```

rwsa.c / rwse.c

```
1  /*
   * File: rwsa.c / rwse.c
   * Author: Gareth Evans (modified from original work by CDW)
   * Last Modified: 16th April 2002
5  *   (making use of strCopy and xListCopy)
   *   (change to i-loop structure in fMonKnuthBendix)
   */

10 #include "utils.c"

FMon
fMonWordReduce( word, rules ) // reduce word as far as possible using rules
FMon word;
FMonPairList rules;
15 {
    // Define variables
    FMon l, r, redword, result;
    FMonPair rule, div;
    FMonPairList rws;
20    Short flag;
    int numrws, pass = 0, num = 1, j = 0, check;

    redword = word; // redword is the 'reduced word'
    numrws = fMonPairListLength( rules );
25    while ( num > 0 )
    {
        num = 0; // num counts the number of substitutions made
        j = 0; // j loops through the rules, 1 <= j <= numrules
        pass++; // pass counts the number of passes made
30
        // Make copy of rules
        rws = fMonPairListCopy( rules );
        while ( j < numrws )
        {
35            j++;
            l = rws -> lft;
            r = rws -> rt;
            rws = rws -> rest;

40            // See if the rule simplifies our word
            check = 0;
            while ( check == 0 )
            {
45                num++;
                result = fMonSubst( redword, l, r );
                check = fMonEqual( redword, result );
                redword = result;
            }

50            num--;
        } // end while ( j < numrws )
    } // end while(num > 0)
    return redword;
}

55 FMonPairList
fMonRulesReduce( K ) // eliminate redundant rules
FMonPairList K;
{
60    // Define variables
    int d;
    ULong i = 0, len = 0;
    FMon pl, pr, ql, qr;
    FMonPairList L, L3, L2, L1, L0 = fMonPairListNul;
65    Bool eq;

    len = fMonPairListLength( K );
    L = fMonPairListRemDups( K ); // remove duplicates
    len = fMonPairListLength( L );
70
    i = 0;
    L2 = fMonPairListCopy( L );
    L3 = fMonPairListCopy( L ); // Assume no reductions to begin with
    while ( L2 )
75    {
        i++;
        ql = L2 -> lft;
        qr = L2 -> rt;
        L2 = L2 -> rest;
80        // L1 is the list without the current element:
        L1 = fMonPairListAppend( L0, L2 );
```

```

// Reduce left and right hand sides
85 pl = fMonWordReduce( ql, L1 );
pr = fMonWordReduce( qr, L1 );
eq = fMonEqual( pl, pr );

// If LHS = RHS, get rid of element...
90 if ( eq == 1 )
{
L3 = fMonPairListCopy( L1 );
}

// ...else keep element
95 else
{
L0 = fMonPairListPush( ql, qr, L0 );
}
} // end while ( L2 )
100 return L3;
}

FMonPairList
fMonKnuthBendix( L ) // apply the KB completion algorithm
105 FMonPairList L;
{
// Define variables
ULong leni, lenj, lenf, lenm, added = 0;
Short flag;
110 FMon li, ri, lj, rj, lf, rf, lm, rm, pf, sf,
pm, sm, w1, w2, af, bf, qf, middle, left, right;
FMonPair pair, div;
FMonPairList Li, Lj, crit, crws;
FILE *kbdata;
115 int i = 0, j = 0, d = 0, passes = 0;

crws = fMonPairListCopy( L ); // Holds the complete rewrite system computed so far
added = 1;
printf( "\nIn fMonKnuthBendix using RemDups:\n" );
120 while ( added > 0 )
{
passes++;
printf( "\nPass number %i:\n", passes );
crit = fMonPairListCopy( crws );
125 added = 0; // enable the escape
Li = fMonPairListCopy( crws );

// We now go through Li and Lj, looking for overlaps
// in each pair of LHS's
130 while ( Li )
{
Lj = fMonPairListCopy( Li );
li = Li -> lft;
ri = Li -> rt;
135 leni = fMonLength(li);
// printf("The length of %s is %i \n", fMonToStr(li), leni);

while ( Lj )
{
140 lj = Lj -> lft;
rj = Lj -> rt;
lenj = fMonLength(lj);
// printf("The length of %s is %i\n", fMonToStr(lj), lenj);

// Decide on the fixed and moving elements
if ( leni <= lenj ) // i moving, j fixed
{
lf = lj; rf = rj; lenf = lenj;
lm = li; rm = ri; lenm = leni;
150 }
else // j moving, i fixed
{
lf = li; rf = ri; lenf = leni;
lm = lj; rm = rj; lenm = lenj;
155 }
// printf( "fixed = (%s, %s), moving = (%s, %s)\n",
// fMonToStr(lf), fMonToStr(rf), fMonToStr(lm), fMonToStr(rm) );
// printf("Length of moving word = %i ", lenm);

160 //
// We now look for overlaps on the left and right
//

//
165 // STAGE 1: Moving word overlaps fixed on the left
//
// (....lf.....)
// (...lm...) ->

```

```

170 //
for (i = 1; i <= lenm-1; i++)
{
    sm = fMonSuffix( lm, i );
    pf = fMonPrefix( lf, i );
175 // printf("i = %i, comparing %s to %s ", i,
// fMonToStr(sm), fMonToStr(pf));

    if ( fMonEqual( sm, pf ) == 1 ) // if overlap found
    {
180 // printf("Overlap (1) found between %s and %s with %s ",
// fMonToStr(lm), fMonToStr(lf), fMonToStr(sm));
pm = fMonPrefix( lm, lenm-i );
sf = fMonSuffix( lf, lenf-i );

185 // Reduce word in two different ways:
w1 = fMonWordReduce( fMonTimes(rm, sf), crws );
w2 = fMonWordReduce( fMonTimes(pm, rf), crws );

// If the reductions are different, add onto list
190 if ( theOrdFun( w1, w2 ) == 1 )
{
    crit = fMonPairListPush( w2, w1, crit );
    added++;
// printf("(%s, %s)\n",fMonToStr(w2),fMonToStr(w1));
195 }
else if ( theOrdFun( w2, w1 ) == 1 )
{
    crit = fMonPairListPush( w1, w2, crit );
    added++;
200 // printf("(%s, %s)\n",fMonToStr(w1),fMonToStr(w2));
} // end if ( fMonEqual( sm, pf ) == 1 )
} // for (i = 1; i <= lenm-1; i++)

205 //
// STAGE 3: Moving word overlaps fixed on the right
//
//          (.....lf.....)
//          <- (...lm...)
210 //
for (i = 1; i <= lenm-1; i++)
{
    pm = fMonPrefix( lm, i );
    sf = fMonSuffix( lf, i );
215 if ( fMonEqual( pm, sf ) == 1 ) // if overlap found
    {
// printf("Overlap (3) found between %s and %s with %s ",
// fMonToStr(lm), fMonToStr(lf), fMonToStr(pm));
sm = fMonSuffix( lm, lenm-i );
220 pf = fMonPrefix( lf, lenf-i );

// Reduce word in two different ways
w1 = fMonWordReduce( fMonTimes(rf, sm), crws );
w2 = fMonWordReduce( fMonTimes(pf, rm), crws );
225 // If the reduced words are different
if ( theOrdFun( w1, w2 ) == 1 )
{
    crit = fMonPairListPush( w2, w1, crit );
    added++;
230 // printf("(%s, %s)\n",fMonToStr(w2),fMonToStr(w1));
}
else if ( theOrdFun( w2, w1 ) == 1 )
{
235     crit = fMonPairListPush( w1, w2, crit );
    added++;
// printf("(%s, %s)\n",fMonToStr(w1),fMonToStr(w2));
}
} // end if ( fMonEqual( pm, sf ) == 1 )
240 } // end for (i = 1; i <= lenm-1; i++)

