Apéndice A: Parche para compilar Festival en gcc 4.x por Nickolay V. Shmyrev

```
diff -upr speech_tools.old/grammar/ngram/EST_Ngrammar.cc
speech_tools/grammar/ngram/EST_Ngrammar.cc
--- speech_tools.old/grammar/ngram/EST_Ngrammar.cc  2005-12-11 20:30:31.000000000 +0300
+++ speech_tools/grammar/ngram/EST_Ngrammar.cc      2005-12-12 01:50:59.000000000 +0300
@@ -306,23 +306,22 @@ void EST_BackoffNgrammarState::print_fre
     // not right - just print out, then recurse through children
     // change to use 'backoff_traverse'

-    int k;
+    EST_DiscreteProbDistribution::Entries k;
     double freq;
-    EST_String name;
-    for (k=p_pdf.item_start();
-       !p_pdf.item_end(k);
-       k = p_pdf.item_next(k))
+    for (k.begin(p_pdf);
+       k != 0;
+       k++)
     {
-      p_pdf.item_freq(k,name,freq);
-      EST_BackoffNgrammarState *s = ((EST_BackoffNgrammarState*)(children.lookup(name)));
+      freq = p_pdf.frequency (*k);
+      EST_BackoffNgrammarState *s = ((EST_BackoffNgrammarState*)(children.lookup(*k)));
       if (p_level==order-1)
       {
           if(freq>0)
-              os << name << " " << followers
+              os << *k << " " << followers
                  << ": " << freq << endl;
       }
       else if (s!=NULL)
-          s->print_freqs(os,order,name+" "+followers);
+          s->print_freqs(os,order,*k+" "+followers);

     }
 }
@@ -373,17 +372,13 @@ void EST_BackoffNgrammarState::zap()
 {

     // recursively delete this state and all its children
-    int k;
-    double freq;
+    EST_DiscreteProbDistribution::Entries k;
     EST_String name;
-    for (k=p_pdf.item_start();
-       !p_pdf.item_end(k);
-       k = p_pdf.item_next(k))
+    for (k.begin(p_pdf); k!=0; k++)
     {
-      p_pdf.item_freq(k,name,freq);
-      EST_BackoffNgrammarState *child = get_child(name);
+      EST_BackoffNgrammarState *child = get_child(*k);
       if (child!=NULL)
-          remove_child(child,name);
+          remove_child(child,*k);
     }

     children.clear();
@@ -456,14 +451,12 @@ bool EST_BackoffNgrammarState::set_backo

 void EST_BackoffNgrammarState::frequency_of_frequencies(EST_DVector &ff)
 {
-    int k,max=ff.n();
+    int max=ff.n();
+    EST_DiscreteProbDistribution::Entries k;
     double freq;
-    EST_String name;
-    for (k=p_pdf.item_start();
-       !p_pdf.item_end(k);
-       k = p_pdf.item_next(k))
+    for (k.begin(p_pdf); k!=0; k++)
     {
-      p_pdf.item_freq(k,name,freq);
+      freq = p_pdf.frequency(*k);
       if(freq < max)
           ff[(int)(freq+0.5)] += 1;
     }
```

```
@@ -1585,38 +1578,33 @@ void EST_Ngrammar::prune_backoff_represe
     // remove any branches with zero frequency count

     // find children of this state with zero freq and zap them
-    int k;
+    EST_DiscreteProbDistribution::Entries k;
     double freq;
-    EST_String name;
-    for (k=start_state->pdf_const().item_start();
-        !start_state->pdf_const().item_end(k);
-        k = start_state->pdf_const().item_next(k))
+    for (k.begin(start_state->pdf_const()); k!=0; k++)
     {
-      start_state->pdf_const().item_freq(k,name,freq);
+      freq = start_state->pdf_const().frequency(*k);
       if (freq < TINY_FREQ)
       {
-          EST_BackoffNgrammarState *child = start_state->get_child(name);
+          EST_BackoffNgrammarState *child = start_state->get_child(*k);

           if (child!=NULL)
           {
-              //cerr << "Zapping  " << name << " : " << child->level()
+              //cerr << "Zapping  " << *k << " : " << child->level()
               //<< " " << child<< endl;
-              start_state->remove_child(child,name);
+              start_state->remove_child(child,*k);
           }
       }

     }

     // then recurse through remaining children
-    for (k=start_state->pdf_const().item_start();
-        !start_state->pdf_const().item_end(k);
-        k = start_state->pdf_const().item_next(k))
+    for (k.begin(start_state->pdf_const());k!=0;k++)
     {
-      start_state->pdf_const().item_freq(k,name,freq);
-      EST_BackoffNgrammarState *child = start_state->get_child(name);
