

Instant (FLA, HOB) Computational Management System KBO Model Design

Fevzi Ünlü

Department of Mathematics, Faculty of Science
Yasar University, Izmir, Turkey
e-Mail: fevzi.unlu@yasar.edu.tr; fevziunlu@ttnet.net.tr

Abstract

KBO is new concept developed by Ünlü[1]. Us-Crop is another new concept developed by Ünlü [2, 3, 4, 5, 6, 7, 8]. The notion of FLA-HOB developed by Ünlü[9]. The background information can be found in Pritsker [10], Pratt[11], Denning[12] and Lewis[13]. The first aim of this paper is to develop a FLA cluster machine type as a set of machines that obtained out of a machine KBO type of a given us-crop and HOB cluster language type as a set of formal languages that obtained out of a language KBO type of an other given us crop, such a way that each HOB[i] in HOB = HOB@HOB has a power (or capacity) to communicate with at least one machine FLA[i] in FLA= FLA@FLA in a given instance of time. The second aim is generating a simulation model of a formal computational management system KBO as a FLAHOB = (FLA, HOB) = {(FLA[i], HOB[i]): FLA[i] \in M, HOB[i] \in L}. Where M is a set of Finite State Machines and L is a set of Programming Languages that in a given instant of time some FLA[i] can be generated to be programmable by some HOB[i] in that instance of time for carrying out a KBO type processing. FLA is a brief vocabulary that stands for “Flower-board of Logic Array,” FLA@FLA is an icon stands for “FLA in/at FLA,” HOB is a brief vocabulary that stands for “Honey Bee of human thought,” and HOB@HOB is an icon stands for “HOB in/at HOB.”

Mathematics Subject Classification: 05B40, 52C17, 94C99, 11H71, 62K05, 68M10

Keywords: KBO, us-crope, FLA, HOB, FLAHOB, formal language KBo, state machine KBO,

1 Introduction

Simulation is a well known technique to represent the dynamic behavior of a (deterministic or probabilistic) virtual machine management system KBO type clusters by moving its members(or components) from one specified state to another in time under a set of rules of operations. Simulation model is also a set of formal (iconic, abstract, and visual) description for representing the dynamic behavior of formal computational management system KBO type clusters [1, 2, 3]. Finite state machine KBO types and formal language KBO types are two important abstract notions in the computer science and they should sometimes be studied as a pair of formal structures being paralleled each other in the same content of a given virtually structured cluster KBO type for creating a formal optimal computation management system KBO type [3, 4]. This presentation has, therefore, been designed for the people who wish to practice the basics of this pair of mentioned abstract notions of finite state machine KBO's and their formal language KBO's while they are on the work of information processing and use them in the problem solving under the name FLAHOB[k] = (FLA, HOB) = { (FLA[i], HOB[i]) = {(FLA[0] or FLA[i-1]@ (FLA[i], HOB[0] or HOB[i-1]@ HOB[i]) : i, k ∈ N, i ← k} programming. Where N is the set of natural numbers, ← is the assignment operator, "FLA" is a brief vocabulary for "Flower-board of Logic Array". As a result FLA represents a set of formal abstract machine KBO's. They were organized as/under an integrated set of algorithms and data structures capable of storing and executing programs in a set of formal languages that we call it as HOB. "HOB" is a brief vocabulary for "Honey Bee of human thought". It represents a set of formal languages to communicate with FLA. The element of FLA and HOB will be represented by FLA[i] and HOB[i], (i = ..., -3,-2, -1, -0, +0, +1, +2, +3, ...) in the pair. The pair of (FLA[i], HOB[i]) for an arbitrary k of i ← k is taken an arbitrary simulation model for a virtually generated formal computational Management system KBO type k, k ∈ N. In order to be understood, we think FLAHOB needs to be designed with a great simplicity by assigning each finite state machine KBO and its special formal language KBO some special properties in this paper. FLAHOB will be simulated later in a high-level programming language as a (FLA[k], HOB[k]) KBO model designer.

2 Instant FLA[k] Machine Model Design

Fundamental Theorem of KBO Processing: Let M be a polarized us-crop with finite state machine type spikes and L be a polarized us-crop with finite formal language type spikes. There is at least one polarized us-crop with FLAHOB = {(FLA[i], HOB[i]): FLA[i] ∈ M and HOB[i] ∈ L, i ← k ∈ N, for

an arbitrary k type KBO spikes that are designed for carrying out KBO type processing.