//
// STAGE 2: Moving word as substring
//
245 //          (.....lf.....)
//          (...lm...) ->
//

if (fMonEqual(lm, lf) != 1) // if the left hand sides are not equal
250 {
    for ( j = 0; j <= lenf-lenm; j++ )
    {
        middle = fMonSubWord(lf, j+1, j+lenm);
        if (fMonEqual(middle, lm) == 1) // if overlap found
    }
}

```

```

255         {
            // printf("Overlap (2) found between %s and %s with %s",
            // fMonToStr(lm), fMonToStr(lf), fMonToStr(middle));

            // Check for existence of elements on left and right
260         if( j != 0 )
            { left = fMonPrefix(lf, j); }
            else left = fMonOne();

            if( j != lenf-lenm )
265         { right = fMonSuffix(lf, lenf-lenm-j); }
            else right = fMonOne();

            // Reduce word
270         w2 = fMonTimes(fMonTimes(left, rm), right);
            w2 = fMonWordReduce( w2, crws );

            // If the reduced words are different
            if ( theOrdFun( rf, w2 ) == 1 )
275         {
                crit = fMonPairListPush( w2, rf, crit );
                added++;
                // printf("(%s, %s)\n",fMonToStr(w2),fMonToStr(rf));
            }
            else if ( theOrdFun( w2, rf ) == 1 )
280         {
                crit = fMonPairListPush( rf, w2, crit );
                added++;
                // printf("(%s, %s)\n",fMonToStr(rf),fMonToStr(w2));
            }
285         } // end if (fMonEqual...)

        } // end for (j)

    } // end if (fMonEqual...)
290     Lj = Lj -> rest; // cycle through Lj

    } // end while ( Lj )

295     Li = Li -> rest; // cycle through Li

    } // end while ( Li )

    printf("%i critical pairs added; ",added);
300     if ( added > 0 ) // if critical pairs were added
    {
        crws = fMonRulesReduce( crit );
        printf("%i rules in the reduced set.\n", fMonPairListLength(crws));
        // d = fMonPairListDisplay( crws );
305     }
    else
    {
        printf("%i rules in the reduced set.\n\n",
310             fMonPairListLength(crws));
    }
    } // end while ( added > 0 )
    printf("Number of passes made = %i.\n", passes );
    return crws;
}

315 IntFMonList
intFMonListWordReduce( word, rules1, rules2, verbose )
// reduce word as far as possible using both sets of rules
IntFMonList word;
320 FMonPairList rules1;
IntFMonPairList rules2;
int verbose; // this input variable decides whether to print out
// information to the screen (0 = yes, 1 = no)
{
    // Define variables
325     IntFMonList back, pass, transfer, new, new2;
    FMonPairList crws;
    IntFMonPairList comparison;
    FMon origWord, redWord, matchMon, newMon, newmonE;
330     ULong len1, len2;
    int check = 0, i = 0, j = 0, k = 0, match = 0, match2 = 0, newint = 0;
    int min = 0;

    // Initialise variables
335     back = intFMonListNul;
    new = intFMonListNul;
    crws = fMonPairListCopy( rules1 );

    // Reduce word iteratively
340     transfer = intFMonListCopy( word );

```

```

check = 1; // escape integer
while( check != 0)
{
345   check = 0; // to enable the escape!
      pass = intFMonListCopy( transfer );
      origWord = pass -> mon;
      match = pass -> i;
      if (verbose == 0)
350   {
          printf("\nIteration %i...\n", i+1);
          printf("Element to be reduced: (%i, %s)\n", match, fMonToStr(origWord));
      }
      i++;
355
      // Reduce the FMon part of our IntFMon
      // using the crws passed into the function
      redWord = fMonWordReduce( origWord, crws );
      if (verbose == 0)
360   {
          printf("FMon reduction: %s -> %s\n",
                fMonToStr(origWord), fMonToStr(redWord));
          printf("New element to be reduced: (%i, %s)\n", match,
                fMonToStr(redWord));
365   printf("Looking for overlaps in Type 2 elements...\n");
      }

      new = intFMonListPush(match, redWord, new);
      transfer = intFMonListCopy( new ); // transfer now holds (match, redWord)
370   new = intFMonListNul;

      comparison = intFMonPairListCopy( rules2 );

      // Now that we have reduced our word using the Type 1 rules,
      // we look for simplifications in the Type 2 rules.

      while(comparison) // While there are rules left to compare with
      {
380         match2 = comparison -> ilft;
              matchMon = comparison -> mlft;
              if (match == match2) // possible transition (matching integers)
              {
                  // Look for overlap
385                 len1 = fMonLength(redWord);
                          len2 = fMonLength(matchMon);

                  if(len2 <= len1) // if list word is <= reduced word
                  {
390                     if (fMonEqual(matchMon, fMonPrefix(redWord, len2)) == 1)
                        {
                            // Overlap Found!
                            // Now changing (x, pp') to (y, qp')
                            newint = comparison -> irt; // irt = y
                            newmonE = comparison -> mrt; // mrt = q
395
                            if (verbose == 0)
                            {
                                printf("Overlap found in (%i, %s), (%i, %s)", match2,
                                        fMonToStr(matchMon), newint, fMonToStr(newmonE));
400                                printf(" of length %i\n", len2);
                            }

                            if (len1 != len2) // i.e. if p' is not empty
                            {
405                                newMon = fMonTimes(newmonE,
                                                            fMonSuffix(redWord, len1-len2));
                            }
                            else // i.e. p' is empty
                            {
410                                newMon = newmonE;
                            }
                            check = 1;
                            new = intFMonListPush(newint, newMon, new);

415                            // Escape from loops
                            transfer = intFMonListCopy( new );
                            new = intFMonListNul;
                            comparison = intFMonPairListNul;
                        } // end if (fMonEqual(matchMon, fMonPrefix(redWord, len2)) == 1)
                    } // end if(len2 <= len1)
                    else
                    {
425                        min = 0;
                    }
                } // end if

```

```

        if (check == 0) // only carry on if no match found so far
        {
430             comparison = comparison -> rest;
        }

    } // end while(comparison)
} // end while(check...)
435 return transfer;
}

IntFMonPairList
intFMonScout( rws )
440 // Look for errors in list e.g. (1, D^2) -> (2, D^2)
IntFMonPairList rws;
{
    // Define variables
    int lftI, lftI2, rtI, rtI2, temp, i;
445 FMon lftM = fMonOne(), rtM = fMonOne(),
    lftM2 = fMonOne(), rtM2 = fMonOne();
    IntFMonPairList L, L1, L2, L3, L0 = intFMonPairListNul;

    // First of all, remove duplicates from the rws
450 L = intFMonPairListRemDups( rws );
    L2 = intFMonPairListCopy( L ); // Used to cycle through the elements
    L3 = intFMonPairListCopy( L ); // Assume all elements are valid to begin with
    i = 0;

455 while ( L2 ) // cycle through the list
    {
        i++;
        // printf("Looking at element number %i\n", i);
        lftI = L2 -> ilft;
460 lftM = L2 -> mlft;
        rtI = L2 -> irt;
        rtM = L2 -> mrt;
        L2 = L2 -> rest; // Cycle through L2

465 // L1 will now hold the list without the current element:
        L1 = intFMonPairListAppend( L0, L2 );

        // Check for invalid entry
        // e.g. (1, D^2) -> (2, D^2)
470 if ((fMonEqual(lftM, rtM) == 1) & (lftI < rtI))
        {
            // Swap Entry - to (2, D^2) -> (1, D^2) in the example above
            printf("(%i, %s), (%i, %s) is incorrectly ordered - ",
475 lftI, fMonToStr(lftM), rtI, fMonToStr(rtM));
            printf("changing to (%i, %s), (%i, %s)...\n",
                rtI, fMonToStr(rtM), lftI, fMonToStr(lftM));
            temp = lftI;
            lftI = rtI;
            rtI = temp;