+      freq = start_state->pdf_const().frequency(*k);
+      EST_BackoffNgrammarState *child = start_state->get_child(*k);
       if (child!=NULL)
       {
-          //cerr << "recursing to " << name << " : " << child->level() << endl;
+          //cerr << "recursing to " << *k << " : " << child->level() << endl;
           //if((child!=NULL) && (child->level() == 3))
           //cerr << *child  << endl;
           prune_backoff_representation(child);
@@ -2324,20 +2312,18 @@ void EST_Ngrammar::print_freqs(ostream &
       backoff_representation->print_freqs(os,p_order);
     else
     {
-      int i,j,k;
+      int i,j;
+       EST_DiscreteProbDistribution::Entries k;
       EST_IVector window(p_order-1);

       for (i=0; i < p_num_states; i++)
       {
           // print out each ngram : freq
-          for (k=p_states[i].pdf().item_start();
-               !p_states[i].pdf().item_end(k);
-               k = p_states[i].pdf().item_next(k))
+          for (k.begin(p_states[i].pdf());k!=0;k++)
           {
               double freq;
-              EST_String name;
               int ind = i;
-              p_states[i].pdf().item_freq(k,name,freq);
+              freq = p_states[i].pdf().frequency(*k);
               if (freq == 0)
                   freq = floor;
               if (freq > 0)
@@ -2349,7 +2335,7 @@ void EST_Ngrammar::print_freqs(ostream &
                   for (j = 0; j < p_order-1; j++)
                      os << wordlist_index(window(j)) << " ";
-                  os << name << " : " << freq << endl;
+                  os << *k << " : " << freq << endl;
               }
```

```
          }
        }
@@ -2665,15 +2651,12 @@ EST_Ngrammar::backoff_traverse(EST_Backo
     function(start_state,params);

     // and recurse down the tree
-    int k;
+    EST_DiscreteProbDistribution::Entries k;
     double freq;
-    EST_String name;
-    for (k=start_state->pdf_const().item_start();
-        !start_state->pdf_const().item_end(k);
-        k = start_state->pdf_const().item_next(k))
+    for (k.begin(start_state->pdf_const()); k!=0;k++)
     {
-      start_state->pdf_const().item_freq(k,name,freq);
-      EST_BackoffNgrammarState *child = start_state->get_child(name);
+      freq = start_state->pdf_const().frequency(*k);
+      EST_BackoffNgrammarState *child = start_state->get_child(*k);
       if (child!=NULL)
           backoff_traverse(child,function,params);

@@ -2696,16 +2679,14 @@ EST_Ngrammar::backoff_traverse(EST_Backo
     {
       // and recurse down the tree if we haven't
       // reached the level yet
-      int k;
+
+       EST_DiscreteProbDistribution::Entries k;
       double freq;
-      EST_String name;

-      for (k=start_state->pdf_const().item_start();
-           !start_state->pdf_const().item_end(k);
-           k = start_state->pdf_const().item_next(k))
+      for (k.begin(start_state->pdf_const()); k!=0; k++)
       {
-          start_state->pdf_const().item_freq(k,name,freq);
-          EST_BackoffNgrammarState *child = start_state->get_child(name);
+          freq = start_state->pdf_const().frequency(*k);
+          EST_BackoffNgrammarState *child = start_state->get_child(*k);
           if (child!=NULL)
              backoff_traverse(child,function,params,level);

diff -upr speech_tools.old/grammar/ngram/EST_PST.cc speech_tools/grammar/ngram/EST_PST.cc
--- speech_tools.old/grammar/ngram/EST_PST.cc       2005-12-11 20:30:31.000000000 +0300
+++ speech_tools/grammar/ngram/EST_PST.cc    2005-12-12 01:41:16.000000000 +0300
@@ -69,14 +69,12 @@ EST_PredictionSuffixTree_tree_node::prin
     if (p_level == 0)
     {
       // Base -- print from pd
-      EST_String s;
+      EST_DiscreteProbDistribution::Entries k;
       double freq;
-      for (int i = pd.item_start();
-           !pd.item_end(i);
-           i=pd.item_next(i))
+      for (k.begin(pd); k!=0; k++)
       {
-          pd.item_freq(i,s,freq);
-          os << get_path() << " " << s << " : " << freq << endl;
+          freq = pd.frequency(*k);
