AESTHETIC Prototype Report

minimum complexity computation in primary auditory patterns

MARCO RENÉ SPRUIT, FEBRUARY 1995

Contents

Prologue

The AESTHETIC Prototype Report[1] presented here describes an implementation of a perceptual model for primary auditory patterns[2].

Context

The research by [Leeuwenberg, 1971][3] was taken as a starting point for two reasons:

a) Inheritance of fundamental notions:

"Finally, since the number of visual patterns that can be stored in long-term memory is so extremely large (Shepard, 1967), the coding for these visual objects is based on efficiency principles. Consequently, the coding system is applicable only to long-term storage. In contrast to the long-term capacity stands the very limited amount of patterns that can be assimilated at the same time or evoked in the imagination. So the imaginal representation is, in content capacity, quite different from the long-term memory. Hence the imaginal representation must be different from the long-term coding (Moran, 1969)." [Leeuwenberg, 1971]

"Unfortunately, the proposed coding language is without doubt inappropriate for music." [Leeuwenberg, 1971]

First of all, the concept of minimum complexity, or efficiency principle, for modelling human perception is inherited from [Leeuwenberg, 1971]. Also, the notion of dividing the perception process into an *exposure* phase and a *storage* phase which need to be processed differently has been considered essential.

Although [Leeuwenberg, 1971] focuses primarily on modelling the perception of visual patterns, it is argued that the fundamental concept could also account for the perception of auditory patterns. However, this does only seem a reasonable assumption if the phrase *perception of visual patterns* is extended to *perception of visual patterns within a monotonously passing timespan* to do just to the most important musical parameter of all: **time**.

b) Inheritance of coding terminology:

Terms imported in AESTHETIC include *iteration, alteration, code, chunk*, [...].

Due to the implementation of auditory patterns only, a specialised set of terms has been introduced to provide for complete timing information during the process, such as position indices. The auditory operation *transposition* is introduced. The

[1] The picture on the front cover of this report is the application icon of the AESTHETIC Prototype.

[2] These are patterns a listener perceives and analyses without modifying the pattern itself. See the section *Epilogue* on secondary auditory patterns for more details.

[3] Leeuwenberg, E.L.J., **A perceptual coding language for visual and auditory patterns**, American Journal of Psychology, 1971, Vol. 84, No. 3.

operation *alteration* is interpreted as a measure of **fuzziness**[4]. For a complete overview of code components, see the sections *Classes* and *Results*.

Interpretation

In the prototype, the exposure- or analysis- or building phase has been incorporated via an implementation of the *chart parser algorithm*[5]. Combining elements recursively into new elements can be considered a parallel process. There is no restraining order in which the elements may be combined. In the implementation of the algorithm, the only time-sensitive aspect consists of the initial combining order.

An essential issue in non-chronological processing is the potential danger of combinatorial explosion of possibilities. To solve this adequately, the time parameter has been introduced in the prototype at a higher level via a *shifting window algorithm*[6]. This part of the process is referred to as the storage- or merging phase. Processing data discontinuously by shifting a static window over the data can be considered a serial process. In the implementation of the algorithm, the only time-insensitive aspect consists of the shifting of dynamic child windows within the static window at the lowest process level. An important advantage of the shifting window algorithm is that the size of the data that needs to be processed is irrelevant, i.e. the process becomes *scaleable*.

The parallel component of the minimisation process consists of shifting dynamic child windows within the static shifting window to generate all elementary codes. These are stored in the window container. The chart parser recursively combines elementary codes until the minimum complexity codes emerge. This is determined by collecting the codes with the lowest atom total. This collection is stored in the document container and control is passed to the serial component. This completes the building phase. It might be comparable to some extent with the working of the auditory short-term memory.

The serial component of the minimisation process consists of shifting a static window over the data. After analysis, the resulting codes are synchronised, stored and merged with their preceding context. Finally, the window is shifted and control is passed again to the parallel component. This comprises the merging phase. It might be comparable to some extent with the working of the auditory long-term memory.

[4] See section *Prototype - Results - Example 8* for details.

[5] See section *Prototype - Internal - Algorithms - 2: Combine()* for details.

[6] See section *Prototype - Internal - Algorithms - 0: Process()* for details.

Prototype

The AESTHETIC Prototype has been implemented with the *Microsoft Visual C++ Development System for Windows, Professional Edition, Version 1.51*. The implementation has been based on the *Microsoft Foundation Classes (MFC) Library*.

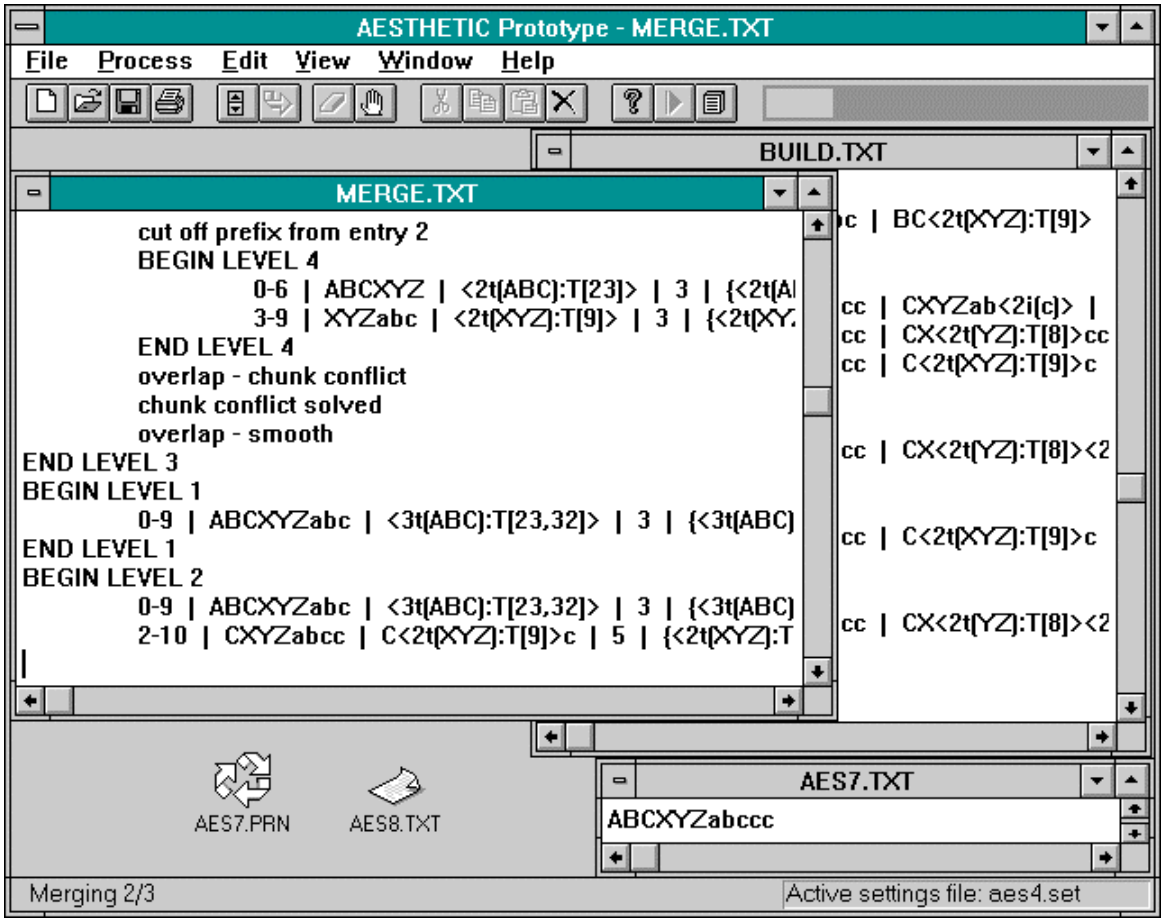
In the next section, *Prototype - External,* the AESTHETIC workspace and a selection of dialogues are presented. These dialogues prompt the user for AESTHETIC-specific information.

In the section *Prototype - Internal - Algorithms* the core algorithms are presented. Although this selection comprises merely about five percent of the source code, it should provide a sufficient impression of the degree of algorithmic complexity.

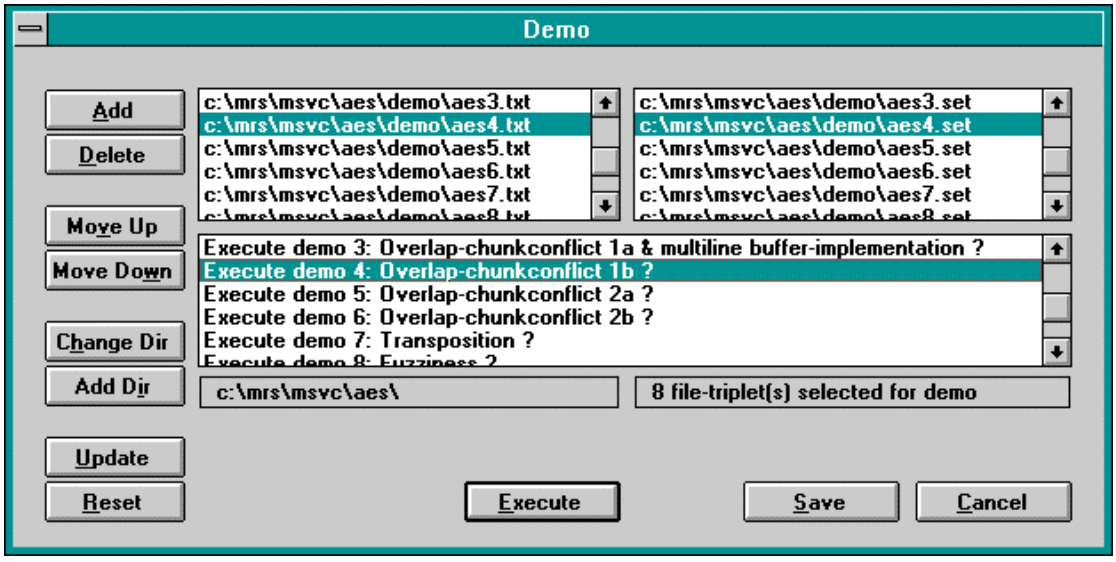
External

This section shows the prototype from a user point of view. AESTHETIC is a Multiple Document Interface (MDI) application with multitasking functionality. This means that any number of documents can be open at the same time and implies that all parameters can be altered at any time. Therefore, the prototype encapsulates maximum flexibility. To further enhance user friendliness the current subprocess is always reported in the statusbar. The actual progress of the entire process is visualised in the fuel gauge meter at the right side of the toolbar. Also, the toolbar contains tooltips.