480 // Change lists accordingly
            L3 = intFMonPairListCopy( L1 );
            L0 = intFMonPairListPush( lftI, lftM, rtI, rtM, L0 );
            L1 = intFMonPairListAppend( L0, L2 );
485 L3 = intFMonPairListCopy( L1 );
            // intFMonPairListDisplay ( L3 );
        }
        // Check for redundant entries e.g. (2, D) -> (2, D)
490 else if ((lftI == rtI) & (fMonEqual(lftM, rtM) == 1))
        {
            // Remove Entry
            L3 = intFMonPairListCopy( L1 );
            printf("Redundant entry (%i, %s), (%i, %s) removed...\n",
495 lftI, fMonToStr(lftM), rtI, fMonToStr(rtM));
            // intFMonPairListDisplay( L3 );
        }
        else // keep things as they are
        {
            L0 = intFMonPairListPush( lftI, lftM, rtI, rtM, L0 );
500 }
    } // end while

    // printf("\n");
    // intFMonPairListDisplay( L3 );
505 return L3;
}

IntFMonPairList
shortCuts( rws )
510 // Find shortcuts
IntFMonPairList rws;
{

```

```

// Define variables
int d, lftI, lftI2, rtI, rtI2, escape, switcher;
515 FMon lftM = fMonOne(), rtM = fMonOne(),
    lftM2 = fMonOne(), rtM2 = fMonOne();
IntFMonPairList L, L4, L3, L2, L1, L0;

// Initialise variables
520 L = rws;
L3 = intFMonPairListCopy( L );
escape = 1;

// Example of shortcut: consider we have rules
525 // (1, D^2) -> (2, D)
// (2, D) -> (1, 1)
//
// Then we can use the 2nd rule to simplify the 1st:
// the new list should be
530 // (1, D^2) -> (1, 1)
// (2, D) -> (1, 1)

while( escape == 1 )
{
535   escape = 0;

   // Remove Duplicates
   L3 = intFMonPairListRemDups( L3 );
   L2 = intFMonPairListCopy( L3 ); // L2 will hold the remaining elements
540   L0 = intFMonPairListNul;

   while ( L2 ) // while there are elements to analyse
   {
     lftI = L2 -> ilft;
545     lftM = L2 -> mlft;
     rtI = L2 -> irt;
     rtM = L2 -> mrt;

     L2 = L2 -> rest; // Cycle through L2
550     // L1 is the list without the current entry:
     L1 = intFMonPairListAppend( L0, L2 );

     L4 = intFMonPairListCopy( L1 ); // L4 will hold the elements we look in for duplicates
     switcher = 0;
555     while( L4 )
     {
       lftI2 = L4 -> ilft;
       lftM2 = L4 -> mlft;
560       rtI2 = L4 -> irt;
       rtM2 = L4 -> mrt;

       L4 = L4 -> rest; // Cycle through L4

565       // Look for shortcuts, i.e. match RHS to a LHS
       if ( fMonEqual( rtM, lftM2 ) == 1 )
       {
         if ( ( rtI == lftI2 ) & ( switcher == 0 ) )
         {
570           // If they are equal push different element
           // printf("Found Shortcut!");
           L0 = intFMonPairListPush( lftI, lftM, rtI, rtM, L0 );
           escape = 1; // to run the algorithm again
           // (there may be more shortcuts)
575           switcher = 1;
         }
       }
     } // end while( L4 )
580     if( switcher == 0 )
     {
       L0 = intFMonPairListPush( lftI, lftM, rtI, rtM, L0 );
       // (keep current element)
585     }

     L3 = intFMonPairListAppend( L0, L2 );
   } // end while( L2 )
590 } // end while( escape == 1 )

return L3;
}

595 IntFMonPairList
intFMonRulesReduce( rws, crwsN )
// Compact rules list
IntFMonPairList rws;

```

```

FMonPairList crwsN;
600 {
    // Define Variables
    int d, lftI, lftI2, rtI, rtI2;
    FMon lftM = fMonOne(), rtM = fMonOne(),
        lftM2 = fMonOne(), rtM2 = fMonOne();
605 ULong i = 0, len = 0;
    IntFMonList pl = intFMonListNul, rl = intFMonListNul;
    IntFMonPairList L, L3, L2, L1, L0 = intFMonPairListNul;

    // Initialise Variables
610 L = intFMonPairListCopy( rws );
    len = intFMonPairListLength( rws );
    L = intFMonPairListRemDups( L );
    len = intFMonPairListLength( L );
    // d = intFMonPairListDisplay( L );

615 i = 0;
    L2 = intFMonPairListCopy( L ); // L2 will hold the remaining elements
    L3 = intFMonPairListCopy( L ); // L3 will hold the returned list
    // (To begin with, we assume that all elements are valid)
620 while ( L2 ) // while there are elements to analyse
    {
        i++;
        lftI = L2 -> ilft;
        lftM = L2 -> mlft;
625 pl = intFMonListPush( lftI, lftM, pl );
        rtI = L2 -> irt;
        rtM = L2 -> mrt;
        rl = intFMonListPush( rtI, rtM, rl );
        L2 = L2 -> rest; // Cycle through L2
630 // L1 is the list without the current entry:
        L1 = intFMonPairListAppend( L0, L2 );

        // Reduce elements

635 // printf("Originals: (%i, %s), (%i, %s); ",
        // lftI, fMonToStr(lftM), rtI, fMonToStr(rtM));

        pl = intFMonListWordReduce( pl, crwsN, L1, 1 );
        rl = intFMonListWordReduce( rl, crwsN, L1, 1 );
640 lftI2 = pl -> i;
        lftM2 = pl -> mon;
        rtI2 = rl -> i;
        rtM2 = rl -> mon;

645 // printf("Reduced: (%i, %s), (%i, %s); \n",
        // lftI2, fMonToStr(lftM2), rtI2, fMonToStr(rtM2));

        // Compare reduced elements
        if ( ( fMonEqual( lftM2, rtM2 ) == 1 ) & ( lftI2 == rtI2 ) )
650 {
            // If they are equal get rid of current element
            // printf("\nGot rid of element!");
            L3 = intFMonPairListCopy( L1 );
        }
655 else
        {
            // If they are not equal, keep current element
            // printf("\nKept element!");
            // Note that the RHS is reduced
660 L0 = intFMonPairListPush( lftI, lftM, rtI2, rtM2, L0 );
        }
    } // end while

    return L3;
665 }

IntFMonPairList
intFMonKnuthBendix( crwsN, irws )
// Compute crws
670 FMonPairList crwsN;
IntFMonPairList irws;
{
    // Define Variables
    int i, j, k, check, lftIA, lftIB, rtIA, rtIB, count = 0, swapI,
675 redIA, redIB, decide, beginK, endK, executeK, added, i1, i2;
    ULong sizei, sizej, lengthA, lengthB, startSize, endSize, midSize;
    FMon lftMA = fMonOne, lftMB = fMonOne,
        rtMA = fMonOne, rtMB = fMonOne,
        redMA = fMonOne, redMB = fMonOne,
680 m1, m2, swapM = fMonOne;
    FMonPairList tempN, cycle3;
    IntFMonPairList tempXW, back, cycle1, cycle2, adder;
    IntFMonList redA, redB;

```

```

685 // Check initial list for mistakes
printf("\nChecking the initial rewrite system...\n");
irws = intFMonPairListRemDups( intFMonScout( irws ) );
printf("...Initial rewrite system checked.\n");

690 // Reduce Initial Type 2 Rules
// i.e. if we have e.g. (1, D^3*C*D) -> (2, D*C*D),
// then we change D^3*C*D and D*C*D to normal forms using crwsN
tempXW = intFMonPairListCopy( irws );
tempN = fMonPairListCopy( crwsN );
695 // irws = intFMonPairListNul;