+          os << get_path() << " " << *k << " : " << freq << endl;
       }
     }
     else
@@ -95,13 +93,11 @@ EST_PredictionSuffixTree_tree_node::prin
     if (p_level == 0)
     {
       // Base -- print from pd
-      EST_String s;
-      double prob;
+       EST_DiscreteProbDistribution::Entries k;
       os << get_path() << " :";
-      for (int i = pd.item_start(); !pd.item_end(i) ; i=pd.item_next(i))
+      for (k.begin(pd);k!=0;k++)
       {
-          pd.item_prob(i,s,prob);
-          os << " " << s << " " << prob;
+          os << " " << *k << " " << pd.probability(*k);
       }
```

```
      os << endl;
    }
diff -upr speech_tools.old/grammar/ngram/freqsmooth.cc speech_tools/grammar/ngram/freqsmooth.cc
--- speech_tools.old/grammar/ngram/freqsmooth.cc    2005-12-11 20:30:31.000000000 +0300
+++ speech_tools/grammar/ngram/freqsmooth.cc 2005-12-12 01:58:10.000000000 +0300
@@ -74,7 +74,7 @@ void fs_build_backoff_ngrams(EST_Ngramma
                              EST_Ngrammar &ngram)
 {
     // Build all the backoff grammars back to uni-grams
-    int i,j,k,l;
+    int i,j,l;

     for (i=0; i < ngram.order()-1; i++)
       backoff_ngrams[i].init(i+1,EST_Ngrammar::dense,
@@ -83,19 +83,17 @@ void fs_build_backoff_ngrams(EST_Ngramma
     for (i=0; i < ngram.num_states(); i++)
     {
       const EST_StrVector words = ngram.make_ngram_from_index(i);
+       EST_DiscreteProbDistribution::Entries k;

-      for (k=ngram.p_states[i].pdf().item_start();
-           !ngram.p_states[i].pdf().item_end(k);
-           k = ngram.p_states[i].pdf().item_next(k))
+      for (k.begin(ngram.p_states[i].pdf()); k != 0; k++)
       {
           double freq;
-          EST_String name;
-          ngram.p_states[i].pdf().item_freq(k,name,freq);
+          freq = ngram.p_states[i].pdf().frequency(*k);
           // Build all the sub-ngrams and accumulate them
           for (j=0; j < ngram.order()-1; j++)
           {
               EST_StrVector nnn(j+1);
-              nnn[j] = name;
+              nnn[j] = *k;
               for (l=0; l < j; l++)
                   nnn[l] = words(ngram.order()-1-j);
               backoff_ngrams[j].accumulate(nnn,freq);
@@ -110,7 +108,7 @@ int fs_backoff_smooth(EST_Ngrammar *back
 {
     // For all ngrams which are too infrequent, adjust their
     // frequencies based on their backoff probabilities
-    int i,j;
+    int i;
     double occurs;
     double backoff_prob;

@@ -129,21 +127,19 @@ int fs_backoff_smooth(EST_Ngrammar *back
               occurs = ngram.p_states[i].pdf().samples();
               EST_StrVector words = ngram.make_ngram_from_index(i);
               words.resize(words.n()+1);
+               EST_DiscreteProbDistribution::Entries j;

-              for (j=pdf.item_start();
-                   ! pdf.item_end(j);
-                   j = pdf.item_next(j))
+              for (j.begin(pdf); j!=0; j++)
               {
-                  EST_String name;
                   double freq;
-                  pdf.item_freq(j,name,freq);
-                  words[words.n()-1] = name;
+                  freq = pdf.frequency(*j);
+                  words[words.n()-1] = *j;
                   backoff_prob =
                       fs_find_backoff_prob(backoff_ngrams,
                                            ngram.order()-1,
                                            words,
                                            smooth_thresh);
-                  pdf.set_frequency(j,occurs*backoff_prob);
+                  pdf.set_frequency(*j,occurs*backoff_prob);
               }
           }
       }
diff -upr speech_tools.old/grammar/ngram/ngrammar_aux.cc
speech_tools/grammar/ngram/ngrammar_aux.cc
--- speech_tools.old/grammar/ngram/ngrammar_aux.cc  2005-12-11 20:30:31.000000000 +0300
+++ speech_tools/grammar/ngram/ngrammar_aux.cc      2005-12-12 01:57:38.000000000 +0300
@@ -117,17 +117,14 @@ smooth_ExponentialFit(EST_DVector &N, in

 void make_f_of_f(EST_BackoffNgrammarState *s,void *params)
```

```
 {
-    int k;
+    EST_DiscreteProbDistribution::Entries k;
     double freq;
-    EST_String name;