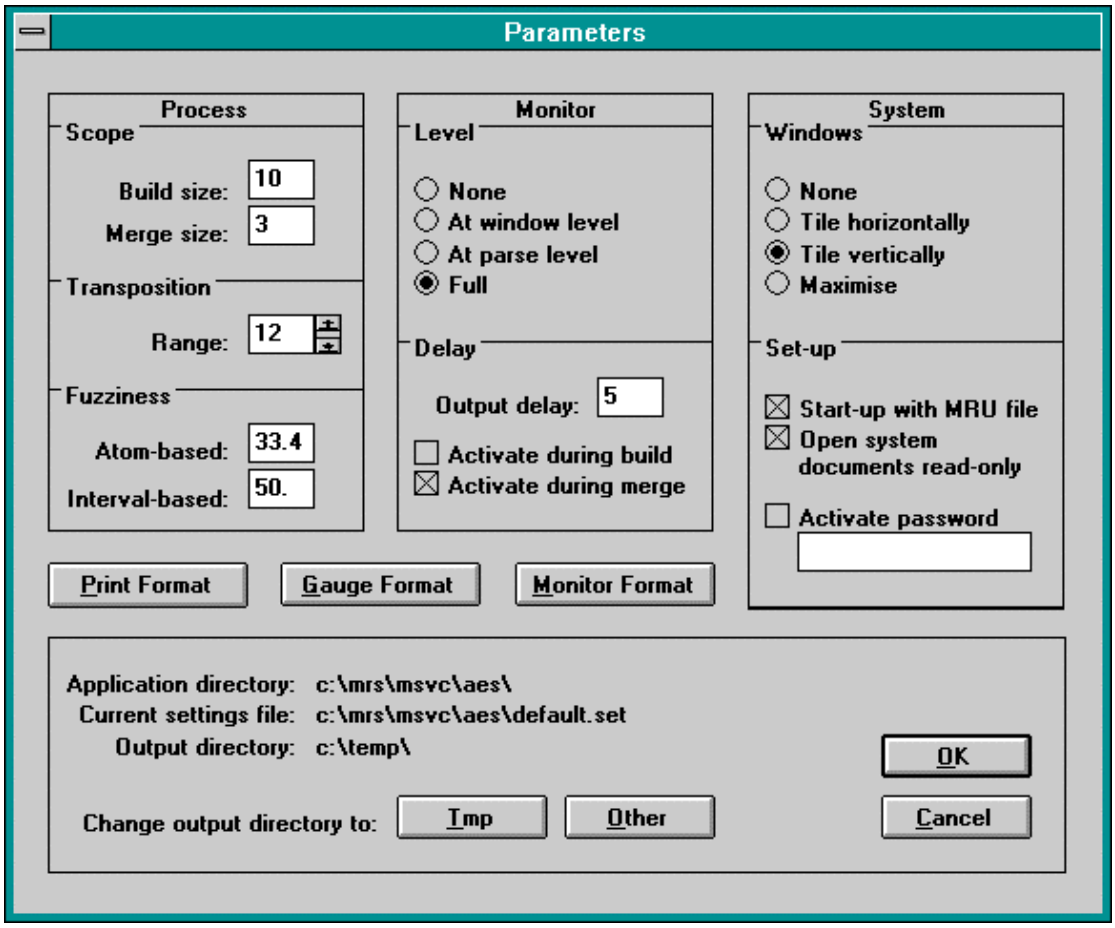
Workspace



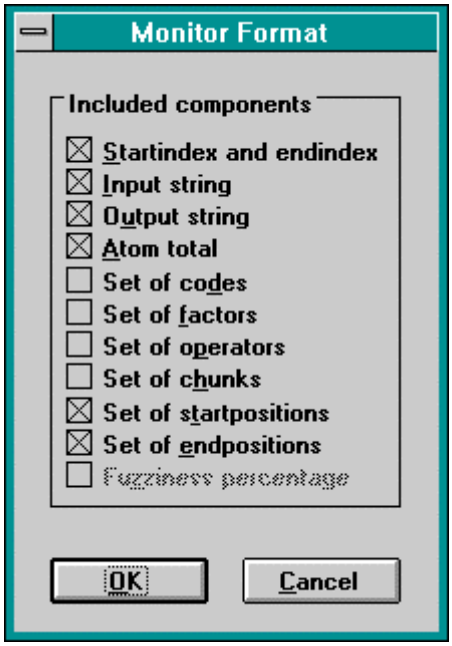
Demo configuration



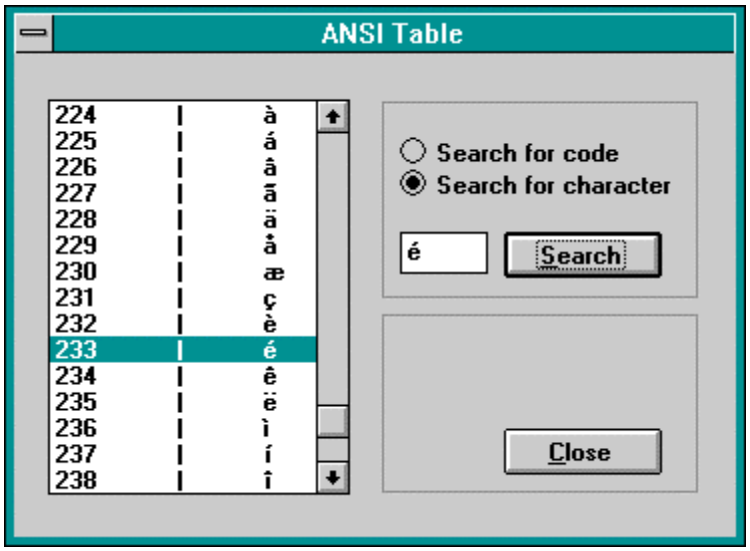
Parameters



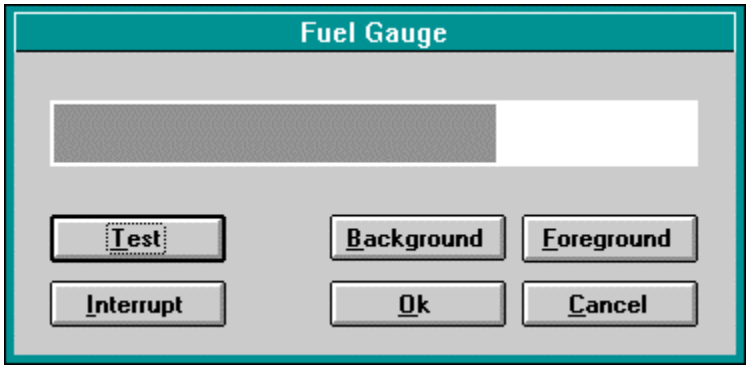
Monitor format



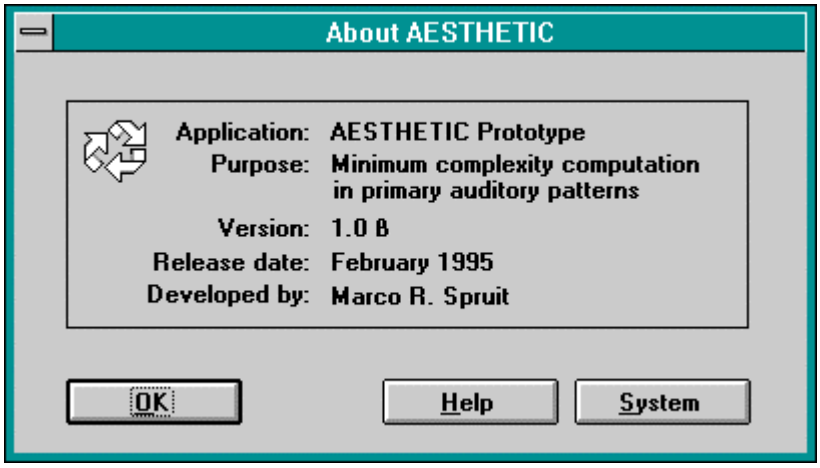
ANSI character table



Fuel gauge meter



About AESTHETIC



Internal

Legend

The extracts provided in the following two sections are selections of the actual implementation in Visual C++. The syntax of the sources can be considered an interpretation of a formal language since mainly higher levels in the function hierarchy are presented here. These sources will even be easier to understand when the following conventions are kept in mind:

* Lines starting with '*//*' are comments and explain the next piece of code.
* Class declaration:
  - Function members with a suffix between curly brackets '{' and '}' indicate an in-line implementation.
  - The declaration keyword 'protected:' means that the successive members can only be accessed from within the object itself.
* Class access:
  - Members of pointer-to-objects are accessed through '->'**,** for example *pWindowContainer->GetSize()*.
  - Members of objects are accessed through '.'**,** for example *aiTranspositions.GetSize()*.
* Variable denotation:
  - Pointer-to-objects are prefixed with ***p***, for example *pWindowContainer*.
  - (Arrays of) integers, (arrays of) strings, (arrays of) floating point-values, etc. are prefixed with ***(a)i***, ***(a)s***, ***(a)f***, etc., for example *aiTranspositions*.
  - Current-class data members are prefixed with ***m_***, for example *m_iWindowSize*.