// while( tempXW )
// {
//     i1 = tempXW -> ilft;
700 //     m1 = fMonWordReduce( tempXW->mlft, crwsN );
//     i2 = tempXW -> irt;
//     m2 = fMonWordReduce( tempXW->mrt, crwsN );
//     irws = intFMonPairListPush( i1, m1, i2, m2, irws );
//     tempXW = tempXW -> rest;
705 // }
// irws = intFMonPairListFXRev( irws );
// tempXW = intFMonPairListCopy( irws );
// printf("\nInitial Type 2 Rules with normal forms:\n");
// intFMonPairListDisplay(irws);
710 // printf("%i rules\n", intFMonPairListLength( irws ));

// Initialise Variables
back = intFMonPairListCopy( irws );
adder = intFMonPairListNul;
715 check = 1;

/*
* General Form of algorithm:
* Iterate:
720 * 1. Look for overlaps of LHS's: (a) type 2 with type 2
*                                     (b) type 1 with type 2
* 2. Add new rules if necessary based on (n, x) < (m, y) iff ...
* 3. Weed out rules using intFMonRulesReduce
* 4. Go to step 1. Continue until no new rules found
725 */

while( check != 0 ) // until no more rules added
{
    check = 0; // enable escape
    added = 0; // reset number of critical pairs added
730 startSize = intFMonPairListLength( tempXW );
    count++; // used to count the number of iterations
    printf("\nIteration %i...\n", count);
    printf("\n...%i elements to begin with\n", startSize);
735
    // *****
    // FIRST STAGE: TYPE 2 WITH TYPE 2
    // *****
    //
740 // Look for overlaps e.g.
// (1, D^2*C) (1, D^2*C) (1, D)
// (1, D^2) OVERLAP! (2, D^2) NO OVERLAP! (1, C) NO OVERLAP!

    cycle1 = intFMonPairListCopy( tempXW );
745 sizei = intFMonPairListLength( cycle1 );
    sizej = sizei;
    for( i = 1; i <= sizei-1; i++ )
    {
        lftIA = cycle1 -> ilft;
750 lftMA = cycle1 -> mlft;
        rtIA = cycle1 -> irt;
        rtMA = cycle1 -> mrt;

        // Set up second list - get to correct part of list
755 cycle2 = intFMonPairListCopy( tempXW );
        for( j = 1; j <= i; j++ )
        {
            cycle2 = cycle2 -> rest;
        }
760
        for( j = i+1; j <= sizej; j++ )
        {
            lftIB = cycle2 -> ilft;
            lftMB = cycle2 -> mlft;
765 rtIB = cycle2 -> irt;
            rtMB = cycle2 -> mrt;
            // printf("Comparing %i with %i with integers %i and %i\n", i, j,
            //         lftIA, lftIB);

770 // WE ARE COMPARING

```

```

// (lftIA, lftMA) -> (rtIA, rtMA) with
// (lftIB, lftMB) -> (rtIB, rtMB)

// Check matching integers
775 if (lftIA == lftIB)
{
    // printf("Match! Now comparing monoids ");
    // Check if one monoid is a submonoid of the other
    lengthA = fMonLength(lftMA);
780 lengthB = fMonLength(lftMB);
    // printf("of lengths %i and %i\n", lengthA, lengthB);

    if (lengthB < lengthA) // swap elements
    {
785 swapI = lftIA; lftIA = lftIB; lftIB = swapI;
        swapI = rtIA; rtIA = rtIB; rtIB = swapI;
        swapM = lftMA; lftMA = lftMB; lftMB = swapM;
        swapM = rtMA; rtMA = rtMB; rtMB = swapM;
        lengthA = fMonLength(lftMA);
790 lengthB = fMonLength(lftMB);
    }

    if (lengthA != 0)
    {
795 // Check if lftMA appears in lftMB
        if (fMonEqual(lftMA, fMonPrefix(lftMB, lengthA)) == 1)
        {
            // Match, add onto list if necessary
            // printf("Subword Found!\n");
            // printf("2/2 Match found between (%i, %s) -> (%i, %s) and ",
800 // lftIA, fMonToStr(lftMA), rtIA, fMonToStr(rtMA));
            // printf("(%i, %s) -> (%i, %s)\n",
            // lftIB, fMonToStr(lftMB), rtIB, fMonToStr(rtMB));

805 redA = intFMonListNul;
            redB = intFMonListNul;

            // Reduce word first way
            redIA = rtIA;
810 redMA = fMonTimes(rtMA, fMonSuffix(lftMB, lengthB-lengthA));
            // printf("First Reduction: (%i, %s)\n",
            // redIA, fMonToStr(redMA));

            // Reduce word second way
815 redIB = rtIB;
            redMB = rtMB;
            // printf("Second Reduction: (%i, %s)\n",
            // redIB, fMonToStr(redMB));

820 // Simplify reduced words using rewrite system
            redA = intFMonListPush(redIA, redMA, redA);
            redA = intFMonListWordReduce( redA, crwsN, tempXW, 1 );
            redB = intFMonListPush(redIB, redMB, redB);
            redB = intFMonListWordReduce( redB, crwsN, tempXW, 1 );

825 redIA = redA -> i;
            redMA = redA -> mon;
            // printf("First Reduction (Reduced): (%i, %s)\n",
            // redIA, fMonToStr(redMA));
830 redIB = redB -> i;
            redMB = redB -> mon;
            // printf("Second Reduction (Reduced): (%i, %s)\n",
            // redIB, fMonToStr(redMB));

835 // printf("Reduced Words (%i, %s) and (%i, %s)\n",
            // redIA, fMonToStr(redMA), redIB, fMonToStr(redMB));

            // Now compare words
840 if (intFMonListEqual(redA, redB) != 1)
            {
                // See which one is the 'largest'
                // printf("Elements different. Adding onto list...\n");

                decide = fMonEqual(redMA, redMB);

845 // FIRST, decide by the monoids
                if ( theOrdFun( redMA, redMB ) == 1 )
                {
                    adder = intFMonPairListNul;
                    adder = intFMonPairListPush(redIB, redMB,
850 redIA, redMA, adder);
                    tempXW = intFMonPairListAppend(tempXW, adder);
                    added++;
                }
                else if ( theOrdFun( redMB, redMA ) == 1 )
            {
855
            }
            }
        }
    }
}

```

```

        adder = intFMonPairListNul;
        adder = intFMonPairListPush(redIA, redMA,
860         redIB, redMB, adder);
        tempXW = intFMonPairListAppend(tempXW, adder);
        added++;
    }

    // SECONDLY, if the monoids are the same,
865    // decide by the integers.
    if (decide == 1)
    {
        if (redIA < redIB)
870        {
            adder = intFMonPairListNul;
            adder = intFMonPairListPush(redIB, redMB,
                redIA, redMA, adder);
            tempXW = intFMonPairListAppend(tempXW, adder);
            added++;
875        }
        else if (redIB < redIA)
        {
            adder = intFMonPairListNul;
            adder = intFMonPairListPush(redIA, redMA,
880             redIB, redMB, adder);
            tempXW = intFMonPairListAppend(tempXW, adder);
            added++;
        }
    } // end if (decide)
885    } // end if (intFMonListEqual...)
    } // end if (fMonEqual..)
    } // end if (lengthA != 0)

    } // end if (IA == IB)
890    cycle2 = cycle2 -> rest;

    } // end for (j)

895    cycle1 = cycle1 -> rest;

    } // end for (i)

    // Tidy Up...
900    // Get rid of unwanted rules
    endSize = intFMonPairListLength( tempXW );
    printf("...%i elements added of Type 2 vs. Type 2\n",
        endSize-startSize);

905    // Reduce Rules
    tempXW = intFMonRulesReduce( tempXW, crwsN );
    back = intFMonPairListCopy( tempXW );
    endSize = intFMonPairListLength( back );
    midSize = endSize-startSize;

910    // *****
    // SECOND STAGE: TYPE 1 WITH TYPE 2
    // *****
    //
915    // Look for overlaps of lftMA in
    // (lftIA, lftMA) -> (rtIA, rtIB)
    // with LHS's of Type 1 Rules

    cycle1 = intFMonPairListCopy( tempXW ); // with new elements
920    sizei = intFMonPairListLength( cycle1 );
    for( i = 1; i <= sizei; i++ ) // for each (x, w)
    {
        lftIA = cycle1 -> ilft;
        lftMA = cycle1 -> mlft;
925        rtIA = cycle1 -> irt;
        rtMA = cycle1 -> mrt;