Proof In this section, we define $FLAHOB = \{(FLA[i], HOB[i]): FLA[i] \in M \text{ and } HOB[i] \in L, i \leftarrow k \in N, \text{ for an arbitrary } k\}$ that carry out KBO type processing. Let us consider that, as we mentioned in the introduction, a machine is an integrated set of algorithm and data structure capable of storing and executing programs in a formal language. For the sake of our purpose, let us further consider that $FLA[k]$ is a simple sequential state machine cluster KBO that one can observe it in the state k and it has eight submachine KBO units. Let us name them as ON/OFF machine KBO unit $M1[k]$, CONTROL machine KBO unit $M2[k]$, INPUT machine KBO unit $M3[k]$, PROCESSING machine KBO unit $M4[k]$ and OUTPUT machine KBO unit $M5[k]$, only one word screen (or Key-Word register) KBO unit $M6[k]$, one ACCUMULATOR(or operation register) KBO unit $M7[k]$ and one word memory KBO unit that it is called as LOCATION KBO unit $M8[k]$. The pictorial organization of $FLA[k]$ KBO is given in Figure 1.1 as a cluster of eight finite state machines.

We point out that the logic design and hardware realization of a $(FLA[k], HOB[k])$ KBO can be carried out in a real or virtual laboratory or work-shop, and its software realization can be created in a high level language program laboratory. In either hardware or software realizations of $FLA[k]$ KBO, the users can be provided with a simple machine such that they may have complete control over the entire $FLA[k]$ KBO. The programmer writes a program on a piece of paper and executes this program instruction by instruction, in case of hardware realization from operator keys or in case of software realization from operator files. An important aspect of $FLA[k]$ KBO environment is that it has a hand-on interactive processing nature on data type of KBO's, the programmer himself can be like an operator of the computational management system KBO. This approach has some definite advantages, it provides maximum flexibility to the user, and user can control the use of a machine $FLA[k]$ KBO in whatever manner he desires by an algorithms in $HOB[k]$ KBO.

In order to complete the $(FLA[k], HOB[k])$ KBO as an abstract machine and abstract language of an instant computational management system KBO design in the state k , we will introduce submachine KBO units in $FLA[k]$ as a sequential state machine cluster. Observe the following that we are going to design only ON/ OFF machine KBO unit as $M1[k]$, CONTROL machine KBO unit as $M2[k]$, INPUT machine KBO unit as $M3[k]$, PROCESSING machine KBO unit as $M4[k]$ and OUTPUT machine KBO unit as $M5[k]$. We may assume that KEY BOARD REGISTER unit (KBD or $M6[k]$), CCUMULATOR (ACC or $M7[k]$) and LOCATION (LOC or $M8[k]$) designs are already exist and given previously to us.

2.1 Instant Virtual ON/OFF Machine KBO Unit M1[k] Design

Let $M1[k]$ be a Moore type of sequential state machine KBO. One can represent it by a quintuple $M1[k] = (S1[k], I1[k], Ol[k], a1[k], b1[k])$, where

- (i) $S1[k] = \{ S11[k], S12[k] \}$ is the set of transition states;
- (ii) $I1[k] = \{ 1, 2 \}$ is the set of inputs;
- (iii) $Ol[k] = \{ OFF, ON \}$ is the set of outputs;
- (iv) $a1[k]: S1[k] \times I1[k] \longrightarrow S1[k]$ is the state transition(or the next state) function;
- (v) $b1[k]: S[k] \longrightarrow Ol[k]$ is the output function.

Hence, $M1[k]$ representation by a transition table can be given as in Table 1.1.