Classes

The implemented class hierarchy is first described to exemplify some important class relationships:

* The workspace object, CMainFrame, contains pointers to its document view objects.
* Each document view object, CEsthView, contains a pointer to its document object.
* Each document object, CEsthDoc, contains a pointer to its database object.

* Each database object, CCodeContainer, contains an object array.
* Each object array, CObArray, contains pointers to code objects.
* Each code object, CCode, contains the information for one event in the process, such as the output string, the atom total, the operator object array, etc.

The declaration of the class-tuple presented in this section, which also contains the in-line-implementation component, maintains the internal database that contains all the information generated by the process.

CCodeContainer

```
class CCodeContainer : public CObject
{
public:
   DECLARE_DYNCREATE(CCodeContainer)//   A WINDOWS MACRO FOR RUN-TIME CLASS CREATION
   CCodeContainer();                  //   THE DEFAULT CONSTRUCTOR
   ~CCodeContainer();                 //   THE OBJECT DESTRUCTOR

//   ATTRIBUTES
protected:
   CEsthDoc* m_pDocument;             //   A POINTER TO THE DOCUMENT OF THIS CONTAINER
   CObArray  m_codeContainer;         //   THE EMBEDDED CONTAINER OBJECT
   CString   m_sDivider;              //   CODE ELEMENT DIVIDER-STRING IN PRINT
   int       m_nWindowLevelInfo;      //   }
   int       m_nCombinationLevelInfo; //   }MONITOR LEVELS
   int       m_iCompleteMonitoring;   //   }
   CReportreport;                  //   INTERFACE LIBRARY
   CLib      lib;                     //   COMMON LIBRARY

//   OPERATIONS
public:
   void      AddEntry(CCode* pNewElement);
   void      AddEntry(int iStartIndex, CString sInPut, CString sOutPut, CString sCode,
                  CString sChunk, char cOperator, int iFactor, int iChunkStart,
                  int iChunkEnd, int iAtomTotal, float fFuzziness);
   void      AddToContainer(CCodeContainer* pContainerToBeAdded, BOOL bAddFirstEntry);
   void      CheckEpilogue();
   void      CheckPrologue();
   BOOL      Combine(CCodeContainer* pContainer, int &iNewEntries);
   CStringConstructNewCode(CCode* p2Entry, CString &sCode, BOOL &bIdenticalRoot,
                  int &iAddFactor, int &iNewFactor, int &iOldFactorDecimals,
                  char &c1LastOperator, char &c2NextOperator);
   CCodeContainer* CopyContainer();
   CCode* CopyEntry(CCode* pCopy);
   char      DetermineOverlapMode(CString &s1LastChunk, CString &s2NextChunk,
                  char &c1LastOperator, char &c2NextOperator);
   int       DetermineEpilogue(CString &sRemainder, int &iGapSize, int
                  iLastCompressed);
   CstringDeterminePrologue(int iStartIndex);
   CstringExcludeFirstCompressionString(CString s);
   CCode*    GetCode(UINT iAtPosition)
               {return (CCode*)m_codeContainer.GetAt(iAtPosition);}
   CEsthDoc* GetLink()
               {return m_pDocument;}
   int       GetSize()
               {return m_codeContainer.GetSize();}
   BOOL      IdenticalChunkRoot(BOOL &bIdenticalRoot, CString &s1LastChunk,
                  CString &s2NextChunk, char &c1LastOperator, char &c2NextOperator);
   void      IncreaseSpecial(int iActiveEntry, int &iEdgeEntry);
   void      InitSpecial(int iActiveEntry, int &iEdgeEntry);
   void      InsertAt(int iPosition, CCode* pEntry)
               {m_codeContainer.InsertAt(iPosition, (CObject*)pEntry);}
   BOOL      Merge();
   CCode* MergeEntryTuple(char cMode, CCode* p1Entry, CCode* p2Entry,
                  int i1AbsComprPosition, int i2AbsComprPosition);
   CCode* MergeTupleCase(CCode* p1Entry, CCode* p2Entry,
```

```
                int i1AbsComprPosition, int i2AbsComprPosition);
  BOOL      PauseProcess();
  void      PreprocessTuple(CCode* p1Entry, CCode* p2Entry,CCode* p3Entry);
  CEsthDoc* PrintAll(CCodeContainer* pMetaContainer = NULL, CString sDocument =
                "ESTHETIC.PRN");
  BOOL      PrintEntry(UINT iAtPosition, int nDebugLevel = 0);
  void      PrintHeader(BOOL bIncludeNrOfEntries = TRUE);
  void      RemoveAt(int iPosition)
              {m_codeContainer.RemoveAt(iPosition);}
  CStringRetrieveGap(int i1AbsComprPosition, int i2AbsComprPosition);
  void      SetLink(CEsthDoc* pDocument)
              {m_pDocument = pDocument;}
  CCode*    SwitchCopy(CCode* pFromEntry, CCode* pToEntry, CCode* pSubjectEntry);
  BOOL      Synchronise();
  void      TupleGap(CCode* p2Entry, CCode* p3Entry,
                int i1AbsComprPosition, int i2AbsComprPosition);
  void      TupleLink(CCode* p2Entry, CCode* p3Entry);
  CCode* TupleOverlap(CCode* p1Entry, CCode* p2Entry, CCode* p3Entry,
                int i1AbsComprPosition, int i2AbsComprPosition);
  void      TupleOverlapChunkConflict(CCode* p1Entry, CCode* p2Entry, CCode* p3Entry,
                int i1AbsComprPosition, int i2AbsComprPosition, int &iComprOverlap,
                int &iInputOverlap, int &i1WindowSize, int &i2WindowSize, int
                &i1LastCompression, int &i2NextCompression, int &i3LastCompression,
                int &i1LastChunkEnd,  int &i2NextChunkEnd, char &c1LastOperator,
                char &c2NextOperator, CString &s1LastChunk, CString &s2NextChunk);
  void      TupleOverlapForceContinuation(CCode* p1Entry, CCode* p2Entry, CCode*
                p3Entry, int i1AbsComprPosition, int i2AbsComprPosition, int
                &iComprOverlap, int &iInputOverlap, int &i1WindowSize, int
              &i2WindowSize, int &i1LastCompression, int &i2NextCompression, int
              &i3LastCompression, int &i1LastChunkEnd, int &i2NextChunkEnd, char
              &c1LastOperator, char &c2NextOperator,CString &s1LastChunk, CString
              &s2NextChunk, int iAddition);
  CCode* TupleOverlapInit(CCode* p1Entry, CCode* p2Entry, CCode* p3Entry,
                int i1AbsComprPosition, int i2AbsComprPosition, int &iComprOverlap,
                int &iInputOverlap, int &i1WindowSize, int &i2WindowSize, int
                &i1LastCompression, int &i2NextCompression, int &i3LastCompression,
                int &i1LastChunkEnd,  int &i2NextChunkEnd, char &c1LastOperator,
                char &c2NextOperator, CString &s1LastChunk, CString &s2NextChunk);
  void      TupleOverlapOperatorConflict(CCode* p1Entry, CCode* p2Entry, CCode*
                p3Entry, int i1AbsComprPosition, int i2AbsComprPosition, int
                &iComprOverlap, int &iInputOverlap, int &i1WindowSize, int
              &i2WindowSize, int &i1LastCompression, int &i2NextCompression, int
              &i3LastCompression, int &i1LastChunkEnd, int &i2NextChunkEnd, char
              &c1LastOperator, char &c2NextOperator, CString &s1LastChunk,
              CString &s2NextChunk);
  void      TupleOverlapSmooth(CCode* p1Entry, CCode* p2Entry, CCode* p3Entry,
                int i1AbsComprPosition, int i2AbsComprPosition, int &iComprOverlap,
                int &iInputOverlap, int &i1WindowSize, int &i2WindowSize, int
                &i1LastCompression, int &i2NextCompression, int &i3LastCompression,
                int &i1LastChunkEnd, int &i2NextChunkEnd, char &c1LastOperator,
                char &c2NextOperator,CString &s1LastChunk, CString &s2NextChunk);
  void      TupleOverlapTotalConflict(CCode* p1Entry, CCode* p2Entry, CCode* p3Entry,
                int i1AbsComprPosition, int i2AbsComprPosition, int &iComprOverlap,
                int &iInputOverlap, int &i1WindowSize, int &i2WindowSize, int
                &i1LastCompression, int &i2NextCompression, int &i3LastCompression,
                int &i1LastChunkEnd, int &i2NextChunkEnd, char &c1LastOperator,
                char &c2NextOperator, CString &s1LastChunk, CString &s2NextChunk);
  CCode*    ValidateEntry(CCode* p1Entry, CCode* p2Entry, CCode* p3Entry);
};

//   ***end of CCodeContainer-declaration***
```