     EST_DVector *ff = (EST_DVector*)params;
     int max = ff->n();
-    for (k=s->pdf_const().item_start();
-        !s->pdf_const().item_end(k);
-        k = s->pdf_const().item_next(k))
+    for (k.begin(s->pdf_const());k!=0; k++)
     {
-      s->pdf_const().item_freq(k,name,freq);
+      freq = s->pdf_const().frequency(*k);
       if(freq+0.5 < max)
           (*ff)[(int)(freq+0.5)] += 1;

@@ -138,44 +135,34 @@ void make_f_of_f(EST_BackoffNgrammarStat

 void get_max_f(EST_BackoffNgrammarState *s,void *params)
 {
-    int k;
+    EST_DiscreteProbDistribution::Entries k;
     double freq;
-    EST_String name;

     double *max = (double*)params;
-    for (k=s->pdf_const().item_start();
-        !s->pdf_const().item_end(k);
-        k = s->pdf_const().item_next(k))
+    for (k.begin(s->pdf_const()); k!=0; k++)
       {
-        s->pdf_const().item_freq(k,name,freq);
+        freq = s->pdf_const().frequency(*k);
         if(freq > *max)
           *max = freq;
-
       }
-
-
 }

 void map_f_of_f(EST_BackoffNgrammarState *s,void *params)
 {
-    int k;
+    EST_DiscreteProbDistribution::Entries k;
     double freq;
-    EST_String name;

     //cerr << "map_f_of_f : visited " << *s << endl;

     EST_DVector *map = (EST_DVector*)params;
     int max = map->n();
-    for (k=s->pdf_const().item_start();
-        !s->pdf_const().item_end(k);
-        k = s->pdf_const().item_next(k))
+    for (k.begin(s->pdf_const());k!=0; k++)
       {
-        s->pdf_const().item_freq(k,name,freq);
+        freq = s->pdf_const().frequency(*k);
         if (freq+0.5 < max)
           {
             double nfreq = (*map)((int)(freq+0.5));
-            s->pdf().set_frequency(name,nfreq);
-
+            s->pdf().set_frequency(*k,nfreq);
           }

       }
@@ -184,27 +171,25 @@ void map_f_of_f(EST_BackoffNgrammarState

 void zero_small_f(EST_BackoffNgrammarState *s,void *params)
 {
-    int k;
+    EST_DiscreteProbDistribution::Entries k;
     double freq;
-    EST_String name;

     double *min = (double*)params;
-    for (k=s->pdf_const().item_start();
```

```
-       !s->pdf_const().item_end(k);
-       k = s->pdf_const().item_next(k))
+    for (k.begin(s->pdf_const()); k!=0; k++)
       {
-        s->pdf_const().item_freq(k,name,freq);
+        freq = s->pdf_const().frequency(*k);
         if (freq < *min)
         {
             //cerr << "zeroing " << freq << " < " << *min << endl;
-            s->pdf().override_frequency(k,0.0);
+            s->pdf().override_frequency(*k,0.0);
         }
       }
 }

 void frequency_of_frequencies(EST_DVector &ff, EST_Ngrammar &n,int this_order)
 {
-  int i,k,size;
+  int i,size;
+  EST_DiscreteProbDistribution::Entries k;
   double max=0.0;

   // if ff has zero size, do complete frequency of frequencies
@@ -233,13 +218,10 @@ void frequency_of_frequencies(EST_DVecto
           // Sum the frequencies
           for(i=0;i<size;i++)
             {
-                for (k=n.p_states[i].pdf().item_start();
-                     !n.p_states[i].pdf().item_end(k);
-                     k = n.p_states[i].pdf().item_next(k))
+                for (k.begin(n.p_states[i].pdf());k!=0;k++)
                   {
-                      EST_String name;
                       double freq;
-                      n.p_states[i].pdf().item_freq(k,name,freq);
+                      freq = n.p_states[i].pdf().frequency(*k);
                       ff[(int)(freq+0.5)] += 1;
                   }
             }
@@ -302,8 +284,8 @@ void frequency_of_frequencies(EST_DVecto

 void map_frequencies(EST_Ngrammar &n, const EST_DVector &map, const int this_order)
 {
-  int i,k;
-
+  int i;
+  EST_DiscreteProbDistribution::Entries k;

   switch(n.representation())
     {
@@ -315,17 +297,12 @@ void map_frequencies(EST_Ngrammar &n, co

           for(i=0;i<size;i++)
             {
-                for (k=n.p_states[i].pdf().item_start();
-                     !n.p_states[i].pdf().item_end(k);
-                     k = n.p_states[i].pdf().item_next(k))
+                for (k.begin(n.p_states[i].pdf());k!=0; k++)