The rules of definition for the state transition (or the next state) function $a1[k]$ and the output function $b1[k]$ can be obtained from this table as:

- (a) $a1[k] (S11[k], 1) = S11[k],$ (b) $b1[k](S11[k]) = OFF,$
 $a1[k] (S12[k], 1) = S11[k],$ $b1[k](S12[k]) = ON.$
 $a1[k] (S11[k], 2) = S12[k],$
 $a1[k] (S12[k], 2) = S12[k].$

2.2 Instant Virtual Control Machine Unit M2[k] Design

Let $M2[k]$ be a Moore type of sequential state machine. We can represent it by a quintuple $M2[k] = (S2[k], I2[k], O2[k], a2[k], b2[k])$, where

- (i) $S2[k] = \{ S21[k], S22[k], S23[k] \}$ is the set of transition states;
- (ii) $I2[k] = \{ 1, 2, 3 \}$ is the set of inputs;
- (iii) $O2[k] = \{ INPUT, PROCESSING, OUTPUT \}$ is the set of outputs;
- (iv) $a2[k]: S2 \times I2[k] \longrightarrow S2[k]$ is the state transition(or the next state) function;
- (v) $b2[k] : S2[k] \longrightarrow O2[k]$ is the output function;

Hence, $M2[k]$ representation by a transition table can be given as in Table 1.2.

The rules of definition for the state transition(or the next state) function $a2[k]$ and the output function $b2[k]$ can be obtained from this table as we have done in the Section 2.1.

2.3 Instant Virtual Input Machine Unit, M3[k], Design

Let $M3[k]$ be a Mealy type of sequential state machine. We can represent it by a quintuple $M3[k] = (S3[k], I3[k], O3[k], a3[k], b3[k])$, where

- (i) $S3[k] = \{ S31[k], S32[k], S33[k] \}$ is the set of state transition;
- (ii) $I3[k] = \{ 0, 1, ..., 9, A, B, ..., Z, \geq, /, \leq, ; \}$ is the set of inputs;

- (iii) $O3[k] = 13$ is the set of outputs;
- (iv) $a3[k] : S3[k] \times I3[k] \longrightarrow S3[k]$ is the state transition function;
- (v) $b3[k] : S3[k] \times I3[k] \longrightarrow O3[k]$ is the output function;

Hence, $M3[k]$ representation by a transition table can be given as in Table 1.3.

The rules of definition for the state transition function $a3[k]$ and the output function $b3[k]$ can be obtained from this table as we have done in the Section 2.1.

2.4 Instant Virtual Processing Machine Unit, M4, Design

Let $M4$ be a Mealy type sequential state machine. We can design $M4[k]$ by a quintuple $M4[k] = (S4[k], I4[k], O4[k], a4[k], b4[k])$, where

- (i) $S4[k] = \{ S41[k], S42[k] \}$ is the set of transition state;
- (ii) $I4[k] = \{ 1, 2 \}$ is the set of inputs;
- (iii) $O4[k] = \{ STO, LDA, ADD, MPY \}$ is the set of output;
- (iv) $a4[k] : S4[k] \times I4[k] \longrightarrow S4[k]$ is the state transition function;
- (vi) $b4[k] : S4[k] \times I4[k] \longrightarrow O4[k]$ is the output function;

Hence, $M4[k]$ representation by a transition table can be given as in Table 1.4.

The rules of definition for the state transition function $a4[k]$ and the output function $b4[k]$ can be obtained from this table.

The followings are the meanings of acronyms used in $O4[k]$:

STO : STOr the content of accumulator into location;

LDA : LOAD Accumulator from the key-board;

ADD: ADD the content of location to the content of the accumulator and keep the result in the accumulator;

MPY : MultiPIY the content of accumulator by the content of the location and keep the result in the accumulator

2.5 Instant Virtual Output Machine Unit, M5[k], Design

Let $M5[k]$ be a Moore type sequential machine. We can represent it by a quintuple $M5[k] = (S5[k], I5[k], O5[k], a5[k], b5[k])$, where

- (i) $S5[k] = \{ S51[k], S52[k] \}$ is the set of state transitions;
- (ii) $I5[k] = \{ 1, 2 \}$ is the set of inputs;
- (iii) $O5[k] = \{ PLOC, PACC \}$ is the set of outputs;
- (iv) $a5[k] : S5[k] \times I5[k] \longrightarrow S5[k]$ is the state transition function;

(v) $b5[k] : S5[k] \longrightarrow O5[k]$ is the output function.
Hence, $M5[k]$ representation by a transition table can be given as in Table 1.5.

The rules of definition for the state transition function $a5[k]$ and the output function $b5[k]$ can be obtained from this table.