        // Set up second list
        cycle3 = fMonPairListCopy( tempN );
930        sizej = fMonPairListLength( cycle3 );

        for( j = 1; j <= sizej; j++ ) // for each monoid LHS of Type 1 rules
        {
            lftMB = cycle3 -> lft;
935            rtMB = cycle3 -> rt;
            // printf("i = %i, j = %i, comparing %s with %s; ",
            // i, j, fMonToStr(lftMA), fMonToStr(lftMB));

            // Now look for overlaps.
940            lengthA = fMonLength(lftMA);
            lengthB = fMonLength(lftMB);

```

```

//
// Overlap type 1: lftMB inside lftMA
945 // e.g. (1, D*C*D*C) -> (2, D*C)
//      C*D
//
//
if ((lengthB <= lengthA) & (lengthA != 0))
950 {
    // Check if lftMB appears somewhere in lftMA
    //
    // (.....lftMA.....)
    // (...lftMB..) ->
955 //
    for(k = 1; k <= lengthA-lengthB+1; k++)
    {
        if(fMonEqual(lftMB, fMonSubWord(lftMA,k,k+lengthB-1)) == 1)
960 {
            // Match found. Now reduce as before
            // printf(" 1/2 Match found between (%i, %s) -> (%i, %s) and ",
            //      lftIA, fMonToStr(lftMA), rtIA, fMonToStr(rtMA));
            // printf("%s\n", fMonToStr(lftMB));

965 // Match, add onto list if necessary
            // printf("Subword Found!\n");
            redA = intFMonListNul;
            redB = intFMonListNul;

970 // Reduce word first way
            redIA = rtIA;
            redMA = rtMA;
            // printf("First Reduction: (%i, %s)\n",
            //      redIA, fMonToStr(redMA));

975 // Reduce word second way
            redIB = lftIA;
            if (k != 1)
980 {
                redMB = fMonTimes(fMonPrefix(lftMA, k-1), rtMB);
            }
            if (k != lengthA-lengthB+1)
            {
                redMB = fMonTimes(redMB,
985 fMonSuffix(lftMA, lengthA-lengthB-k+1));
            }
            // printf("Second Reduction: (%i, %s)\n",
            //      redIB, fMonToStr(redMB));

990 // Simplify reduced words using rewrite system
            redA = intFMonListPush(redIA, redMA, redA);
            redA = intFMonListWordReduce( redA, crwsN, tempXW, 1 );
            redB = intFMonListPush(redIB, redMB, redB);
            redB = intFMonListWordReduce( redB, crwsN, tempXW, 1 );

995 redIA = redA -> i;
            redMA = redA -> mon;
            // printf("First Reduction (Reduced): (%i, %s)\n",
            //      redIA, fMonToStr(redMA));

1000 redIB = redB -> i;
            redMB = redB -> mon;
            // printf("Second Reduction (Reduced): (%i, %s)\n",
            //      redIB, fMonToStr(redMB));

1005 // printf("Reduced Words (%i, %s) and (%i, %s)\n",
            //      redIA, fMonToStr(redMA), redIB, fMonToStr(redMB));

// Now compare words
1010 if (intFMonListEqual(redA, redB) != 1)
    {
        // See which one is the 'largest'
        // printf("Elements different. Adding onto list...\n");
        decide = fMonEqual(redMA, redMB);

1015 // FIRST, decide by the monoids
        if ( theOrdFun( redMA, redMB ) == 1 )
        {
            adder = intFMonPairListNul;
            adder = intFMonPairListPush(redIB, redMB,
1020 redIA, redMA, adder);
            tempXW = intFMonPairListAppend(tempXW, adder);
            added++;
        }
        else if ( theOrdFun( redMB, redMA ) == 1 )
1025 {
            adder = intFMonPairListNul;
            adder = intFMonPairListPush(redIA, redMA,

```

```

1030         redIB, redMB, adder);
tempXW = intFMonPairListAppend(tempXW, adder);
added++;
    }

1035     // SECONDLY, if the monoids are the same,
    // decide by the integers.
    if (decide == 1)
    {
1040         if (redIA < redIB)
        {
            adder = intFMonPairListNul;
            adder = intFMonPairListPush(redIB, redMB,
1045             redIA, redMA, adder);
            tempXW = intFMonPairListAppend(tempXW, adder);
            added++;
        }
        else if (redIB < redIA)
        {
1050             adder = intFMonPairListNul;
            adder = intFMonPairListPush(redIA, redMA,
            redIB, redMB, adder);
            tempXW = intFMonPairListAppend(tempXW, adder);
            added++;
        }
    } // end if (decide)
1055 } //end if intMonListEqual...
} // end if fMonEqual...
} // end for (k)
} // end if(lengthB <= lengthA)

1060 //
// Overlap type 2: lftMB overlaps on right
//
// (.....lftMA.....)
//          (...lftMB...) ->
1065 //

// To begin with, set up the conditions on the loop
beginK = 0;
endK = lengthA-1;
1070 executeK = 0;
if (lengthB <= lengthA)
{
    beginK = 1;
    endK = lengthB-1;
1075 }

// Look for overlaps on right hand side
for( k = beginK; k <= endK; k++ )
{
1080     executeK = 0; // reset switch
    if((lengthB <= lengthA) & (lengthA != 0))
    {
        // printf("Comparing (%i, %s) with %s ", lftIA,
        // fMonToStr(lftMA),
        // fMonToStr(lftMB));
1085         if(fMonEqual(fMonSuffix(lftMA, k),
            fMonPrefix(lftMB, k)) == 1)
        {
            executeK = 1;
1090         }
    }
    else if (lengthA != 0) // i.e. and lengthB > lengthA
    {
        // printf("Comparing (%i, %s) with %s ", lftIA,
        // fMonToStr(lftMA),
        // fMonToStr(lftMB));
1095         if(fMonEqual(fMonSuffix(lftMA, lengthA-k),
            fMonPrefix(lftMB, lengthA-k)) == 1)
        {
            executeK = 1;
1100         }
    }
}

1105 if(executeK == 1)
{
    // Match found. Now reduce as before
    // printf(" 1/2 Match found between (%i, %s) -> (%i, %s) and ",
    //         lftIA, fMonToStr(lftMA), rtIA, fMonToStr(rtMA));
    // printf("%s\n", fMonToStr(lftMB));
1110
    redA = intFMonListNul;
    redB = intFMonListNul;

    if (lengthB <= lengthA)

```

```

1115     {
        // Reduce word first way
        redIA = rtIA;
        redMA = fMonTimes(rtMA, fMonSuffix(lftMB, lengthB-k));
1120         // printf("1st Reduction: (%i, %s) ",
        //         redIA, fMonToStr(redMA));

        // Reduce word second way
        redIB = lftIA;
        redMB = fMonTimes(fMonPrefix(lftMA, lengthA-k), rtMB);
1125         // printf("2nd Reduction: (%i, %s) ",
        //         redIB, fMonToStr(redMB));
    }
    else
1130     {
        // Reduce word first way
        redIA = rtIA;
        redMA = fMonTimes(rtMA, fMonSuffix(lftMB,
            lengthB-lengthA+k));
1135         // printf("(%i)", lengthB-lengthA+k);
        // printf("1st Reduction: (%i, %s) ",
        //         redIA, fMonToStr(redMA));

        // Reduce word second way
        redIB = lftIA;
        redMB = rtMB;
1140         if (k != 0)
        {
            redMB = fMonTimes(fMonPrefix(lftMA, k), redMB);
        }
1145         // printf("2nd Reduction: (%i, %s) ",
        //         redIB, fMonToStr(redMB));
    }