                   {
-                      EST_String name;
                       double freq,nfreq;
-                      n.p_states[i].pdf().item_freq(k,name,freq);
+                      freq = n.p_states[i].pdf().frequency(*k);
                       nfreq = map((int)(freq+0.5));
-                      n.p_states[i].pdf().set_frequency(name,nfreq);
-
-
+                      n.p_states[i].pdf().set_frequency(*k,nfreq);
                   }
             }
       }
diff -upr speech_tools.old/grammar/ngram/ngrammar_io.cc speech_tools/grammar/ngram/ngrammar_io.cc
--- speech_tools.old/grammar/ngram/ngrammar_io.cc   2005-12-11 20:30:31.000000000 +0300
+++ speech_tools/grammar/ngram/ngrammar_io.cc       2005-12-12 01:52:04.000000000 +0300
@@ -281,7 +281,8 @@ EST_read_status
 load_ngram_cstr_bin(const EST_String filename, EST_Ngrammar &n)
 {
     EST_TokenStream ts;
-    int i,j,k,order;
+    EST_DiscreteProbDistribution::Entries k;
+    int i,j,order;
```

```
     int num_entries;
     double approx_num_samples = 0.0;
     long freq_data_start, freq_data_end;
@@ -368,14 +369,14 @@ load_ngram_cstr_bin(const EST_String fil
            fclose(ifd);
            return misc_read_error;
        }
-       for (k=n.p_states[i].pdf().item_start();
-            !n.p_states[i].pdf().item_end(k);
-            k = n.p_states[i].pdf().item_next(k))
+       for (k.begin(n.p_states[i].pdf());
+            k != 0;
+            k++)
        {
-           n.p_states[i].pdf().set_frequency(k,dd[j]);
+           n.p_states[i].pdf().set_frequency(*k,dd[j]);
            // Update global info too
            approx_num_samples += dd[j]; // probably not right
-           n.vocab_pdf.cumulate(k,dd[j]);
+           n.vocab_pdf.cumulate(*k,dd[j]);

            // Number of consecutive occurrences of this frequency as in
            // dd[j+1] if its a negative number
@@ -407,7 +408,7 @@ EST_write_status
 save_ngram_htk_ascii_sub(const EST_String &word, ostream *ost,
                          EST_Ngrammar &n, double floor)
 {
-    int k;
+    EST_DiscreteProbDistribution::Entries k;
     EST_String name;
     double freq;
     EST_StrVector this_ngram(2); // assumes bigram
@@ -436,12 +437,10 @@ save_ngram_htk_ascii_sub(const EST_Strin
     }

     // not efficient but who cares ?
-    for (k=this_pdf.item_start();
-        !this_pdf.item_end(k);
-        k = this_pdf.item_next(k))
+    for (k.begin(this_pdf); k!=0; k++)
     {
-       this_pdf.item_freq(k,name,freq);
-       if(name != n.p_sentence_start_marker)
+       freq = this_pdf.frequency(*k);
+       if((*k) != n.p_sentence_start_marker)
        {
            total_freq += freq;
        }
@@ -460,15 +459,13 @@ save_ngram_htk_ascii_sub(const EST_Strin
     {
       lfreq=-1;

-       for (k=this_pdf.item_start();
-            !this_pdf.item_end(k);
-            k = this_pdf.item_next(k))
+       for (k.begin(this_pdf);k!=0; k++)
        {
-           this_pdf.item_freq(k,name,freq);
-
-           if ( (name == n.p_sentence_start_marker) ||
-               (name == n.p_sentence_end_marker) ||
-               (name == OOV_MARKER) )
+           freq = this_pdf.frequency (*k);
+
+           if ( (*k == n.p_sentence_start_marker) ||
+               (*k == n.p_sentence_end_marker) ||
+               (*k == OOV_MARKER) )
               continue;

           if (freq == lfreq)
@@ -734,8 +731,9 @@ save_ngram_cstr_ascii(const EST_String f
     // awb's format
     (void)trace;
     ostream *ost;
-    int i,k;
-
+    int i;
+    EST_DiscreteProbDistribution::Entries k;
+
     if (filename == "-")
```

```
       ost = &cout;
     else
@@ -764,17 +762,14 @@ save_ngram_cstr_ascii(const EST_String f
            const EST_StrVector this_ngram = n.make_ngram_from_index(i);
            this_pdf = n.prob_dist(this_ngram);

-           for (k=this_pdf.item_start();
-                !this_pdf.item_end(k);
-                k = this_pdf.item_next(k))
+           for (k.begin(this_pdf); k!=0; k++)
            {
               double freq;
-              EST_String name;
-              this_pdf.item_freq(k,name,freq);
