Diplomarbeit

Thema: Implementierung eines Klassifizierungssystems

(Programmbeschreibung)

Bearbeiter: Marcus Hutter

Betreuer: Gerhard Weiß

Abgabedatum: 15. Mai 1991

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Ein Klassifizierungssystem (CFS, engl. *Classifiersystem*) ist ein massiv paralleles regelbasiertes System, dessen Komponenten *(Classifier)* Nachrichten *(Messages)* austauschen können, dessen Verhalten von einem Lehrer beurteilt wird *(Reinforcement)* und das mittels *Credit-Assignment* und genetischen Algorithmen fähig ist zu lernen.

Für eine einführende Darstellung muß auf die inzwischen sehr umfangreiche Literatur, insbesondere [4], verwiesen werden.

Das Konzept des CFS wurde zuerst von Holland [1] entwickelt, inzwischen gibt es aber eine Vielzahl von Varianten und Erweiterungen (↑[2, 3]).

Bisher ist es nicht möglich, diese Varianten in ihrer Performance zu vergleichen, eine Aussage über die Güte der verschiedenen Ansätze ist somit kaum oder überhaupt nicht möglich.

Das in dieser Diplomarbeit erstellte Programm gestattet erstmals bzgl. der wichtigsten Varianten einen *direkten* Vergleich. In den folgenden Kapiteln wird dieses Programm, bei dem besonders auf eine effiziente Implementierung geachtet wurde, beschrieben.

2 Grundfunktionen

In diesem Kapitel sollen die Funktionalität und die Implementierung einiger elementarer Routinen beschrieben werden. Obwohl diese eher allgemeiner Natur und nicht unbedingt spezifisch für CFS sind, sollen sie aus folgenden Gründen trotzdem näher erläutert werden:

- Nur eine genaue Kenntnis der Funktionalität der Grundfunktionen kann eine klare Vorstellung des gesamten CFS vermitteln und gezielte Experimente ermöglichen.
- Für jemand, der dieses CFS erweitern oder als Motivation für ein eigenes verwenden will, stellen die Grundfunktionen eine solide Basis dar.

2.1 Selektion (Funktionalität)

Ein zentraler Punkt in CFS ist die Selektion. Darunter ist folgendes zu verstehen:
Aus einer (Multi)Menge M von n bewerteten Elementen, d.h. $M = \{(Id_1, Val_1), \ldots, (Id_n, Val_n)\}$ wird eine Teil(multi)menge $N = \{Id_{i_1}, \ldots, Id_{i_k}\}$ von k Elementen (die Bewertung wird weggelassen) in Abhängigkeit der Bewertungen Val_i ausgewählt, wobei Einzelheiten vom Selektions-Modus abhängen. Generell kann man sagen, daß Elemente mit höheren Bewertungen bevorzugt werden.

Die verschiedenen Selektions-Modi existieren in je zwei Varianten:

1. Elemente können mehrfach selektiert werden, d.h. mehrfach in der Ziel(multi)menge auftreten.
2. Elemente werden höchstens einmal selektiert. Da die Elemente sequentiell selektiert werden (eines nach dem anderen), wird nach Auswahl eines Elements dieses einfach aus M gelöscht.

Kommen wir nun zu den einzelnen Selektionsarten:

2.1.1 Dummy-Selection (`SelMode=1`)

Es werden beliebige (gleiche oder verschiedene) Elemente selektiert.
Diese Variante dient eigentlich internen Zwecken (wenn z.B. $k = N$), kann aber auch für eigene Zwecke verwendet werden, wenn absolut egal ist, welche k Elemente selektiert werden sollen, z.B. Reduktion einer Menge bei keinerlei (nicht einmal statistischer) Zusatzinformation.

2.1.2 Probability-Selection (`SelMode=2 oder 6`)

Dies ist eine statistische Selektion der im wesentlichen die Bewertung der Elemente als Wahrscheinlichkeitsverteilung zugrunde liegt, d.h. Id_i wird mit Wahrscheinlichkeit $Val_i / \sum_{j=1}^{n} Val_j$ selektiert. Dieses Verfahren wird k-mal (mit oder ohne Löschen der bereits selektierten Elemente aus M) wiederholt.

`SelMode=2` und `SelMode=6` unterscheiden sich nur hinsichtlich ihrer Implementierung (↑2.2).

2.1.3 Best-Selection (`SelMode=3`)

Es werden die k (verschiedenen!) Elemente mit der größten Bewertung selektiert.
k-mal das gleiche beste Element zu wählen macht nicht besonders viel Sinn und ergibt deshalb eine Warnung.

2.1.4 Scaled-Selection (`SelMode=4`)

Die skalierte Selektion kann auf zwei verschiedene Weisen interpretiert werden, deren Äuivalenz in 2.2.4 auf Seite 10 bewiesen wird.

1. Betrachte zwei Elemente und selektiere das höher bewertete mit Wahrscheinlichkeit p (> 0.5).
2. Sortiere die Elemente nach ihrer Bewertung, nummeriere sie linear, d.h. mit $(s, s + k, s + 2k, \ldots, s + nk)$ durch (Linearisierung) und selektiere mit Wahrscheinlichkeit proportional zu dieser Nummer (wie in `SelMode=2`).

2.1.5 Random-Selection (`SelMode=5`)

Selektiere gleichverteilt, d.h. unabhängig von der Bewertung wird ein Element mit Wahrscheinlichkeit $1/n$ selektiert.
Diese Methode dient zum Vergleich zwischen rein zufälligen und bewertungsabhängigen Selektionsarten.

2.2 Selektion (Implementierung)

Da Selektion eine zentrale Funktion ist, wurde der effizienten Implementierung höchste Aufmerksamkeit gewidmet. Der Mindestanspruch lag, wie auch bei allen anderen Routinen, keinen Algorithmus mit quadratischer Ordnung, sondern höchstens Ordnung $O(n \log n)$ oder wenn möglich sogar $O(n)$ zu verwenden. Die Funktion `Select` wird mit folgenden Parametern aufgerufen:

`TIdVal A[]:` `A` ist ein Array von Tupeln, deren erste Komponenten Identifier (`index=int`) der Elemente darstellen (z.B. *Message*-Nummer) und deren zweite die Bewertung (`fixp=int`) enthalten.

`int *NP:` `*NP` gibt die Anzahle der Elemente in `A` an und wird bei Selektion ohne Vielfachheit um `k` vermindert.

`index B[]:` `B` nimmt die selektierten Identifier auf.

`int k:` `abs(k)` ist gleich der Anzahl der zu selektierenden Elemente. Möchte man mehrmals (zu versch. Zeitpunkten) Elemente aus `A` selektieren, so muß `A` zuvor mittels `MultSelInit(A,N,Mode)` initialisiert werden und im folgenden `sign(k)=-1` gewählt werden. Bei einmaliger Selektion (`sign(k)=1`) initialisiert `Select` automatisch.

`sbyte Mode:` `abs(Mode)` ist der Selektions-Mode (-6 bis 6),
`sign(Mode)` gibt an, ob mit (-1) oder ohne ($+1$) Vielfachheit selektiert werden soll.

Beispiel:
Skalierte Selektion aus `A` der Länge 10 ohne Vielfachheit.
Zuerst 3 Elemente, später noch einmal 2 Elemente.

```
N=10;
MultSelInit(A,N,4);
Select(A,&N,B,-3,4);      /* nun ist N=7 */
Select(A,&N,B,-2,4);      /* nun ist N=5 */
```

2.2.1 Probability-Selection (`SelMode=2`)

Das Prinzip ist einfach:
Man denke sich einen Streifen aus lauter Stücken zusammengesetzt, deren Längen gerade den Bewertungen entsprechen. Der Gesamtstreifen hat so die Länge $S = \sum_{i=0}^{n-1} Val_i$. Tippt man nun blind (gleichverteilt zufällig) auf den Streifen, so erwischt man ein Stück mit Wahrscheinlichkeit proportional zu seiner Länge.

Bei naiver Implementierung benötigt man im Mittel $n/2$ Additionen zur Selektion eines Elements, bei geschickterer (Zwischenspeichern der Partialsummen $\sum_{i=0}^{k} Val_i \quad \forall\, 0 \leq k < n$ und *Divide & Conquere*) $\log n$ Vergleiche, aber weiterhin $n/2$ um ein Element zu löschen. Dieses auch auf $\log n$ zu drücken macht die Sache etwas verzwickter. Die *Divide & Conquere* Idee wird beibehalten, aber die Partialsummen durch eine Baumstruktur ersetzt.

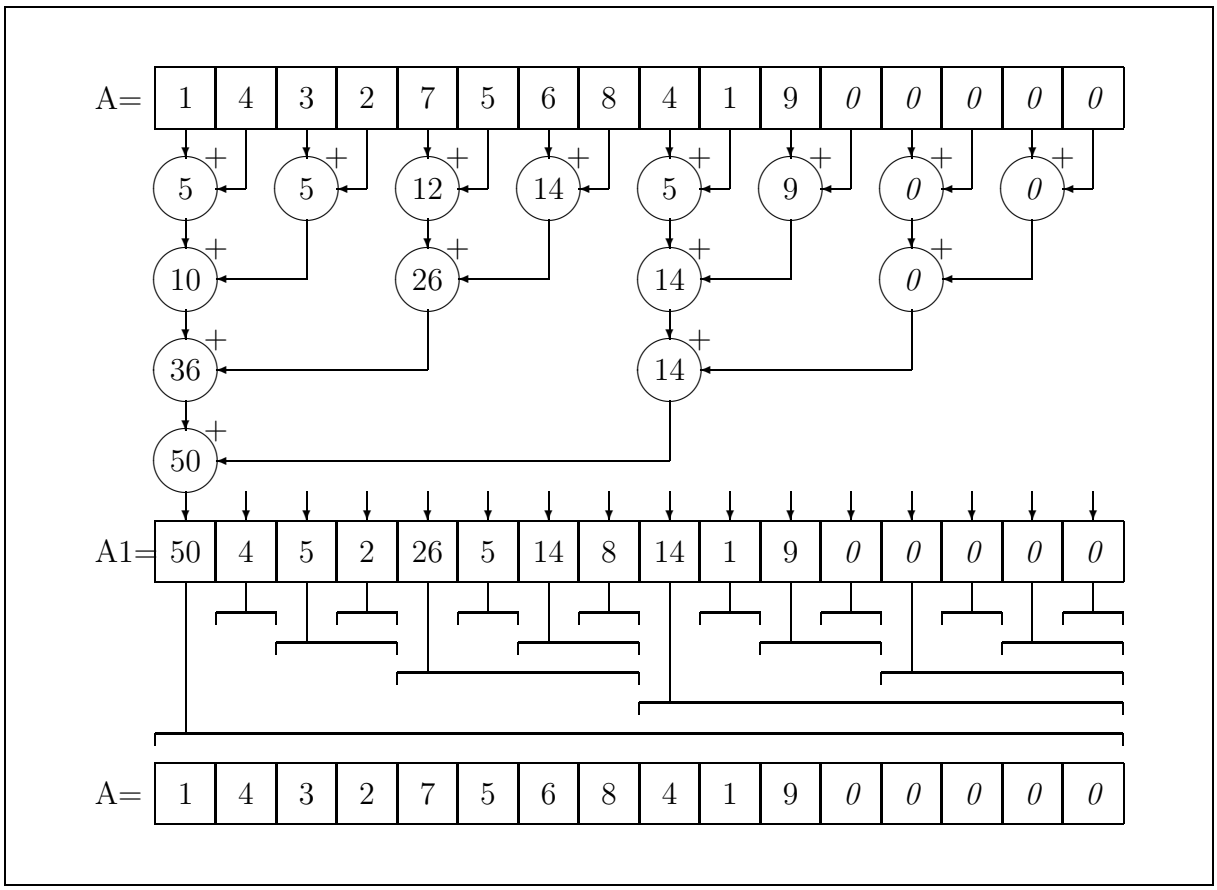

Abbildung 1: Initialisierung von A (`ProbSelInit`)

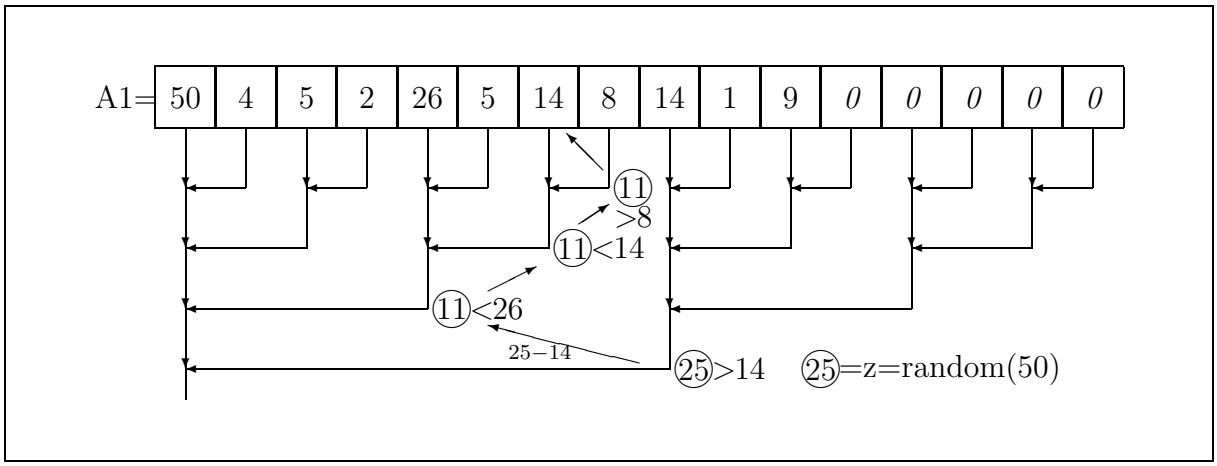

Abbildung 2: Selektion (`ProbSel`)

Die Initialisierung von `A` soll das Beispiel in Abbildung 1 verdeutlichen. Bei Arraylängen, die keine Zweierpotenz sind, kann man sich dieses durch Nullen auf die nächste Zweierpotenz verlängert denken.

Selektiert wird ein Element wie folgt (siehe auch Abbildung 2):

1. Ziehen einer gleichverteilten Zufallszahl `z` zwischen `0` und `S=A1[0]-1`.
2. Man denke sich `z` von rechts abgetragen. Da `A1[n/2]` die Summe der rechten Hälfte von Elementen aus `A` enthält gilt:
   - ist `z` < `A1[n/2]`, so liegt `z` in der rechten Hälfte von `A`. Betrachte im folgenden diese Hälfte als neues Array `A2[0..n/2-1]=A1[n/2..n-1]`.
   - ist `z` ≥ `A1[n/2]`, so liegt `z` in der linken Hälfte von `A`. Betrachte im folgenden diese Hälfe als neues Array `A2[0..n/2-1]=A[0..n/2-1]`. Da sich das rechte Ende geändert hat, muß zur Korrektur `A1[n/2]` von `z` abgezogen werden.
3. Wiederhole Schritt 2 mit `A2`, solange `n` > 1.
4. Die Komponente, bei der man am Ende dieser Iteration angelangt ist wird selektiert.

Um ein Element k in `A` zu ersetzen, bestimmt man die ursprüngliche Bewertung und korrigiert alle in der Hierarchie höher stehenden Elemente, indem man die neue minus die alte Bewertung hinzuaddiert (↑Abbildung 3).

Um ein Element k in `A` zu löschen, ersetzt man dieses durch das Element $n-1$, löscht das Element $n-1$ und verkürzt das Array um 1.

Die Initialisierung benötigt $n-1$ Additionen, die Selektion und das Löschen eines Elements $O(\log n)$ Operationen. Damit ergibt sich die Gesamtkomplexität der Selektion von k Elementen zu $O(n + k \log n)$.

In der Praxis treten zusätzliche Probleme bei der Initialisierung auf; da der integer-Bereich beschränkt ist, läuft man Gefahr diesen bei der Summation zu überschreiten. Deshalb wurde jede Summe (`a+b`) durch eine Mittelwertsbildung (`(a+b)/2`) ersetzt und sämtliche Routinen dieser Korrektur angepaßt.

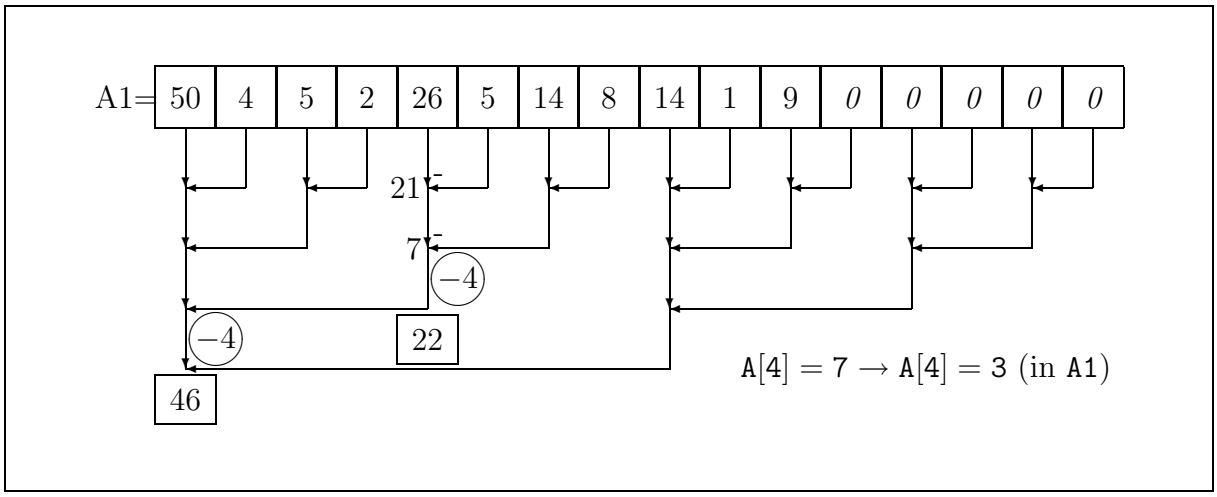

Abbildung 3: Rekonstruktion und Modifikation einer Bewertung

Bei Verminderung der Arraylänge von $2^n + 1$ auf 2^n, nimmt dabei die Hierarchieebene von `A[0]` um eins ab und `A[0]` muß entsprechend korrigiert werden.

Der Test auf eine Zweierpotenz kann im übrigen leicht wie folgt implementiert werden:

`N` ist Zweierpotenz $\Longleftrightarrow$ `N&(N-1)==0` (bitweises `&`)

2.2.2 Probability-Selection (`SelMode=6`)

Funktional ist dieser Selektions-Modus identisch mit vorherigem, der Algorithmus ist aber wesentlich einfacher und unter gewissen Umständen sogar schneller. Das Prinzip ist auch hier wiederum einfach:

Es wird eine gleichverteilte Zufallszahl i zwischen 0 und $n-1$ wie in *Random-Selection* gezogen, allerdings wird das Element i nur mit Wahrscheinlichkeit A_i/K (K konstant) akzeptiert. Wird das Element i nicht akzeptiert,so wird nach diesem Verfahren weitergesucht. Damit $A_i/K \leq 1$ für alle i ist, muß $K \geq \max(A_i)$ gewählt werden. Die mittlere Zahl an Zyklen, bis eine Zufallszahl akzeptiert wird, ist somit

$$C \;=\; \frac{1}{<A_i/K>_i} \;=\; \frac{K \cdot n}{\sum_{i=0}^{n-1} A_i} \quad .$$

Man wählt also K zweckmäßigerweise möglichst klein, also $K = \max(A_i)$. C hängt somit wesentlich vom Verhältnis des Flächeninhalts des der Funktion umbeschriebenen Rechtecks und der Funktion selbst ab (↑Abbildung 4).

Bei einigermaßen gleichverteilten Zufallsgrößen ist C klein und damit die Zeitkomplexität eine kleine Konstante, also der Implementierung für `SelMode=2` mit Komplexität $O(\log n)$ vorzuziehen.

2.2.3 Best-Selection (`SelMode=3`)

Bei iterierter Selektion wird A mittels Quicksort (`QSort`) zuerst aufsteigend sortiert und dann bei jedem Zugriff die obersten Elemente geliefert und gelöscht (Die Länge von A wird reduziert). Mittlere Gesamtkomplexität: $O(n \log n)$

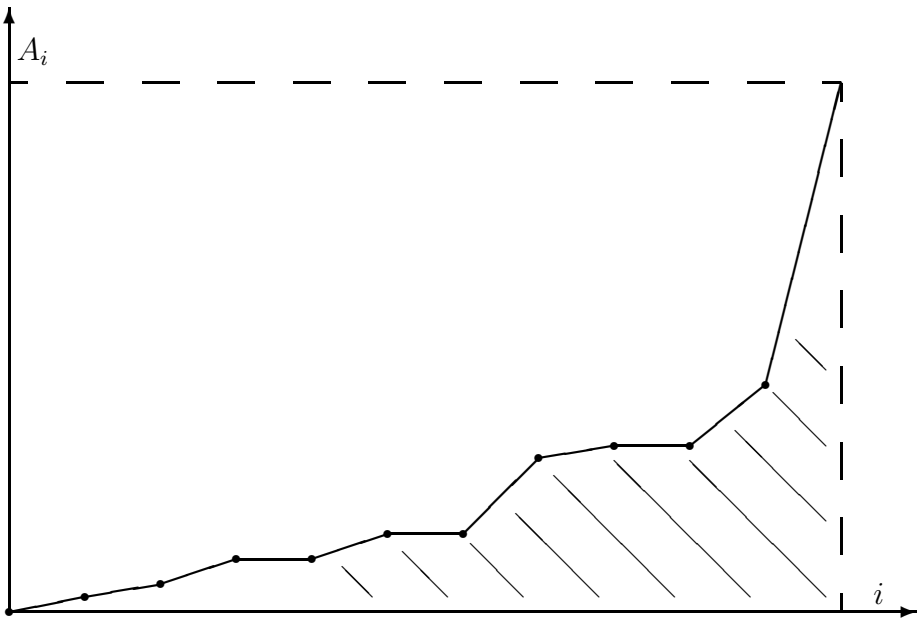

Abbildung 4: Verteilung + einhüllendes Rechteck

Bei einmaliger Selektion der k größten Elemente aus A kann auf das Sortieren verzichtet werden und durch einen quicksort-ähnlichen Algorithmus (`QMax`) mit nur noch linearer Rekursion ersetzt werden. Wählt man wie in Quicksort ein Trennelement und sortiert die kleineren Elemente an den Anfang des Arrays und die größeren an das Ende, so kann man wie folgt weiter argumentieren:

- Liegen weniger als k Elemente in der oberen Hälfte, so werden *diese* garantiert und noch *einige* der unteren Hälfte selektiert. Man muß also nur noch die untere Hälfte betrachten und `QMax` (rekurviv) mit der restlichen Zahl zu selektierender Elemente aufrufen.
- Liegen mehr als k Elemente in der oberen Hälfte, so werden *einige* dieser und garantiert *keine* der unteren Hälfte selektiert. Man betrachtet hier also nur noch die obere Hälfte und ruft `QMax` (rekursiv) mit der oberen Hälfte und k auf.