    // Simplify reduced words using rewrite system
1150     redA = intFMonListPush(redIA, redMA, redA);
    redA = intFMonListWordReduce( redA, crwsN, tempXW, 1 );
    redB = intFMonListPush(redIB, redMB, redB);
    redB = intFMonListWordReduce( redB, crwsN, tempXW, 1 );

1155     redIA = redA -> i;
    redMA = redA -> mon;
    // printf("First Reduction (Reduced): (%i, %s)\n",
    //         redIA, fMonToStr(redMA));

1160     redIB = redB -> i;
    redMB = redB -> mon;
    // printf("Second Reduction (Reduced): (%i, %s)\n",
    //         redIB, fMonToStr(redMB));

1165     // printf("Reduced Words (%i, %s) and (%i, %s)\n",
    //         redIA, fMonToStr(redMA), redIB, fMonToStr(redMB));

    // Now compare words
1170     if (intFMonListEqual(redA, redB) != 1)
    {
        // See which one is the 'largest'
        // printf("Elements different. Adding onto list...\n");
        decide = fMonEqual(redMA, redMB);

1175         // FIRST, decide by the monoids
        if ( theOrdFun( redMA, redMB ) == 1 )
        {
            adder = intFMonPairListNul;
            adder = intFMonPairListPush(redIB, redMB,
1180                                     redIA, redMA, adder);
            tempXW = intFMonPairListAppend(tempXW, adder);
            added++;
        }
        else if ( theOrdFun( redMB, redMA ) == 1 )
1185         {
            adder = intFMonPairListNul;
            adder = intFMonPairListPush(redIA, redMA,
            redIB, redMB, adder);
            tempXW = intFMonPairListAppend(tempXW, adder);
1190            added++;
        }
    }

    // SECONDLY, if the monoids are the same,
    // decide by the integers.
1195     if (decide == 1)
    {
        if (redIA < redIB)
        {
1200            adder = intFMonPairListNul;
            adder = intFMonPairListPush(redIB, redMB,

```

```

                                redIA, redMA, adder);
                                tempXW = intFMonPairListAppend(tempXW, adder);
                                added++;
1205                                }
                                else if (redIB < redIA)
                                {
                                    adder = intFMonPairListNul;
                                    adder = intFMonPairListPush(redIA, redMA,
1210                                        redIB, redMB, adder);
                                    tempXW = intFMonPairListAppend(tempXW, adder);
                                    added++;
                                }
                                } // end if (decide)
                                } //end if intMonListEqual...
1215                                } // end if (executeK)
                                } // end for (k)
1220                                cycle3 = cycle3 -> rest;
                                } // end for (j)
                                cycle1 = cycle1 -> rest;
1225                                } // end for(i)
                                // *****
                                // END OF STAGE 2
1230                                // *****

                                // Get rid of unwanted rules
                                back = intFMonPairListCopy( tempXW );
                                endSize = intFMonPairListLength( back );
1235                                printf("...%i elements added of Type 1 vs. Type 2\n",
                                    endSize-startSize-midSize);
                                // intFMonPairListDisplay(tempXW);
                                tempXW = intFMonRulesReduce( tempXW, crwsN );
                                back = intFMonPairListCopy( tempXW );
1240                                // Look for shortcuts
                                back = shortCuts( back );

                                endSize = intFMonPairListLength( back );
1245                                printf("...%i elements in the reduced set\n", endSize);

                                // if the size of the list has increased, do the algorithm again
                                if (added > 0)
                                {
1250                                    check = 1;
                                }

                                } // end while

1255                                return back;
                                } // end IntFMonPairList

FMonList
1260 fMonMonoidElements( gens, crws ) // use complete rws to list elements
FMonList gens;
FMonPairList crws;
{
    ULong len;
1265    Short flag;
    FMon w, g, wg;
    FMonList used = fMonListNul, found = fMonListNul, revf, gen2;
    int i = 0, d = 0, lenu = 0, lenf = 0, inf, inu;

1270    found = fMonListPush( fMonOne(), found );
    lenf = 1;
    while ( lenf > 0 )
    {
        revf = fMonListFXRev( found );
1275        w = revf -> first;
        used = fMonListPush( w, used );
        found = fMonListFXRev( revf -> rest );
        lenf = lenf-1;
        lenu = lenu+1;
1280        gen2 = fMonListCopy( gens );
        while ( gen2 )
        {
            g = gen2 -> first;
            gen2 = gen2 -> rest;
1285            wg = fMonWordReduce( fMonTimes(w, g), crws );
            inu = fMonListIsMember( wg, used );

```

```

        inf = fMonListIsMember( wg, found );
        if ( ( inf == 0 ) && ( inu == 0 ) )
1290     {
            found = fMonListPush( wg, found );
            lenf = lenf+1;
        } // end if

    } // end while ( gen2 )
1295 } // end while ( lenf > 0 )

    return fMonListFXRev( used );

} // end FMonMonoidElements
1300
IntFMonList
intFMonMonoidElements( size, gens, crwsN, crwsXW )
// use complete rws to list elements
int size;
1305 FMonList gens;
FMonPairList crwsN;
IntFMonPairList crwsXW;
{
    FMon g, monE, monE2, M, M2;
1310 FMonList gen2;
    IntFMonList used = intFMonListNul, found = intFMonListNul, revf,
        answer = intFMonListNul, L1, L2, L3, L0 = intFMonListNul,
        check = intFMonListNul;
    int i = 0, lenu = 0, lenf = 0, inf, inu, intE, intE2, I, I2;
1315
    // Assume that all [x, 1] are elements
    for(i = 1; i <= size; i++)
    {
        found = intFMonListPush( i, fMonOne(), found );
1320    }
    // printf("Initial Set = \n");
    // intFMonListDisplay(found);

    // Check that all [x, 1] are indeed elements
1325 L2 = intFMonListCopy( found ); // L2 will hold the remaining elements
    L3 = intFMonListCopy( found ); // L3 will hold the returned list
        // (To begin with, we assume that all elements are valid)
    while ( L2 ) // while there are elements to analyse
    {
1330     I = L2 -> i;
        M = L2 -> mon;
        check = intFMonListPush( I, M, check );
        L2 = L2 -> rest; // Cycle through L2
        // L1 is the list without the current entry:
1335     L1 = intFMonListAppend( L0, L2 );

        // Reduce elements
        check = intFMonListWordReduce( check, crwsN, crwsXW, 1 );
1340     I2 = check -> i;
        M2 = check -> mon;

        if (intFMonListIsMember( I2, M2, L1 ) == 1)
        {
            // 'element' can be reduced to another element - get rid!!
1345         L3 = intFMonListCopy( L1 );
        }
        else
        {
            // keep element
1350         L0 = intFMonListPush( I, M, L0 );
        }
    } // end while

    // finished checking - now set found to be checked elements
1355 found = intFMonListCopy( L3 );

    lenf = intFMonListLength( found );
    while ( lenf > 0 )
    {
1360     revf = intFMonListFXRev( found );
        intE = revf -> i;
        monE = revf -> mon;
        used = intFMonListPush( intE, monE, used );
        found = intFMonListFXRev( revf -> rest );
1365     lenf = lenf-1;
        lenu = lenu+1;
        gen2 = fMonListCopy( gens );

        while ( gen2 )
1370     {
            g = gen2 -> first;
            gen2 = gen2 -> rest;

```

```

    answer = intFMonListPush(intE, fMonTimes(monE, g), answer);
    answer = intFMonListWordReduce( answer, crwsN, crwsXW, 1 );
1375   intE2 = answer -> i;
        monE2 = answer -> mon;
        inu = intFMonListIsMember( intE2, monE2, used );
        inf = intFMonListIsMember( intE2, monE2, found );
        if ( ( inf == 0 ) && ( inu == 0 ) )
1380     {
            found = intFMonListPush( answer -> i, answer -> mon, found );
            lenf = lenf+1;
        } // end if
        answer = intFMonListNul;
1385     } // end while ( gen2 )