+              freq = this_pdf.frequency (*k);

               for (int jj=0; jj < this_ngram.n(); jj++)
                   *ost << this_ngram(jj) << " ";
-              *ost << name << " : " << freq << endl;
+              *ost << *k << " : " << freq << endl;
            }
       }
     }
@@ -831,7 +826,8 @@ save_ngram_cstr_bin(const EST_String fil
     if (n.representation() == EST_Ngrammar::sparse)
       return misc_write_error;

-    int i,k;
+    int i;
+    EST_DiscreteProbDistribution::Entries k;
     FILE *ofd;
     double lfreq = -1;
     double count = -1;
@@ -872,13 +868,10 @@ save_ngram_cstr_bin(const EST_String fil
            if ( trace )
               cerr << "\r" << i*100/n.num_states() << "%";

-           for (k=n.p_states[i].pdf().item_start();
-                !n.p_states[i].pdf().item_end(k);
-                k = n.p_states[i].pdf().item_next(k))
+           for (k.begin(n.p_states[i].pdf()); k!= 0; k++)
            {
               double freq;
-              EST_String name;
-              n.p_states[i].pdf().item_freq(k,name,freq);
+              freq = n.p_states[i].pdf().frequency(*k);
               if (freq == 0.0)
                   freq = floor;
               if (freq == lfreq)
@@ -915,14 +908,11 @@ save_ngram_cstr_bin(const EST_String fil
            const EST_StrVector this_ngram = n.make_ngram_from_index(i);
            this_pdf = n.prob_dist(this_ngram);

-           for (k=this_pdf.item_start();
-                !this_pdf.item_end(k);
-                k = this_pdf.item_next(k))
+           for (k.begin(this_pdf);k!=0;k++)
            {

               double freq;
-              EST_String name;
-              this_pdf.item_freq(k,name,freq);
+              freq = this_pdf.frequency(*k);
               if (freq == lfreq)
                   count--;
               else
diff -upr speech_tools.old/grammar/wfst/wfst_train.cc speech_tools/grammar/wfst/wfst_train.cc
--- speech_tools.old/grammar/wfst/wfst_train.cc     2005-12-11 20:30:31.000000000 +0300
+++ speech_tools/grammar/wfst/wfst_train.cc 2005-12-12 01:15:36.000000000 +0300
@@ -314,10 +314,9 @@ static LISP find_best_split(EST_WFST &wf
     LISP splits,s,dd,r;
     LISP *ssplits;
     gc_protect(&splits);
-    EST_String sname;
-    int b,best_b,i;
+    int b,best_b;
     int num_pdfs;
-    double best_score, score, sfreq;
+    double best_score, score;

     for (dd = data[split_state_name]; dd; dd = cdr(dd))
```

```
       pdf_all.cumulate(get_c_int(car(car(dd))));
@@ -354,6 +353,7 @@ static LISP find_best_split(EST_WFST &wf
           break;
       else
       {
+          EST_DiscreteProbDistribution::Entries i;
           // combine a and b
           // Add trans to 0
           setcar(cdr(ssplits[0]),
@@ -362,11 +362,9 @@ static LISP find_best_split(EST_WFST &wf
           setcar(ssplits[0], flocons(best_score));
           // Update 0's pdf with values from best_b's
           b_pdf = pdf(car(cdr(cdr(ssplits[best_b]))));
-          for (i=b_pdf->item_start(); !b_pdf->item_end(i);
-               i = b_pdf->item_next(i))
+          for (i.begin(*b_pdf); i != 0; i++)
           {
-             b_pdf->item_freq(i,sname,sfreq);
-             a_pdf->cumulate(i,sfreq);
+             a_pdf->cumulate(*i,b_pdf->frequency (*i));
           }
           ssplits[best_b] = NIL;
       }
@@ -390,21 +388,17 @@ static double score_pdf_combine(EST_Disc
    // Find score of (a+b) vs (all-(a+b))
    EST_DiscreteProbDistribution ab(a);
    EST_DiscreteProbDistribution all_but_ab(all);
-   int i;
+   EST_DiscreteProbDistribution::Entries i;
    EST_String sname;
-   double sfreq, score;
-   for (i=b.item_start(); !b.item_end(i);
-        i = b.item_next(i))
+   double score;
+   for (i.begin (b); i!= 0; i++)
    {
-     b.item_freq(i,sname,sfreq);
-     ab.cumulate(i,sfreq);
+     ab.cumulate(*i,b.frequency(*i));
    }

-   for (i=ab.item_start(); !ab.item_end(i);
-        i = ab.item_next(i))
+   for (i.begin (ab); i!=0; i++)
    {
-     ab.item_freq(i,sname,sfreq);
-     all_but_ab.cumulate(i,-1*sfreq);