The followings are the meanings of the notations in $O5$:

PACC : Print the content of ACCumulator onto output media;

PLOC : Print the content of LOCation onto output media.

We have completed instance machine design for $FLA[k]$. Now we can start instant $HOB[k]$ language design For $FLA[k]$.

3 Instant $HOB[k]$ Language Design For $FLA[k]$

Any set of notations for the description of algorithms and data structures may be termed as a formal language. A formal language that has an implementation on a computer is termed a programming language. $HOB[k]$ is a simple formal language designed by the following rules of grammar to communicate with $FLA[k]$.

3.1 Semantic Rules for $HOB[k]$

Let $:=$ be a grammatical assignment operator, s be a finite string and n be a finite integer. If “ $\langle s1 \rangle$ or $\langle s2 \rangle$ or ... or $\langle sn \rangle$ ” = “ $\langle s1, s2, ..., sn \rangle$ ” for any finite number of finite strings $s1, s2, ..., sn$, then:

SER 1: (a) $SM1S1[k] := \langle A1S11[k], A1S21[k] \rangle$: “Set INPUT machine unit $M1[k]$ to the state $S11[k]$ ”,

(b) $SM1S2[k] := \langle A1S12[k], A1S22[k] \rangle$: “Set INPUT machine unit $M1[k]$ to the state $S12[k]$ ”,

(c) OFF := “Call OFF the $FLA1[k]$ system ”

(d) ON := “Call ON the $FLA1[k]$ system,”

where

(i) $A1S11[k] := a1[k] (S11[k], 1)$,

(ii) $A1S21[k] := a1[k] (S12[k], 1)$,

(iii) $A1S12[k] := a1[k] (S11[k], 2)$,

(iv) $A1S22[k] := a1[k] (S12[k], 2)$.

SER 2: (a) $SM2S1[k] := \langle A2S11[k], A2S21[k], A2S32[k] \rangle$:

“Set CONTROL machine unit $M2[k]$ to the state $S21[k]$ ”,

(b) $SM2S2[k] := \langle A2S12[k], A2S22[k], A2S32[k] \rangle$:

“Set CONTROL machine unit $M2[k]$ to the state $S22[k]$ ”,

(c) $SM2S3[k] := \langle A2S13[k], A2S23[k], A2S33[k] \rangle$:

“Set CONTROL machine unit $M2[k]$ to the state $S23[k]$ ”,

(d) INPUT $\quad \quad \quad := b2[k] (S21[k])$: “Call INPUT machine unit M3[k] by bringing CONTROL machine unit M2[k] to the state S21[k]” ,
 (e) PROCESSING $\quad \quad \quad := b2[k] (S22[k])$: “Call PROCESSING machine unit M4[k] by bringing CONTROL machine unit M2[k] to the state S22[k]” ,
 (f) OUTPUT $\quad \quad \quad := b2[k] (S23[k])$: “ Call OUTPUT machine unit M5[k] by bringing CONTROL machine unit M2[k] to the state S23[k]”

where

- (i) $A2S11[k] := a2[k] (S21[k], 1)$,
- (ii) $A2S21[k] := a2[k] (S22[k], 1)$,
- (iii) $A2S31[k] := a2[k] (S23[k], 1)$,
- (iv) $A2S12[k] := a2[k] (S21[k], 2)$,
- (v) $A2S22[k] := a2[k] (S22[k], 2)$,
- (vi) $A2S32[k] := a2[k] (S23[k], 2)$,
- (vii) $A2S13[k] := a2[k] (S21[k], 3)$,
- (viii) $A2S23[k] := a2[k] (S22[k], 3)$,
- (ix) $A2S33[k] := a2[k] (S23[k], 3)$.