Dieser Prozess bewirkt ein sukzessives Verschieben der Grenze, bis diese "paßt".

Mittlere Gesamtkomplexität: $O(n)$

2.2.4 Scaled-Selection (`SelMode=4`)

An dieser Stelle sollen die Äquivalenz der beiden in 2.1.4 auf Seite 5 angedeuteten Interpretationen bewiesen, einige weitere mathematische Untersuchungen angestellt und die Vorteile dieser Selektionsart behandelt werden.

Die Implementierung der 1. Interpretation ist trivial:

Ziehe zwei gleichverteilte Zufallszahlen i und j zwischen 0 und $n-1$ und wähle aus $\{A_i, A_j\}$ das größere Element mit Wahrscheinlichkeit p (entsprechend das kleinere mit Wahrscheinlichkeit $q = 1 - p$) (*).

Mathematische Analyse
Seien $A_1, \ldots, A_n$ in aufsteigender Reihenfolge sortiert, d.h. $A_1 \leq \ldots \leq A_n$. [1]

Seien X_1 und X_2 die zwei gleichverteilten Zufallsgrößen, Y die sich aus obigem Prozeß (*) für das selektierte Element ergebende Zufallsgröße, d.h.

$$P(X_i = k) \;=\; 1/n \quad \text{für} \quad k \in \{1, \ldots, n\}$$

(*) mathematisch übersetzt:

$$\begin{aligned} P(Y = k) \quad &= \quad p \cdot P(\max(X_1, X_2) = k) + q \cdot P(\min(X_1, X_2) = k) \\ &\qquad \text{da } X_1 < X_2 \Longleftrightarrow A_{X_1} < A_{X_2} \end{aligned}$$

Als Verteilung geschrieben:

$$\begin{aligned} P(Y \leq k) \;&=\; p \cdot P(X_1 \leq k \wedge X_2 \leq k) + q \cdot P(X_1 \leq k \vee X_2 \leq k) \quad = \\ &=\; p \cdot \underbrace{P(X_1 \leq k) \cdot P(X_2 \leq k)}_{\text{unabhängig}} + q \cdot \Big[1 - \underbrace{P(X_1 > k) \cdot P(X_2 > k)}_{\text{unabhängig}}\Big] \quad = \\ &=\; p \cdot \frac{k}{n} \cdot \frac{k}{n} + q \cdot \Big[1 - (1 - \frac{k}{n}) \cdot (1 - \frac{k}{n})\Big] \quad = \\ &=\; q\frac{2k}{n} + (p - q)\Big(\frac{k}{n}\Big)^2 \end{aligned}$$

und wieder als Wahrscheinlichkeit ausgedrückt:

$$\begin{aligned} P(Y = k) \;&=\; P(Y \leq k) - P(Y \leq k - 1) \quad = \\ &=\; \Big[q\frac{2k}{n} + (p - q)\Big(\frac{k}{n}\Big)^2\Big] - \Big[q\frac{2(k-1)}{n} + (p - q)\Big(\frac{k-1}{n}\Big)^2\Big] \quad = \\ &=\; q\frac{2}{n} + (p - q)\frac{2k - 1}{n^2} \quad = \quad \frac{2(p - q)}{n^2} \cdot k + \Big(\frac{2q}{n} - \frac{p - q}{n^2}\Big) \end{aligned}$$

$$\Longrightarrow \qquad \boxed{P(Y = k) \;=\; ak + b \quad \text{mit} \quad a = \frac{2(p - q)}{n^2} \quad , \quad b = \frac{2q}{n} - \frac{p - q}{n^2}}$$

Y ist also eine linear verteilte Zufallsgröße, wie in 2.1.4 behauptet wird.

$(A_1, \ldots, A_n)$ zu sortieren und mit Wahrscheinlichkeit proportional zu $(a+b, a+2b, \ldots, a+nb)$ zu selektieren, ist also zu obiger Methode äquivalent.

Frei ist noch der Parameter p, der z.B. so gewählt werden kann, daß die Selektions-Wahrscheinlichkeit des $\frac{\text{größten}}{\text{kleinsten}}$ Elements gleich C ist, d.h.

$$\begin{aligned} C \quad &:= \quad \frac{P(Y = n)}{P(Y = 1)} \quad = \quad \frac{an + b}{a + b} \quad = \quad \frac{2(p - q)n + 2qn - (p - q)}{2(p - q) + 2qn - (p - q)} \quad = \\ &= \quad \frac{2n(2p - 1) + 2n(1 - p) - (2p - 1)}{(2p - 1) + 2n(1 - p)} \quad = \end{aligned}$$

[1] Man mache sich klar, daß die Nummerierung keinen Einfluß auf die Selektion hat (da gleichverteilt) und somit im Programm beliebig (also einfach die Array-Indizes) gewählt werden kann und nicht wie man vermuten könnte, das Array zuerst sortiert werden muß.

$$
\begin{aligned}
= \quad & \frac{2np-(2p-1)}{2n(1-p)+(2p-1)} \quad \stackrel{n\to\infty}{\longrightarrow} \quad \frac{p}{1-p} \\
\iff \quad & [2n(1-p)+(2p-1)]\cdot C \quad = \quad 2np-(2p-1) \\
\iff \quad & [2(1-n)p+(2n-1)]\cdot C \quad = \quad 2(n-1)p+1 \\
\iff \quad & 2(1-n)p(C+1)+(2n-1)C \quad = \quad 1 \\
\iff \quad & p \quad = \quad \frac{(2n-1)C-1}{2(n-1)(C+1)} \quad \stackrel{n\to\infty}{\longrightarrow} \quad \frac{C}{C+1}
\end{aligned}
$$

Beispiel 1:
$C = 1$: Kein Element wird bevorzugt $\Longrightarrow$
$p = \frac{(2n-1)-1}{2(n-1)(1+1)} = \frac{1}{2}$: Rein gleichverteilter Zufall (klar) $\Longrightarrow$
$P(Y = k) = 1/n$: Jedes Element mit Wahrscheinlichkeit $1/n$

Beispiel 2:
$C = 2$: Wahrscheinlichkeit des $\frac{\text{größten}}{\text{kleinsten}} = 2 \quad \Longrightarrow \quad p = \frac{4n-3}{6n-6} \stackrel{n\to\infty}{\longrightarrow} \frac{2}{3}$

Beispiel 3:
$p = 1 \quad \Longrightarrow \quad P(Y = k) = \frac{1}{n^2}(2k-1) \quad , \quad C = 2n-1$: Sehr starke Selektion

Beispiel 4:
$p = 2/3 \quad \Longrightarrow \quad P(Y = k) = \frac{1}{3n^2}(2k+2n-1) \quad , \quad C = \frac{4n-1}{2n+1} \stackrel{n\to\infty}{\longrightarrow} 2$: (↑Beispiel 2)

Vorteile:

- **Einfache Implementierung:**
  Linearisierung, ohne sortieren zu müssen (nicht zu verwechseln mit linerarer Skalierung, die häufig verwendet wird).
- **Gute Skalierung:**
  Linearisierung auch unangenehmer Funktionen (↑Abbildung 4).
- **Biologische Plausibilität:**
  Zwei Individuen (Elemente) kämpfen ums Überleben (selektiert zu werden), und dasjenige mit höherer Fitneß (Bewertung) siegt mit größerer Wahrscheinlichkeit ($p > q$!).

2.2.5 Random-Selection (`SelMode=5`)

Selektiere gleichverteilt, d.h. unabhängig von der Bewertung wird ein Element mit Wahrscheinlichkeit $1/n$ selektiert.

2.3 Tupelalgorithmen

Die Routine *Bid*, die aus den *Matching-Tupels* die *Bid-Tupels* erzeugt, muß u.a. folgendes berechnen:
Gegeben ist eine Menge M von *Messages*, wobei jede *Message* eine Intensität besitzt, die durch $f : M \to \Re$ gegeben ist. Teilmengen $M_i \subseteq M$ *matchen* den Bedingungsteil i eines festen *Classifiers*. Nun werden sämtliche *Matching-Tupels* $T = M_1 \times \cdots \times M_r$ gebildet und hiervon die Gesamtintensität gleich dem *Support*

$$S \quad := \sum_{(m_1,\ldots,m_r)\in T} f(m_1) + \cdots + f(m_r)$$

berechnet. Durch eine kleine Transformation können diese $|T| = |M_1| \cdot \ldots \cdot |M_r|$ Additionen auf $|M_1| + \ldots + |M_r|$ Additionen reduziert werden.

$$\text{Betrachte:} \quad \sum_{(m_1,\ldots,m_r)\in T} f(m_i) \;=\; \sum_{\substack{(m_1,\ldots,m_{i-1},m_{i+1},\ldots,m_r)\\ \in M_1\times\cdots\times M_{i-1}\times M_{i+1}\times\cdots\times M_r}} \;\sum_{m_i\in M_i} f(m_i) \;=$$

$$= \; |M_1 \times \cdots \times M_{i-1} \times M_{i+1} \times \cdots \times M_r| \cdot \sum_{m_i\in M_i} f(m_i) \;=$$

$$= \; \frac{|T|}{|M_i|} \cdot \sum_{m_i\in M_i} f(m_i) \;=\; |T| \,<f>_{M_i}$$

$$\Longrightarrow \quad S \;= \sum_{(m_1,\ldots,m_r)\in T} f(m_1) + \cdots + f(m_r) \;=\; |T| \,\{<f>_{M_1} + \cdots + <f>_{M_r}\}$$

Für zwei Bedingungen ergibt sich:

$$\boxed{\sum_{m_1\in M_1, m_2\in M_2} f(m_1) + f(m_2) \quad = \quad |M_2| \sum_{m_1\in M_1} f(m_1) \;+\; |M_1| \sum_{m_2\in M_2} f(m_2)}$$

Diese Äquivalenz wird in der Routine *Bid* verwendet.

2.4 Zufallszahlen-Generatoren

2.4.1 Gleichverteilte Zufallszahlen

Erstaunlicherweise sind selbst im hochentwickelten Compiler TC 2.0 von Borland die angeblich gleichverteilten Zufallszahlen katastrophal ungleichverteilt. Aus diesem Grund wurde ein Einzeiler `random2()` geschrieben, der `random()` ersetzt (und funktioniert !).

2.4.2 `flip`

`flip(p)` liefert mit Wahrscheinlichkeit `p` das Ergebnis `TRUE`, wobei zu beachten ist, daß `p` eine Fixpunktzahl ist (↑3.1.2).

2.4.3 Normalverteilung

Sehr schnell ist folgende Approximation der Normalverteilung, da:

- keine Fließkommazahlen verwendet werden,
- nur *eine* 16-Bit Zufallszahl gezogen wird,
- nur elementare (maschinennahe) Rechenoperationen ausgeführt werden.

Es werden die 1-Bits (Bits, deren Wert gleich 1 ist) der 16-Bit Zufallszahl gezählt, durch 2 geteilt und 4 abgezogen. Da es 0 bis 16 1-Bits gibt ist die Zufallsgröße durch ± 4 beschränkt.

Mathematische Analyse
Jedes Bit i ist eine $\{0, 1\}$ gleichverteilte Zufallsgröße B_i, d.h. 0 mit Wahrscheinlichkeit 1/2 und 1 mit Wahrscheinlichkeit 1/2.

$$\Longrightarrow \quad E(B_i) = 1/2 \quad , \quad Var(B_i) = 1/4$$

$$\text{Sei} \qquad N = \frac{1}{2} \sum_{i=1}^{16} B_i - 4$$

$$\Longrightarrow \quad E(N) = \frac{1}{2} \cdot 16 \cdot \frac{1}{2} - 4 = 0 \quad , \quad Var(N) = \left(\frac{1}{2}\right)^2 \cdot 16 \cdot \frac{1}{4} = 1$$

Nach dem zentralen Grenzwert-Satz geht eine n-fach-Summe einer beliebigen Verteilung für $n \to \infty$ gegen eine Normalverteilung.

$\Longrightarrow N$ ist ungefähr Standard-Normalverteilung ($n = 16 \approx \infty$), wobei N bei den Werten ± 4 abgeschnitten ist.

3 Datenstrukturen

3.1 Konventionen

3.1.1 Anfangsbuchstaben

Um die Variablennamen einerseits nicht zu lang, das Progamm aber andererseits nicht durch kryptische Namensgebung unleserlich zu machen, wurden einige ein- oder mehrbuchstabige Namenskonventionen so weit wie möglich das ganze Programm durchgehalten.

Mittels Tabelle 1 und 2 dürften die meisten Namensgebungen verständlich werden.

3.1.2 Konstanten und primitive Datentypen

Einige zweckmäßige Definitionen von Konstanten und primitiven Datentypen, wie `bool`, `byte`, `index` u.a. wurden eingeführt und können dem Programmlisting entnommen werden. Im folgenden soll nur der Datentyp `fixp`,der `float` ersetzt, näher erläutert werden.

S = Size of	Konstante für den Compiler (z.B. physikalische Arraylänge)
M = Maximal	Maximaler Wert (z.B. größter jemals benutzter Array-Index)
N = Number of	Anzahl, Zahl (z.B. größter aktuell benutzter Array-Index)
T = Type of	Typen werden mit T begonnen
pr = print	Routinen, die etwas im Textmode anzeigen
in = input	Routinen, die Eingaben von der Tastatur erwarten
N = not/non	Verneinung der Bedeutung der folgenden Buchstaben
D = Default	Defaultwerte (Konstanten)
X =	Variablen, auf die nur indirekt über Zeiger zugegriffen wird
P = Pointer	häufig auch als Endbuchstabe bei Ergebnisparametern (z.B. NP = Pointer auf N)

Tabelle 1: Bedeutung der Anfangsbuchstaben von Identifikatoren

CFS	Classifier-System
M	matching
M	Message
MM	matching Message
C	Classifier
T	Tupel
B	Bid
H	Help...

Tabelle 2: Abkürzungen in Identifikatoren

FixPunkt-Zahlen

`fixp` ist eine 2-Byte `integer` Zahl, wobei ein Byte als Vorkommateil, der andere als Nachkommateil interpretiert wird. Eine FixPunkt-Zahl `fp` ist also als reelle Zahl `fp/256` zu interpretieren; umgekehrt kann eine reele Zahl `r` mittels `(int)(r*256)` in FixPunkt-Format verwandelt werden, zu dem das Makro `FixP` zur Verfügung steht. Aus diesem Grund werden im `EXAMPLE1.C`-File auch sämtliche reelle Konstanten mittels

```
fixp Name = FixP( xx.xx )
```

definiert.

FixPunkt-Zahlen können wie `int` mittels + und − addiert und subtrahiert werden. Zur Multiplikation, Division und Potenzierung stehen `FPMult`, `FPDiv` und `FPPow` zur Verfügung, die entweder mit Fehlerabfragen (`SpeedF=FALSE`) implementiert sind oder als schnelle Makros (`SpeedF=TRUE`) zur Verfügung stehen.

Strings, Messages, Conditions und Actions

Eine Message ist eine Folge von 0en und 1en, deren Länge mittels `SStr` (Size of String) festgelegt ist. Die *Condition*- und *Action*-Teile von *Classifiern* sind Folgen von 0en, 1en und #en, deren Längen ebenfalls durch `SStr` festgelegt sind.

Zur Codierung von *Messages*, *Conditions* und *Actions* wird als gemeinsamer Datentyp `TString` verwendet, der je nach `SStr` 1 bis 4 Bytes im Speicher belegt:

SStr	TString	Bytes
1–8	byte	1
9–16	int	2
17–32	long	4

Tabelle 3: Der Datentyp `TString`

Eine 0/1-Stelle einer Message entspricht einfach einem Bit in `TString`. Unbenutzte Bits sind im Normalfall 0.

Zur Codierung von *Conditions* werden zwei Variable (C und B) vom Typ `TString` verwendet und eine 0/1/#-Stelle in einem korrespondierenden Paar von Bits in C und B gespeichert, wobei C als *Condition* und B als Maske bezeichnet wird.

Bei einer spezifischen Stelle (0/1) wird in C (wie bei Messages) ein Bit auf 0/1 gesetzt, das korrespondierende in C auf 1. Bei einer unspezifischen Stelle (#) werden in C und B die korrespondierenden Bits beide auf 0 gesetzt.

Actions werden analog behandelt.

Diese Art der Codierung erlaubt ein effizientes *Message-Condition-Matching* und benötigt, nebenbei bemerkt, nur wenig Speicher.

Condition		Bit 15 0
01##1101#1	$\Longleftrightarrow$	C=0000000100110101
		B=0000001100111101

Tabelle 4: Beispiel einer *Condition*-Codierung

Der Datentyp `TIdVal`

Ein zentraler Datentyp zur Selektion ist `TIdVal` oder besser `TIdVal[]`, ein Array von `TIdVal`.

```
typedef struct { index Id; fixp  Val; } TIdVal;
```

`TIdVal` ist eine Struktur, die aus 2 integer-Werten besteht, die im Normalfall folgende Bedeutung haben.

`Id` : Ein Array-Index, der als `Id` dieses Elements gilt.
`Val`: Eine FixPunkt-Bewertung.

Da `TIdVal` 4 Byte im Speicher belegt, kann `TIdVal` auch als `long` aufgefaßt werden, was häufig vorteilhaft (vor allem beim Kopieren) im Programm Verwendung findet.

3.2 Datenstruktur des Classifier-Systems

An dieser Stelle soll die gesamte Datenstruktur incl. deren Zugriff zu Referenzzwecken graphisch dargestellt werden, Dokumentation der einzelnen Strukturkomponenten findet man im Programmlisting.

- Eine mit `x` beschriftete Linie bedeutet, daß der untere Datentyp eine Komponente des oberen ist, auf die mittels `.x` zugegriffen werden kann.
- Eine mit `x` beschrifteter Pfeil bedeutet, daß der obere Datentyp eine Pointerkomponente `->x` besitzt, die auf den unteren zeigt.
- Eine unbeschriftete Kante zwischen zwei Komponenten deutet ihre Äquivalenz an.

Mehrfachbeschriftungen sind als mehrfaches Auftreten von Komponenten gleichen Typs zu interpretieren.