CCode

```
class CCode : public CObject
{
  DECLARE_DYNCREATE(CCode)
protected:
  CCode();                  // PROTECTED CONSTRUCTOR USED FOR RUN-TIME CREATION

//   ATTRIBUTES
protected:
```

```
  int           m_iStartIndex;
  CString     m_sInput;
  CString     m_sOutput;
  int           m_iAtomTotal;
  float         m_fFuzziness;
  CStringArray  m_sCodes;
  CStringArray  m_sChunks;
  CWordArray    m_cOperators;
  CWordArray    m_iFactors;
  CWordArray    m_iChunkStarts;
  CWordArray    m_iChunkEnds;

//   OPERATIONS
public:
  //   PUBLIC CONSTRUCTORS & DESTRUCTOR
  CCode(int iStartIndex, CString sInPut, CString sOutPut, CString sCode,
               CString sChunk, char cOperator, int iFactor, int iChunkStart,
               int iChunkEnd, int iAtomTotal, float fFuzziness);
  CCode(int iStartIndex, CString sInPut, CString sOutPut, CStringArray* asCodes,
               CStringArray* asChunks, CWordArray* acOperators, CWordArray*
               aiFactors, CWordArray* aiChunkStarts, CWordArray* aiChunkEnds, int
               iAtomTotal, float fFuzziness);
  virtual ~CCode();

  CString       GetPrint();
  void          UpdateEntry(CCode* pCopy);


  //   ACCESS TO DATA:
  //
  //   GET GENERAL INFO THROUGH..
  int           GetCompressionRatio()
                  {return m_iChunkEnds.GetSize();}
  int           EOC()     //   i.e. END OF COMPRESSION RATIO
                  {return m_iChunkEnds.GetSize();}
  int           GetMinimisationScore(CCode* pEntry);
  int           GetNearestProceedingChunkEnd(int iChunkStart);
  BOOL          IdenticalTo(CCode* pEntry);
  void          IncreaseIndex(int& iIndex);
  void          InitialiseIndex(int& iIndex);
  BOOL          IsMorePrimal(CCode* pThanEntry);
  BOOL          WithinOverlap(int iCurrentPosition, int iOverlap);

  //   GET DATA THROUGH..
  int           GetStartIndex()
                  {return m_iStartIndex;}
  CString       GetInput()
                  {return m_sInput;}
  CString       GetOutput()
                  {return m_sOutput;}
  int           GetAtomTotal()
                  {return m_iAtomTotal;}
  float         GetFuzziness()
                  {return m_fFuzziness;}
  CStringArray* GetCodes()
                  {return &m_sCodes;}
  CStringArray* GetChunks()
                  {return &m_sChunks;}
  CWordArray*   GetOperators()
                  {return &m_cOperators;}
  CWordArray*   GetFactors()
                  {return &m_iFactors;}
  CWordArray*   GetChunkStarts()
                  {return &m_iChunkStarts;}
  CWordArray*   GetChunkEnds()
                  {return &m_iChunkEnds;}
  CString       GetCode(int iAtPosition)
                  {return m_sCodes.GetAt(iAtPosition);}
  CString       GetChunk(int iAtPosition)
                  {return m_sChunks.GetAt(iAtPosition);}
  char          GetOperator(int iAtPosition)
                  {return (char)m_cOperators.GetAt(iAtPosition);}
  int           GetFactor(int iAtPosition)
                  {return (int)m_iFactors.GetAt(iAtPosition);}
  int           GetChunkStart(int iAtPosition)
                  {return (int)m_iChunkStarts.GetAt(iAtPosition);}
```

```
  int           GetChunkEnd(int iAtPosition)
                  {return (int)m_iChunkEnds.GetAt(iAtPosition);}

  //   ADD DATA THROUGH..
  void          AddEntry(CString sCode)
                  {m_sCodes.Add(sCode);}
  void          AddChunk(CString sChunk)
                  {m_sChunks.Add(sChunk);}
  void          AddOperator(char cOperator)
                  {m_cOperators.Add((WORD)cOperator);}
  void          AddFactor(int iFactor)
                  {m_iFactors.Add((WORD)iFactor);}
  void          AddChunkStart(int iStart)
                  {m_iChunkStarts.Add((WORD)iStart);}
  void          AddChunkEnd(int iEnd)
                  {m_iChunkEnds.Add((WORD)iEnd);}
  void          AddCodes(CStringArray* asCodes)
                  {m_sCodes.InsertAt(EOC(),asCodes);}
  void          AddChunks(CStringArray* asChunks)
                {m_sChunks.InsertAt(EOC(),asChunks);}
  void          AddOperators(CWordArray* acOperators)
                  {m_cOperators.InsertAt(EOC(),acOperators);}
  void          AddFactors(CWordArray* aiFactors)
                {m_iFactors.InsertAt(EOC(),aiFactors);}
  void          AddChunkStarts(CWordArray* aiStarts)
                  {m_iChunkStarts.InsertAt(EOC(),aiStarts);}
  void          AddChunkEnds(CWordArray* aiEnds)
                  {m_iChunkEnds.InsertAt(EOC(),aiEnds);}

  //   INSERT DATA THROUGH..
  void          InsertCode(CString sCode, int iPos)
                  {m_sCodes.InsertAt(iPos, sCode);}
  void          InsertChunk(CString sChunk, int iPos)
                  {m_sChunks.InsertAt(iPos, sChunk);}
  void          InsertOperator(char cOperator, int iPos)
                  {m_cOperators.InsertAt(iPos, (WORD)cOperator);}
  void          InsertFactor(int iFactor, int iPos)
                  {m_iFactors.InsertAt(iPos, (WORD)iFactor);}
  void          InsertChunkStart(int iStart, int iPos)
                  {m_iChunkStarts.InsertAt(iPos, (WORD)iStart);}
  void          InsertChunkEnd(int iEnd, int iPos)
                  {m_iChunkEnds.InsertAt(iPos, (WORD)iEnd);}

  //   MODIFY DATA THROUGH..
  void          UpdateAtomTotal(char cMode = 'r', int iNewAtomTotal = 0);
  void          UpdateChunkEnd(int iUpdatePosition, int iNewChunkEnd);
  void          UpdateChunkStart(int iUpdatePosition, int iNewChunkStart);
  void          UpdateCode(int iUpdatePosition, CString sNewCode);
  void          UpdateFactor(int iUpdatePosition, int iNewFactor);
  void          UpdateFuzziness(int iOldAtomTotal);
  void          UpdateInput(char cMode, CString sNewInput);
  void          UpdateOutput(char cMode = 'r', CString sNewInput = "");
  void          UpdateStartIndex(char cMode, int iValue);

  //   REMOVE DATA THROUGH..
  void          RemoveCode(int iPosition)
                  {m_sCodes.RemoveAt(iPosition);}
  void          RemoveChunk(int iPosition)
                  {m_sChunks.RemoveAt(iPosition);}
  void          RemoveOperator(int iPosition)
                  {m_cOperators.RemoveAt(iPosition);}
  void          RemoveFactor(int iPosition)
                  {m_iFactors.RemoveAt(iPosition);}
  void          RemoveChunkStart(int iPosition)
                  {m_iChunkStarts.RemoveAt(iPosition);}
  void          RemoveChunkEnd(int iPosition)
                  {m_iChunkEnds.RemoveAt(iPosition);}
};

//   ***end of CCode-declaration***
```

Algorithms

0 Process()

Overview

PARSE all lines by shifting a static window over the data:
BUILD the minimum complexity-codes for each window;
store these codes in the document container and
MERGE the document container after each static period of time.

Implementation

```
//   READ DATAFILE PER LINE...
//
for (iLine = 0;
     iLine < iLineTotal;
     iLine++)
{
  iBytesInLine = rEditControl.GetLine(iLine,acLine);

  //   SHIFT THE STATIC WINDOW OVER THE DATA LINE...
  //
  for (iWindowStart = 0;
       (iWindowStart+_GETAPP()->m_iWindowSize) <= (iBufferSize+iBytesInLine);
       iWindowStart++)
  {
    sWindowChunk = sBuffer + ((CString)acLine).Mid(max(iWindowStart-iBufferSize, 0),
                      min(iBytesInLine, _GETAPP()->m_iWindowSize-iBufferSize));

    //1:   GENERATE ALL ELEMENTARY WINDOW CODES AND
    //     STORE THESE IN THE WINDOW-CONTAINER...
    //
    Generate(iWindowIndex, sWindowChunk);

    //2:   COMBINE ALL ELEMENTARY WINDOW CODES AND
    //     STORE THESE IN THE DOCUMENT-CONTAINER AND
    //     RE-INITIALISE THE WINDOW CONTAINER FOR THE NEXT CYCLE...
    //
    m_pWindowContainer->Combine(m_pDocumentContainer, iNewEntries);
    UpdateWindowContainer();

    //3:   IF THE DOCUMENT CONTAINER-SIZE REACHES OR EXCEEDS THE STATIC SIZE
    //     THEN MERGE IT AGAIN INTO ONE CODE...
    //
    if (m_pDocumentContainer->GetSize() >= _GETAPP()->m_iMemorySize)
      Merge(iMergeCounter++);

    //   UPDATE CHARACTER-INDEX AND READ-BUFFER...
  } //...end of level 2
} //...end of level 1

//   MERGE REMAINDER...
Merge(iMergeCounter++, TRUE);

//   ***end of main***
```

1 Generate()

Overview

GENERATE all unique minimisations for the current input by shifting dynamic child windows over the static window and STORE these elementary codes in the window container.