SER 3 : (a) $SM3S1[k] \quad \quad \quad := < A3S11[k], A3S21[k], A3S31[k] >$:
 “Set INPUT machine unit M3 to the state S31” ,
 (b) $SM3S2[k] \quad \quad \quad := < A3S12[k] , A3S22[k], A3S32[k] >$:
 “Set INPUT machine M3[k] to the state S32[k]” ,
 (c) $SM3S3[k] \quad \quad \quad := < A3S13[k], A3S23[k], A3S33[k] >$:
 “Set INPUT machine unit M3[k] to the state S33[k]”

where

- (i) $A3S11[k] := a3[k] (S31[k], 1)$,
 - (ii) $A3S21[k] := a3[k] (S32[k], 1)$,
 - (iii) $A3S31[k] := a3[k] (S33[k], 1)$,
 - (iv) $A3S12[k] := a3[k] (S31[k], 2)$,
 - (v) $A3S22[k] := a3[k] (S32[k], 2)$,
 - (vi) $A3S32[k] := a3[k] (S33[k], 2)$,
 - (vii) $A3S13[k] := a3[k] (S31[k], 3)$,
 - (viii) $A3S23[k] := a3[k] (S32[k], 3)$,
 - (ix) $A3S33[k] := a3[k] (S33[k], 3)$.
- (d) $0 := < b3[k] (S31[k], 0) , b3[k] (S33[k], 0) >$: “zero” ,
 $1 := < b3[k] (S31[k], 1) , b3[k] (S33[k], 1) >$: “one” ,
 $2 := < b3[k] (S31[k], 2) , b3[k] (S33[k], 2) >$: “two” ,
 ...
 $9 := < b3[k] (S31[k], 9) , b3[k] (S33[k], 9) >$: “nine” ,
 $A := < b3[k] (S32[k], A) , b3[k] (S33[k], A) >$: “letter A” ,
 $B := < b3[k] (S32[k], B) , b3[k] (S33[k], B) >$: “letter B” ,
 ...

$Z := < b3[k] (S32[k], Z) , \quad b3[k] (S33[k], Z) > : \text{“letter Z”} ,$
 $>> := b3[k] (S33[k], >>) : \text{“right shift with one blank space”} ,$
 $/ := b3[k] (S33[k], /) : \text{“go to the next line”} ,$
 $<< := b3[k] (S33[k], <<) : \text{“left shift with one blank space”} ,$
 $:= b3[k] (S33[k], ;) : \text{“end of the instruction”} .$

SER 4: (a) $SM4S1[k] := < A4S11[k], A4S21[k] > : \text{“Set PROCESSING machine unit to M4[k] to the state S41[k]”} ,$

(b) $SM4S2[k] := < A4S12[k] , A4S22[k] > : \text{“Set PROCESSING machine unit to M4 to the state S42[k]”} ,$

(c) $STO := b4[k] (S41[k], 1) : \text{“STOre the content of ACCUMULATOR from the Key – Board”} ,$

(d) $LDA := b4[k] (S42[k], 2) : \text{“LoaD ACCUMULATOR from the Key-Board ”} ,$

(f) $ADD := b4[k] (S42[k], 1) : \text{“ADD the content of LOCATION to the content of ACCUMULATOR and keep the result in the ACCUMULATOR”} ,$

(e) $MPY := b4[k] (S42[k], 2) : \text{“MultiPIY the content of CCUMULATOR with the content of LOCATION and keep the result in the ACCUMULATOR”} .$

SER 5: (a) $SM5S1[k] := < A5S11[k] , A5S21[k] > : \text{“ Set OUTPUT machine unit M5[k] to the state S51[k] ”} ,$

(b) $SM5S1[k] := < A5S12[k] , A5S22[k] : \text{“Set OUTPUT machine unit M5[k] to the state S52[k] ”} ,$

(c) $PLOC := b5[k] (S51[k]) : \text{“Print the content of the LOCATION onto output media”} ,$

(d) $PACC := b5[k] (S52[k]) : \text{“Print the content of the ACCUMULATOR onto output media”} ,$

where

(i) $A5S11[k] := a5[k] (S51[k], 1) ,$

(ii) $A5S21[k] := a5[k] (S52[k], 1) ,$

(iii) $A5S12[k] := a5[k] (S51[k], 2) ,$

(iv) $A5S22[k] := a5[k] (S52[k], 2) .$

3.2 Syntax Rules for HOB[k]

SYR1 : $<\text{letter}> := A | B | C | \dots | Z .$

SYR2 : $<\text{numeral}> := 0 | 1 | 2 | \dots | 9 .$

SYR3 : $<\text{special-character}> := << | >> | ; | / .$

SYR4 : $<\text{HOB1-character}> := <\text{letter}> | <\text{numeral}> | <\text{special – character}> .$