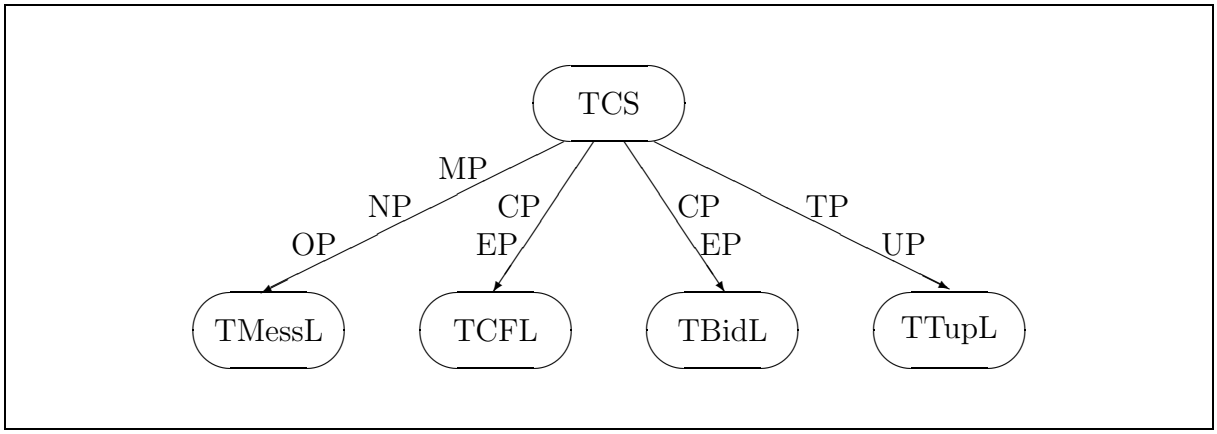

Abbildung 5: Globale *Classifier*-Datenstruktur TCS

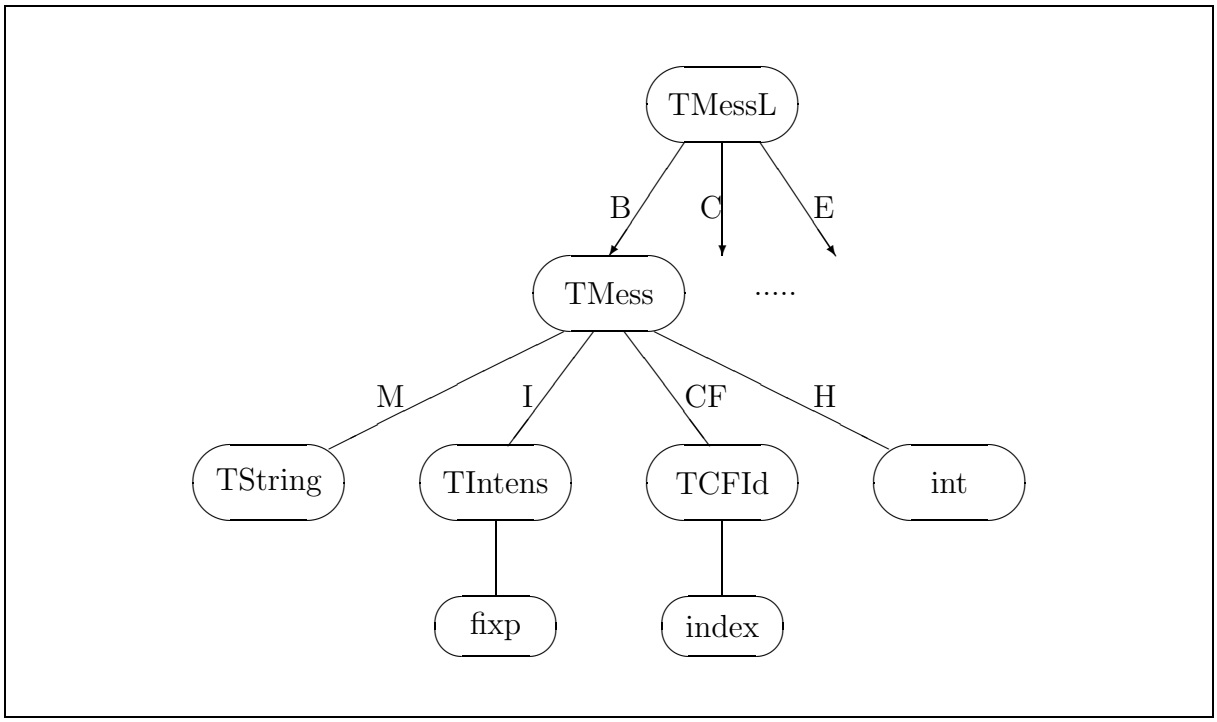

Abbildung 6: *MessageList*-Datenstruktur `TMessL`

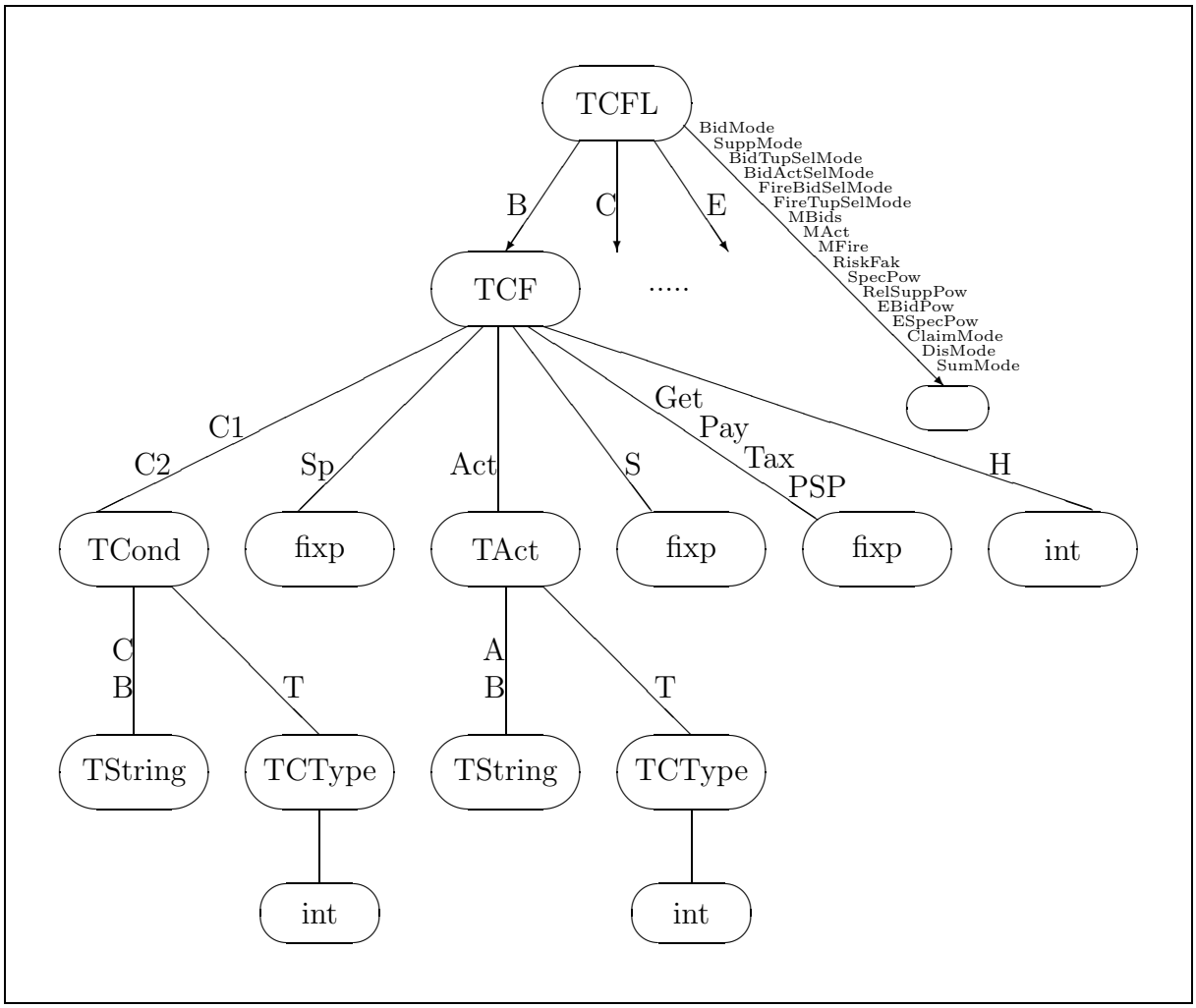

Abbildung 7: `ClassifierList- & EffectorList`-Datenstruktur `TCFL`

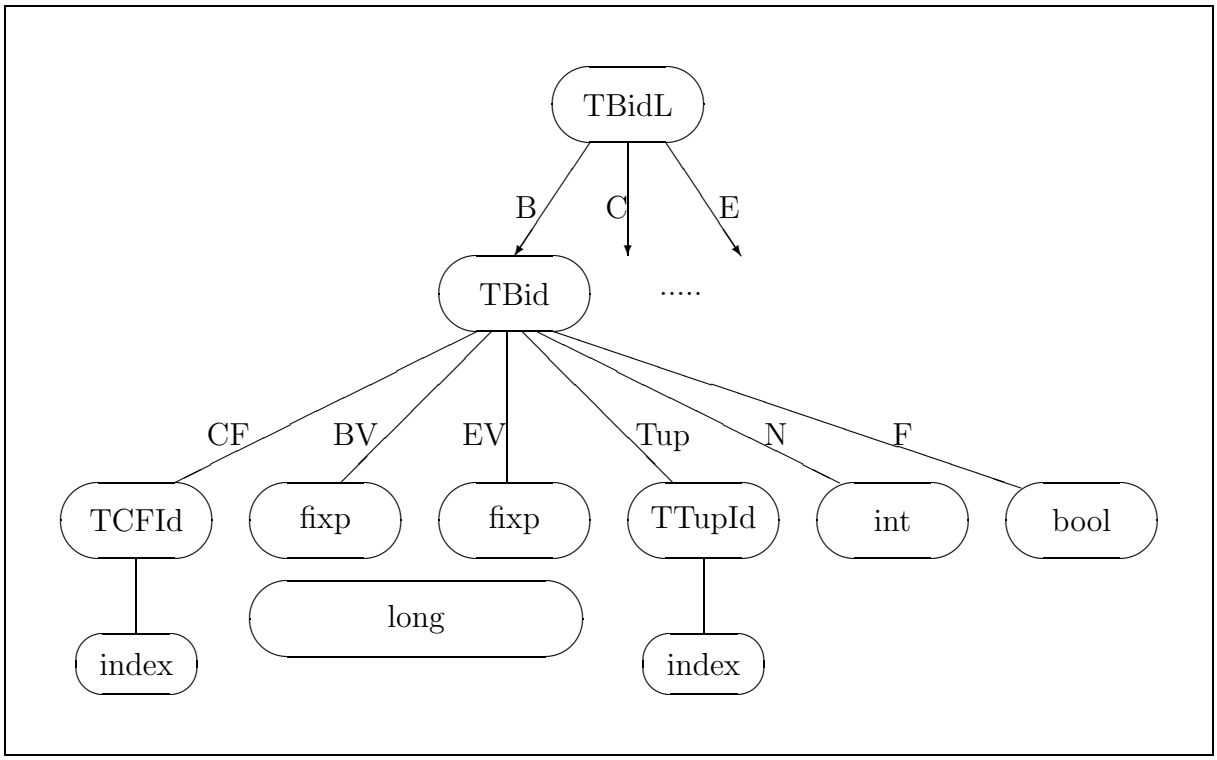

Abbildung 8: *BidList*-Datenstruktur `TBidL`

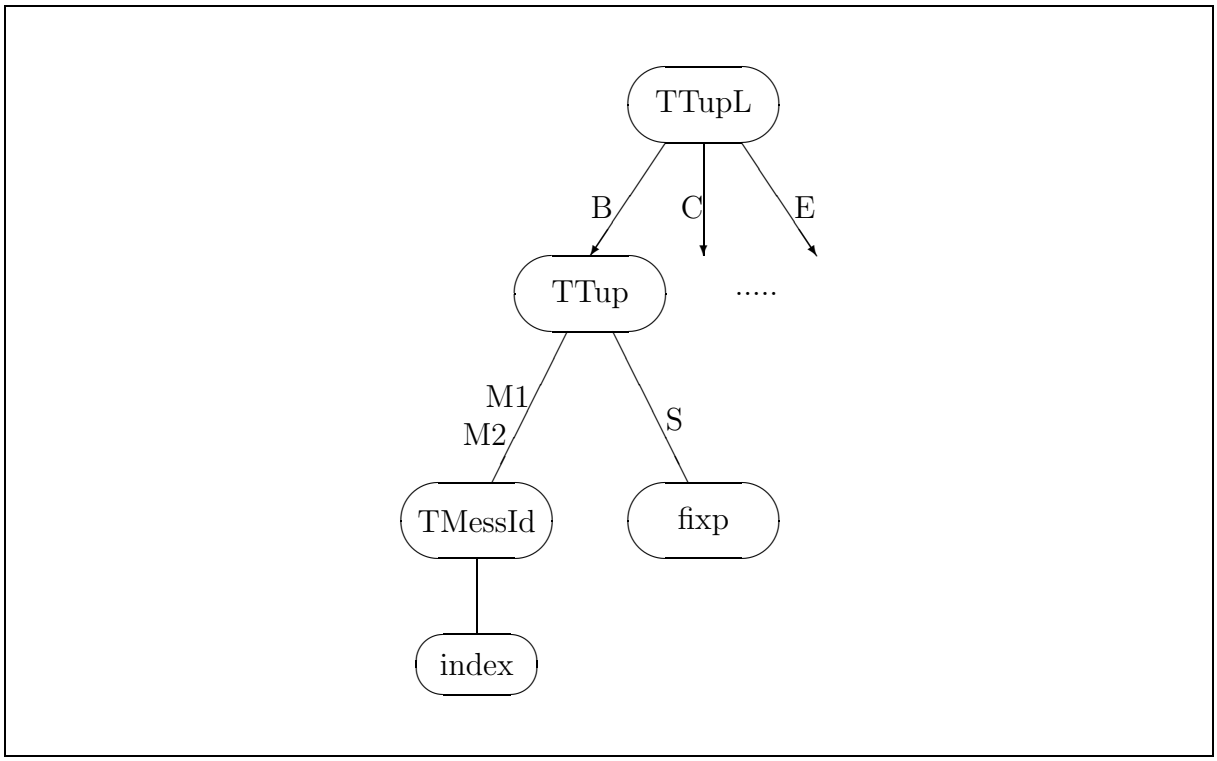

Abbildung 9: *TupelList*-Datenstruktur `TTupL`

4 Benutzer-Oberfläche

4.1 Start des Klassifizierungs-Systems

Es ist vorteilhaft (aber nicht zwingend notwendig) das CFS unter einem C-Compiler mit integrierter Entwicklungsumgebung (IDE) oder von einem C-Debugger aus zu starten, da jene Umgebungen auch als Arbeitsumgebung des CFS verwendbar sind. Hier soll die Benutzung des CFS bei Verwendung des Turbo-C 2.0 oder Turbo-C++ Compilers beschrieben werden.

Benötigt werden zwei C-Files:

`CFSSIM.C` : Das Klassifizierungs-System

`EXAMPLEn.C`: Die beispielspezifische Daten, wie Benutzer-Parameter & Benutzer-Funktionen

`EXAMPLE1.C` enthält als Beispiel den 4:2-Decoder, der in [4] und [5] beschrieben ist, auf das ich mich im folgenden beziehe.

Da die Beispiele einige Parameter (die mit `#define` definierten) enthalten, die den Objekt-Code von `CFSSIM.OBJ` bestimmen, kann `CFSSIM.C` nicht *einmal* vorweg compiliert werden. Daher wurde folgender Weg gewählt.

`EXAMPLE`*n*`.C` ist *ihr* Bezugsfile (Main-File) den Sie mit dem Turbo-C-Editor ggf. modifizieren und starten. Die `#include`-Zeile bewirkt dann, daß `CFSSIM.C` mit den aktuellen `#define`-Paramtern compiliert wird.

Bevor Sie das Programm starten sollten Sie unbedingt in die Zeile, in der `BreakPoint()` steht, mit `Ctrl-F8` einen Breakpoint setzen. Damit können Sie wie in 4.2.4 beschrieben wird jederzeit das CFS kurz unterbrechen und in die IDE gelangen, Parameter modifizieren und das CFS an der Unterbrechungsstelle fortsetzen.

Die Schritte sind im einzelnen:

Eingabe	**Bedeutung**
`TC o.ä.`	Aufruf von Turbo-C
`Alt-F L EXAMPLE1.C`	Laden des Beispiels
??	Eventuell modifizieren
`Ctrl-F8`	Breakpoint in Zeile `BreakPoint()` setzen
`Alt-R R`	Start des CFS
??	Verschiedene CFS-Menu-Eingaben 4.2
`B`	Rückkehr in die IDE von Turbo-C
`Ctrl-F4` *Name=Wert*	Modifikation von Paramtern
`Alt-R R`	Fortsetzen der CFS-Simulation
...	

4.2 Menusteuerung

Nachdem Sie das CFS mittles `Alt-R R` gestartet haben, erscheint in der 1. Zeile eine Menu-Leiste. Die Großbuchstaben geben die den Menupunkt aktivierenden Tasten an, deren Bedeutung im einzelnen in 4.2.1 bis 4.2.5 erklärt wird. Unterhalb der Menu-Leiste erscheint die Simulationsüberschrift, die auch ins Protokoll aufgenommen wird. Danach wird das Protokoll abgeschaltet und die Simulation gestartet (Initialisierung u.a.) und knapp vor Eintritt in den 1. Zyklus unterbrochen. Danach erwartet das CFS eine Menu-Eingabe (z.B. `C` um die Simulation fortzusetzen). Solche Menu-Eingaben können auch später *jederzeit*, d.h. auch während die Simulation läuft, erfolgen.

4.2.1 Ausgabe (`0123`) (`Display`)

Mit der Taste D wird der aktuelle Status des CFS angezeigt. Mit den Tasten 0 bis 3 wird die Ausführlichkeit sämtlicher Ausgaben beeinflußt. Für Details ↑4.2.6 und Anhang C.

4.2.2 Einzelschritt-Abarbeitung (`stEp`) (`ESC`) (`Cont`)

Im `SingleStep-Mode` wird die Simulation nach jedem Zyklus oder jeder Episode angehalten. Ist `SingleStep=OFF`, so läuft die Simulation solange, bis sie mit ESC (oder A oder B) unterbrochen wird. In beiden Fällen kann die Simulation mit C oder *space* fortgesetzt werden. Mit E kann der `SingleStep-Mode` aus- und auch wieder eingeschaltet werden.

4.2.3 Protokoll (`Prot`)

Die Überschrift incl. Zeitpunkt des Simulationsstarts wird immer mitprotokolliert. Zu jedem Zeitpunkt kann man mit P die Protokollierung ein- und auch wieder ausschalten. Ist `Protocoll=ON`, so wird *alles*, was auf dem Bildschirm erscheint auch an das File, deren Name in `EXAMPLE`*n*`.C` durch `ProtDev[]` angegeben wurde, *angehängt*; d.h. ein evtl. bestehendes Protokoll wird nicht gelöscht, sondern erweitert.

Ändert man den Protokollnamen während der Simulation, so ist zu beachten, daß dieser neue Name erst nach einmaligem Tastendruck P aktiviert wird.

Tip: Den Protokoll-File kann man einfach und schnell mit dem Turbo-Editor inspizieren und evtl. aufbereiten.

4.2.4 Parameter-Modifikation (`Break`)

Parameter können nach Beendigung der Simulation mittels A im `EXAMPLE`*n*`.C`-File modifiziert und das CFS neu compiliert und gestartet werden.

Wesentlich schneller kann man dies auch erreichen, indem man mit B die Simulation unterbricht (nur wenn der Breakpoint gesetzt wurde, da dieser aufgerufen wird !).

Nun befinden Sie sich in der IDE (oder im Debugger) und dürfen alles machen, was ihnen die IDE gestattet; nur den Source-File sollten Sie nicht modifizieren.

Die häufigste Anwendung wird wohl die Einsichtnahme in oder die Modifikation von Benutzer-Parametern (↑Kapitel 5) sein. Dies können Sie mittels

`Ctrl-F4` *Name* [*return*] *bzw.* `Ctrl-F4` *Name* = *Wert*

erreichen. Für Details siehe Turbo-C-Beschreibung.

Die Simulation können Sie mit [Alt-R] [R] fortsetzen, wollen Sie die Simulation neu starten, drücken Sie anschließend noch R (Restart). Auf keinen Fall sollten Sie den Neustart von der IDE aus z.B. mittels Alt-R P Alt-R R vornehmen, da diese ihre Parametermodifikationen rückgängig macht.

4.2.5 Sonstige Menu-Befehle

Mit [R] (Restart) können Sie das CFS neu starten. R ist in jeder Hinsicht ein absoluter Neustart, auch der Zufallszahlen-Generator wird wie beim Erstaufruf gesetzt. Somit lassen sich Simulationsläufe exakt reproduzieren.
Mit [Z] (`rdZ`) kann der Zufallszahlen-Generator zufällig (nicht reproduzierbar) initialisiert werden.
Der Warnton bei Warnungen und Fehlern läßt sich mittels [S] (Sound) aus- und einschalten.

[T] (`Test`) ruft die Routine `Test` auf, die sich in `CFSSIM.C` befindet, aber auch nach `EXAMPLE`*n*`.C` gelegt werden kann und bei Bedarf mit beliebigem Inhalt gefüllt werden kann.

4.2.6 Status- & Statistik-Ausgabe

Neben Fehlermeldungen, Warnungen und anderem "Kleinkram" können der *Classifier*status (CS) und die Statistik (ST) als Einzeiler oder ausführlich angezeigt werden. Dies erfolgt entweder mit der Taste D (Display) oder automatisch während der Simulation. Wie und wann was erscheint kann der Tabelle 5 und dem Beispielprotokoll im Anhang C entnommen werden.

`prMode`	**Anzeige pro Schritt**	**Anzeige pro Episode**	**weiteres**
3	CS ausführlich	CS & ST ausführlich	*Warnings & Errors*
2	CS einzeilig	CS & ST ausführlich	*Warnings & Errors*
1	—	Stat einzeilig	*Warnings & Errors*
0	—	—	*Errors*

Tabelle 5: Ausgabe in den verschiedenen *Print-Modes*

Jeder Warnung ist die Häufigkeit ihres bisherigen Auftretens vorangestellt. Dies hat folgenden Sinn:

Name	Bereich	Beschreibung
SMessL	$1 \ldots \infty$	Max. Länge der *Message*-Liste incl. Detektor-*Message*.
SOutL	$1 \ldots \infty$	Max. Anzahl von Aktionen (*Output-Message*).
SCFL	$1 \ldots \infty$	Anzahl der *Classifier*.
SEffL	$1 \ldots \infty$	Anzahl der Effektoren.
SBidL	$1 \ldots \infty$	Max. Anzahl von *Bid's* bzw. *Bid-Tupels*.
STupL	$1 \ldots \infty$	Wählen Sie SBidL und STupL so groß wie möglich.
SStr	$1 \ldots 32$	*Message*,*Condition* & *Action*-Länge, d.h. Anzahl der 0/1/#-Stellen.
TwoCond	FALSE	*Classifier* haben eine *Condition*.
	TRUE	*Classifier* haben zwei *Conditions*.
SpeedF	FALSE	Evtl. auftretende Fehler werden überprüft und als *Warning* oder *Error* angezeigt.
	TRUE	CFS ist auf Zeit optimiert.

Tabelle 6: Array-Größen und andere Compiler-Information

Da manchmal Warnungen penetrant häufig auftreten, man deren Ursache aber erst später beheben möchte, wird ab der vierten Warnung (eines Typs) diese nur noch stichprobenartig ausgegeben, und zwar im Laufe der Zeit immer seltener. An der Nummerierung erkennt man aber dennoch wie häufig die Warnung in der Zwischenzeit erzeugt wurde. Dieses Konzept hat den Vorteil, daß schon nach kurzer Zeit die Anzeige frei von lästigen Warnungen bleibt. Es versteht sich von selbst, daß man sich trotzdem Gedanken darüber machen sollte, ob eine Warnung nicht doch ihre Berechtigung hat.

5 Benutzer-Parameter

Eine ausführliche Beschreibung aller Benutzer-Parameter und deren Wirkungsweise findet man in [6]. Tabellarische Übersichten sind hier abgebildet.

Zusätzlich zu den theoretischen Bereichen ergeben sich noch Einschränkungen durch die physikalischen Datentypen, wie `int`, `long`, ...