Implementation

```
//   FOR ALL CHILD WINDOW-SIZES WITHIN THE SHIFTING WINDOW AND...
//
for (iChunkSize = 1;
     iChunkSize <= (iStringLength/2);
     iChunkSize++)
{
  //   FOR ALL CHILD WINDOW-POSITIONS WITHIN THE SHIFTING WINDOW...
  //
  for (iChunkStart = 0;
       iChunkStart <= (iStringLength-(2*iChunkSize));
       iChunkStart++)
  {
    aiTranspositions.RemoveAll();
    afFuzziness.RemoveAll();
    sChunk = sInput.Mid(iChunkStart, iChunkSize);
    iChunkRepeatPosition = iChunkStart + iChunkSize;
    iFactor = 1;
    iTransposition = IntervalLists(sInput.Mid(iChunkStart, iChunkSize), iIntervals1,
                                   sInput.Mid(iChunkRepeatPosition, iChunkSize),
                                   aiIntervals2, fFuzzyPercentage);

    //   ADD THE SUCCESSIVE CHUNK-MINIMISATIONS (BE IT ITERATION,
    //   TRANSPOSITION OR ALTERATION) TO THE WINDOW-CONTAINER AND
    //   DETERMINE VALUES FOR THE NEXT WINDOW-CHUNK
    //   WHILE THIS CURRENT WINDOW-CHUNK CAN BE MINIMISED.
    //
    while ((WithinTranspositionRange(iTransposition)) &&
           (CorrelatedEnough(aiIntervals1,aiIntervals2,fFuzzyPercentage)) &&
           ((iChunkRepeatPosition+iChunkSize) <= iStringLength))
    {
      aiTranspositions.Add(iTransposition);
      afFuzziness.Add(fFuzzyPercentage);
      iChunkRepeatPosition += iChunkSize;
      iFactor++;

      m_pWindowContainer->AddEntry(iWindowIndex, sInput, sOutput, sCode, sChunk,
                          cOperator, iFactor, iChunkStart, iChunkStart+
                          (iChunkSize*iFactor), iAtomTotal, fFuzzyPercentage);

      iTransposition = IntervalLists(sInput.Mid(iChunkStart, iChunkSize),
                                     aiIntervals1, sInput.Mid(iChunkRepeatPosition,
                                    iChunkSize), aiIntervals2, fFuzzyPercentage);
    } //...end of level 3
  } //...end of level 2
} //...end of level 1

//   ***end of generate***
```

2 Combine()

Overview

COMBINE all original container entries with
all container tuples for
all active entry compressions with
all container entry compressions and add the new combinations to the container
if the tuple-compressions are non-overlapping.
EVALUATE the container afterwards and
add the container entries with the lowest atomtotal to the document container.

Implementation

```
//   INITIALISE AN ORIGINAL CONTAINER ENTRY AS THE CURRENT ACTIVE ENTRY...
//
for (iActiveEntry = 0;
     iActiveEntry < iOriginalContainerSize;
     iActiveEntry++)
{
  pActiveEdge = GetCode(iActiveEntry);
  iActiveCompressionRatio = pActiveEdge->GetCompressionRatio();

  //   COMBINE THIS ACTIVE ENTRY WITH ALL REMAINING CONTAINER ENTRIES...
  //
  iContainerStepSize = GetSize();
  for (InitSpecial(iActiveEntry, iEdgeEntry);
       iEdgeEntry < iContainerStepSize;
       IncreaseSpecial(iActiveEntry, iEdgeEntry))
  {
    pCurrentEdge = GetCode(iEdgeEntry);
    iCurrentCompressionRatio = pCurrentEdge->GetCompressionRatio();

    //   MATCH ALL ACTIVE ENTRY/CONTAINER ENTRY-TUPLES...
    //
    for (iActiveSize = 0;
         iActiveSize < iActiveCompressionRatio;
         iActiveSize++)
    {
      iActiveStart = pActiveEdge->GetChunkStart(iActiveSize);
      iActiveEnd = pActiveEdge->GetChunkEnd(iActiveSize);

      //   IN ALL POSSIBLE POSITION-COMBINATIONS...
      //
      for (iCurrentSize = 0;
           iCurrentSize < iCurrentCompressionRatio;
           iCurrentSize++)
      {
        iCurrentStart = pCurrentEdge->GetChunkStart(iCurrentSize);
        iPreviousEnd = pCurrentEdge->GetNearestProceedingChunkEnd(iCurrentStart);

        //   AND ADD ALL NON-OVERLAPPING COMBINATIONS TO THE CONTAINER.
        //
        if ((iPreviousEnd <= iActiveStart) && (iActiveEnd <= iCurrentStart))
        {
          pNewEntry = CopyEntry(pActiveEdge);
          pNewEntry->UpdateEntry(pCurrentEdge);
```

```
            AddEntry(pNewEntry);
          }
        } //...end of level 4
      } //...end of level 3
    } //...end of level 2
} //...end of level 1

//   FIND THE ENTRIES WITH THE LOWEST ATOMTOTAL...
//
iCurrentAtomMinimum = INT_MAX;
iOriginalContainerSize = GetSize();
for (iEdgeEntry = 0;
     iEdgeEntry < iOriginalContainerSize;
     iEdgeEntry++)
{
  pCurrentEdge = GetCode(iEdgeEntry);
  if (pCurrentEdge->GetAtomTotal() < iCurrentAtomMinimum)
  { awMinimumCodePositions.RemoveAll();
    awMinimumCodePositions.Add(iEdgeEntry);
    iCurrentAtomMinimum = pCurrentEdge->GetAtomTotal();
  }
  else
  { if (pCurrentEdge->GetAtomTotal() == iCurrentAtomMinimum)
      awMinimumCodePositions.Add(iEdgeEntry);
  }
} //...end of level 1


//   ADD THOSE ENTRIES TO THE DOCUMENT CONTAINER.
//
iNewEntries = awMinimumCodePositions.GetSize();
for (iNewCode = 0;
     iNewCode < iNewEntries;
     iNewCode++)
{
  pDocumentContainer->AddEntry(GetCode(awMinimumCodePositions.GetAt(iNewCode)));
} //...end of level 1

//   ***end of combine***
```

3 Merge()

Overview

STORE the window-level codes permanently in the history container and
SYNCHRONISE the new entries internally and
MERGE all tuples in the document container.

Implementation

```
//   STORE ALL WINDOW-LEVEL CODES IN THE HISTORY CONTAINER
//
m_pHistoryContainer->AddToContainer(m_pDocumentContainer);

//   IN THE COMBINE ALGORITHM, THE TIME PARAMETER IS IRRELEVANT.
//   HOWEVER, MERGING MUST BE PERFORMED IN STRICT SPATIAL ORDER.
//   THEREFORE, THE INTERNAL CHUNK-COMPRESSIONS MUST FIRST BE SYNCHRONISED...
m_pDocumentContainer->Synchronise();

//   FIRST MERGING ONLY:
//   RETRIEVE THE POSSIBLY UNCOMPRESSED PROLOGUE...
```

```
if (iMergeCounter == 0)
  m_pDocumentContainer->CheckPrologue();

//3.1: MERGE THE FIRST TWO ENTRIES IN THE DOCUMENT CONTAINER
//   WHILE THE CONTAINER CONSISTS OF MULTIPLE CODES.
iDynamicSize = iStaticSize = m_pDocumentContainer->GetSize();
while (iDynamicSize > 1)
{
  m_pDocumentContainer->MergeTuple();
  iDynamicSize--;
} //...end of level 1

//   LAST MERGING ONLY:
//   RETRIEVE THE POSSIBLY UNCOMPRESSED EPILOGUE...
if (bFinalMerge)
  m_pDocumentContainer->CheckEpilogue();

//   ***end of merge***
```

3.1 MergeTuple()

Overview

MERGE the first tuple in the document container based on the absolute end position of the last compression in the first entry and the absolute start position of the first compression in the second entry.

VALIDATE the result and

REPLACE this tuple with the merged version.

Implementation

```
//   GET THE FIRST TUPLE FROM THE DOCUMENT CONTAINER AND
//   INITIALISE THE NEW ENTRY WITH THE CONTENTS OF THE FIRST CODE...
//
p1Entry = GetCode(0);
p2Entry = GetCode(1);
p3Entry = CopyEntry(p1Entry);

//   DETERMINE THE ABSOLUTE END POSITION OF THE FIRST ENTRY (i.e. p1Entry) AND
//   THE ABSOLUTE START POSITION OF THE SECOND ENTRY (i.e. p2Entry)...
//
i1InputLength = p1Entry->GetInput().GetLength();
i1LastChunkEnd = p1Entry->GetChunkEnds()->GetAt(p1Entry->EOC()-1);
i1RemainderLength = i1InputLength - i1LastChunkEnd;
i1AbsComprPosition = (i1InputLength - i1RemainderLength) + p1Entry->GetStartIndex();

i2FirstChunkStart  = p2Entry->GetChunkStarts()->GetAt(0);
i2AbsComprPosition = i2FirstChunkStart + p2Entry->GetStartIndex();

//3.1.1: MERGE THE TUPLE BASED ON THE DETERMINED ABSOLUTE
//   POSITIONS AND RETURN THE MERGED VERSION (i.e. p3Entry)...
//
if (i1AbsComprPosition > i2AbsComprPosition)
  p3Entry = MergeTupleOverlap(p1Entry, p2Entry, p3Entry,
                              i1AbsComprPosition, i2AbsComprPosition);
else
  p3Entry = MergeTupleLink(p1Entry, p2Entry, p3Entry,
                              i1AbsComprPosition, i2AbsComprPosition);

//   VALIDATE THE RESULT...
```

```
//
p3Entry = ValidateEntry(p1Entry, p2Entry, p3Entry);


//   REPLACE THE FIRST TUPLE IN THE DOCUMENT CONTAINER WITH THE MERGED VERSION...
//
RemoveAt(0);
RemoveAt(0);
InsertAt(0, p3Entry);

//   ***end of mergeTuple***
```

3.1.1 MergeTupleOverlap()

Overview

PREPROCESS the tuple by removing noise and

REDIRECT the process to specialised instances.