        } // end while ( lenf > 0 )

1390     return intFMonListFXRev( used );
} // end intFMonMonoidElements

```

kba.c

```
1  /*
   * File: kba.c
   * Author: Gareth Evans (modified from original work by CDW)
   * Last Modified: 16th April 2002
5  */

#include "frmon.h"

int
10 main(argc, argv)
    int argc;
    char *argv[];
    {
15     #include "rwsa.h"

        // Define Variables
        FMonPairList rws, crws, crwsN;
        FILE *kldata;
        FILE *otherdata;
20     FILE *testdata;
        int d = 0, nOfGen = 0, rowLength = 0, i = 0, j = 0, lengthElts;
        ULong numbersLength;
        FMon empty, build;
        FMonList Mgenerators, cycle, transferGen, N, elts;
25     IntFMonList numbers, transferNum, inputTest, inputTest2, redWord, eltsIW;
        IntFMonPairList rws2, crws2, crwsXW;

        // Set Order
        theOrdFun = fMonTlex;
30

        // Check for Files
        if ( ( argc < 2 ) | ( argc > 4 ) )
        {
35             printf("\nYou have not given between one ");
             printf("and three files on the command line!");
             printf("\nThe correct format is 'kba FILE1.in' or\n");
             printf("'kba FILE1.in FILE2.in' or\n");
             printf("'kba FILE1.in FILE2.in FILE3.in',\n");
             printf("where FILE1 is the presentation for N,\n");
40             printf("the optional FILE2 is the information for building the M-action,\n");
             printf("and the optional FILE3 is an arbitrary word (x, n).\n\n");
             printf("See the user manual for more information.\n\n");
             exit (1);
        }
45     // Open files specified on the command line
        if ((kldata = fopen (argv[1], "r")) == NULL)
        {
            printf ("%s\n", "Error opening the first file");
            exit (1);
50         }
        if (argc > 2)
        {
            if ((otherdata = fopen (argv[2], "r")) == NULL)
            {
55                 printf ("%s\n", "Error opening the second file");
                 exit (1);
            }
            if (argc == 4)
            {
60                 if ((testdata = fopen (argv[3], "r")) == NULL)
                    {
                        printf ("%s\n", "Error opening the third file");
                        exit (1);
65                     }
            }
        }

        printf("\nData read in. Now processing...\n");

70     // Read in the Information for N
        N = fMonListFromFile( kldata );
        printf("\nGenerating set for monoid N = \n");
        fMonListDisplay( N );
        printf("[%i generator(s)]\n", fMonListLength( N ));
75     rws = fMonPairListFromFile( kldata );

        // Read in the other information if required
        if (argc > 2)
        {
80             nOfGen = firstLineInt( otherdata );
             Mgenerators = secondLineFmons( otherdata, nOfGen );
             numbers = remainingLines( otherdata, nOfGen );
        }
    }
}
```

```

}

85 // Display some information here
printf("\nN is presented as follows:\n");
fMonPairListDisplay( rws );
printf("[%i rule(s)]\n", fMonPairListLength( rws ));

90 // Compute crws for N
printf("\nNow computing a complete rewrite system for N....\n");
crwsN = fMonKnuthBendix( rws );
printf("\n...Complete rewrite system for N computed.\n");

95 // Display some information dependent on the amount of files
// given on the command line
if (argc == 2)
{
100 printf("\nThis is the Complete rewrite system for N:\n");
fMonPairListDisplay(crwsN);
printf("[%i rule(s)]\n", fMonPairListLength( crwsN ));

// Compute the elements of N
105 elts = fMonMonoidElements( N, crwsN );
lengthElts = fMonListLength( elts );
printf("\nElements of the monoid N:\n");
fMonListDisplay( elts );
printf("[%i element(s)]\n", lengthElts);
}
110 else
{
printf("\nThese are the Type 1 rules in the Complete Rewrite System:\n");
fMonPairListDisplay(crwsN);
printf("[%i rule(s)]\n", fMonPairListLength( crwsN ));
115 }

// Continue to process the data if required
if (argc > 2)
{
120 printf("\nContinuing to process the data...\n");
printf("\nThe number of generators of M is %i.\n", nOfGen);
printf("\nImages of M generators: \n");
fMonListDisplay( Mgenerators );
printf("[%i image(s)]\n", fMonListLength( Mgenerators ));
125

// printf("\nTable Information: \n");
// intFMonListDisplay( numbers );
printf("\n");

130 // Now convert the numbers information into an IntFMonPairList
numbersLength = intFMonListLength( numbers );
// printf("Numbers has length %i\n", numbersLength);
rowLength = numbersLength/nOfGen;
rws2 = intFMonPairListNul;
135 empty = fMonOne();

// Generate initial rws
transferNum = intFMonListCopy( numbers );
transferGen = fMonListCopy( Mgenerators );
140 for(i = 1; i <= nOfGen; i++)
{
build = transferGen -> first;
build = fMonWordReduce(build, crwsN);
145 for(j = 1; j <= rowLength; j++)
{
// Build up the Pair List rules
rws2 = intFMonPairListPush(j, build,
transferNum -> i, empty, rws2);
transferNum = transferNum -> rest;
150 }
transferGen = transferGen -> rest;
}
rws2 = intFMonPairListFXRev( rws2 );
printf("Type 2 rules for the initial rewrite system:\n");
155 intFMonPairListDisplay(rws2);
printf("[%i rule(s)]\n", intFMonPairListLength( rws2 ));

// Now compute a crws for elements of the form (x, n)
printf("\nNow computing the complete rewrite system for ");
160 printf("elements of the form (x, n):\n");
crwsXW = intFMonKnuthBendix( crwsN, rws2 );
printf("\n...Complete rewrite system computed for elements");
printf(" of the form (x, n).\n");
printf("\nThe following is the Complete Rewrite System:\n");
165 printf("\n-----TYPE 1 RULES-----\n");
fMonPairListDisplay(crwsN);
printf("[%i rule(s)]\n", fMonPairListLength( crwsN ));
printf("\n-----TYPE 2 RULES-----\n");

```

```

170     intFMonPairListDisplay(crwsXW);
        printf("[%i rule(s)]\n", intFMonPairListLength( crwsXW ));

        // Compute the elements for the induced set
        printf("\n-----ELEMENTS-----\n");
        eltsIW = intFMonMonoidElements(rowLength, N, crwsN, crwsXW);
175     intFMonListDisplay(eltsIW);
        printf("[%i element(s)]\n", intFMonListLength(eltsIW));

        printf("\n-----\n\n");

180     // Now, if specified, we try to reduce an intFMon to normal form
        if (argc == 4)
        {
            printf("The word (x, n) to reduce is as follows:\n");
            inputTest = intFMonListFromFile( testdata );
185     inputTest2 = intFMonListCopy( inputTest );
            printf("[%i, %s]", inputTest2 -> i, fMonToStr(inputTest2 -> mon));
            // intFMonListDisplay( inputTest );

            printf("\nNow Reducing....\n");
190     redWord = intFMonListWordReduce( inputTest, crwsN, crwsXW, 0 );
            printf("\n...Reduction Finished.\nThe normalised form");
            printf(" of the word is as follows\n");
            printf("(it belongs to the following equivalence class):\n");
            intFMonListDisplay( redWord );
195     }
        }
        return 0;
    }

200 #include "rwsa.c"

```

kbe.c

```
1  /*
   * File: kbe.c
   * Author: Gareth Evans (modified from original work by CDW)
   * Last Modified: 15th April 2002
5  */