Name	Bereich	Beschreibung
StRate	$1 \ldots \infty$	Episodenlänge: Anzahl der Zyklen, nach denen und für die eine Statistik berechnet und angezeigt wird.
SclSel	$50\% \ldots 100\%$	Skalierungsfaktor bei SelMode=4 (↑2.2.4).
ProtDev		Protokoll-Filename als String.
DetMessTag		*Tag's* für Detektor-*Message* als String
EffMessTag		*Tag's* für Effektor-*Message* als String

Tabelle 7: Allgemeine Variablen

Name	Bereich	Beschreibung
`DetRate`	$1 \ldots \infty$	Detektor-Aufruf-Rate
`MDetM`	$1 \ldots$ `SMessL`	Max. Anzahl von *Detector-Messages*
`DefDetMI`	$0.0 \ldots 9.0$	*Default-Detector-Message-Intensity*
`DefStr`	$0.0 \ldots 9.0$	*Default-Classifier-Strength*
`DefStrDev`	$0 \ldots 25\%$	*Default-Classifier-Strength-Deviation*

Tabelle 8: `Detector`-Variablen

Name	Bereich	Beschreibung
`DefNMI`	$0.0 \ldots 9.0$	*Default-Non-Match-Intensity:* *Message-Intensity*-Ersatzwert bei negativen *Conditions*
`Dev2`	$0 \ldots 25\%$	*EBid-Deviation*
`DelMessSelMode`	$-6 \ldots 6$	*Selection-Mode* bei zu vielen *Messages* in abh. der *Intensity*
`CleanHallF`	`FALSE` `TRUE`	*Classifier* mit *Detector-Tag* werden nicht gelöscht *Classifier* mit *Detector-Tag* werden gelöscht
`EffRate`	$1 \ldots \infty$	Effektor-Aufruf-Rate

Tabelle 9: `Matching-, Bid-, Clean- & Effector-`Variablen

Name	Bereich	Beschreibung
`ReRate`	$1 \ldots \infty$	*Reinforcement*-Aufruf-Rate
`CreditMode`	$1 \ldots 4$	1 = Expliziter *Bucket-Brigade* 2 = Impliziter *Bucket-Brigade* (nicht impl.) 3 = *Profit-Sharing-Plan* (PSP) 4 = Kein *Credit-Assignment*
`NMFrac`	$0 \ldots 100\%$	Reduzierter *Reinforcement / Credit* für *Classifier* mit negativer *Condition*
`DetFrac`	$0 \ldots 100\%$	Reduzierter *Reinforcement / Credit* für *Detector-Messages*
`PSPFrac`	$0 \ldots 100\%$	PSP-Reduktions-Konstante
`PSPDisMode`	$1 \ldots 4$	PSP-*Reinf.-Distr.* für *Classifier* ist prop. 1 = eins 2 = Anzahl der *Bids* 3 = maximalem *Bid*-Wert 4 = Summe der *Bid*-Werte

Tabelle 10: `Credit`-Variablen

Name	Bereich	Beschreibung
HeadTax	0.0...9.0	*Taxes* pro Zyklus
MessTax	0.0...9.0	*Taxes* für das Senden einer *Message*
BidTax	0.0...9.0	*Taxes* für das Bieten (*Bid*)
BidTaxMode	1...2	2 = Zahle nur, falls Bieten nicht zum Feuern führte 1 = Zahle immer
MultBidTaxF	TRUE	Zahle für jeden *Bid* extra
	FALSE	Zahle einmal für alle *Bids*
MultMessTaxF	TRUE	Zahle für jede gefeuerte *Message*
	FALSE	Zahle einmal für alle gefeuerten *Messages* extra

Tabelle 11: Taxes-Variablen

Name	Bereich	Beschreibung
MaxStr	1.0...20.0	Max. *Classifier-Strength*
MinStr	0.0...1.0	Min. *Classifier-Strength*

Tabelle 12: StrUpDate-Variablen

Name	Bereich	Beschreibung
GenRate	1...∞	Aufruf-Rate des genetischen Algorithmus'
NCFRepl	0...SCFL	Anzahl der zu ersetzenden *Classifier*
ReplSelMode	-6...6	*Selection-Mode* beim Ersetzen der *Classifier* in abh. von 1/*Strength*
NCFMut	0...SCFL	Anzahl der zu mutierenden *Classifier*
NMutate	0...SStr	Mutationen pro *Condition / Action*
MutSelMode	-6...6	*Selection-Mode* beim Mutieren der *Classifier* in abh. von 1/*Strength*
CondSp	0...100%	Spezifizität der *Condition bzw. Action*, d.h. Wahrscheinlichkeit einer nicht-#-Stelle
ActSp	0...100%	
NegCondP	0...100%	Wahrscheinlichkeit einer negativen *Condition*

Tabelle 13: Genetic-Variablen

Name	Bereich	Beschreibung
DelMessMode	0...2	0 = lösche keine *Messages*. 1 = lösche *Messages* mit *Effector-Tag*. 2 = lösche alle *Messages*

Tabelle 14: ReInit-Variablen

Name	**Bereich**	**Beschreibung**
`BidMode`	1...4	1 = ein *Bid* pro *Classifier* 2 = ein *Bid* pro *Matching-Tupel* 3 = Wie 1 – nur: Aktionsliste statt Tupel-Liste 4 = Wie 2 – nur: ein *Bid* pro Aktion
`SuppMode`	1...2	1 = Zähle die *Intensity* bei *BidTupels* mit zwei gleichen *Messages* nur einfach 2 = Addiere die *Intensity* beider *Messages*
`BidTupSelMode`	-6...6	Mode zum Selektieren von Tupeln in `MatchBid`
`BidActSelMode`	-6...6	Mode zum Selektieren von Aktionen in `MatchBid`
`FireBidSelMode`	-6...6	Mode zum Selektieren von *Bids* in `Fire`
`FireTupSelMode`	-6...6	Mode zum Selektieren von Tupeln in `Fire`
`MBids`	$1...\infty$	Max. Anzahl von *Bids* pro *Classifier*
`MAct`	$1...\infty$	Max. Anzahl von *Actions* pro *Classifier*
`MFire`	$1...\infty$	Max. Anzahl von gefeuerten *Messages* pro *Classifier*
`RiskFak`	0...100%	Berechnung des (effektiven) *Bids* mittels:
`SpecPow`	0...2	$Bid = RiskFak \cdot Strength \cdot Spec^{SpecPow} \cdot RelSupp^{RelSuppPow}$
`RelSuppPow`	0...2	
`EBidPow`	0...2	$EffBid = [Bid + Noise(Bid \cdot Dev2)]^{EBidPow} \cdot Spec^{ESpecPow}$
`ESpecPow`	0...2	
		Modes für expliziten *Bucket-Brigade*
`ClaimMode`	1...2	Mehrfache Verwendung einer *Message* wird dem Sender 1 = nur einfach, 2 = mehrfach vergütet.
`DisMode`	1...2	1 = *Credit* ist unabhängig von der *Message-Intensity* 2 = *Credit* ist prop. zur *Message-Intensity*
`SumMode`	1...3	1 = Teil des *Reinf.* bzw. *Bids* geht verloren (`NMFrac`) 2 = *Reinf.* bzw. *Bid* wird restlos aufgeteilt 3 = Jeder *Classifier* bekommt vollen *Reinf.* bzw *Bid*

Tabelle 15: *Classifier/Effector*-Variablen

6 Ergänzungen und Ausblicke

Das Programm ist so konzipiert, daß es in vielerlei Hinsicht erweitert werden kann. Dies betrifft sowohl programmspezifische Erweiterungen, wie z.B.

- Menu-Punkte, um Simulations-Status auf Diskette abzuspeichern und von Diskette zu laden.
- Compileroptionen (besonders um *Compiler-Warnings* zu unterdrücken) mittels `#pragma warn` *xyz* ins Programm aufnehmen.
- Per Menu ein- und ausschaltbare *Warnings*.
- Flexible Skalierung von FixPunkt-Zahlen.
- Unterstützung von CFS-Aufrufen durch *Batch*-Prozesse.

als auch Erweiterungen des CFS selbst, wie z.B.

- Profit-Sharing-Plan (`CreditMode=2`).
- Erweiteter genetischer Algorithmus (Kreuzung, Häufung, getriggerter GA,...) ↑[4] (Wird bis Ende '91 zur Verfügung stehen).
- Tree & Food Beispiel von Wilson [5] (und andere Beispiele).

Eine Benutzerbeschreibung des Klassifizierungs-Systems findet sich in [6]

Abschließend sei noch darauf hingewiesen, daß für die Fehlerfreiheit dieses Programms – natürlich – keine Garantie übernommen werden kann.

7 Was tun im Fehlerfall ?

Im Folgenden werden eine Reihe von Warnungen und Fehlern beschrieben, die evtl. beim Simulationslauf auftreten können und deren Ursache meist in fehlerhaften oder kritischen Benutzer-Variablen oder Benutzer-Funktionen liegt.

7.1 Warnungen

Warnungen sind im Normallfall ebenso wie Fehlermeldungen vom Benutzer ernst zu nehmen und bei der nächsten Gelegenheit zu beheben. Allerdings wird bei Warnungen, im Gegensatz zu Fehlermeldungen, die Simulation fortgesetzt (↑4.2.6).

`Warning 0: Unspecific Warning`
: Programmfehler: Benachrichtigen Sie den Programmierer.

`Warning 1: Division underflow`
Bei der Division wird eine Zahl derart klein, daß sie in der FixPunkt-Darstellung zu Null wird
Tritt diese Warnung häufiger auf, und sind generell die FixPunkt-Werte, die in Status und Statistik angezeigt werden recht klein (alle kleiner als 5), so sollten Sie versuchen, alle Benutzer-Parameter *(Taxes, Reinf, ...)* neu zu skalieren, d.h. alle mit einer Konstanten zu multiplizieren, was ja nichts an der Funktionalität ändert.

`Warning 2: QMax called with k=0`
`QMax` *sollte die Null größten Elemente suchen*
Programmfehler: Benachrichtigen Sie den Programmierer.

`Warning 3: ProbSel called with N=0`
`ProbSel` *sollte aus Null Elementen* $k > 0$ *auswählen*
Programmfehler: Benachrichtigen Sie den Programmierer.

`Warning 4: Tupellist is full`
In der Liste der Bidtupels ist kein Platz mehr für weitere Eintragungen
Dimensionieren Sie die Tupelliste mit `STupL` so groß wie möglich (es Ihre Speicherkapazität zuläßt). Haben Sie trotz eines relativ kleinen Beispiels (wenig *Classifier* und *Messages*) und großer Tupelliste Schwierigkeiten, so sollten Sie in Betracht ziehen, `MBids` bzw. `MAct` zu verkleinern. Eine übervolle Tupelliste kann auch Ursache zu unspezifischer *Classifier-Conditions* sein. Ist dies der Fall, so sollten Sie `CondSp` erhöhen.

`Warning 5: Bidlist is full`
In der Liste der Bids ist kein Platz mehr für weitere Eintragungen
Dimensionieren Sie die Bidliste mit `SBidL` so groß wie möglich (es Ihre Speicherkapazität zuläßt). Eine übervolle Tupelliste kann auch Ursache zu unspezifischer Classifier-Conditions sein. Ist dies der Fall, so sollten Sie `CondSp` erhöhen.

`Warning 6: Selection of Zero-Values`
Bei einem Selektionsvorgang wurden Elemente mit Bewertung (`Val`*) Null ausgewählt, die meist Ursache vorangegangener Unterläufe sind*
Beachten Sie die unter Warning 1 angegebenen Hinweise.

`Warning 7: BestSelMode must be positive`
Die k besten Elemente mit Wiederholung zu selektieren macht keinen Sinn
Ersetzen Sie x`SelMode=-3` durch x`SelMode=+3`.

`Warning 8: Too many matching tuples`
Es gibt einen *Classifier, den zu viele Tupels matchen*
Beachten Sie die unter Warning 4 angegebenen Hinweise.

7.2 Fehlermeldungen

`Error 0: Unspecific Error`
Programmfehler: Benachrichtigen Sie den Programmierer.

`Error 1: Multiplication Overflow`
Überlauf beim Multiplizieren zweier FixPunkt-Zahlen
Sie sollten versuchen alle Benutzer-Parameter (*Taxes, Reinf, ...*) neu zu skalieren, d.h. alle durch eine Konstante zu dividieren, was ja nichts an der Funktionalität ändert.
Fortsetzung kritisch – System fängt sich aber meist nach einigen Runden

`Error 2: Division Overflow`
Überlauf beim Dividieren zweier FixPunkt-Zahlen
Beachten Sie die Hinweise unter Error 1.
Fortsetzung kritisch – System fängt sich aber meist nach einigen Runden

`Error 3: Wrong Selection-Mode`
x`SelMode` *ist auf ungültige Werte gesetzt (erlaubt sind -6..6)*
Fortsetzung nicht sinnvoll

`Error 4: Select more than possible`
Es sollten mehr (verschiedene) Elemente selektiert werden als vorhanden sind
Programmfehler: Benachrichtigen Sie den Programmierer.

`Error 5: Wrong User-Classifier/Message`
Das Format der vordefinierten User-Classifier / Message-Strings ist falsch
Überprüfen Sie `MessSL`, `CFSL` und `EffSL`. Vielleicht ist die in `SStr` angegebene Länge inkonsistent besetzt.
Fortsetzung nicht sinnvoll

`Error 6: Couldn't allocate Memory`
Fehler in der Speicherreservierung für die verschiedenen Listen
Wählen Sie die mit `#define S`*x* definierten Listenlängen kleiner oder compilieren Sie mit dem Compiler-Modell `Compact`, um mehr als 64 KByte für die Listen zur Verfügung zu haben.
Fortsetzung katastrophal (Absturzgefahr)

`Error 7: Wrong BidMode`
`BidMode` *ist auf ungültige Werte gesetzt (erlaubt sind 1..4)*
Fortsetzung nicht sinnvoll

`Error 8: Wrong Credit-Assign-Mode`
`CreditMode` *ist auf ungültige Werte gesetzt (erlaubt sind 1,3 und 4)*
Fortsetzung nicht sinnvoll

`Error 9: prf is called with prMode=0`
Es wurde versucht etwas anzuzeigen, obwohl `prMode` *auf 0 gesetzt ist*
Programmfehler: Benachrichtigen Sie den Programmierer.