+     all_but_ab.cumulate(*i,-1*ab.frequency(*i));
    }

    score = (ab.entropy() * ab.samples()) +
@@ -522,9 +516,7 @@ static double find_score_if_split(EST_WF
    EST_DiscreteProbDistribution pdf_split(&wfst.in_symbols());
    EST_DiscreteProbDistribution pdf_remain(&wfst.in_symbols());
    int in, tostate, id;
-   int i;
-   double sfreq;
-   EST_String sname;
+   EST_DiscreteProbDistribution::Entries i;

    ent_split = ent_remain = 32*32*32*32;
    LISP dd;
@@ -555,11 +547,9 @@ static double find_score_if_split(EST_WF
    for (dd = data[tostate]; dd; dd = cdr(dd))
      pdf_remain.cumulate(get_c_int(car(car(dd))));
    // Subtract the bit thats split
-   for (i=pdf_split.item_start(); !pdf_split.item_end(i);
-        i = pdf_split.item_next(i))
+   for (i.begin(pdf_split); i != 0; i++)
    {
-     pdf_split.item_freq(i,sname,sfreq);
-     pdf_remain.cumulate(i,-1*sfreq);
+     pdf_remain.cumulate(*i,-1*pdf_split.frequency(*i));
    }
    if (pdf_remain.samples() > 0)
      ent_remain = pdf_remain.entropy();
diff -upr speech_tools.old/include/EST_simplestats.h speech_tools/include/EST_simplestats.h
--- speech_tools.old/include/EST_simplestats.h      2005-12-11 20:30:31.000000000 +0300
+++ speech_tools/include/EST_simplestats.h  2005-12-12 01:27:00.000000000 +0300
@@ -190,22 +190,24 @@ enum EST_tprob_type {tprob_string, tprob
    integers are too restrictive so they are actually represented as
```

```
     doubles.

-    Methods are provided to iterate over the values in a distribution,
-    for example
+    We provide iterator over the values in a distribution, for example
     \begin{verbatim}
-       EST_DiscreteProbistribution pdf;
-       for (int i=pdf.item_start(); i < pdf.item_end(); i=pdf.item_next(i))
+       EST_DiscreteProbDistribution pdf;
+       EST_DiscreteProbDistribution::Entries i;
+
+       for (i.begin(pdf); i != 0; i++)
        {
-          EST_String name;
-          double prob;
-          item_prob(i,name,prob);
-          cout << name << ": prob " << prob << endl;
+          cout << *i << ": prob " << pdf.probability (*i) << endl;
        }
     \end{verbatim}

     @author Alan W Black (awb@cstr.ed.ac.uk): July 1996
 */
 class EST_DiscreteProbDistribution {
+
+    struct IPointer_s { int idx; EST_UItem *item; };
+    typedef struct IPointer_s IPointer;
+
 private:
     double num_samples;         // because frequencies don't have to be integers
     EST_tprob_type type;
@@ -215,6 +217,16 @@ private:
     EST_DVector icounts;
     /* For unknown vocabularies: tprob_string */
     EST_StrD_KVL scounts;
+
+protected:
+
+    void point_to_first(IPointer &i) const;
+    void move_pointer_forwards(IPointer &i) const;
+    bool points_to_something(const IPointer &i) const;
+    const EST_String &points_at(const IPointer &i) const;
+
+    friend class EST_TIterator< EST_DiscreteProbDistribution, IPointer, EST_String>;
+
 public:
     EST_DiscreteProbDistribution() : type(tprob_string), discrete(NULL), icounts(0), scounts()
{init();}
     /// Create with copying from an existing distribution.
@@ -264,18 +276,6 @@ public:
     double frequency(const EST_String &s) const;
     ///
     double frequency(const int i) const;
-    /// Used for iterating through members of the distribution
-    int item_start() const;
-    /// Used for iterating through members of the distribution
-    int item_next(int idx) const;
-    /// Used for iterating through members of the distribution
-    int item_end(int idx) const;
-    /// During iteration returns name given index
-    const EST_String &item_name(int idx) const;
-    /// During iteration returns name and frequency given index
-    void item_freq(int idx,EST_String &s,double &freq) const;
-    /// During iteration returns name and probability given index
-    void item_prob(int idx,EST_String &s,double &prob) const;

     /// Returns discrete vocabulary of distribution
     inline const EST_Discrete *const get_discrete() const { return discrete; };
@@ -298,8 +298,11 @@ public:
         as it will affect how probabilities are calculated.