Implementation

```
//   PREPARE THE TUPLE BY CUTTING OFF NOISE FROM THE END OF ENTRY 1 AND
//   OF THE BEGINNING OF ENTRY 2
//
PreprocessTuple(p1Entry, p2Entry, p3Entry);

//   DETERMINE THE TYPE OF OVERLAP BASED ON THE OPERATORS/CHUNKS-COMBINATION...
//
switch (MergeTupleOverlapMode(s1LastChunk,s2NextChunk,c1LastOperator,c2NextOperator))
{
  case 's':
  { MergeTupleOverlapSmooth(p1Entry, p2Entry, p3Entry, i1AbsComprPosition,
              i2AbsComprPosition, iComprOverlap, iInputOverlap, i1WindowSize,
              i2WindowSize, i1LastCompression, i2NextCompression, i3LastCompression,
              i1LastChunkEnd, i2NextChunkEnd,c1LastOperator,
            c2NextOperator,s1LastChunk, s2NextChunk);
    break;
  }
  case 'c':
  { // SEE 3.1.1.1
    //
    MergeTupleOverlapChunkConflict(p1Entry, p2Entry, p3Entry, i1AbsComprPosition,
             i2AbsComprPosition, iComprOverlap, iInputOverlap, i1WindowSize,
             i2WindowSize, i1LastCompression, i2NextCompression, i3LastCompression,
               i1LastChunkEnd, i2NextChunkEnd, c1LastOperator,
               c2NextOperator,s1LastChunk, s2NextChunk);
    break;
  }
  case 'o':
  { MergeTupleOverlapOperatorConflict(p1Entry, p2Entry, p3Entry, i1AbsComprPosition,
              i2AbsComprPosition, iComprOverlap, iInputOverlap, i1WindowSize,
              i2WindowSize, i1LastCompression, i2NextCompression, i3LastCompression,
              i1LastChunkEnd, i2NextChunkEnd, c1LastOperator,
              c2NextOperator,s1LastChunk, s2NextChunk);
    break;
  }
  case 't':
  { MergeTupleOverlapTotalConflict(p1Entry, p2Entry, p3Entry, i1AbsComprPosition,
             i2AbsComprPosition, iComprOverlap, iInputOverlap, i1WindowSize,
             i2WindowSize, i1LastCompression, i2NextCompression, i3LastCompression,
               i1LastChunkEnd, i2NextChunkEnd, c1LastOperator,
               c2NextOperator,s1LastChunk, s2NextChunk);
    break;
```

```
  }
}

//   ***end of mergeTupleOverlap***
```

3.1.1.1 MergeTupleOverlapChunkConflict()

Overview

DETERMINE which entry-chunk offers the highest compression (to start the actual merging with) but

REDIRECT the process again if the addition is a compression by itself.

Implementation

```
//   DETERMINE THE HIGHEST COMPRESSED CHUNK
//   (TO BE USED AS THE STARTING POINT FOR THE MERGED VERSION)...
//
i2NextChunkStart = p2Entry->GetChunkStart(i2NextCompression);
i1RelLastChunkStart = p1Entry->GetChunkStart(i1LastCompression);
i1RelLastChunkEnd = p1Entry->GetChunkEnd(i1LastCompression);

if ((i1RelLastChunkEnd - i1RelLastChunkStart) >= (i2NextChunkEnd - i2NextChunkStart))
{
  //   THE FIRST ENTRY IS THE BEST STARTING POINT...
  //
  iAddition = (p2Entry->GetStartIndex() + p2Entry->GetInput().GetLength()) -
              (p1Entry->GetStartIndex() + p1Entry->GetInput().GetLength());
  sAddition = p2Entry->GetInput().Right(iAddition);

  if (iAddition > 1)
  {
    //   THE NON-OVERLAPPING PART OF THE SECOND ENTRY IS A COMPRESSION IN ITSELF,
    //   SO REDIRECT THE PROCESS AGAIN...
    //
    if ((p2Entry->GetFactor(i2NextCompression)*
         p2Entry->GetChunk(i2NextCompression).GetLength()) > 2)
    {
      //   SEE 3.1.1.1a
      //
      TupleOverlapForceContinuation(p1Entry, p2Entry, p3Entry, ...);
    }
  }
  else
  {
    //   THE NON-OVERLAPPING PART OF THE SECOND ENTRY IS NOISE,
    //   SO A SIMPLE ADDITION WILL RESULT IN THE MERGED VERSION...
    //
    p3Entry->UpdateInput('a', sAddition);
    p3Entry->UpdateAtomTotal('a', iAddition);
    p3Entry->UpdateOutput('r');
  }
}
else
{
  if (_GETAPP()->m_iMonitorLevel == m_iCompleteMonitoring)
    report.Add(TAB + "NOT IMPLEMENTED YET: overlap - chunk conflict - " +
               "<<Longest code is in entry two>>" + NL);
}

//   ***end of mergeTupleOverlapChunkConflict***
```

3.1.1.1a MergeTupleOverlapForceContinuation()

Overview

FORCE a smooth overlap between the overlapping entries by
REMOVING the overlap from the second entry and
REDIRECT the process again.

Implementation

```
//   DETERMINE THE SIZE OF THE CHUNK THAT IS TO BE REMOVED FROM THE SECOND ENTRY AND
//   HOW THIS WILL EFFECT ITS ENTRY...
//
iDecrease = p2Entry->GetInput().GetLength() - iAddition;
s2DecreaseChunk = p2Entry->GetInput().Left(iDecrease);

bSkipRemoval = FALSE;
if ( (p2Entry->GetChunk(i2NextCompression) == s2DecreaseChunk) &&
     (IdenticalChunkRoot(bIdenticalRoot, s1LastChunk, s2DecreaseChunk,
                         c1LastOperator, c2NextOperator)))
{
  //   THE CHUNKS MATCH INDIRECTLY (AS IS THE CASE WITH, FOR EXAMPLE, A TRANSPOSITION-
  //   TUPLE), SO DON'T REMOVE THE OVERLAP (INVOKED MESSAGE: CHUNK-CONFLICT SOLVED)...
  //
  bSkipRemoval = TRUE;
}

if (!bSkipRemoval)
{
  p2Entry->UpdateStartIndex('a', iDecrease);
  p2Entry->UpdateInput('r', p2Entry->GetInput().Right(iAddition));

  if ((p2Entry->GetFactor(i2NextCompression)) <= 2)
  {
    //   AFTER REMOVAL, THE OVERLAPPING CHUNK IS NOT COMPRESSED ANYMORE,
    //   SO REMOVE ALL ITS COMPONENTS...
    //
    p2Entry->RemoveCode(i2NextCompression);
    p2Entry->RemoveChunk(i2NextCompression);
    p2Entry->RemoveOperator(i2NextCompression);
    p2Entry->RemoveFactor(i2NextCompression);
    p2Entry->RemoveChunkStart(i2NextCompression);
    p2Entry->RemoveChunkEnd(i2NextCompression);
    p2Entry->UpdateOutput('r');
  }
  else
  {
    //   AFTER REMOVAL, THE OVERLAPPING CHUNK IS STILL A COMPRESSION,
    //   SO UPDATE ALL ITS COMPONENTS...
    //
    iOldFactorDecimals = lib.ItoS(p2Entry->GetFactor(i2NextCompression)).GetLength();
    sNewCode = p2Entry->GetCode(i2NextCompression);
    sNewCode = sNewCode.Right(sNewCode.GetLength() - sNewCode.Find('>')+1);
    sNewCode =  sNewCode.Left(1) + lib.ItoS(p2Entry->GetFactor(i2NextCompression) - 1)
                + sNewCode.Right(sNewCode.GetLength() - 1 - iOldFactorDecimals);

    p2Entry->UpdateFactor(i2NextCompression,
                          p2Entry->GetFactor(i2NextCompression) - 1);
    p2Entry->UpdateChunkEnd(i2NextCompression,
                            p2Entry-GetChunkEnd(i2NextCompression) - iDecrease);
    p2Entry->UpdateCode(i2NextCompression, sNewCode);
```

```
      p2Entry->UpdateOutput('r');
   }
}

//   THE OVERLAP HAS BEEN REMOVED, SO THE TUPLE IS READY TO BE MERGED SMOOTHLY...
//
TupleOverlapSmooth(p1Entry, p2Entry, p3Entry, i1AbsComprPosition, i2AbsComprPosition,
                   iComprOverlap, iInputOverlap, i1WindowSize, i2WindowSize,
                   i1LastCompression, i2NextCompression, i3LastCompression,
                   i1LastChunkEnd, i2NextChunkEnd, c1LastOperator, c2NextOperator,
                   s1LastChunk, s2NextChunk);

//   ***end of mergeTupleOverlapForceContinuation***
```

Results

The results provided in this section are copies of AESTHETIC Print files, the final output of the minimisation process.

This section is concluded with a print of the system documents *BUILD.TXT* and *MERGE.TXT* to exemplify the minimisation process from a user point of view.