A Programmlisting CFSSIM.C

```
/*****************************************************************/
/*                                                               */
/*     C l a s s i f i e r - S y s t e m - S i m u l a t o r     */
/*                                                               */
/*****************************************************************/
/*                  in Turbo-C 2.0 on IBM-AT                     */
/*       Diplomarbeit: Copyright 1991 by Marcus Hutter           */
/*                      Stand: 13.05.1991                        */
/*****************************************************************/

/*****************************************************************/
/*              R e m a r k s                                    */
/*****************************************************************

- It is assumed, that 'int' is 16-bit and 'long' is 32-bit signed.
- Use Compact-Model (instead of Small), if your CFS-Example is so
  large, that you get Errors about 'too many dynamic Variables'
  (Mess,CF,..) and ignore warnings about loosing digits.
- Set an Breakpoint in and compile the EXAMPLE.C-File;
  this file is included.

 *****************************************************************/
/*              D e f i n i t i o n s                            */
/*****************************************************************/

/*-----------------------------*/
/*        Include-Files        */
/*-----------------------------*/

#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <conio.h>
#include <dos.h>
#include <setjmp.h>

/*-----------------------------*/
/*         Definitions         */
/*-----------------------------*/

#define then    /**/            /* if then else                 */
#define ESC     27              /* define the escape key        */
#define FALSE   0               /* FALSE for bool               */
#define TRUE    1               /* TRUE  for bool               */
#define MaxInt  32767           /* Maximal signed integer       */
#define FOREVER for(;;)         /* infinite loop                */
#define PSize   sizeof(*)       /* Pointer-Size                 */
#define NScrL   50              /* # of Screen-Lines 25/43/50   */
#define STDCOL  YELLOW          /* Standard Color               */

#define MMask   (TString)~(~(long)0<<SStr) /* Cond/Act-Mask     */
#define NonId   -1              /* Not existing Id              */
```

```
#define DetId   -1              /* 'CF-Id' of Detector          */

#if TwoCond                     /* ifTC(x) eval. x if TwoCond   */
  #define ifTC(x) x             /* ifnTC(x)  "   x if not "     */
  #define ifnTC(x)
#else
  #define ifTC(x)
  #define ifnTC(x) x
#endif

/*------------------------------*/
/*         Struct. & Types      */
/*------------------------------*/

typedef char            bool;   /* bool = { TRUE , FALSE }      */
typedef unsigned char   byte;   /* 0..255                       */
typedef signed char     sbyte;  /* -128..127                    */
typedef signed int      index;  /* Array-Indices                */
typedef signed int      fixp;   /* FixPoint: Range -128 to 127  */
                                /*         : step 1/256         */
                                /* real(constants) to FixPoint  */
#define FixP(x) ((fixp)(((float)x)*256))

#if FALSE /* (SStr<=sizeof(byte)*8) *//* Type of String         */
  typedef byte TString;         /* Test, da sonst Anz. blöd */
#elif (SStr<=sizeof(int)*8)
  typedef int  TString;
#elif (SStr<=sizeof(long)*8)
  typedef long TString;
#else
  #error Message too long
#endif

typedef byte    TCType;         /* CF & Cond.-Type (ev.BitArray)*/
typedef index   TMessId;        /* Message-Index/Id             */
typedef index   TCFId;          /* Classifier-Index/Id          */
typedef index   TBidId;         /* Bid-Index/Id                 */
typedef index   TTupId;         /* MessTupel-Index/Id           */

typedef struct                  /* = long int for Sort & Select */
      { index   Id;             /* Inedx/Id                     */
        fixp    Val; }          /* Value/Prorpability/Helpinfo  */
        TIdVal;
#define Id(x)   (((TIdVal*)&(x))->Id)  /* first half of long    */
#define Val(x)  (((TIdVal*)&(x))->Val) /* second "   of long    */

typedef struct                  /*         CF-Condition         */
      { TString C;              /* String  ------------         */
        TString B;              /* #-Mask                       */
        TCType  T; }            /* Type                         */
        TCond;

typedef struct                  /*          CF-Action           */
      { TString A;              /* String   ---------           */
        TString B;              /* #-Mask                       */
        TCType  T; }            /* Type                         */
```

```
        TAct;

typedef struct                  /*          Classifier          */
      { TCond   C1;             /* Condition----------          */
  ifTC( TCond   C2; )           /* Condition 2                  */
        TAct    Act;            /* Action                       */
        fixp    Sp;             /* Specifity of Cond1 (& Cond2) */
        fixp    S;              /* Strength                     */
        fixp    Get;            /* Reinf + Pay from others      */
        fixp    Pay;            /* Pay to others                */
        fixp    Tax;            /* Taxes                        */
        fixp    PSP;            /* Bid-Sums over Reinf-Expisode */
        int     H; }            /* Help-Value                   */
        TCF;

typedef struct                  /*           CF-List            */
      { TCF     *B;             /* Begin     -------            */
        TCF     *C;             /* Current End                  */
        TCF     *E;             /* End of List                  */
        byte    BidMode;
        byte    SuppMode;
        sbyte   BidTupSelMode;
        sbyte   BidActSelMode;
        sbyte   FireBidSelMode;
        sbyte   FireTupSelMode;
        int     MBids;
        int     MAct;
        int     MFire;
        fixp    RiskFak;
        byte    SpecPow;
        byte    RelSuppPow;
        byte    EBidPow;
        byte    ESpecPow;
        byte    ClaimMode;
        byte    DisMode;
        byte    SumMode; }
        TCFL;

typedef struct                  /*            Message           */
      { TString M;              /* String     -------           */
        fixp    I;              /* Intensity                    */
        TCFId   CF;             /* Id of sending CF             */
        int     H; }            /* Help-Value                   */
        TMess;

typedef struct                  /*         Message-List         */
      { TMess   *B;             /*         ------------         */
        TMess   *C;
        TMess   *E; }
        TMessL;

typedef struct                  /*             Bid              */
      { TCFId   CF;             /* Bidding CF  ---              */
        fixp    BV;             /* Bid-Value                    */
        fixp    EV;             /* Effective-Bid-Value          */
        TTupId  Tup;            /* Matching-Tuples              */
```

```
        int     N;              /* Number of Bid-Tuples         */
        bool    F; }            /* Fired-Flag                   */
        TBid;

typedef struct                  /*           Bid-List           */
      { TBid    *B;             /*           --------           */
        TBid    *C;
        TBid    *E; }
        TBidL;

typedef struct                  /*           MessTupel          */
      { TMessId M1;             /* Message 1 ---------          */
  ifTC( TMessId M2; )           /* Message 2                    */
        fixp    S; }            /* Support                      */
        TTup;

typedef struct                  /*        MessTupel-List        */
      { TTup    *B;             /*        --------------        */
        TTup    *C;
        TTup    *E; }
        TTupL;

typedef struct                  /*      Classifier-System       */
      { TMessL  *MP;            /* Mess =================       */
        TMessL  *NP;            /* New-Mess                     */
        TMessL  *OP;            /* Output-Mess (from Eff)       */
        TCFL    *CP;            /* Classifier                   */
        TCFL    *EP;            /* Effectors                    */
        TBidL   *BP;            /* Bids                         */
        TBidL   *DP;            /* Eff-Bids (same list)         */
        TTupL   *TP;            /* MessTupel                    */
        TTupL   *UP; }          /* Eff-MessTupel (same list)    */
        TCS;

typedef struct                  /*              Statistic       */
      { long    CycleC;         /* Cycle-Counter---------       */
        long    ReVal;          /*  of Reinf                   */
        long    Str;            /*  of CF-Stregnth             */
        long    NCFMinStr;      /* # of CF with MinStr          */
        long    NCFMaxStr;      /* # of CF with MaxStr          */
        long    MCF;            /* # of NonMatched CF           */
        long    MMess;          /* # of NonMatched Mess.        */
        long    NCF;            /* # of CF                      */
        long    NBid;           /* # of Bids                    */
        long    NTup;           /* # of Tupels                  */
        long    NMess; }        /* # of Messages                */
        TStat;

/*------------------------------*/
/*      Variables               */
/*------------------------------*/

TMessL  XM,XN,XO;
TBidL   XB,XD;
extern TCFL XC,XE;
TTupL   XT,XU;
```

```
TStat   cy,ep;

TCS     XCS = { &XM,&XN,&XO,&XC,&XE,&XB,&XD,&XT,&XU };
TCS     *o = &XCS;

long    CycleC;                 /* Main-Cycle-Step-Counter      */
fixp    ReVal;                  /* Reinforcement-Value          */
TString DetMTagC,DetMTagB;      /* Detector-Tags                */
TString EffMTagC,EffMTagB;      /* Effector-Tags                */

TMess   *GMB;                   /* Global-Mess-Pointer          */
TString *GStr;                  /* Global Strting-Pointer       */

jmp_buf jmp;                    /* jump-label                   */
FILE    *devP,*lpt1;            /* Output-Device                */
bool    ProtF;                  /* Protocoll is Off/On          */
bool    SStepF;                 /* Single-Step-Mode             */
bool    prM1F;                  /* printmode 1 - first flag     */
bool    SoundF = TRUE;          /* Sound On / Off               */
byte    prMode = 3;             /* Print-Mode 0=No ... 3=All    */

/*-----------------------------*/
/*      Constants              */
/*-----------------------------*/

char    FName[14] = "CFSSIM.C"; /* File-Name (not needed)       */
char    *ErrMess[] =            /* Error-Messages               */
      { "Unspecific Error",
        "Mutliplication Overflow",
        "Division Overflow",
        "Wrong Selection-Mode",
        "Select more than possible",
        "Wrong User-Classifier/Message",
        "Couldn't allocate Memory",
        "Wrong BidMode",
        "Wrong Credit-Assign-Mode",
        "pprint is call with prMode=0" };

char    *WarnMess[] =           /* Warnning-Messages            */
      { "Unspecific Warning",
        "Division Underflow",
        "QMax called with k=0",
        "ProbSel called with N=0",
        "Tupellist is full",
        "Bidlist is full",
        "Selection of Zero-Values",
        "BestSelMode must be positive",
        "Too many matching tuples" };

#define NWarn (sizeof(WarnMess)/sizeof(char*))
int     WC[NWarn];
int     WP[NWarn];

/*-----------------------------*/
/*      Test-Variables         */
/*-----------------------------*/
```

```
/*-----------------------------*/
/*      Externals              */
/*-----------------------------*/

/* Example-Specific variables and functions                    */
/* For explanation refer to Example1/2/...                     */
extern  int     StRate,DetRate,EffRate,ReRate,GenRate;
extern  int     MDetM,MEqMess,NMutate,NCFMut,NCFRepl;
extern  fixp    DefNMI,SclSel,DefDetMI,DefStr,DefStrDev;
extern  fixp    Dev1,Dev2,NMFrac,DetFrac,HeadTax,MessTax,BidTax;
extern  fixp    MaxStr,MinStr,CondSp,ActSp,NegCondP,PSPFrac;
extern  byte    NMIMode,CreditMode,BidTaxMode,PSPDisMode;
extern  byte    DelMessMode;
extern  sbyte   DelMessSelMode,MutSelMode,ReplSelMode;
extern  bool    CleanHallF,MultBidTaxF,MultMessTaxF;
extern  char    DetMessTag[],EffMessTag[],ProtDev[];
extern  char    *CFSL[],*EffSL[];
extern          InitEnv(),Detect(TMessL*,int),Effect(TMessL*);
extern          Display();
extern  fixp    Reinf();

/*-----------------------------*/
/*      Forward Declarations   */
/*-----------------------------*/

extern          Menu();
extern          main();
extern  char    event();

/*************************************************************/
/*              G e n e r a l   P r o c e d u r e s           */
/*************************************************************/

#define Sound(h,l)                                            \
        { if (SoundF) { sound(h); delay(l); nosound(); } }

/*-----------------------------*/
 prf(int Col,char*fmt,...)      /* printf to dev & devP        */
/*-----------------------------*/
{ char  str[999];
  extern Error(int);
  if (Col!=0) then textcolor(Col);
  if (prMode>0) then
  { vsprintf(str,fmt,...);
    cprintf(str);
    if (ProtF) then fprintf(devP,str); }
  else Error(9);
  if (Col!=0) then textcolor(STDCOL);
}
/*-----------------------------*/
        Error(n)                /* break with Error # 10+n     */
/*-----------------------------*/
{ prMode++;
  textbackground(LIGHTRED);
  prf(WHITE,"Error #%d: %s\r\n",10+n,ErrMess[n]);
```

```
  textbackground(BLACK); clreol();
  textbackground(LIGHTRED);
  prf(WHITE,"Press any Menu-Key, but its risky to continue !!\r\n");
  textbackground(BLACK); clreol();
  prMode--;
  Sound(500,100);
  Menu();
}
/*-----------------------------*/
       Warning(n)              /* write Warning # n           */
/*-----------------------------*/
{ WC[n]++;
  if (WC[n]%((WP[n]>>2)+1)==0)  /* /4 = Warning decay-Rate     */
  { if (prMode>0) then prf(LIGHTGRAY,
      "%d.Warning #%d: %s\r\n",WC[n],20+n,WarnMess[n]);
    WP[n]+=1+(WP[n]>>1);
    Sound(1000,5);
} }
/* Random-Number between 0 and n-1 */
#define random2(n) ((int)(((long)rand()*n)>>15))

/* TRUE with Prob. p (p is fixp. !) */
#define flip(p)    ((p)>(rand()>>7))

/*-----------------------------*/
char*   ActTime()              /* Actual Time in String       */
/*-----------------------------*/
{ char  *s;
  time_t t;                    /* Actual Time                 */
  time(&t);
  s=ctime(&t);
  s[strlen(s)-1]='\0';
  return(s);
}
/*-----------------------------*/
       memswap(P,Q,n)          /* swap n Bytes from P and Q   */
/*-----------------------------*/
char    *P,*Q;
{ char  h,*E=P+n;
  while(P<E) { h=*P; *P++=*Q; *Q++=h; }
}
#if SpeedF                     /* fast Fix-Point-Operations   */
#define FPMult(x,y) (fixp)((long)(x)*(y)>>8)
#define FPDiv(x,y)  (fixp)(((long)(x)<<8)/(y))
#else
/*-----------------------------*/
fixp    FPMult2(x,y)           /* Fix-Point-Mult with Test    */
/*-----------------------------*/
long    x,y;
{ long h=(x*y)>>8;
  if ((h>0?h:-h)>MaxInt) then Error(1);
  return((fixp)h);
}
/*-----------------------------*/
fixp    FPDiv2(x,y)            /* Fix-Point-Division with Test */
/*-----------------------------*/
```

```
long    x,y;
{ long h=(x<<8)/y;
  if ((h>0?h:-h)>MaxInt) then Error(2);
  if (x!=0&&h==0) then Warning(1);
  return((fixp)h);
}                               /* secure Fix-Point-Operations  */
#define FPMult(x,y) FPMult2((long)(x),(long)(y))
#define FPDiv(x,y) FPDiv2((long)(x),(long)(y))
#endif
/*-----------------------------*/
fixp    FPPow(y,x,n)            /* Fix-Point-Pow y*x^n          */
/*-----------------------------*/
fixp y,x; byte n;
{ int i;
  for(i=0; i<n; i++) y=FPMult(y,x);
  return(y);
}
/*-----------------------------*/
int     Noise(Dev)              /* Standard-Normal-Distribution */
/*-----------------------------*  Cut at +-4 in .5 Steps       */
int     Dev;                    /* * deviation Dev              */
{ static int s=8;
  byte  c;
  int   b;
  if (Dev) then
  { c=0; b=rand();
    while(b) { c+=b&1; b>>=1; }
    return(Dev*(c-(s=15-s)) >> 1); }
  else return(0);
}
/* Order-Functions must be Reflexive, that means InOrder(x,x) ! */
/*-----------------------------*/
bool    InStdOrd(x,y)           /* Standard-Order-Function      */
/*-----------------------------*/
TIdVal  x,y;
{ return(x.Val<=y.Val);
}
/*-----------------------------*/
bool    InMessOrder(x,y)        /* Mess-Order-Function          */
/*-----------------------------*/
TIdVal  x,y;
{ return(GMB[x.Id].M<=GMB[y.Id].M);
}
/*-----------------------------*/
bool    InStrOrder(x,y)         /* String-Order-Function        */
/*-----------------------------*/
TIdVal  x,y;
{ return(GStr[x.Id]<=GStr[y.Id]);
}
/*-----------------------------*/
byte    Count1s(s)              /* Count # of 1 Bits            */
/*-----------------------------*/
TString s;
{ byte c=0;
  while(s) { if (s&1) then c++; s>>=1; }
  return(c);
```

```
}

/*------------------------------*/
        Shuffle(A,N)            /* Shuffle N El. of A           */
/*------------------------------*/
long    A[];
int     N;
{ index m;
  long  h;
  while(N>1)
  { m=random2(N);
    h=A[m]; A[m]=A[--N]; A[N]=h;
} }
/*------------------------------*/
#define QDivide()               /* Used in QMax and QSort */    \
/*------------------------------*/              \
{ d=*M;                                         \
  while(B<C)                                    \
  { if (M-B<C-M) then                           \
      if (InOrder(d,h=*C)) then C--;            \
      else { *C=*B; *B++=h; }                   \
    else                                        \
      if (InOrder(h=*B,d)) then B++;            \
      else { *B=*C; *C--=h; } }                 \
  if (InOrder(d,*C)) then B--; else C++;        \
}
/*------------------------------*/
        QMax(A,N,k,InOrder)     /* k biggest El. to bottom of A */
/*------------------------------*/
long    A[];                    /* Array of length N            */
int     N;                      /* Length of Array              */
int     k;                      /* # of El. to select           */
bool    InOrder(long,long);     /* Order-Criterium              */
{ long  *B,*B2,*C,*C2;          /* Begin & End of part of A     */
  long  *M,d;                   /* Div.-Point & -Element of A   */
  long  h;                      /* help-variable for exchange   */
  if (k==0) then { Warning(2); return; }
  B=B2=A; C=C2=A+N-1; M=C-k+1;
  FOREVER
  { QDivide();
    if (C<M)      then { B2=B; C=C2; }
    else if (C>M) then { B=B2; C2=C; }
    else break;
} }
/*------------------------------*/
        QSort(B2,C2,InOrder)    /* Sort A In-Order              */
/*------------------------------*/
long    *B2;                    /* Begin of Index-Array         */
long    *C2;                    /* End of Index-Array (B2+#El-1)*/
bool    InOrder(long,long);     /* Order-Criterium              */
{ long  *B,*C;                  /* Actual Begin & End -Pointer  */
  long  *M,d;                   /* Div.-Point & -Element of A   */
  long  h;                      /* help-variable for exchange   */
  if (B2<C2) then
  { B=B2; C=C2; M=B+((C-B+1)>>1);
    QDivide();
```

```
    QSort(B2,B,InOrder);
    QSort(C,C2,InOrder);
} }
/* TIdVal and long are used as synonyms                          */

/*------------------------------*/
int     MaxVal(A,N)             /* Maximal Value of A            */
/*------------------------------*/
TIdVal  A[];                    /* Elements                      */
int     N;                      /* Number of Elements            */
{ int   i;
  int   m=A[0].Val;
  for(i=1; i<N; i++) if (A[i].Val>m) then m=A[i].Val;
}
#define AL         ((long*)A)   /* long synonym for A            */

/*------------------------------*/
        ProbSelInit(A,N)        /* Init Probability-Selection    */
/*------------------------------*/
TIdVal  A[];                    /* Elements                      */
int     N;                      /* Number of Elements            */
{ index i,k;
  fixp  m=MaxVal(A,N);
  if (m<8000 && m) then
  { m=16000/m;
    for(i=0; i<N; i++) A[i].Val*=m; }
  for(k=1; k<N; k<<=1)
    for(i=0; i<N; i+=k<<1)
    { if (i+k<N) then A[i].Val+=A[i+k].Val;
      A[i].Val>>=1; }
}
/*------------------------------*/
index   ProbSel(A,N)            /* Probability-Selection         */
/*------------------------------*/
TIdVal  A[];                    /* Elements                      */
int     N;                      /* Number of Elements            */
{ index i,k=0;
  TIdVal z;
  z.Id=rand()<<1;
  z.Val=random2(A[0].Val);
  for(i=1; i<N; i<<=1);
  for(i>>=1; i!=0; i>>=1)
  { *(long*)&z<<=1;
    if (i+k<N) then
      if (z.Val<A[k+i].Val) then k+=i;
      else z.Val-=A[k+i].Val; }
  return(k);
}
/*------------------------------*/
        ProbChg(A,N,k,p)        /* Probability-Selection         */
/*------------------------------*/
TIdVal  A[];                    /* Elements                      */
int     N;                      /* Number of Elements            */
index   k;                      /* Change Element k              */
fixp    p;                      /* ... to Probability p          */
{ int   i;
```

```
  for(i=1; !(i&k) && i<N; i<<=1)
  { if (i+k<N) then p+=A[i+k].Val;
    p>>=1; }
  p-=A[k].Val;
  for(; i<N; i<<=1)
  { if (k&i) then { A[k].Val+=p; k-=i; }
    p>>=1; }
  A[0].Val+=p;
}
/*-----------------------------*/
        ProbSelInit2(A,N)       /* Init Probability-Selection   */
/*-----------------------------*/
TIdVal  A[];                    /* Elements                     */
int     N;                      /* Number of Elements           */
{ index i,m=0;
  long  h;
  if (N==0) then Warning(3);
  for(i=1; i<N; i++) if (A[i].Val>A[m].Val) then m=i;
  h=AL[m]; AL[m]=AL[0]; AL[0]=h;
}
/*-----------------------------*/
index   ProbSel2(A,N)           /* Probability-Selection        */
/*-----------------------------*/
TIdVal  A[];                    /* Elements                     */
int     N;                      /* Number of Elements           */
{ index k;
  while(random2(A[0].Val)>=A[k=random2(N)].Val);
  return(k);
}
/*-----------------------------*/
      MultSelInit(A,N,Mode)     /* Init Selection (for k<0)     */
/*-----------------------------*/
TIdVal  A[];                    /* Elements & Values            */
int     N;                      /* Number of Elements           */
sbyte   Mode;                   /* Selection-Mode               */
{ switch(abs(Mode))
  { case 1: break;
    case 2: ProbSelInit(A,N); break;
    case 3: QSort(A,A+N-1,InStdOrd); break;
    case 4: break;
    case 5: break;
    case 6: ProbSelInit2(A,N); break;
    default:Error(3);
} }

/*-----------------------------*/
        Select(A,NP,B,k,Mode)   /* Select k of N Elements       */
/*-----------------------------*/
TIdVal  A[];                    /* Elements & Values            */
int     *NP;                    /* Number of Elements           */
index   B[];                    /* Selected k Elements          */
int     k;                      /* Selection-Mode               */
sbyte   Mode;                   /* >0:different; <0:with repeat */
{ index i,h,m,m2;
  int   an;
  int   N=*NP;
```

```
  bool  once=(k>0);

/* treat special cases */
  k=abs(k);
  if (k==0) then return;
  if (k>N && (Mode>0 || N==0)) then { Error(4); Mode=-Mode; }
  if (k==N && Mode>0) then Mode=1;
#if !SpeedF                     /* Test slows down execution    */
  for(i=h=0; i<N; i++)
  if (A[i].Val) then h++;
  if (h<k) then Warning(6);
#endif

/* branch on selection-mode */
  switch(abs(Mode))
  { case  1:                    /* Dummy-Selection              */
      for(i=0; i<k; i++)
        B[i]=A[Mode>0?--N:0].Id;
      break;
    case  2:                    /* Probability-Selection        */
      if (once) then ProbSelInit(A,N);
      for(i=0; i<k; i++)
      { m=ProbSel(A,N);
        B[i]=A[m].Id;
        if (Mode>0 && N>1) then
        { an=A[N-1].Val;
          for(h=N-1; !(h&1); h>>=1) an<<=1;
          ProbChg(A,N,m,an);
          ProbChg(A,N,N-1,0);
          A[m].Id=A[--N].Id;
          if (!(N&N-1)) then A[0].Val<<=1;
      } }
      N=*NP-k; break;
    case  3:                    /* Best-Selection (Different)   */
      if (Mode<0) then Warning(7);
      if (once) then QMax(A,N,k,InStdOrd);
      for(i=0; i<k; i++) B[i]=A[--N].Id;
      break;
    case  4:                    /* Scaled-Selection             */
      for(i=0; i<k; i++)
      { m=random2(N); m2=random2(N);
        if ((A[m].Val>A[m2].Val)^flip(SclSel)) then m=m2;
        B[i]=A[m].Id;
        if (Mode>0) then AL[m]=AL[--N];
      } break;
    case  5:                    /* Random-Selection             */
      for(i=0; i<k; i++)
      { m=random2(N); B[i]=A[m].Id;
        if (Mode>0) then AL[m]=AL[--N];
      } break;
    case  6:                    /* Prob2-Selection (simple vers) */
      if (once) then m=0; else m=1;
      for(i=0; i<k; i++)
      { if (m==0) then ProbSelInit2(A,N);
        m=ProbSel2(A,N);
        B[i]=A[m].Id;
```

```
        if (Mode>0) then AL[m]=AL[--N];
      } break;
    default: Error(3);
  }
  *NP=N;
}
#define pprFlag(F,S)    \
        prf(LIGHTMAGENTA,"%s %s\r\n",S,((F)==TRUE)?"ON":"OFF")
#define SetSStep(F) pprFlag(SStepF=(F),"SingleStep")
#define SetSound(F) pprFlag(SoundF=(F),"Sound")

/*-----------------------------*/
        SetProt(F)              /* Toggle Protocoll On/Off      */
/*-----------------------------*/
bool    F;
{ if (F)
  then devP=fopen(ProtDev,"ab");
  else fclose(devP);
  pprFlag(ProtF=F,"Protocoll");
}
/*-----------------------------*/
        SetprMode(n)            /* Set Print-Mode to 0,1,2 or 3 */
/*-----------------------------*/
int     n;
{ prM1F= (n==1||n==2);
  prf(LIGHTMAGENTA,"PrintMode = %d\r\n",prMode=n);
}

/*****************************************************************/
/*             C F S - B a s i c - P r o c e d u r e s           */
/*****************************************************************/

/* Bits 0 to SStr-1 of M define 0 and 1 positions                */
/* Corresponding Bits in C and B define CF-Cond/Action pos.      */
/* (0,1)=0, (1,1)=1, (0,1)=(0,0)=#                               */

/* Does Message M match Condition (C,B) ?                        */
#define MatchMC(M,C,B) !((M^C)&B)  /* for doc. look further      */

/* Match Message M with Action (A,B) to result                   */
#define MatchMA(M,A,B) ((A&B)|(M&~B))

/*-----------------------------*/
        CalcSpec(C)             /* Calculate Specifity          */
/*-----------------------------*/
TCF     *C;
{ fixp  Sp1,Sp2;
  Sp1=Count1s((long)C->C1.B);
  if (!(C->C1.T&2)) then Sp1=SStr-Sp1;
ifnTC( C->Sp=FPDiv(Sp1,SStr); )
ifTC( Sp2=Count1s((long)C->C2.B);
      if (!(C->C2.T&2)) then Sp2=SStr-Sp2;
      C->Sp=FPDiv(Sp1+Sp2,SStr<<1); )
}
#define CrSumM(n) M=MP->B+n;                                    \
        if (M->H>0 || ModeP->ClaimMode==2) then                 \
```

```
        { M->H=-abs(M->H); /* Visited */                          \
          R= ModeP->SumMode==1 ? FixP(1) : -M->H;                 \
          Sum+= ModeP->DisMode==1 ? R : FPMult(R,M->I); }

/*------------------------------*/
long    CrSum(TS,N,MP,ModeP)    /* Calculate Credit-Sum         */
/*------------------------------*/
TTup    *TS;                    /* Start of Tupellist           */
int     N;                      /* # of Tupels to Check         */
TMessL  *MP;
TCFL    *ModeP;
{ TMess *M;
  TTup  *T;
  fixp  R;
  long  Sum=0;
  if (ModeP->SumMode==3) then return(FixP(1));
  for(T=TS; T<TS+N; T++)
  { CrSumM(T->M1); ifTC( CrSumM(T->M2); ) }
  return(Sum);
}
#define DistrM(n) M=MP->B+n;                                      \
        if (M->H<0 || ModeP->ClaimMode==2) then                   \
        { M->H=abs(M->H); /* Visited */                           \
          R= FPMult(M->H,Val);                                    \
          R= ModeP->DisMode==1 ? R : FPMult(R,M->I);              \
          Sum+=R; CP->B[M->CF].Get+=R; }

/*------------------------------*/
  Distr(TS,N,MP,ModeP,CP,Val)   /* Distribute Val. to CF        */
/*------------------------------*/
TTup    *TS;                    /* Start of Tupellist           */
int     N;                      /* # of Tupels to Check         */
TMessL  *MP;
TCFL    *ModeP,*CP;
fixp    Val;
{ TMess *M;