SYR5 : $<\text{identifier}> := <\text{letter}> | <\text{letter}> <\text{letter}> | <\text{letter}> <\text{numeral}> | <\text{identifier}> <\text{letter}> | <\text{identifier}> <\text{numeral}> .$

- SYR6:** <reserved-word> := SM1S1[k] | SM1S2[k] | OFF | ON |
 SM2S1[k] | SM2S2[k] | INPUT |
 PROCESSING | OUTPUT | SM3S1[k] |
 SM3S2[k] | STO | LDA | ADD | MPY | SM4S1[k] |
 SM4S2[k] | PLOC | PACC | SM5S1 | SM5S2 |
 ENTER (pseudo-word) .
- SYR7:** <natural-number> := <numeral> | <numeral> <numeral>
 | <numeral> <natural – number> .
- SYR8:** <ordering-word> := <natural-number> .
- SYR9:** <state-transfer-instruction> := SM1S1[k] | SM1S2[k] | SM2S1[k]
 | SM2S2[k] | SM2S3[k] | SM3S1[k] |
 SM3S2[k] | SM3S3[k] | SM4S1[k] |
 SM4S2[k] | SM1S2[k] | SM5S2[k] .
- SYR10:** <call-instruction> := INPUT | PROCESSING | OUTPUT .
- SYR11:** <operation-instruction> := STO | LDA | MPY | PLOC |
 PACC | ENTER <identifier> .
- SYR12:** <instruction> := <state-transfer-instruction> :
 <call-instruction> : <operation-instruction> .
- SYR13:** <instruction-list> := <instruction> < > ; / :
 <ordering-word> <instruction> < > ; / :
 <instruction-list> < instruction-list> .
- SYR14:** <HOB[k]-program> := ON < instruction> /
 <instruction-list> OFF .

3.3 Pragmatic Rules for HOB[k]

PRR 1: HOB[k] is a machine dependent programming language KBO type. A program is written in this type, from the left to the right and from the top to the bottom, on the (FLA[k], HOB[k]) paper tape (diskette or disk); it is executed as instruction by instruction from this paper tape (diskette or disk) on the hand-on interactive FLA[k] machine KBO type.

PRR 2: Each line on a (FLA[k], HOB[k]) paper tape (diskette or disk) has a length of 32 columns, each can hold only one (FLA[k], HOB[k]) system character. Further each line on a (FLA[k], HOB[k]) paper tape (diskette or disk) has four special fields which are called as Field A, Field B, Field C, Field D. Field A occupies 6 columns from column 1 to column 6. Field B occupies 25 columns from column 8 to column 32. Field C occupies 23 columns from column 10 to column 32. Field D occupies 21 column from column 12 to column 32.

PRR 3: In a HOB[k] program, an instruction with missing ordering word is called a HOB[k] statement. In a structured HOB[k] program:

- (a) Ordering words appear in the Field A as right justified,

- (b) ON/OFF statements appear in the Field B as left justified,
- (c) Call statements appear in the Fields C as left justified,
- (d) State transfer statements and other operational statements appear in the Field D as left justified.

PRR 4: The output of HOB[k] program are printed, line by line, from the left to the right and from the top to the bottom, on the (FLA[k], HOB[k]) output paper tape media as left justified. In the paper tape media, each line has 32 columns and each column holds only one (FLA[k], HOB[k]) system character.

4 Conclusions

We have developed an instant abstract machine KBO cluster and an instant abstract language KBO cluster for generating a (FLA, HOB) computational management system KBO type us-Crop under a name of an instant (FLA, HOB) KBO model design. FLA[k] Virtual Machine and its Language HOB[k] is the k-th member of a polarized us-crop of FLAHOB = (FLA, HOB) = { (FLA[k], HOB[k]: k = ..., -3, -2, -1, -0, +0, 1, 2, 3, ..} for a simulation model of a computational management system KBO type. FLA[k] is a formal sequential state machine KBO type composed of eight submachine M1[k], M2[k], M3[k], M4[k], M5[k], M6[k], M7[k] and M8[k]. It has only one word screen (or Key-Word register), one accumulator (or operation register) and one word memory called as location and organized M8[k]. They were formally designed in such a way that one can easily or readily obtain their simulation on an IBM compatible personal computer. HOB[k] is a formal language KBO type which has been defined in terms of three finite set of grammar rules. Namely these are a set of semantic rules, a set of syntax rules, and a set of pragmatic rules. One can easily and effectively communicate with FLA[k] while programming it in HOB[k] in their simulation. Therefore, instant (FLA[k], HOB[k]) Virtual Machine KBO type and its Formal Language KBO type can be used in a variety of purposes such as a teaching tool in a classroom environment of different computer science courses to expose the students to the concept of virtual machines and formal languages.