Examples

1 Simple link

PROCESS PARAMETERS:
Build codes at size : 2
Merge codes at size : 2
Transposition range : 0
Fuzzy Atom percentage : 0.000000
Fuzzy Interval percentage : 0.000000

INPUT

xxyy

OUTPUT

Current date : Friday, February 24, 1995
Current time : 12:31:24

Column 1: startindex and endindex of input;
Column 2: input string;
Column 3: output string;
Column 4: atom total;
Column 5: set of compressions;
Column 6: set of factors;
Column 7: set of operators;

Number of entries in window-level container: 2

MINIMUM COMPLEXITY CODE:

0-4 | xxyy | <2i(x)><2i(y)> | 2 | {<2i(x)>, <2i(y)>} | {2, 2} | {i, i}

EMERGED FROM WINDOW-LEVEL CODES:

0-2 | xx | <2i(x)> | 1 | {<2i(x)>} | {2} | {i}
2-4 | yy | <2i(y)> | 1 | {<2i(y)>} | {2} | {l}

2 Prologue-, epilogue- and gap-implementation

PROCESS PARAMETERS:
Build codes at size : 5
Merge codes at size : 3
Transposition range : 256
Fuzzy Atom percentage : 0.000000
Fuzzy Interval percentage : 0.000000

INPUT

qwefra
a
juy

hl

oaaaaarubc
p

OUTPUT

Current date : Sunday, February 12, 1995
Current time : 17:57:08

Column 1: startindex and endindex of input;
Column 2: input string;
Column 3: output string;
Column 4: atom total;
Column 5: set of compressions;

Number of entries in window-level container: 13

MINIMUM COMPLEXITY CODE:

0-23 | qwefraajuyhloaaaaarubcp | qwefr<2i(a)>j<2t(uy):T[-13]>o<5i(a)>rubcp | 16 | {<2i(a)>, <2t(uy):T[-13]>, <5i(a)>}

EMERGED FROM WINDOW-LEVEL CODES:

2-7 | efraa | efr<2i(a)> | 4 | {<2i(a)>}
3-8 | fraaj | fr<2i(a)>j | 4 | {<2i(a)>}
4-9 | raaju | r<2i(a)>ju | 4 | {<2i(a)>}
5-10 | aajuy | <2i(a)>juy | 4 | {<2i(a)>}
7-12 | juyhl | j<2t(uy):T[-13]> | 3 | {<2t(uy):T[-13]>}
8-13 | uyhlo | <2t(uy):T[-13]>o | 3 | {<2t(uy):T[-13]>}
10-15 | hloaa | hlo<2i(a)> | 4 | {<2i(a)>}
11-16 | loaaa | lo<3i(a)> | 3 | {<3i(a)>}
12-17 | oaaaa | o<4i(a)> | 2 | {<4i(a)>}
13-18 | aaaaa | <5i(a)> | 1 | {<5i(a)>}
14-19 | aaaar | <4i(a)>r | 2 | {<4i(a)>}
15-20 | aaaru | <3i(a)>ru | 3 | {<3i(a)>}
16-21 | aarub | <2i(a)>rub | 4 | {<2i(a)>}

3 Overlap-chunkconflict[1a] & multiline buffer-implementation

PROCESS PARAMETERS:
Build codes at size : 6
Merge codes at size : 3
Transposition range : 256
Fuzzy Atom percentage : 0.000000
Fuzzy Interval percentage : 0.000000

INPUT

QWQ
W

W
Wax
sAXS

e

OUTPUT

Current date : Saturday, February 11, 1995
Current time : 22:20:02

Column 1: startindex and endindex of input;
Column 2: input string;
Column 3: output string;
Column 4: atom total;
Column 5: set of start positions of chunks;
Column 6: set of end positions of chunks;

Number of entries in window-level container: 6

MINIMUM COMPLEXITY CODE:

0-13 | QWQWWWaxsAXSe | <2i(QW)><2i(W)><2t(axs):T[-32]>e | 7 | {0, 4, 6} | {4, 6, 12}

EMERGED FROM WINDOW-LEVEL CODES:

0-6 | QWQWWW | <2i(QW)><2i(W)> | 3 | {0, 4} | {4, 6}
1-7 | WQWWWa | WQ<3i(W)>a | 4 | {2} | {5}
2-8 | QWWWax | Q<3i(W)>ax | 4 | {1} | {4}
3-9 | WWWaxs | <3i(W)>axs | 4 | {0} | {3}
4-10 | WWaxsA | <2i(W)>axsA | 5 | {0} | {2}
6-12 | axsAXS | <2t(axs):T[-32]> | 3 | {0} | {6}

4 Overlap-chunkconflict[1b]

PROCESS PARAMETERS:
Build codes at size : 6
Merge codes at size : 3
Transposition range : 256
Fuzzy Atom percentage : 0.000000
Fuzzy Interval percentage : 0.000000

INPUT

ABCABCBCBC

OUTPUT

Current date : Sunday, February 12, 1995
Current time : 17:33:30

Column 1: startindex and endindex of input;
Column 2: input string;
Column 3: output string;
Column 4: atom total;
Column 5: set of factors;
Column 6: set of operators;
Column 7: set of chunks;

Number of entries in window-level container: 6

MINIMUM COMPLEXITY CODE:

0-10 | ABCABCBCBC | <2i(ABC)><2i(BC)> | 5 | {2, 2} | {i, i} | {ABC, BC}

EMERGED FROM WINDOW-LEVEL CODES:

0-6 | ABCABC | <2i(ABC)> | 3 | {2} | {i} | {ABC}
1-7 | BCABCB | <2t(BC):T[-1]>CB | 4 | {2} | {t} | {BC}
2-8 | CABCBC | CA<2i(BC)> | 4 | {2} | {i} | {BC}
3-9 | ABCBCB | A<2i(BC)>B | 4 | {2} | {i} | {BC}
3-9 | ABCBCB | AB<2i(CB)> | 4 | {2} | {i} | {CB}
4-10 | BCBCBC | <3i(BC)> | 2 | {3} | {i} | {BC}

5 Overlap-chunkconflict[2a]

PROCESS PARAMETERS:
Build codes at size : 5
Merge codes at size : 3
Transposition range : 256
Fuzzy Atom percentage : 0.000000
Fuzzy Interval percentage : 0.000000

INPUT

AABABABX

OUTPUT

Current date : Sunday, February 12, 1995
Current time : 17:53:21

Column 1: startindex and endindex of input;
Column 2: input string;
Column 3: output string;
Column 4: atom total;
Column 5: set of compressions;

Number of entries in window-level container: 6

MINIMUM COMPLEXITY CODE:

0-8 | AABABABX | A<3i(AB)>X | 4 | {<3i(AB)>}

EMERGED FROM WINDOW-LEVEL CODES:

0-5 | AABAB | A<2i(AB)> | 3 | {<2i(AB)>}
1-6 | ABABA | <2i(AB)>A | 3 | {<2i(AB)>}
1-6 | ABABA | A<2i(BA)> | 3 | {<2i(BA)>}
2-7 | BABAB | <2i(BA)>B | 3 | {<2i(BA)>}
2-7 | BABAB | B<2i(AB)> | 3 | {<2i(AB)>}

3-8 | ABABX | <2i(AB)>X | 3 | {<2i(AB)>}

6 Overlap-chunkconflict[2b]

PROCESS PARAMETERS:
Build codes at size : 10
Merge codes at size : 3
Transposition range : 256
Fuzzy Atom percentage : 0.000000
Fuzzy Interval percentage : 0.000000

INPUT

ABABABCABCABC

OUTPUT

Current date : Friday, February 24, 1995
Current time : 12:22:37

Column 1: startindex and endindex of input;
Column 2: input string;
Column 3: output string;
Column 4: atom total;
Column 5: set of start positions of chunks;
Column 6: set of end positions of chunks;

Number of entries in window-level container: 4

MINIMUM COMPLEXITY CODE:

0-13 | ABABABCABCABC | <2i(AB)><3i(ABC)> | 5 | {0, 4} | {4, 13}

EMERGED FROM WINDOW-LEVEL CODES:

0-10 | ABABABCABC | <2i(AB)><2i(ABC)> | 5 | {0, 4} | {4, 10}
1-11 | BABABCABCA | <2i(BA)><2i(BCA)> | 5 | {0, 4} | {4, 10}
2-12 | ABABCABCAB | <2i(AB)><2i(CAB)> | 5 | {0, 4} | {4, 10}
3-13 | BABCABCABC | B<3i(ABC)> | 4 | {1} | {10}

7 Transposition

PROCESS PARAMETERS:
Build codes at size : 7
Merge codes at size : 3
Transposition range : 48 *(Less minimisation if Transposition range < 32, no minimisation if Transposition range < 23.)*
Fuzzy Atom percentage : 0.000000
Fuzzy Interval percentage : 0.000000

INPUT

ABCXYZabccc

OUTPUT

Current date : Sunday, February 19, 1995
Current time : 18:59:42

Column 1: startindex and endindex of input;
Column 2: input string;
Column 3: output string;
Column 4: atom total;
Column 5: set of compressions;
Column 6: set of factors;
Column 7: set of operators;
Column 8: set of chunks;
Column 9: set of start positions of chunks;
Column 10: set of end positions of chunks;

Number of entries in window-level container: 7

MINIMUM COMPLEXITY CODE:

0-11 | ABCXYZabccc | <3t(ABC):T[23,32]><2i(c)> | 4 | {<3t(ABC):T[23,32]>, <2i(c)>} | {3, 2} | {t, i} | {ABC, c} | {0, 13} | {9, 15}

EMERGED FROM WINDOW-LEVEL CODES:

0-7 | ABCXYZa | <2t(ABC):T[23]>a | 4 | {<2t(ABC):T[23]>} | {2} | {t} | {ABC} | {0} | {6}
1-8 | BCXYZab | <2t(BC):T[22]>Zab | 5 | {<2t(BC):T[22]>} | {2} | {t} | {BC} | {0} | {4}
1-8 | BCXYZab | BCX<2t(YZ):T[8]> | 5 | {<2t(YZ):T[8]>} | {2} | {t} | {YZ} | {3} | {7}
2-9 | CXYZabc | C<2t(XYZ):T[9]> | 4 | {<2t(XYZ):T[9]>} | {2} | {t} | {XYZ} | {1} | {7}
3-10 | XYZabcc | <2t(XYZ):T[9]>c | 4 | {<2t(XYZ):T[9]>} | {2} | {t} | {XYZ} | {0} | {6}
3-10 | XYZabcc | X<2t(YZ):T[8]><2i(c)> | 4 | {<2t(YZ):T[8]>, <2i(c)>} | {2, 2} | {t, i} | {YZ, c} | {1, 5} | {5, 7}
4-11 | YZabccc | <2t(YZ):T[8]><3i(c)> | 3 | {<2t(YZ):T[8]>, <3i(c)>} | {2, 3} | {t, i} | {YZ, c} | {0, 4} | {4, 7}