  TTup  *T;
  fixp  R,Sum=FixP(0);
  for(T=TS; T<TS+N; T++)
  { DistrM(T->M1); ifTC( DistrM(T->M2); ) }
  return(Sum);
}
/*------------------------------*/
        SetMFrac(MP)            /* Set Mess.-Credit-Fraction    */
/*------------------------------*/
TMessL  *MP;
{ TMess *M;
  MP->B[-1].H=NMFrac;
  for(M=MP->B; M<MP->C; M++)
    M->H= (M->CF==DetId) ? DetFrac : FixP(1);
}
/*------------------------------*/
TString RandStr(p)              /* Create Random-String         */
/*------------------------------*/
fixp p;                         /* p/256 = Prob. for a 1-Bit    */
{ TString s=1;
```

```
  while(!(s&~MMask))
  { s<<=1; if flip(p) then s|=1; }
  return(s&MMask);
}
/*-----------------------------*/
       RandMess(M)             /* Create Random-Message        */
/*-----------------------------*/
TMess   *M;
{ M->M=RandStr(FixP(0.5));
  M->I=DefNMI;
  M->CF=NonId;
}
/*-----------------------------*/
       RandCF(C)               /* Create Random-Classifier     */
/*-----------------------------*/
TCF     *C;                    /* Pointer to CF                */
{ C->C1.T=flip(NegCondP) ? 4 : 6 ;
  C->C1.B=RandStr(CondSp);
  C->C1.C= C->C1.B & RandStr(FixP(0.5));
ifTC( C->C2.T=flip(NegCondP) ? 4 : 6 ;
      C->C2.B=RandStr(CondSp);
      C->C2.C= C->C2.B & RandStr(FixP(0.5)); )
  C->Act.B=RandStr(ActSp);
  C->Act.A= C->Act.B & RandStr(FixP(0.5));
  C->S=DefStr+Noise(FPMult(DefStr,DefStrDev));
  C->Sp=CalcSpec(C);
  C->Get=C->Pay=C->Tax=C->PSP=FixP(0);
}
/*-----------------------------*/
       MutateCA(CP,BP,Sp)      /* Mutate Cond/Act              */
/*-----------------------------*/
TString *CP,*BP;               /* (*CP,*BP)= Cond/Act          */
fixp    Sp;                    /* Mean Specifity = 1- #Prob.   */
{ TString m;
  m=1<<random2(SStr);
  if flip(Sp) { *BP|=m;
    if flip(FixP(0.5)) then *CP|=m; else *CP&=~m; }
  else { *BP&=~m; *CP&=~m; }
}

/*****************************************************************/
/*              I n p u t   /   O u t p u t                      */
/*****************************************************************/

/*-----------------------------*/
char    *StoCB(S,PC,PB)        /* CharString to Condition      */
/*-----------------------------*/
char    *S;                    /* String like "01##10"         */
TString *PC,*PB;               /* Condition (*PC,*PB)          */
{ char  *T;
  *PB=*PC=0;
  for(T=S+SStr; S<T; S++)
  { *PB<<=1; *PC<<=1;
    switch(*S)
    { case '0': (*PB)++;          break;
      case '1': (*PB)++; (*PC)++; break;
```

```
      case '#':                    break;
      case '*':          (*PC)++; break;
      default : Error(5);
  } }
  return(T);                   /* Pointer behind string        */
}
/*-----------------------------*/
        StoCF(S,C,Str,Dev)     /* CharString to Classifier     */
/*-----------------------------*/
char    *S;                    /* String like "+01##/110#"     */
TCF     *C;                    /* Pointer to CF                */
fixp    Str,Dev;               /* Strength and Deviation       */
{ C->C1.T=4+'-'-*S++;
  S=StoCB(S,&C->C1.C,&C->C1.B);
ifTC( C->C2.T=4+'-'-*S++;
      S=StoCB(S,&C->C2.C,&C->C2.B); )
  S++;
  C->S=Str+Noise(FPMult(Str,Dev));
  C->Sp=CalcSpec(C);
  S=StoCB(S,&C->Act.A,&C->Act.B);
  C->Get=C->Pay=C->Tax=C->PSP=FixP(0);
  if (*S) then Error(5);
}
/*-----------------------------*/
        StoM(S,MP)             /* CharString to MessageString  */
/*-----------------------------*/
char    *S;                    /* String like "010011"         */
TMess   *MP;
{ TString B;
  StoCB(S,&MP->M,&B);
  if (MMask^B) then Error(5);
  MP->CF=DetId;
  MP->I=DefDetMI;
}
/*-----------------------------*/
char    *CBtoS(C,B)            /* Condition to CharString      */
/*-----------------------------*/
TString C,B;
{ static char S[SStr+1];
  int   i;
  S[SStr]=0;
  for(i=SStr-1; i>=0; i--)
  { S[i]= (B&1) ? (C&1?'1':'0') : (C&1?'*':'#');
    B>>=1; C>>=1; }
  return(S);
}
/*-----------------------------*/
        pprInfo()              /* Printf Help-Information      */
/*-----------------------------*/
{ prf(LIGHTCYAN,"%s\r\n%s\r\n",
  "Press one of the upcase letters in Menu-Field ...",
  "For more information refer to documentation     ");
}
/*-----------------------------*/
        pprCFL(CP)             /* pprint Classifier-List       */
/*-----------------------------*/
```

```
TCFL    *CP;
{ TCF   *C;
  int   i=0;
  prf(LIGHTGREEN,
  "CFId   Str    Get    Pay  Taxes Condion/Action\r\n");
  for(C=CP->B; C<CP->C; C++)
  { prf(0,"%3d:%6.2f %6.2f %6.2f %6.2f %c%s",
      i++,C->S/256.0,C->Get/256.0,C->Pay/256.0,C->Tax/256.0,
      '-'-(C->C1.T&2),CBtoS(C->C1.C,C->C1.B));
ifTC( prf(0,"%c%s",'-'-(C->C2.T&2),CBtoS(C->C2.C,C->C2.B)); )
    prf(0,"/%s\r\n",CBtoS(C->Act.A,C->Act.B));
} }
/*-----------------------------*/
        pprMessL(MP)            /* pprint Message-List          */
/*-----------------------------*/
TMessL  *MP;
{ TMess *M;
  int   i=0;
  prf(LIGHTGREEN,"MId Intens Sender Message\r\n");
  for(M=MP->B; M<MP->C; M++)
  { prf(0,"%3d:%6.2f ",i++,M->I/256.0);
    if (M->CF==DetId)
    then prf(0,"  D    ");
    else prf(0,"%3d    ",M->CF);
    prf(0,"%s\r\n",CBtoS(M->M,MMask));
} }
/*-----------------------------*/
        pprBidL(BP,TP)          /* pprint Bid-List              */
/*-----------------------------*/
TBidL   *BP;
TTupL   *TP;
{ TBid  *B;
  TTup  *T,*TS;
  int   i=0;
  prf(LIGHTGREEN,
    "BidId CFId Bid EBid Fired BidTupel\r\n");
  for(B=BP->B; B<BP->C; B++)
  { prf(0,"%3d: %3d %6.2f %6.2f %c {",
      i,B->CF,B->BV/256.0,B->EV/256.0,B->F?'F':'-');
    for(T=TS=TP->B+B->Tup; T<TS+B->N; T++)
    { if (T->M1==NonId) then prf(0," N");
      else prf(0," %d",T->M1);
      ifTC( if (T->M2==NonId) then prf(0,":N");
            else prf(0,":%d",T->M2); )
    }
    prf(0," }\r\n");
    i++;
} }
/*-----------------------------*/
        pprCS(prMode,o)         /* pprint whole CFS             */
/*-----------------------------*/
TCS     *o;
{ extern pprStat(int,TStat*);
  Display();
  switch(prMode) {
  case 3:
```

```
    prf(WHITE,
    "-------------- Cycle %ld --------------\r\n",CycleC);
    prf(LIGHTGREEN,"\r\nMessage-List:\r\n");
    pprMessL(o->MP);
    prf(LIGHTGREEN,"\r\nClassifier-List:\r\n");
    pprCFL(o->CP);
    prf(LIGHTGREEN,"\r\nBid-List:\r\n");
    pprBidL(o->BP,o->TP);
    prf(LIGHTGREEN,"\r\nNew-Message-List:\r\n");
    pprMessL(o->NP);
    if (CycleC%EffRate==0) then
    { prf(LIGHTGREEN,"\r\nEffector-List:\r\n");
      pprCFL(o->EP);
      prf(LIGHTGREEN,"\r\nEff-Bid-List:\r\n");
      pprBidL(o->DP,o->UP);
      prf(LIGHTGREEN,"\r\nOutput-Message-List:\r\n");
      pprMessL(o->OP);
    }
    prf(0,"\r\n");
    prM1F=TRUE;
  case 2:
    pprStat(1,&cy); break;
} }
/*-----------------------------*/
        pprStat(prMode,epp)     /* Print Statistic              */
/*-----------------------------*/
int     prMode;
TStat   *epp;
{ int   n         = (int)max(epp->CycleC,1);
  float ReVal     = epp->ReVal/(256.0*n);
  float Str       = epp->Str/(256.0*n*(o->CP->C-o->CP->B));
  int   NCFMinStr = (int)(epp->NCFMinStr/n);
  int   NCFMaxStr = (int)(epp->NCFMaxStr/n);
  int   MCF       = (int)(epp->MCF/n);
  int   MMess     = (int)(epp->MMess/n);
  int   NCF       = (int)(epp->NCF/n);
  int   NBid      = (int)(epp->NBid/n);
  int   NTup      = (int)(epp->NTup/n);
  int   NMess     = (int)(epp->NMess/n);

  switch(prMode) {
  case 2: pprCS(3,o); /* no break; */
  case 3: prf(LIGHTGREEN,
    "\r\n  Statistic (Mean-Values) of Cycle %ld-%ld:\r\n",
            CycleC-n+1,CycleC);
    prf(0,"  External Reinforcement  : %.3f\r\n",ReVal);
    prf(0,"  mean CF-Strength        : %.3f\r\n",Str);
    prf(0,"  min/maximal Strength CF : %d/%d\r\n",
                                       NCFMinStr,NCFMaxStr);
    prf(0,"  (matched) Messages      : (%d) %d\r\n",MMess,NMess);
    prf(0,"  (matched) Classifers    : (%d) %d\r\n",MCF,NCF);
    prf(0,"  Bids (-Tupels)          : %d (%d)\r\n",NBid,NTup);
    prf(0,"\r\n");
    prM1F=TRUE;
    break;
  case 1:
```

```
    if (prM1F) then
    { prf(LIGHTGREEN,"%s\r\n%s\r\n",
      "Cycle-Cycle  Reinf Strength Min Max Mch Mch Bid Tup",
      "             Value          Str Str  CF Mess 's els");
      prM1F=FALSE; }
    prf(0,"%5ld-%-5ld:%7.3f %7.3f %3d %3d %3d %3d %3d %3d\r\n",
      CycleC-n+1,CycleC,ReVal,Str,NCFMinStr,
      NCFMaxStr,MCF,MMess,NBid,NTup);
    break;
  case 0: break;
  default: Error(8);
} }

/*****************************************************************/
/*              C F S - H i g h - P r o c e d u r e s            */
/*****************************************************************/

/* Support of Tupel (m1,m2)  (used by MatchBid)                  */
#define TupSupp1(m1,m2) m1.Val                                    \
  ifTC(+ ((m1.Id==m2.Id && CP->SuppMode==1) ?  0 : m2.Val) )

/* Support of Tupel (m1,m2) (used by Fire)
*/
#define TupSupp2(m1,m2) (MP->B[m1].I                              \
  ifTC(+ ((m1==m2 && CP->SuppMode==1) ? 0 : MP->B[m2].I) ) )

/*------------------------------*/
       MatchBid(CP,MP,BP,TP)   /* Create BidList                 */
/*------------------------------*/
TCFL    *CP;
TMessL  *MP;
TBidL   *BP;
TTupL   *TP;
{ TMess *M;
  TIdVal M1[SMessL];
ifTC( TIdVal M2[SMessL];
      TIdVal MM[SMessL+1];
      long s1; long s2; )
  TIdVal F[STupL];
  index G[STupL];
  int   H[STupL];               /* Copy of A[].Val               */
  TString Str[SMessL];
  TString as;
  int   n1,n2=1,nn;
  TCF   *C;
  int   h,i,k,k2,N;
  bool  mf;                     /* Matching Flag                 */
  long  s,ss=0;                 /* Support                       */
  TBid  *B,*BPB=BP->C;          /* relevant beginning for this   */

for(M=MP->B; M<MP->C; M++) M->H=FALSE;          /* H=Matched     */

for(C=CP->B; C<CP->C; C++) /* for all CF do *//*** Match CF ***/

{ n1=0; ifTC( n2=nn=0; )
  for(i=0; i<MP->C-MP->B; i++) /* for all Mess do               */
```

```
  { mf=FALSE;
    if (MatchMC(MP->B[i].M,C->C1.C,C->C1.B)) then
    { M1[n1].Id=i; M1[n1++].Val=MP->B[i].I; mf=TRUE; }
ifTC( if (MatchMC(MP->B[i].M,C->C2.C,C->C2.B)) then
      { M2[n2].Id=i; M2[n2++].Val=MP->B[i].I; mf=TRUE; }
      if (mf) then
      { MM[nn].Id=i; MM[nn++].Val=MP->B[i].I; MP->B[i].H=TRUE; } )
  }
  if (!(C->C1.T&2)) then
  { if (n1) then n1=0;
    else { M1[n1].Id=NonId; M1[n1++].Val=DefNMI;
     ifTC( MM[nn].Id=NonId; M1[nn++].Val=DefNMI; ) } }
ifTC(
  if (!(C->C2.T&2)) then
  { if (n2) then n2=0;
    else
    { M2[n2].Id=NonId; M2[n2++].Val=DefNMI;
      if (MM[nn].Id!=NonId)
      then MM[nn].Id=NonId; MM[nn++].Val=DefNMI; }
  } )
  if (n1&&n2) then
  { cy.MCF++;                    /* for Statistic                */
    if (CP->BidMode>2) then
    { for(i=0; i<n1; i++)
        Str[M1[i].Id]=MatchMA(MP->B[M1[i].Id].M,C->Act.A,C->Act.B);
      GStr=Str;
      QSort(M1,M1+n1-1,InStrOrder);
      for(i=k=0; i<n1; k++)
      { as=Str[M1[i].Id];
        for(h=i,s=0; i<n1 && as==Str[M1[i].Id]; i++)
        { s+=M1[i].Val;          /* maybe max better             */
          if (M1[i].Val>M1[h].Val) then h=i;
        }
        M1[k].Id=M1[h].Id; M1[k].Val=(int)min(s,MaxInt);
      }
      n1=k;
ifTC( for(s=h=i=0; i<n2; i++)
      { s+=M2[i].Val;
        if (M2[i].Val>M2[h].Val) then h=i; }
      M2[0].Id=M2[h].Id;
      M2[0].Val=(int)min(s,MaxInt); n2=1; )
    }
/*** Bid CF ***/
    switch(CP->BidMode) {
    case 1: case 3:                /* BidMode = 1              */
      if (BP->C>=BP->E) then { Warning(5); goto Full; }
      BP->C->CF=C-CP->B;
ifTC( if (CP->SuppMode==1) then    /* SuppMode = 1 / 2         */
      { for(s=i=0; i<nn; i++) s+=MM[i].Val; }
      else
      { for(s1=i=0; i<n1; i++) s1+=M1[i].Val;
        for(s2=i=0; i<n2; i++) s2+=M2[i].Val;
        s=n2*s1+n1*s2; } )         /* =Sum over all Tuples */
ifnTC( for(s=i=0; i<n1; i++) s+=M1[i].Val; )
      *(long*)&BP->C->BV=s;        /* Supp in .BV & .EV        */
      ss+=s;
```

```
      BP->C->N=n1*n2;                    /* Est-Calc will follow */
      BP->C->Tup=TP->C-TP->B;
      for(i=0; i<n1; i++) ifTC( for(k=0; k<n2; k++) )
      { if (TP->C>=TP->E) then { Warning(4); goto Full; }
        TP->C->M1=M1[i].Id;
ifTC(   TP->C->M2=M2[k].Id; )
        TP->C++; }
      BP->C->F=FALSE;
      BP->C++;
      break;
    case 2: case 4:                      /* BidMode = 2          */
      for(i=h=0; i<n1; i++) ifTC( for(k=0; k<n2; k++) )
      { F[h].Id=h;
        F[h].Val=H[h]=TupSupp1(M1[i],M2[k]);
        if (++h>STupL) then { Warning(8); break; }
      }
      k=min(CP->MBids,h);
      if (k>(k2=TP->E-TP->C)) then { k=k2; Warning(4); }
      if (k>(k2=BP->E-BP->C)) then { k=k2; Warning(5); }
      Select(F,&h,G,k,CP->BidTupSelMode); /* normally h=n1*n2 */
      for(h=0; h<k; h++)
      { BP->C->CF=C-CP->B;
        BP->C->BV=H[G[h]];
        BP->C->N=1;
        BP->C->Tup=TP->C-TP->B;
        BP->C->F=FALSE;
        TP->C->M1=M1[G[h]/n2].Id;
ifTC(   TP->C->M2=M2[G[h]%n2].Id; )
        BP->C++; TP->C++; }
      break;
    default: Error(7);
} } }
Full:;
for(M=MP->B; M<MP->C; M++)             /* matching statistic   */
  if (M->H) then cy.MMess++;

/*** Calc BidVal ***/
for(B=BPB; B<BP->C; B++)               /* for all Bids do      */
{ if (CP->BidMode==1) then             /* BV is Supp           */
    B->BV=FPDiv(*(long*)&B->BV,ss);    /* BV is RelSupp / Est  */
  C=CP->B+B->CF;
  B->BV=FPMult(FPPow(CP->RiskFak,B->BV,CP->RelSuppPow),
               FPPow(C->S,C->Sp,CP->SpecPow));/* BV is BidValue */
  B->EV=B->BV+Noise(FPMult(B->BV,Dev2));      /* EV=BV+Noise    */
  B->EV=FPMult(FPPow(FixP(1),B->EV,CP->EBidPow),
                 FPPow(FixP(1),C->Sp,CP->ESpecPow));/* Eff-Bid  */
} }

/*-----------------------------*/
       Fire(CP,MP,BP,TP,NP)    /* Fire Messages                */
/*-----------------------------*/
TCFL    *CP;
TMessL  *MP;
TBidL   *BP;
TTupL   *TP;
TMessL  *NP;
```

```
{ TIdVal FB[SBidL],FT[STupL];
  index S,GT[STupL];
  int   i,k,NT;
  TCF   *C;
  TBid  *B;
  TTup  *T;
  int   N=BP->C-BP->B;
  bool  Stop=FALSE;
  int   r=NP->E-NP->C;

  for(C=CP->B; C<CP->C; C++) C->H=CP->MFire; /* Fire-Counter    */
  for(i=0; i<N; i++)
  { FB[i].Id=i;
    FB[i].Val=BP->B[i].EV; }
  MultSelInit(FB,N,CP->FireBidSelMode);

  while(r>0 && N!=0)
  { Select(FB,&N,&S,-1,CP->FireBidSelMode); /* -1 = multiple sel*/
    B=BP->B+S; C=CP->B+B->CF; NT=B->N; T=TP->B+B->Tup;
    k=min(NT,min(r,C->H));
    if (k>0) then
    { for(i=0; i<NT; i++)
      { FT[i].Id=B->Tup+i;
        FT[i].Val=TupSupp2(T->M1,T->M2); }
      Select(FT,&NT,GT,k,CP->FireTupSelMode);
      for(i=0; i<k; i++)
      { NP->C->M=MatchMA(MP->B[TP->B[GT[i]].M1].M,
                         C->Act.A,C->Act.B);
        NP->C->I=B->BV;
        NP->C->CF=B->CF;
        NP->C++; }
      C->H-=k; r-=k;
      B->F=TRUE;
    }
} }
/*------------------------------*/
        Clean(MP)               /* Clean Message-List           */
/*------------------------------*/
TMessL  *MP;                    /* MessList to Clean MP --> MP  */
{ TMess  H[SMessL],*HC=H;       /* should and will be empty     */
  TIdVal F[SMessL+1],*F1=F;
  index  G[SMessL];
  int   i;
  TString as;
  TMess *M;
  int   N=MP->C-MP->B;
  TIdVal *F2,*FE=F+N;

  if (N>=MEqMess) then
  { GMB=MP->B;
    for(i=0; i<N; i++)
    { F[i].Id=i; F[i].Val=GMB[i].I; }
    QSort(F,F+N-1,InMessOrder);

    while(F1<FE)
    { as=GMB[F1->Id].M;
```

```
      for(F2=F1+1; F2<FE && GMB[F2->Id].M==as; F2++);
      N=F2-F1;
      if (N>MEqMess) then
      { Select(F1,&N,G,MEqMess,DelMessSelMode);
        for(i=0; i<MEqMess; i++)
          memcpy(HC++,GMB+G[i],sizeof(TMess));
        F1=F2; }
      else
        for(; F1<F2; F1++)
          memcpy(HC++,GMB+F1->Id,sizeof(TMess));
    }
    memcpy(MP->B,H,(HC-H)*sizeof(TMess));
    MP->C=GMB+(HC-H);
  }
  if (CleanHallF) then
  { for(M=MP->B; M<MP->C;)
      if (MatchMC(M->M,DetMTagC,DetMTagB)) then
        memcpy(M,--MP->C,sizeof(TMess));
      else M++;
} }
/*------------------------------*/
Credit(CP,EP,MP,NP,BP,DP,TP,UP) /* Credit-Assignment            */
/*------------------------------*/
TCFL    *CP,*EP;
TMessL  *MP,*NP;
TBidL   *BP,*DP;
TTupL   *TP,*UP;
{ index BidN;
  fixp  Val;
  long  Sum;
  TCF   *C;
  TMess *M;
  TBid  *B;
  switch(CreditMode) {
  case 1:                       /* Explicit Bucket-Brigade      */
    for(C=CP->B-1; C<CP->C; C++) /* Test */
      C->Get=C->Pay=FixP(0);
  /* Set .H = +-Rate, - = Visited */
    SetMFrac(NP); SetMFrac(MP);
  /* Distribute Reinforcement */
    Sum=0;
    for(B=DP->B; B<DP->C; B++)
      if (B->F) Sum+=CrSum(UP->B+B->Tup,B->N,NP,EP);
    if (Sum) then
    { Val=FPDiv(ReVal,Sum);
      for(B=DP->B; B<DP->C; B++)
        if (B->F) Distr(UP->B+B->Tup,B->N,NP,EP,CP,Val);
    }
  /* Distribute Bids */
    for(B=BP->B; B<BP->C; B++) if (B->F) then
      if (B->N) then
      { Sum=CrSum(TP->B+B->Tup,B->N,MP,CP);
        if (Sum) then
        { Val=FPDiv(B->BV,Sum);
          CP->B[B->CF].Pay=Distr(TP->B+B->Tup,B->N,MP,CP,CP,Val);
      } }
```

```
    break;
  case 2: Error(8); break;      /* Implicit Bucket-Brigade      */
  case 3:                       /* Profit Sharing Plan          */
    for(B=BP->B; B<BP->C; B++) if (B->F) then
    { C=CP->B+B->CF;
      Val= (PSPDisMode<3) ? FixP(1) : B->BV;
      if (PSPDisMode&1)
      then C->PSP=max(C->PSP,Val);
      else if ((FixP(127)-C->PSP)<Val)
           then C->PSP=FixP(127);
           else C->PSP+=Val;
    }
    if (CycleC%ReRate==0) then
      for(C=CP->B; C<CP->C; C++)
      if (C->PSP) then
      { C->Pay=FPMult(PSPFrac,C->PSP);
        C->Get=FPMult(PSPFrac,ReVal);
        C->PSP=FixP(0); }
      else { C->Pay=C->Get=0; }
    break;
  case 4: /* no Credit-Asignment */
  default: Error(8);            /* No Credit Assignment         */}
}

/*------------------------------*/
        Taxes(CP,NP,BP)         /* Tax-Payment                  */
/*------------------------------*/
TCFL    *CP;
TMessL  *NP;
TBidL   *BP;
{ TCF   *C;
  TMess *M;
  TBid  *B;

  for(C=CP->B-1; C<CP->C; C++)
  { C->Tax=HeadTax; C->H=0; }

  for(M=NP->B; M<NP->C; M++)
  { C=CP->B+M->CF;
    if (!C->H || MultMessTaxF) then
    { C->Tax+=MessTax; C->H=1; }
  }
  for(B=BP->B; B<BP->C; B++)
  { C=CP->B+B->CF;
    if ((!(C->H&2) || MultBidTaxF) &&
        (!(C->H&1) || BidTaxMode==1)) then
    { C->Tax+=BidTax; C->H|=2; }
  }
  for(C=CP->B; C<CP->C; C++)
    C->Tax=FPMult(C->S,C->Tax);
}
/*------------------------------*/
        StrUpDate(CP)           /* Strength-Update (Test !!)    */
/*------------------------------*/
TCFL    *CP;
{ TCF   *C;
```

```
  for(C=CP->B; C<CP->C; C++)
    C->S=max(MinStr,min(MaxStr,
         C->S+C->Get-C->Pay-C->Tax));
}
/*-----------------------------*/
        Genetic(CP)             /* Genetic Operations          */
/*-----------------------------*/
TCFL    *CP;
{ TIdVal F[SCFL];
  index  G[SCFL];
  TCF   *C;
  int   i,k;
  int   N=CP->C-CP->B;

  if (prMode>1) then
    prf(LIGHTBLUE,
    "%ld.genetic algorithm envocation\r\n",CycleC/GenRate);

/* Replace CF (and Sender in MessL, not impl.) */
  for(i=0; i<N; i++)
  { F[i].Id=i;
    F[i].Val= CP->B[i].S ? FPDiv(FixP(0.49),CP->B[i].S) : FixP(127);
  }
  Select(F,&N,G,NCFRepl,ReplSelMode);
  for(i=0; i<NCFRepl; i++) RandCF(CP->B+G[i]);

/* Mutate CF */
  for(i=0; i<N; i++)
  { F[i].Id=i;
    F[i].Val= CP->B[i].S ? FPDiv(FixP(0.5),CP->B[i].S) : FixP(127);
  }
  Select(F,&N,G,NCFMut,MutSelMode);
  for(i=0; i<NCFMut; i++)
  { C=CP->B+G[i];
    for(k=0; k<NMutate; k++)
    { MutateCA(&C->C1.C,&C->C1.B,CondSp);
      ifTC( MutateCA(&C->C2.C,&C->C2.B,CondSp); )
      MutateCA(&C->Act.A,&C->Act.B,ActSp);
  } }
}
/*-----------------------------*/
        ReInit(o)               /* Re-Initialize               */
/*-----------------------------*/
TCS     *o;
{ TMessL *h;
  TMess  *M;
  memswap(o->MP,o->NP,sizeof(TMessL));
  o->NP->C=o->NP->B;
  o->OP->C=o->OP->B;
  o->BP->C=o->BP->B;
  o->TP->C=o->TP->B;

  if (DelMessMode) then
  { for(M=o->MP->B; M<o->MP->C;)
      if (DelMessMode==2 || MatchMC(M->M,EffMTagC,EffMTagB)) then
        memcpy(M,--o->MP->C,sizeof(TMess));
```

```
      else M++;
} }
/*-----------------------------*/
        InitStat(stp)           /* Initialize Statistic         */
/*-----------------------------*/
TStat   *stp;
{ memset(stp,0,sizeof(TStat));
}
/*-----------------------------*/
        UpdStat(o)              /* Update Statistic             */
/*-----------------------------*/
TCS     *o;
{ TCF   *C;
  int   i;
  cy.CycleC=1;
  cy.ReVal=ReVal;
  cy.NCFMinStr=cy.NCFMaxStr=cy.Str=0;
  for(C=o->CP->B; C<o->CP->C; C++)
  { cy.Str+=C->S;
    if (C->S==MinStr) then cy.NCFMinStr++;
    if (C->S==MaxStr) then cy.NCFMaxStr++; }
  /* cy.MCF and cy.MMess are updated in MatchBid() */
  cy.NCF  =o->CP->C-o->CP->B;
  cy.NBid =o->BP->C-o->BP->B;
  cy.NTup =o->TP->C-o->TP->B;
  cy.NMess=o->MP->C-o->MP->B;
  for(i=0; i<sizeof(TStat)/sizeof(long); i++)   /* TStat =      */
    ((long*)&ep)[i]+=((long*)&cy)[i];           /* long A[8]    */
}
/*-----------------------------*/
        MainStep(o)             /* Main Evaluation Cycle        */
/*-----------------------------*/
TCS     *o;
{ if (CycleC%StRate==0) then InitStat(&ep);
  CycleC++;  Detect(o->MP,MDetM);
  cy.MCF=cy.MMess=0;                          /* for Stat       */

  MatchBid(o->CP,o->MP,o->BP,o->TP);
  if (DelMessMode!=2) then o->NP->E-=MDetM; /* res. for DetMess */
  Fire(o->CP,o->MP,o->BP,o->TP,o->NP);
  if (DelMessMode!=2) then o->NP->E+=MDetM; /* free reserved "  */
  Clean(o->NP);

  if (CycleC%EffRate==0) then
  { o->DP->B=o->DP->C=o->BP->C; /* because DP=BP,UP=TP          */
    o->UP->B=o->UP->C=o->TP->C;
    MatchBid(o->EP,o->NP,o->DP,o->UP);
    Fire(o->EP,o->NP,o->DP,o->UP,o->OP);
    /* maybe create EMess, if OP is Empty */
    Effect(o->OP);
  }
  if (CycleC%ReRate==0)
  then ReVal=Reinf(o->OP);
  else ReVal=FixP(0);
  Credit(o->CP,o->EP,o->MP,o->NP,o->BP,o->DP,o->TP,o->UP);
  Taxes(o->CP,o->NP,o->BP);
```

```
  pprCS(prMode,o);
  StrUpDate(o->CP);
  if (CycleC%GenRate==0) then Genetic(o->CP);
  UpdStat(o);
  if (CycleC%StRate==0) then pprStat(prMode,&ep);
  if (SStepF && ((prMode>1)||(CycleC%StRate==0)))
  then  Menu(); else event();
  ReInit(o);
}

/*****************************************************************/
/*              M e n u - C o n t r o l                          */
/*****************************************************************/

/*-----------------------------*/
char    GetKey()                /* Get a Key                    */
/*-----------------------------*/
{ char c=getch();
  if (c==0) then c=128+getch();
}
/*-----------------------------*/
        Menu()                  /* Wait for Key/Menu-Selection  */
/*-----------------------------*/
{ char  c;
  extern chdo(char c);
  do { c=GetKey(); if (c!=ESC) chdo(c); }
  while(!strchr("Cc ",c));
  if (!SStepF) prf(LIGHTGREEN,"Simulation Continued\r\n");
}
/*-----------------------------*/
        prMenu()                /* Display Menu                 */
/*-----------------------------*/
{ window(1,1,80,1);
  textbackground(BLUE);
  clrscr();
  cprintf("Retart ESC Cont Break Abort Prot stEp ");
  cprintf("Help Display Sound 0123 rdZ Test\r");
  window(1,3,80,NScrL);
  textbackground(BLACK);
}
/*-----------------------------*/
        chdo(char c)            /* Branch on Menu-Selection c   */
/*-----------------------------*/
{ extern CFSMain(),CFSExit(),TestFunc();
  switch(toupper(c)) {
  case 'R': CFSExit(); longjmp(jmp,1);     /* ReStart CFS-Simul.*/
  case 'B': BreakPoint();                  /* Break to TC++     */
  case ESC: prf(LIGHTGREEN,                /* Interrupt Simul.  */
    "Simulation Interrupted in Cycle %ld\r\n",CycleC);
    Menu(); break;
  case 'C': case ' ': break;               /* Continue Simul.   */
  case 'A': CFSExit(); exit(0);            /* Abort to CFS      */
  case 'P': SetProt(!ProtF);  break;       /* Protocoll on/off  */
  case 'E': SetSStep(!SStepF);break;       /* SingleStep on/off */
  case 'H': case 187: pprInfo(); break;    /* Printf Help-Info  */
  case 'D': pprCS(max(prMode,2),o); break; /* Print Status      */
```

```
  case '0': case '1': case '2':            /* PrintMode 0123    */
  case '3': SetprMode(c-'0'); break;
  case 'S': SetSound(!SoundF); break;      /* Sound on/off      */
  case 'Z': randomize(); break;            /* randomize         */
  case 'T': TestFunc(); break;             /* Testfunction      */
  default : Sound(2500,10);                /* Wrong Key         */
} }
/*-----------------------------*/
char    event()                  /* look for KeyBoard-Event      */
/*-----------------------------*/
{ char  c;
  if (!kbhit()) then return(0);
  c=GetKey(); chdo(c); return(c);
}
/*****************************************************************/
/*               T e s t - P r o c e d u r e s                   */
/*****************************************************************/

/*-----------------------------*/
        TestFunc()               /* Testfunction                 */
/*-----------------------------*/
{ int i;
  prf(LIGHTCYAN,"This is a test ...\r\n");
  Error(0);
}

/*****************************************************************/
/*      I n i t - E x i t - M a i n - P r o c e d u r e s        */
/*****************************************************************/

#define oalloc(P,S)                                 \
        P->B=malloc((S+1)*sizeof(*P->B));           \
        if (P->B==NULL) then Error(6);              \
        P->B=P->C=P->B+1; P->E=P->B+S;

/*-----------------------------*/
        InitMessL(MP,S)          /* Init Mess-List               */
/*-----------------------------*/
TMessL  *MP;
int     S;
{ MP->B=malloc((S+1)*sizeof(TMess));
  if (MP->B==NULL) then Error(6);
  MP->B->M=0; MP->B->I=DefNMI; MP->B->CF=NonId;
  MP->B=MP->C=MP->B+1; MP->E=MP->B+S;
}
/*-----------------------------*/
        InitCFL(CP,S)            /* Init CF-List                 */
/*-----------------------------*/
TCFL    *CP;
int     S;
{ CP->B=malloc((S+1)*sizeof(TCF));
  if (CP->B==NULL) then Error(6);
  CP->B->S=FixP(1); /* ... */
  CP->B=CP->C=CP->B+1; CP->E=CP->B+S;
}
/*-----------------------------*/
```

```
            CFSInit()          /*      Initialisierung         */
/*-----------------------------*/
{ char  **S;
  char  T[SStr+1];

/* Other Initializations */
  memset(WC,0,sizeof(WC));
  memset(WP,0,sizeof(WP));
  srand(12345);

/* Create Arrays */
  InitMessL(o->MP,SMessL);
  InitMessL(o->NP,SMessL);
  InitMessL(o->OP,SOutL);
  InitCFL(o->CP,SCFL);
  InitCFL(o->EP,SEffL);
  oalloc(o->BP,SBidL);
  memcpy(o->DP,o->BP,sizeof(TBidL));
  oalloc(o->TP,STupL);
  memcpy(o->UP,o->TP,sizeof(TTupL));

/* Init Classifier & Effectors & Enviroment */
  for(S=CFSL; *S; S++)
    StoCF(*S,o->CP->C++,DefStr,DefStrDev);
  while(o->CP->C<o->CP->E) RandCF(o->CP->C++);
  for(S=EffSL;*S; S++)
    StoCF(*S,o->EP->C++,FixP(1),FixP(0));
  InitEnv();

/* Read Detector/Effector-Tags */
  memset(T,'#',SStr); T[SStr]='\0';
  memcpy(T,DetMessTag,min(strlen(DetMessTag),SStr));
  StoCB(T,&DetMTagC,&DetMTagB);
  memset(T,'#',SStr); T[SStr]='\0';
  memcpy(T,EffMessTag,min(strlen(EffMessTag),SStr));
  StoCB(T,&EffMTagC,&EffMTagB);

/* Open Devices */
  textmode(C4350); textcolor(YELLOW);
  devP=fopen(ProtDev,"ab");
  CycleC=0; ProtF=TRUE; SStepF=FALSE;
  textbackground(BLACK);
  clrscr(); prMenu();
  if (prMode>0) then { prf(LIGHTCYAN,
    "%s\r\n%s\r\n%s\r\n%s\r\n%s\r\n%s\r\n%s%s%s\r\n%s\r\n\n",
    "+-------------------------------------------------------+",
    "|  C l a s s i f i e r - S y s t e m - S i m u l a t o r  |",
    "|-------------------------------------------------------|",
    "|  Copyright by Marcus Hutter     01.04.91     Vers. 1.0  |",
    "|-------------------------------------------------------|",
    "|       Simulation Run ("  ,   ActTime()   ,   ")        |",
    "+-------------------------------------------------------+");
  }
  SetProt(FALSE);
  SetSStep(TRUE);
}
```

```
/*-----------------------------*/
            CFSMain()          /*      Main-Routinen           */
/*-----------------------------*/
{ prf(LIGHTGREEN,
  "Press <C> to continue ... or any other Menu-Key\r\n");
  Menu();
  FOREVER MainStep(o);
}
/*-----------------------------*/
            CFSExit()          /*      Exit-Routinen           */
/*-----------------------------*/
{ fclose(devP); fclose(lpt1);
  free(o->TP->B-1); free(o->BP->B-1);
  free(o->EP->B-1); free(o->CP->B-1);
  free(o->OP->B-1); free(o->NP->B-1); free(o->MP->B-1);
}
/*-----------------------------*/
            main()             /*      Main-Program            */
/*-----------------------------*/
{ setjmp(jmp);
  CFSInit();
  CFSMain();
  CFSExit();                   /* not needed */
  exit(0);                     /* not needed */
}
```