     */
     void set_num_samples(const double c) { num_samples = c;}
+
+    friend class EST_TStructIterator<EST_DiscreteProbDistribution, IPointer, EST_String>;
+    typedef EST_TStructIterator<EST_DiscreteProbDistribution, IPointer, EST_String> Entries;

-friend ostream & operator <<(ostream &s, const EST_DiscreteProbDistribution &p);
+    friend ostream & operator <<(ostream &s, const EST_DiscreteProbDistribution &p);
     EST_DiscreteProbDistribution &operator=(const EST_DiscreteProbDistribution &a);
 };
```

```
diff -upr speech_tools.old/stats/EST_DProbDist.cc speech_tools/stats/EST_DProbDist.cc
--- speech_tools.old/stats/EST_DProbDist.cc  2005-12-11 20:30:31.000000000 +0300
+++ speech_tools/stats/EST_DProbDist.cc      2005-12-12 01:25:23.000000000 +0300
@@ -305,79 +305,55 @@ double EST_DiscreteProbDistribution::ent
 }

 //  For iterating through members of a probability distribution
-int EST_DiscreteProbDistribution::item_start(void) const
+void EST_DiscreteProbDistribution::point_to_first(IPointer &i) const
 {
     if (type == tprob_discrete)
-       return 0;
+       i.idx = 0 ;
     else
-       return (int)scounts.list.head();
+       i.item = scounts.list.head();
+    return;
 }

-int EST_DiscreteProbDistribution::item_end(int idx) const
+void EST_DiscreteProbDistribution::move_pointer_forwards(IPointer &i) const
 {
     if (type == tprob_discrete)
-       return (idx >= icounts.length());
+       i.idx++;
     else
-       return ((EST_Litem *)idx == 0);
+       i.item = next(i.item);
+       return;
 }

-int EST_DiscreteProbDistribution::item_next(int idx) const
+bool EST_DiscreteProbDistribution::points_to_something(const IPointer &i) const
 {
     if (type == tprob_discrete)
-       return ++idx;
+       return (i.idx < icounts.length());
     else
-       return (int)next((EST_Litem *)idx);
+       return (i.item != NULL);
 }

-const EST_String &EST_DiscreteProbDistribution::item_name(int idx) const
-{
-    if (type == tprob_discrete)
-       return discrete->name(idx);
-    else
-       return scounts.list((EST_Litem *)idx).k;
-}

-void EST_DiscreteProbDistribution::item_freq(int idx,EST_String &s,double &freq) const
+const EST_String &EST_DiscreteProbDistribution::points_at(const IPointer &i) const
 {
     if (type == tprob_discrete)
     {
-       s = discrete->name(idx);
-       freq = icounts(idx);
+       return discrete->name(i.idx);
     }
-    else
-    {
-       s = scounts.list((EST_Litem *)idx).k;
-       freq = scounts.list((EST_Litem *)idx).v;
-    }
-}

-void EST_DiscreteProbDistribution::item_prob(int idx,EST_String &s,double &prob) const
-{
-    if (type == tprob_discrete)
-    {
-       prob = probability(idx);
-       s = discrete->name(idx);
-    }
-    else
-    {
-       s = scounts.list((EST_Litem *)idx).k;
-       prob = (double)scounts.list((EST_Litem *)idx).v/num_samples;
-    }
+    return scounts.list(i.item).k;
 }
```

```
 ostream & operator<<(ostream &s, const EST_DiscreteProbDistribution &pd)
 {
     // Output best with probabilities
-    int i;
+    EST_DiscreteProbDistribution::Entries i;
     double prob;
     double sum=0;
-    EST_String name;

     s << "(";
-    for (i=pd.item_start(); !pd.item_end(i); i=pd.item_next(i))
-    {
-      pd.item_prob(i,name,prob);
-      s << "(" << name << "=" << prob << ") ";
+    for (i.begin (pd); i != 0; i++)
+    {
+      prob = pd.probability (*i);
+      s << "(" << *i << "=" << prob << ") ";
       sum+=prob;
     }
     s << "best=" << pd.most_probable(&prob) << " samples="
diff -upr speech_tools.old/stats/wagon/wagon_aux.cc speech_tools/stats/wagon/wagon_aux.cc
--- speech_tools.old/stats/wagon/wagon_aux.cc       2005-12-11 20:30:31.000000000 +0300
+++ speech_tools/stats/wagon/wagon_aux.cc    2005-12-12 01:15:36.000000000 +0300
@@ -603,15 +603,13 @@ ostream & operator <<(ostream &s, WImpur
     }
     else if (imp.t == wnim_class)
     {
-      int i;
-      EST_String name;
+      EST_DiscreteProbDistribution::Entries i;
       double prob;

       s << "(";
-      for (i=imp.p.item_start(); !imp.p.item_end(i); i=imp.p.item_next(i))
+      for (i.begin (imp.p); i != 0; i++)
       {
-          imp.p.item_prob(i,name,prob);
-          s << "(" << name << " " << prob << ") ";
+          s << "(" << *i << " " << imp.p.probability (*i) << ") ";
       }
       s << imp.p.most_probable(&prob) << ")";
     }
```