#include "frmon.h"

int
10 main(argc, argv)
    int argc;
    char *argv[];
    {
#include "rwse.h"
15
    // Define Variables
    FMonPairList rwsM, rwsN, crwsM, crwsN;
    FILE *Mdata;
    FILE *Ndata;
20    FILE *otherdata;
    FILE *testdata;
    int d = 0, nOfGen = 0, rowLength = 0, i = 0, j = 0,
        lengthElts = 0, lengthM = 0, position;
    ULong numbersLength, Mlength;
25    FMon empty, answer, eltsElement, genElement, build;
    FMonList Mgenerators, cycle, transferGen, M, Mcopy, N, elts, eltscopy;
    IntFMonList numbers, transferNum, inputTest, inputTest2, redWord, eltsIW;
    IntFMonPairList rws2, crws2, crwsXW;

30    // Set Order
    theOrdFun = fMonTLex;

    // Check for Files
    if ( ( argc < 2 ) | ( argc > 5 ) | ( argc == 3 ) )
35    {
        printf("\nYou have not given one, three ");
        printf("or four files on the command line!\n");
        printf("\nThe correct format is 'kbe FILE1.in' or\n");
        printf("'kbe FILE1.in FILE2.in FILE3.in' or\n");
40        printf("'kbe FILE1.in FILE2.in FILE3.in FILE4.in',\n");
        printf("where FILE1 is the presentation for M,\n");
        printf("FILE2 is the presentation for N,\n");
        printf("FILE3 is the images of the M generators, and\n");
        printf("the optional FILE4 is an arbitrary word (x, n).\n\n");
45        printf("See the user manual for more information.\n\n");
        exit (1);
    }

    // Open files specified on the command line
    if ((Mdata = fopen (argv[1], "r")) == NULL)
50    {
        printf ("%s\n", "Error opening the first file");
        exit (1);
    }
    if (argc > 2)
55    {
        if ((Ndata = fopen (argv[2], "r")) == NULL)
        {
            printf ("%s\n", "Error opening the second file");
            exit (1);
60        }
        if (argc > 3)
        {
            if ((otherdata = fopen (argv[3], "r")) == NULL)
            {
65                printf ("%s\n", "Error opening the third file");
                exit (1);
            }
            if (argc == 5)
            {
70                if ((testdata = fopen (argv[4], "r")) == NULL)
                {
                    printf ("%s\n", "Error opening the fourth file");
                    exit (1);
75                }
            }
        }
    }

    // Read in the Information for M
80    M = fMonListFromFile( Mdata );
    lengthM = fMonListLength( M );
    nOfGen = lengthM;
```

```

rwsM = fMonPairListFromFile( Mdata );

85 // Read in the other information if required
if (argc > 2)
{
    N = fMonListFromFile( Ndata );
    rwsN = fMonPairListFromFile( Ndata );
90     Mgenerators = secondLineFMons( otherdata, nOfGen );
}

printf("\nData read in. Now processing...\n");

95 // Display some information here
printf("\nGenerating set for monoid M = \n");
fMonListDisplay( M );
printf("[%i generator(s)]\n", fMonListLength( M ));
printf("\nM is presented as follows:\n");
100 fMonPairListDisplay( rwsM );
printf("[%i rule(s)]\n", fMonPairListLength( rwsM ));

// Compute crws for M
printf("\nNow computing a complete rewrite system for M....\n");
105 crwsM = fMonKnuthBendix( rwsM );
printf("\n...Complete rewrite system for M computed.\n");

printf("\nThis is the Complete rewrite system for M:\n");
fMonPairListDisplay( crwsM );
110 printf("[%i rule(s)]\n", fMonPairListLength( crwsM ));

// Compute the elements of M
elts = fMonMonoidElements( M, crwsM );
lengthElts = fMonListLength( elts );
115 printf("\nElements of the monoid M:\n");
fMonListDisplay( elts );
printf("[%i element(s)]\n", lengthElts);

// Continue to process the data if required
120 if (argc > 2)
{
    // Calculate the information for N.
    printf("\nContinuing to process the data...\n");

125     printf("\nGenerating set for monoid N = \n");
    fMonListDisplay( N );
    printf("[%i generator(s)]\n", fMonListLength( N ));
    printf("\nN is presented as follows:\n");
    fMonPairListDisplay( rwsN );
130     printf("[%i rule(s)]\n", fMonPairListLength( rwsN ));

    // Compute crws for N
    printf("\nNow computing a complete rewrite system for N....\n");
    crwsN = fMonKnuthBendix( rwsN );
135     printf("\n...Complete rewrite system for N computed.\n");

    printf("\nThis is the complete rewrite system for N:\n");
    fMonPairListDisplay( crwsN );
    printf("[%i rule(s)]\n", fMonPairListLength( crwsN ));

140     printf("\nContinuing to process the data...\n");
    printf("\nThe number of generators in M is %i.\n", nOfGen);
    printf("\nImages of M generators: \n");
    fMonListDisplay( Mgenerators );
145     printf("[%i image(s)]\n", fMonListLength( Mgenerators ));

    printf("\n");

    // Now construct an initial rewrite system of type (x, n)
    // by computing the action for M
    rws2 = intFMonPairListNul;
    empty = fMonOne();

    // Generate initial rws
    transferGen = fMonListCopy( Mgenerators );
    mcopy = fMonListCopy( M );
    for(i = 1; i <= nOfGen; i++)
    {
        eltscopy = fMonListCopy( elts );
        genElement = Mgenerators -> first;
        genElement = fMonWordReduce( genElement, crwsN );
        for(j = 1; j <= lengthElts; j++)
        {
            // Build up the Pair List rules
            eltsElement = eltscopy -> first;
            eltscopy = eltscopy -> rest;
165             // printf("Multiplying %s with %s and reducing...\n",
            //             fMonToStr(eltsElement), fMonToStr(genElement));

```

```

    build = Mcopy -> first;
170   answer = fMonWordReduce( fMonTimes(eltsElement, build), crwsM );
    position = fMonListPosition( answer, elts );
    rws2 = intFMonPairListPush( j, genElement, position, empty, rws2 );
}
Mcopy = Mcopy -> rest;
175 Mgenerators = Mgenerators -> rest;
}
rws2 = intFMonPairListFXRev( rws2 );
printf("Type 2 rules for the initial rewrite system:\n");
intFMonPairListDisplay( rws2 );
180 printf("[%i rule(s)]\n", intFMonPairListLength( rws2 ));

// Now compute a crws for elements of the form (x, n)
printf("\nNow computing a complete rewrite system for ");
printf("elements of the form (x, n):\n");
185 crwsXW = intFMonKnuthBendix( crwsN, rws2 );
printf("\n...Complete rewrite system computed for elements");
printf(" of the form (x, n).\n");
printf("\nThe following is the Complete Rewrite System:\n");
printf("\n-----TYPE 1 RULES-----\n");
190 fMonPairListDisplay( crwsN );
printf("[%i rule(s)]\n", fMonPairListLength( crwsN ));
printf("\n-----TYPE 2 RULES-----\n");
intFMonPairListDisplay( crwsXW );
printf("[%i rule(s)]\n", intFMonPairListLength( crwsXW ));
195

// Compute the elements for the induced set
printf("\n-----ELEMENTS-----\n");
eltsIW = intFMonMonoidElements( lengthElts, N, crwsN, crwsXW );
intFMonListDisplay( eltsIW );
200 printf("[%i element(s)]\n", intFMonListLength(eltsIW));

printf("\n-----\n\n");

// Now, if specified, we try to reduce an intFMon to normal form
205 if (argc == 5)
{
    printf("The word (x, n) to reduce is as follows:\n");
    inputTest = intFMonListFromFile( testdata );
    inputTest2 = intFMonListCopy( inputTest );
210 printf("(%i, %s)", inputTest2 -> i, fMonToStr(inputTest2 -> mon));
    // intFMonListDisplay( inputTest );

    printf("\nNow Reducing....\n");
    redWord = intFMonListWordReduce( inputTest, crwsN, crwsXW, 0 );
215 printf("\n...Reduction Finished.\nThe normalised form");
    printf(" of the word is as follows\n");
    printf("(it belongs to the following equivalence class):\n");
    intFMonListDisplay( redWord );
}
}
220 return 0;
}

#include "rwse.c"

```