B Beispiellisting EXAMPLE1.C

```
/*****************************************************************/
/*      C l a s s i f i e r - S y s t e m - E x a m p l e   1    */
/*****************************************************************/
/*      T e s t    (c) by Marcus Hutter  01.04.1991              */
/*****************************************************************/

/* USER-Variables are Model-dependend                            */
/* user-Variables are Program-Specific                           */

/*-------------------------------*/
/*      Variables                */
/*-------------------------------*/

/* Array-Sizes and other Compiler-Information */
#define SMessL          1       /* USER: # of Messages          */
#define SOutL           1       /* USER: # of Mess prod. by Eff */
#define SCFL          100       /* USER: # of Classifiers       */
#define SEffL           1       /* USER: # of Effectors         */
#define SBidL         100       /* user: # of matching-mess     */
#define STupL         100       /* user: # of matching-tuples   */

#define SStr            6       /* USER: # of Bits per String   */
#define TwoCond      FALSE      /* USER: two Conditions (or one)*/
#define SpeedF        TRUE      /* user: Optimize for Speed     */

BreakPoint() {}         /* Set a BreakPoint in this Line !!     */

#include "D:\TC\MYPROGS\CFSSIM.C" /* Classifier-System-Sim.     */

/* General - Variables  */
int     StRate  =      100;     /* USER: Statistic-Activ-Rare   */
fixp    SclSel  =  FixP(2/3);   /* USER: Scale-Factor  50%-100% */
char ProtDev[30]="example1.lst";/* USER: Protocoll-Device       */
char    DetMessTag[] =  "#";    /* USER: {0,1,#}*               */
char    EffMessTag[] =  "##";   /* USER: {0,1,#}*               */

/* Detector - Variables */
int     DetRate  =      1;      /* USER: Det.-Sampling-Rate     */
int     MDetM    =      1;      /* USER: # of Det.Mess.         */
fixp    DefDetMI = FixP(1.0);   /* USER: Detector-Mess-Intens.  */
fixp    DefStr   = FixP(2.0);   /* USER: DefaultStrength 0..9   */
fixp    DefStrDev= FixP(0.1);   /* USER: DefaultStrDev   0..25% */

/* Matching - Variables */
fixp    DefNMI  =  FixP(1.0);   /* USER: Default NonMatchIntens */

/* Bid - Variables */
fixp    Dev2    =  FixP(0.0);   /* USER: EBid-Deviation  0..25% */

/* Clean - Variables */
int     MEqMess       = MaxInt; /* USER: # of Equal Mess.       */
sbyte   DelMessSelMode= 2;      /* USER: DelMode       -6..6    */
bool    CleanHallF    = FALSE;  /* USER: Clean Hallunc.Mess.    */
```

```
/* Effector - Variables */
int     EffRate       = 1;      /* USER: Eff.-Sampling-Rate     */

/* Credit - Variables   */
int     ReRate    =     1;      /* USER: Reinf.Act.Rate         */
byte    CreditMode=     1;      /* USER: Credit-Assign-Mode     */
fixp    NMFrac    =FixP(0.3);   /* USER: NonMatchCreditFrac.in% */
fixp    DetFrac   =FixP(0.5);   /* USER: DetectorCreditFrac.in% */
fixpPSPFrac   =FixP(0.1);       /* USER: PSP-Reduc.-Constant.in%*/
byte    PSPDisMode=     4;      /* USRR: PSP-Distr.-Mode 1..4   */

/* Taxes - Variables    */
fixp    HeadTax  = FixP(0.00);  /* USER: Head-Tax (CF-Tax)      */
fixp    MessTax  = FixP(1.00);  /* USER: Mess-Tax               */
fixp    BidTax   = FixP(0.00);  /* USER: Bid-Tax                */
byte    BidTaxMode   =  2;      /* USER: BidTaxMode      1..2   */
bool    MultBidTaxF  =  FALSE;  /* USER: Tax for each Bid       */
bool    MultMessTaxF =  FALSE;  /* USER: Tax for each Mess      */

/* StrUpDate - Variables*/
fixp    MaxStr  =   FixP(20.0); /* USER: Maximal Strength of CF */
fixp    MinStr  =   FixP( 0.0); /* USER: Minimal Strength of CF */

/* Genetic - Variables  */
int     GenRate   =     50;     /* USER: Genetic-Activ.-Rate    */
int     NCFRepl   =     50;     /* USER: # of CF to Replace     */
sbyte   ReplSelMode=    3;      /* USER: Replace-SelMode        */
int     NCFMut    =     0;      /* USER: # of CF to Mutate      */
int     NMutate   =     1;      /* USER: # of Mutations per CF  */
sbyte   MutSelMode=     3;      /* USER: Mutation-SelMode       */
fixp    CondSp    =FixP(0.50);  /* USER: Mean Cond-Specifity  % */
fixp    ActSp     =FixP(1.0);   /* USER: Mean Act.-Specifity  % */
fixp    NegCondP  =FixP(0.00);  /* USER: Prob. of neg. Cond   % */

/* ReInit - Variables   */
byte    DelMessMode =   2;      /* USER: Del.Mess-Mode   0..2   */

TCFL    XC      = { 0,0,0,      /* Classifier-Parameters        */
        2,                      /* USER: BidMode         1..4   */
        2,                      /* USER: SuppMode        1..2   */
        2,                      /* USER: BidTupSelMode  -6..6   */
        2,                      /* USER: BidActSelMode  -6..6   */
        3,                      /* USER: FireBidSelMode -6..6   */
        2,                      /* USER: FireTupSelMode -6..6   */
        5,                      /* USER: MBids      1..MaxInt   */
        MaxInt,                 /* USER: MAct       1..MaxInt   */
        MaxInt,                 /* USER: MFire      1..MaxInt   */
        FixP(1.0),              /* USER: RiskFak   FixP(0..1) % */
        1,                      /* USER: SpecPow         0..2   */
        1,                      /* USER: RelSuppPow      0..2   */
        1,                      /* USER: EBidPow         0..2   */
        0,                      /* USER: ESpecPow        0..2   */
        1,                      /* USER: ClaimMode       1..2   */
        1,                      /* USER: DisMode         1..5   */
        2 };                    /* USER: SumMode         1..3   */
```

```
TCFL    XE      = { 0,0,0,      /* Effector-Parameters          */
        2,                      /* USER: BidMode         1..4   */
        2,                      /* USER: SuppMode        1..2   */
        2,                      /* USER: BidTupSelMode  -6..6   */
        2,                      /* USER: BidActSelMode  -6..6   */
        2,                      /* USER: FireBidSelMode -6..6   */
        2,                      /* USER: FireTupSelMode -6..6   */
        5,                      /* USER: MBids      1..MaxInt   */
        MaxInt,                 /* USER: MAct       1..MaxInt   */
        1,                      /* Should be 1 (MFire)          */
        FixP(0.5),              /* USER: RiskFak   FixP(0..1) % */
        1,                      /* USER: SpecPow         0..2   */
        1,                      /* USER: RelSuppPow      0..2   */
        1,                      /* USER: EBidPow         0..2   */
        0,                      /* USER: ESpecPow        0..2   */
        1,                      /* USER: ClaimMode       1..2   */
        1,                      /* USER: DisMode         1..5   */
        2 };                    /* USER: SumMode         1..3   */


char    *MessSL[] =             /* Predefined Messages          */
      { /* "1111",
        "1000",
        "1001",
        "0000", */
        NULL };

char    *CFSL[] =               /* Predefined Classifiers       */
      { /* "+1###+#11#/01#1",
        "-0###-0###/111#",
        "+0###-0000/1###", */
        NULL };

char    *EffSL[] =              /* Predefined Effectors         */
      { "+######/00000#",
        NULL };

typedef struct                  /* Enviroment                   */
      { int     inp;
        int     out;
        bool    success;
        /*...*/ }
        TEnv;

TEnv    Env;                    /* Current Enviroment           */

/*-----------------------------*/
        InitEnv()               /* Init Enviroment              */
/*-----------------------------*/
{ memset(&Env,0,sizeof(Env));
}
/*-----------------------------*/
        Detect(MP,N)            /* Detect Messages              */
/*-----------------------------*/
TMessL  *MP;                    /* Message-List                 */
int     N;                      /* Max.# of Det.Mess            */
```

```
{ Env.inp=rand()&63;            /* 2 Addr. & 4 Inp. Bits        */
  Env.out=(Env.inp>>(5-(Env.inp&3)))&1;
  MP->C->M = Env.inp;
  MP->C->I = DefDetMI;
  MP->C->CF= DetId;
  MP->C++;
}
/*------------------------------*/
        Effect(OP)              /* Effect Messages              */
/*------------------------------*/
TMessL  *OP;
{/* Change Enviroment in dependence on MP */
 /* prf(0,"Output-Mess = Env.Change:\n");
  pprMessL(MP); */
}
/*------------------------------*/
fixp    Reinf(OP)               /* Calculate Reinforcement      */
/*------------------------------*/
TMessL  *OP;
{ Env.success = (OP->C>OP->B) && (Env.out==(OP->B->M));
  return(FixP((Env.success) ? 10 : 0 ));
}
/*------------------------------*/
        Display()               /* Display Information          */
/*------------------------------*/
{ if (prMode>1) then
    prf(LIGHTRED,"4to1Decoder: %s->%d %s\r\n",
      CBtoS(Env.inp,MMask),Env.out,
      (Env.success) ? "Success" : "Failure");
}

/********************** End of Example 1 ************************/
```