A partial simulation of (FLA[k], HOB[k]) Virtual Machine and its Language was tried out by some group of students of computer science taking "Programming Language: Design and Implementation." The results obtained were well beyond our expectations.

The instant design of "(FLA[k], HOB[k]) Virtual Machine KBO type and its Language KBO type" is on the design of a computer system KBO type and its language KBO type. Different types of virtual machines KBO types and their languages KBO types can be designed for some other purposes by the same technique as an us-culture KBO type. Their instant simulations may

give birth to them; brings them into life if we want them in different shapes and configurations in a very short period of time. We claim that our technique of design is also applicable to all existing microprocessors KBO type.

References

- [1] F. Ünlü, FTD Grammar Graphs, *International Journal of Computer Mathematics*, Vol. 80, no. 1, pp1-9, January 2003, MR1952093, 68N19(05C85 68Q42 68T30).
- [2] F. Ünlü, tüze-Channel and tüze-Channeled Discrete Convolution Process, *Karadeniz University Math. Journal*, Vol. 5, no. 1, pp 104-116, MR07354449, 93B07.
- [3] F. Ünlü and S. Sönmez, I@I: Tıkız Internet Tabanlı Tıkız Internet, *II. Proceeding of Information Technology Congress*, pp. 246-248, 1-May 2003, Pamukkale University-Denizli.
- [4] F. Ünlü, An Intuitive Differential Equation Model for Knowledge Based Objects(KBO) Representation of Science, ISCISXIV, *Proceeding of The Fourteenth International Symposium on Computer and Information Science*, pp1066-1068, October, 18-20, Kuşadası, Aydın, Turkey.
- [5] F. Ünlü, Chance Constrained Threshold KBO System Design, *International Mathematical Journal*, Vol. 5. no. 4, pp 321-328, 2004, MR2036709 93B51(93C95).
- [6] F. Ünlü, A Generalized us-Culture Job Scheduling for Forecasting Problems, *International Mathematical Journal*, Vol. 4, no. 4, pp 313-320, 2003.
- [7] F. Ünlü, S. Sönmez, and Z. I. Ünlü, Circular Convolved I@I KBO Cluster KBO Generating us-Crop, *International Mathematical Journal*, Vol. 5. no. 4, pp 329 -338, 2004.
- [8] F. Ünlü, FLA2 & HOB2: A Pair Design Of A Virtual Machine And Its Language As A Simulation Model Of An Experimental Computational System, *DIRASAT in Science*, Vol. XV, No. 9, pp 304-324, the University of Jordan, Amman, 1988.
- [9] Pritsker, A.A. and Pegden, C.D.: *Introduction to Simulation and SLAM*, A Halsted Press Book, John Wiley & Sons, New York (1979) .

- [10] Pratt, T.W. :*Programming Languages: Design Implementation*, Prentice-Hall International, Engelwood, New Jersey(1984).
- [11] Denning, P.J., Dennis, J.B. and Qualitz, J.E.: *Machines, Languages, and Computations*, Prentice-Hall, Inc., Engelwood, New Jersey(1978).
- [12] Lewis II, P. M., and at all: *Compiler Design Theory*, Addison-Wesley Publishing Company, Inc., Reading, Massachusetts(1976).

Received: October 3, 2005

Figure Captions:

Figure 1.1: A pictorial organization of FLA[k] KBO cluster.

Table 1.1: Transition table of M1[k].

Table 1.2 : Transition table of M2[k].

Table 1.3 : Transition table of M3[k].

Table 1.4: Transition table of M4[k].

Table 1.5: Transition table of M5[k].