8 Fuzziness

PROCESS PARAMETERS:
Build codes at size : 9
Merge codes at size : 3
Transposition range : 256
Fuzzy Atom percentage : 34.000000 *(No minimisation if Atom percentage < 33.3 % ÓR if Interval percentage < 50 %.)*
Fuzzy Interval percentage : 0.000000

INPUT

ABzBCyABz

OUTPUT

Current date : Friday, February 24, 1995
Current time : 12:00:59

Column 1: startindex and endindex of input;
Column 2: input string;
Column 3: output string;
Column 4: atom total;
Column 5: set of factors;
Column 6: set of operators;
Column 7: set of chunks;

Number of entries in window-level container: 6

MINIMUM COMPLEXITY CODE:

0-9 | ABzBCyABz | <3a(ABz):T[1,0]:F[33.3,33.3]> | 3 | {3} | {a} | {ABz}

EMERGED FROM WINDOW-LEVEL CODES:

```
0-7 | ABzBCyA | <2a(ABz):T[1]:F[33.3]>A | 4 | {2} | {a} | {ABz}
0-7 | ABzBCyA | A<2a(BzB):T[-1]:F[33.3]> | 4 | {2} | {a} | {BzB}
1-8 | BzBCyAB | <2a(BzB):T[-1]:F[33.3]>B | 4 | {2} | {a} | {BzB}
1-8 | BzBCyAB | B<2t(zBC):T[-1]> | 4 | {2} | {t} | {zBC}
2-9 | zBCyABz | <2t(zBC):T[-1]>z | 4 | {2} | {t} | {zBC}
2-9 | zBCyABz | z<2a(BCy):T[-1]:F[33.3]> | 4 | {2} | {a} | {Bcy}
```

Trace

build.txt

[fragment of example 6]

```
Processing esth4.txt (13 characters)

BEGIN LEVEL 2
   0-10 | ABABABCABC | <2i(AB)>ABCABC | 8 | {0} | {4}
   0-10 | ABABABCABC | <3i(AB)>CABC | 6 | {0} | {6}
   0-10 | ABABABCABC | A<2i(BA)>BCABC | 8 | {1} | {5}
   0-10 | ABABABCABC | AB<2i(AB)>CABC | 8 | {2} | {6}
   0-10 | ABABABCABC | ABABA<2t(BC):T[-1]>C | 8 | {5} | {9}
   0-10 | ABABABCABC | ABAB<2i(ABC)> | 7 | {4} | {10}
END LEVEL 2
BEGIN LEVEL 3
   0-10 | ABABABCABC | <2i(AB)>A<2t(BC):T[-1]>C | 6 | {0, 5} | {4, 9}
   0-10 | ABABABCABC | <2i(AB)><2i(ABC)> | 5 | {0, 4} | {4, 10}
   0-10 | ABABABCABC | A<2i(BA)><2t(BC):T[-1]>C | 6 | {1, 5} | {5, 9}
END LEVEL 3
BEGIN LEVEL 1
   0-10 | ABABABCABC | <2i(AB)><2i(ABC)> | 5 | {0, 4} | {4, 10}
END LEVEL 1
[...]
```

merge.txt

[fragment of example 7]

```
BEGIN LEVEL 2
   0-7 | ABCXYZa | <2t(ABC):T[23]>a | 4 | {<2t(ABC):T[23]>} | {2} | {t} | {ABC} | {0} | {6}
   1-8 | BCXYZab | <2t(BC):T[22]>Zab | 5 | {<2t(BC):T[22]>} | {2} | {t} | {BC} | {0} | {4}
END LEVEL 2
BEGIN LEVEL 3
   cut off suffix from entry 1
   remove redundant compression from entry 2
   BEGIN LEVEL 4
      0-6 | ABCXYZ | <2t(ABC):T[23]> | 3 | {<2t(ABC):T[23]>} | {2} | {t} | {ABC} | {0} | {6}
      5-8 | Zab | Zab | 3 | {} | {} | {} | {} | {} | {}
   END LEVEL 4
   compression index information of entry 2 has been modified - assume removal
   overlap - smooth
END LEVEL 3
BEGIN LEVEL 1
   0-8 | ABCXYZab | <2t(ABC):T[23]>ab | 5 | {<2t(ABC):T[23]>} | {2} | {t} | {ABC} | {0} | {6}
END LEVEL 1
BEGIN LEVEL 2
   0-8 | ABCXYZab | <2t(ABC):T[23]>ab | 5 | {<2t(ABC):T[23]>} | {2} | {t} | {ABC} | {0} | {6}
   1-8 | BCXYZab | BCX<2t(YZ):T[8]> | 5 | {<2t(YZ):T[8]>} | {2} | {t} | {YZ} | {3} | {7}
END LEVEL 2
BEGIN LEVEL 3
   cut off suffix from entry 1
   cut off prefix from entry 2
   BEGIN LEVEL 4
      0-6 | ABCXYZ | <2t(ABC):T[23]> | 3 | {<2t(ABC):T[23]>} | {2} | {t} | {ABC} | {0} | {6}
```

```
        4-8 | YZab | <2t(YZ):T[8]> | 2 | {<2t(YZ):T[8]>} | {2} | {t} | {YZ} | {0} | {4}
    END LEVEL 4
    overlap - chunk conflict
    cut off overlap from entry 2
    BEGIN LEVEL 4
        0-6 | ABCXYZ | <2t(ABC):T[23]> | 3 | {<2t(ABC):T[23]>} | {2} | {t} | {ABC} | {0} | {6}
        6-8 | ab | ab | 2 | {} | {} | {} | {} | {} | {}
    END LEVEL 4
    overlap - smooth
END LEVEL 3
BEGIN LEVEL 1
    0-8 | ABCXYZab | <2t(ABC):T[23]>ab | 5 | {<2t(ABC):T[23]>} | {2} | {t} | {ABC} | {0} | {6}
END LEVEL 1
BEGIN LEVEL 2
    0-8 | ABCXYZab | <2t(ABC):T[23]>ab | 5 | {<2t(ABC):T[23]>} | {2} | {t} | {ABC} | {0} | {6}
    2-9 | CXYZabc | C<2t(XYZ):T[9]> | 4 | {<2t(XYZ):T[9]>} | {2} | {t} | {XYZ} | {1} | {7}
END LEVEL 2
BEGIN LEVEL 3
    cut off suffix from entry 1
    cut off prefix from entry 2
    BEGIN LEVEL 4
        0-6 | ABCXYZ | <2t(ABC):T[23]> | 3 | {<2t(ABC):T[23]>} | {2} | {t} | {ABC} | {0} | {6}
        3-9 | XYZabc | <2t(XYZ):T[9]> | 3 | {<2t(XYZ):T[9]>} | {2} | {t} | {XYZ} | {0} | {6}
    END LEVEL 4
    overlap - chunk conflict
    chunk conflict solved
    overlap - smooth
END LEVEL 3
BEGIN LEVEL 1
    0-9 | ABCXYZabc | <3t(ABC):T[23,32]> | 3 | {<3t(ABC):T[23,32]>} | {3} | {t} | {ABC} | {0} | {9}
END LEVEL 1
[...]
```

Epilogue

The shifting window mechanism with multiline buffering functions flawlessly. This is also true for the generation of elementary minimisation codes. The combination of these elementary codes until the minimum codes emerge also works as expected, although the selection procedure of the minimum complexity codes could be optimised with respect to, for example, operator-complexity. (This functionality has already been incorporated in the merging process.) The merging process can handle a wide variety of cases though not all of them. Although the unhandled cases are already determined by the algorithm and reported to the user, they have not been explicitly implemented yet. The serious complexity of interacting code-information results in initialising and maintaining a large number of variables[7].

Obviously, incomplete processing should not result in chaos. Therefore, the validation mechanism has been implemented. Just before replacing the current tuple the merged version is verified. If an absurd component is encountered such as a time index that points to the future, then the first code of the current tuple replaces the merged version so that this code will be part of a different tuple the next cycle. This way, rather *robust behaviour* is ensured.

Although relatively easy to incorporate, all *secondary similarities*, i.e. similarities that emerge only after chunk-manipulation, such as mirroring and reversal have been omitted from the minimisation process. This has been done because these similarities are of a primarily rational nature. For a listener to notice these similarities, he or she must first have the knowledge of the existence of these similarities within this context. Also, he must have a certain skill in abstract reasoning. Finally, he must become aware of such similarities during the original analysis. Recognition of similarities that become evident only after the original analysis should fall out of the scope of primary auditory perception.

One possible extension to the AESTHETIC model could be the incorporation of an idle-time processing mechanism for updating and optimising long-term memory information. If the short-term memory processes all information within the available time-frame, then the remainder of the time-frame can be used to optimise the long-term memory. This would of course also imply an implementation of the time-frame mechanism.

[7] For example, see section *Prototype - Internal - Classes - CCodeContainer - TupleOverlapInit().*

This concludes the AESTHETIC project about minimum complexity calculation in primary auditory patterns, although it will be evident that this point has been chosen for other reasons than project completion.

Marco René Spruit,
Department of Computational Linguistics,
Faculty of Arts,
University of Amsterdam,
February 1995.