C Beispielprotokoll EXAMPLE1.LST

```
+-------------------------------------------------------+
|  C l a s s i f i e r - S y s t e m - S i m u l a t o r  |
|-------------------------------------------------------|
|  Copyright by Marcus Hutter     01.04.91     Vers. 1.0  |
|-------------------------------------------------------|
|       Simulation Run (Sun May 12 09:01:02 1991)        |
+-------------------------------------------------------+
Protocoll ON
4to1Decoder: 101011->0 Failure
-------------- Cycle 1 --------------

Message-List:
MId Intens Sender Message
  0:  1.00   D    101011

Classifier-List:
CFId   Str    Get    Pay  Taxes Condion/Action
  0:  1.71   0.00   0.00   0.00 +#0##0#/111011
  1:  1.90   0.00   0.00   0.00 +10#01#/000000
  2:  1.90   0.00   0.00   0.00 +#11#00/100100
  3:  2.00   0.00   0.00   0.00 +1101#1/110110
  4:  2.29   0.00   0.00   0.00 +##0#1#/000110
  5:  1.80   0.00   0.00   0.00 +##000#/001010
  6:  1.90   0.00   0.00   0.00 +0#1###/111111
  7:  2.39   0.00   0.00   0.00 +11####/010001
  8:  2.00   0.00   0.00   0.00 +0#10#0/010111
  9:  2.10   0.00   0.00   0.00 +##10#0/110001
 10:  2.20   0.00   0.00   0.00 +0#1101/011010
 11:  1.90   0.00   0.00   0.00 +#0#01#/100000
 12:  1.80   0.00   0.00   0.00 +#00#00/110101
 13:  2.00   0.00   0.00   0.00 +##100#/111111
 14:  2.00   0.00   0.00   0.00 +####1#/110000
 15:  2.00   0.00   0.00   0.00 +000##1/010100
 16:  1.80   0.00   0.00   0.00 +##0#10/100100
 17:  1.90   0.00   0.00   0.00 +###1#0/110111
 18:  2.10   0.00   0.00   0.00 +0###1#/001110
 19:  1.71   0.00   0.00   0.00 +1#1###/011011
 20:  1.80   0.00   0.00   0.00 +#01110/001101
 21:  1.90   0.00   0.00   0.00 +#01#1#/011100
 22:  1.51   0.00   0.00   0.00 +##0##1/111100
 23:  2.10   0.00   0.00   0.00 +110###/000011
 24:  2.10   0.00   0.00   0.00 +###1##/000010
 25:  2.10   0.00   0.00   0.00 +10##1#/010010
 26:  2.20   0.00   0.00   0.00 +1##11#/010110
 27:  1.90   0.00   0.00   0.00 +##1#11/100000
 28:  2.39   0.00   0.00   0.00 +###0##/000010
 29:  1.90   0.00   0.00   0.00 +##001#/011100
 30:  2.29   0.00   0.00   0.00 +0#0###/000101
 31:  2.10   0.00   0.00   0.00 +#0#001/100000
 32:  1.90   0.00   0.00   0.00 +#0##0#/001101
 33:  1.90   0.00   0.00   0.00 +#0#100/100111
 34:  1.90   0.00   0.00   0.00 +101#10/001100
 35:  1.71   0.00   0.00   0.00 +######/011000
 36:  2.10   0.00   0.00   0.00 +0#10#0/111010
```

```
37:  1.90   0.00   0.00   0.00 +100#1#/010100
38:  1.90   0.00   0.00   0.00 +##1#1#/011110
39:  1.71   0.00   0.00   0.00 +0#####/000111
40:  2.20   0.00   0.00   0.00 +10#00#/001010
41:  2.10   0.00   1.74   2.10 +10101#/100011
42:  2.20   0.00   0.00   0.00 +###01#/111010
43:  2.00   0.00   0.00   0.00 +1#####/000011
44:  2.20   0.00   0.00   0.00 +###0#0/011001
45:  2.00   0.00   0.00   0.00 +0#00##/010111
46:  2.10   0.00   0.00   0.00 +10####/110001
47:  1.90   0.00   0.00   0.00 +10####/001011
48:  1.32   0.00   0.00   0.00 +#0###1/000001
49:  1.90   0.00   0.00   0.00 +0000##/001101
50:  1.80   0.00   0.00   0.00 +#11###/000011
51:  2.10   0.00   0.00   0.00 +10#10#/011100
52:  2.00   0.00   0.00   0.00 +##010#/010111
53:  1.80   0.00   0.00   0.00 +#00##1/100100
54:  1.90   0.00   0.00   0.00 +1#1011/111001
55:  2.10   0.00   0.00   0.00 +00##1#/000110
56:  1.90   0.00   0.00   0.00 +1#1111/111111
57:  2.20   0.00   0.00   0.00 +10##0#/011111
58:  2.00   0.00   0.00   0.00 +0#11##/001100
59:  1.71   0.00   0.00   0.00 +#0#1#0/001010
60:  2.39   0.00   0.00   0.00 +####10/010111
61:  2.00   0.00   0.00   0.00 +11#0##/101000
62:  2.10   0.00   0.00   0.00 +0#00##/001100
63:  2.39   0.00   0.00   0.00 +##001#/101010
64:  1.71   0.00   0.00   0.00 +1##1#1/100000
65:  2.00   0.00   0.00   0.00 +11##01/001001
66:  2.29   0.00   0.00   0.00 +#10110/010011
67:  2.00   0.00   0.00   0.00 +1#11##/010000
68:  2.00   0.00   0.00   0.00 +10001#/011000
69:  2.10   0.00   0.00   0.00 +#11###/100100
70:  2.20   0.00   0.00   0.00 +##0001/111010
71:  2.00   0.00   0.00   0.00 +1##111/010111
72:  1.80   0.00   0.00   0.00 +###0##/001100
73:  2.00   0.00   0.00   0.00 +0##1#1/100010
74:  2.10   0.00   0.00   0.00 +0####0/101110
75:  2.00   0.00   0.00   0.00 +#110##/000010
76:  1.90   0.00   0.00   0.00 +#0##11/101101
77:  2.10   0.00   0.00   0.00 +##01#1/100101
78:  1.90   0.00   0.00   0.00 +1#####/100010
79:  1.90   0.00   0.00   0.00 +0#00#1/010001
80:  2.00   0.00   0.00   0.00 +##1##1/111110
81:  2.20   0.00   0.00   0.00 +#101#1/000101
82:  1.90   0.00   0.00   0.00 +00100#/100101
83:  2.10   0.00   0.00   0.00 +##0111/010100
84:  1.90   0.00   0.00   0.00 +00####/001010
85:  2.10   0.00   0.00   0.00 +1111#1/011001
86:  2.10   0.00   0.00   0.00 +#001#1/001000
87:  2.10   0.00   0.00   0.00 +1##1#0/100010
88:  2.39   0.00   0.00   0.00 +###0#0/110000
89:  1.90   0.00   0.00   0.00 +1###0#/001111
90:  2.00   0.00   0.00   0.00 +#0###1/001000
91:  1.80   0.00   0.00   0.00 +1####1/110100
92:  2.10   0.00   0.00   0.00 +##11#0/110100
```

```
 93:  1.71   0.00   0.00   0.00 +0###1#/011011
 94:  2.00   0.00   0.00   0.00 +100#11/000001
 95:  1.41   0.00   0.00   0.00 +###10#/110111
 96:  2.20   0.00   0.00   0.00 +##11##/111101
 97:  2.00   0.00   0.00   0.00 +1#0###/101010
 98:  1.90   0.00   0.00   0.00 +##01#0/001000
 99:  2.10   0.00   0.00   0.00 +#0#10#/000111

Bid-List:
BidId CFId Bid EBid Fired BidTupel
  0:   1   1.26   1.26 - { 0 }
  1:  11   0.95   0.95 - { 0 }
  2:  14   0.33   0.33 - { 0 }
  3:  19   0.57   0.57 - { 0 }
  4:  21   0.95   0.95 - { 0 }
  5:  25   1.05   1.05 - { 0 }
  6:  27   0.95   0.95 - { 0 }
  7:  28   0.39   0.39 - { 0 }
  8:  35   0.00   0.00 - { 0 }
  9:  38   0.63   0.63 - { 0 }
 10:  41   1.74   1.74 F { 0 }
 11:  42   0.73   0.73 - { 0 }
 12:  43   0.33   0.33 - { 0 }
 13:  46   0.70   0.70 - { 0 }
 14:  47   0.63   0.63 - { 0 }
 15:  48   0.43   0.43 - { 0 }
 16:  54   1.58   1.58 - { 0 }
 17:  72   0.29   0.29 - { 0 }
 18:  76   0.95   0.95 - { 0 }
 19:  78   0.31   0.31 - { 0 }
 20:  80   0.66   0.66 - { 0 }
 21:  90   0.66   0.66 - { 0 }
 22:  91   0.60   0.60 - { 0 }

New-Message-List:
MId Intens Sender Message
  0:  1.74  41    100011

Effector-List:
CFId   Str    Get    Pay  Taxes Condion/Action
  0:  1.00   0.00   0.00   0.00 +######/00000#

Eff-Bid-List:
BidId CFId Bid EBid Fired BidTupel
  0:   0   0.00   0.00 F { 0 }

Output-Message-List:
MId Intens Sender Message
  0:  0.00   0    000001

Cycle-Cycle  Reinf Strength Min Max Mch Mch Bid Tup
             Value          Str Str  CF Mess 's els
    1-1    :  0.000   0.000   0   0  24   0   0   0
PrintMode = 2
4to1Decoder: 011101->1 Failure
Cycle-Cycle  Reinf Strength Min Max Mch Mch Bid Tup
```

```
             Value          Str Str  CF Mess 's els
    2-2    :  0.000   1.968   1   0  13   0  23  23
4to1Decoder: 011111->1 Failure
    3-3    :  0.000   1.946   2   0  16   0  12  12
4to1Decoder: 111010->1 Success
    4-4    :  0.000   1.926   3   0  19   0  15  15
4to1Decoder: 000110->0 Success
    5-5    : 10.000   1.995   3   0  17   0  18  18
4to1Decoder: 011000->0 Failure
    6-6    : 10.000   2.063   3   0  17   0  16  16

PrintMode = 1
SingleStep OFF
Simulation Continued
Cycle-Cycle  Reinf Strength Min Max Mch Mch Bid Tup
             Value          Str Str  CF Mess 's els
    1-100  :  6.400   2.268   8   0  18   0  17  17
  101-200  :  7.800   2.525   5   0  19   0  18  18
  201-300  :  8.900   2.718   2   0  18   0  17  17
  301-400  :  8.900   2.887   2   0  19   0  18  18
  401-500  :  8.700   2.851   3   0  17   0  16  16
  501-600  :  8.600   2.792   4   0  18   0  17  17
  601-700  :  9.300   2.926   1   0  19   0  18  18
  701-800  :  9.600   2.999   1   0  17   0  16  16
  801-900  :  9.500   2.991   1   0  17   0  16  16
  901-1000 :  9.700   3.042   0   0  18   0  17  17
 1001-1100 :  9.300   3.101   1   0  17   0  16  16
 1101-1200 :  9.800   3.156   0   0  16   0  15  15
 1201-1300 :  9.600   3.100   0   0  16   0  15  15
 1301-1400 :  9.600   3.094   1   0  17   0  16  16
 1401-1500 :  9.900   3.145   0   0  17   0  16  16
 1501-1600 :  9.900   3.165   0   0  18   0  17  17
 1601-1700 :  9.900   3.174   0   0  17   0  16  16
 1701-1800 :  9.900   3.163   0   0  16   0  15  15
 1801-1900 : 10.000   3.204   0   0  17   0  16  16
 1901-2000 : 10.000   3.207   0   0  16   0  15  15
 2001-2100 : 10.000   3.195   0   0  16   0  15  15
Simulation Interrupted in Cycle 2151
PrintMode = 3
4to1Decoder: 111100->1 Success
-------------- Cycle 2151 --------------

Message-List:
MId Intens Sender Message
  0:  1.00   D    111100

Classifier-List:
CFId   Str    Get    Pay  Taxes Condion/Action
  0:  6.01   0.00   0.00   0.00 +0#1#00/110010
  1:  5.45   0.00   0.00   0.00 +011#01/000001
  2:  6.01   0.00   0.00   0.00 +0#1#10/011001
  3:  2.59   0.00   0.00   0.00 +110#11/111010
  4:  1.80   0.00   0.00   0.00 +11####/101011
  5:  2.49   0.00   0.00   0.00 +##11##/101111
  6:  2.10   0.00   0.00   0.00 +01#10#/101010
  7:  1.80   0.00   0.00   0.00 +##10##/110100
```

```
  8:  2.10   0.00   0.00   0.00 +1####0/000010
  9:  1.80   0.00   0.00   0.00 +000###/011111
 10:  2.39   0.00   0.00   0.00 +0#10#0/011101
 11:  1.90   0.00   0.00   0.00 +###0##/011111
 12:  1.80   0.00   0.00   0.00 +1##000/101011
 13:  6.67   0.00   0.00   0.00 +#1#10#/110111
 14:  2.20   0.00   0.00   0.00 +#0#1##/011011
 15:  1.90   0.00   0.00   0.00 +#01#00/110111
 16:  1.80   0.00   0.00   0.00 +##00#0/011101
 17:  2.49   0.00   0.00   0.00 +#0###1/111101
 18:  1.80   0.00   0.00   0.00 +#####0/011111
 19:  6.01   0.00   0.00   0.00 +1##000/101001
 20:  2.00   0.00   0.00   0.00 +0#10##/110100
 21:  2.39   0.00   0.00   0.00 +#101#1/101000
 22:  2.59   0.00   0.00   0.00 +###011/000010
 23:  5.46   0.00   0.00   0.00 +10#100/110011
 24:  1.71   0.00   0.00   0.00 +##10##/101101
 25:  1.51   0.00   0.00   0.00 +11##1#/111010
 26:  6.67   0.00   0.00   0.00 +###111/011101
 27:  2.10   0.00   0.00   0.00 +0##001/111101
 28:  2.00   0.00   0.00   0.00 +#00#1#/100111
 29:  2.49   0.00   0.00   0.00 +01#0#1/101110
 30:  2.29   0.00   0.00   0.00 +0###1#/101001
 31:  6.01   0.00   0.00   0.00 +#0#001/100000
 32:  3.13   0.00   0.00   0.00 +010#10/011100
 33:  6.01   0.00   0.00   0.00 +1#0#10/010010
 34:  2.00   0.00   0.00   0.00 +####1#/101000
 35:  1.90   0.00   0.00   0.00 +#1#01#/101110
 36:  2.29   0.00   0.00   0.00 +#11##0/000110
 37:  2.10   0.00   0.00   0.00 +0##100/011111
 38:  2.49   0.00   0.00   0.00 +###1##/011101
 39:  1.80   0.00   0.00   0.00 +0#1100/010001
 40:  2.10   0.00   0.00   0.00 +010#1#/101101
 41:  2.00   0.00   0.00   0.00 +00#0##/101000
 42:  1.71   0.00   0.00   0.00 +###1#1/011001
 43:  1.90   0.00   0.00   0.00 +1#0##1/010111
 44:  2.49   0.00   0.00   0.00 +#11###/111011
 45:  2.20   0.00   0.00   0.00 +#00010/110001
 46:  1.80   0.00   0.00   0.00 +0##1#0/010110
 47:  1.90   0.00   0.00   0.00 +#1#1#1/001101
 48:  1.90   0.00   0.00   0.00 +##1###/101001
 49:  2.49   0.00   0.00   0.00 +0##0#0/111000
 50:  6.01   0.00   0.00   0.00 +10##01/100010
 51:  2.20   0.00   0.00   0.00 +1###1#/010011
 52:  2.10   0.00   0.00   0.00 +110##1/001110
 53:  1.90   0.00   0.00   0.00 +##1000/110000
 54:  2.10   0.00   0.00   0.00 +0#0#1#/110100
 55:  5.45   0.00   0.00   0.00 +01#000/101100
 56:  2.10   0.00   0.00   0.00 +###1#1/000111
 57:  6.01   0.00   0.00   0.00 +00#00#/111110
 58:  2.20   0.00   0.00   0.00 +#0####/010010
 59:  1.71   0.00   0.00   0.00 +0##10#/100111
 60:  7.51   0.00   0.00   0.00 +####10/010111
 61:  2.10   0.00   0.00   0.00 +1#01##/000010
 62:  2.49   0.00   0.00   0.00 +##1000/111110
 63:  6.67   0.00   0.00   0.00 +##001#/101010
```

```
 64:  2.49   0.00   0.00   0.00 +#10#0#/110101
 65:  6.01   0.00   0.00   0.00 +11##01/001001
 66:  2.00   0.00   0.00   0.00 +###100/010000
 67:  2.59   0.00   0.00   0.00 +##1###/111011
 68:  3.07   0.00   0.00   0.00 +10001#/011000
 69:  6.67   0.00   0.00   0.00 +##10#1/010110
 70:  2.00   0.00   0.00   0.00 +0###0#/111011
 71:  2.39   0.00   0.00   0.00 +#1#1#0/100001
 72:  2.59   0.00   0.00   0.00 +1##111/111100
 73:  2.49   0.00   0.00   0.00 +0#1##1/100100
 74:  2.49   0.00   0.00   0.00 +1#110#/001110
 75:  2.00   0.00   0.00   0.00 +10#1#0/110000
 76:  2.59   0.00   0.00   0.00 +0####1/010001
 77:  2.49   0.00   0.00   0.00 +011#01/001101
 78:  2.29   0.00   0.00   0.00 +1####0/000110
 79:  2.49   0.00   0.00   0.00 +###0##/010010
 80:  2.49   0.00   0.00   0.00 +010100/000101
 81:  6.01   0.00   0.00   0.00 +#101#1/000101
 82:  8.59   0.00   0.00   0.00 +0#####/111010
 83:  2.59   0.00   0.00   0.00 +##1#1#/100101
 84:  2.10   0.00   0.00   0.00 +#111#1/011011
 85:  5.45   0.00   0.00   0.00 +1111#1/011001
 86:  2.39   0.00   0.00   0.00 +##0##0/110001
 87:  7.51   0.00   0.00   0.00 +##00##/001011
 88:  2.39   0.00   0.00   0.00 +###0##/101110
 89:  2.10   0.00   0.00   0.00 +001100/011111
 90:  5.50   0.00   0.00   0.00 +00#101/100100
 91:  6.67  10.00   3.33   6.67 +#11##0/110111
 92:  1.90   0.00   0.00   0.00 +##1111/110011
 93:  2.00   0.00   0.00   0.00 +10###0/100111
 94:  2.29   0.00   0.00   0.00 +####01/010110
 95:  6.67   0.00   0.00   0.00 +##1#01/110100
 96:  6.01   0.00   0.00   0.00 +##0110/010110
 97:  6.01   0.00   0.00   0.00 +0##100/001100
 98:  1.80   0.00   0.00   0.00 +######/000001
 99:  2.49   0.00   0.00   0.00 +1##011/001110

Bid-List:
BidId CFId Bid EBid Fired BidTupel
  0:   4   0.60   0.60 - { 0 }
  1:   5   0.82   0.82 - { 0 }
  2:   8   0.70   0.70 - { 0 }
  3:  13   3.33   3.33 - { 0 }
  4:  18   0.29   0.29 - { 0 }
  5:  36   1.14   1.14 - { 0 }
  6:  38   0.41   0.41 - { 0 }
  7:  44   0.82   0.82 - { 0 }
  8:  48   0.31   0.31 - { 0 }
  9:  66   1.00   1.00 - { 0 }
 10:  67   0.42   0.42 - { 0 }
 11:  71   1.20   1.20 - { 0 }
 12:  74   1.65   1.65 - { 0 }
 13:  78   0.76   0.76 - { 0 }
 14:  91   3.33   3.33 F { 0 }
 15:  98   0.00   0.00 - { 0 }
```

```
New-Message-List:
MId Intens Sender Message
  0:  3.33  91    110111

Effector-List:
CFId   Str    Get    Pay  Taxes Condion/Action
  0:  1.00   0.00   0.00   0.00 +######/00000#

Eff-Bid-List:
BidId CFId Bid EBid Fired BidTupel
  0:   0   0.00   0.00 F { 0 }

Output-Message-List:
MId Intens Sender Message
  0:  0.00   0    000001

Cycle-Cycle  Reinf Strength Min Max Mch Mch Bid Tup
             Value          Str Str  CF Mess 's els
 2151-2151 : 10.000   3.207   0   0  17   0  16  16
```

Literatur

[1] **Holland J.H.**: *Escaping Brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems*; Machine Learning Vol 2 Kaufmann Publ., 1986

[2] **Booker L.B., Goldberg D.E., Holland J.H.**: *Classifier systems and genetic algorithms*; Artificial Intelligence 40,235-282, 1989

[3] *Proceedings of the third international conference on genetic algorithms*; Kaufmann Publ., 1989

[4] **Goldberg D.E.**: *Genetic algorithms in search, optimization machine learning*; Addison-Wesley, 1989

[5] **Wilson S.W.**: *Classifier systems and the animat problem*; Kluwer Academic Publishers, Bosten, 1987

[6] **Weiss G.**: *Benutzerbeschreibung zum Klassifizierungssystem* TU-München, 1991