XFL3: A NEW FUZZY SYSTEM SPECIFICATION LANGUAGE

F. J. Moreno-Velo, S. Sánchez-Solano, A. Barriga, I. Baturone, D. R. López

Instituto de Microelectrónica de Sevilla - Centro Nacional de Microelectrónica
Avda. Reina Mercedes s/n, (Edif. CICA),
E-41012, Sevilla, Spain

5th WSES /IEEE Multiconference on Circuits, Systems, Communications and Computers (CSCC 2001), pp. 361-366, Rethymnon, Julio 2001

XFL3: A New Fuzzy System Specification Language

F. J. MORENO-VELO, S. SÁNCHEZ-SOLANO, A. BARRIGA, I. BATURONE, D. R. LÓPEZ
Instituto de Microelectrónica de Sevilla (IMSE-CNM)
Avda. Reina Mercedes, s/n Edif. CICA. E-41012 Sevilla.
SPAIN
xfuzzy-team@imse.cnm.es. http://www.imse.cnm.es

Abstract: - This paper presents the main features of XFL3, a new language for fuzzy system specification, which has been defined as the starting point for the 3.0 version of our fuzzy system design environment, Xfuzzy [1]. Its main advantages with respect to its precursor, XFL [2], are its capability to admit user-defined membership functions, parametric operators, and linguistic hedges. Taking this language as the basis, different fuzzy system development tools are being implementing, which are also summarized briefly.

Key-Words: - Formal languages, CAD tools, Fuzzy systems.

1 Introduction

The definition of formal languages for fuzzy system specification is usual for its several advantages [3][4]. However, two objectives may conflict. On one side, a high expressive language, able to apply all the fuzzy logic-based formalisms, is desired. On the other side, the final system implementation constraints have to be considered. In this sense, some languages focus on expressiveness [5][6], while others are focused on software or hardware implementations [7]. When our group begun to design a fuzzy system environment, our aim was to meet both objectives. This led us to the definition of the formal language XFL [2], which is the base for several hardware- and software-oriented development tools that constitute the Xfuzzy 2.0 design environment [1].

As a starting point for the 3.0 version of Xfuzzy, a new language, XFL3, which extends the advantages of XFL, has been defined. XFL3 allows the user to define new membership functions and parametric operators, and admits the use of linguistic hedges which permit to describe more complex relationships among variables [8]. In order to incorporate these improvements, some modifications have been made in the XFL syntax. In addition, the new language XFL3, together with the tools based on it, employ Java as programming language. This means the use of an advantageous object-oriented methodology and the flexibility of executing the new version of Xfuzzy in any platform with JRE (Java Runtime Environment) installed.

2 The XFL3 language

XFL3 is a fuzzy system specification language which provides the user with a great flexibility to define the functions associated with the fuzzy operators and linguistic variables and which allows to express complex rule bases.

An XFL3 specification consists of several objects defining operator sets, variable types, and rule bases. The definition format of these elements is described in the following.

2.1 Operator sets

An operator set in XFL3 is an object containing the mathematical functions that are assigned to each fuzzy operator. Fuzzy operators can be binary (like the T-norms and S-norms employed to represent linguistic variable connections, implication, or rule aggregations), unary (like the C-norms or the operators related with linguistic hedges), or can be associated with defuzzification methods [9].

XFL3 defines the operator sets with the following format (Figure 1):

operatorset identifier **{**
 operator assigned_function**(**parameter_list**);**
 operator assigned_function**(**parameter_list**);**
 .... **}**

It is not required to specify all the operators. When one of them is not defined, its default function is as-

This work has been partially supported by the Spanish CICYT Project TIC98-0869 and the FEDER Project (Nº 1FD97-0956-C3-02).

```
operatorset systemop {
  and  xfl.min();
  or    xfl.max();
  imp xfl.min();
  strongly xfl.pow(3);
  moreorless xfl.pow(0.4);
 }
```

Fig. 1: Example of operator set definition.

sumed. Table 1 shows the operators (and their default functions) currently used in XFL3.

Table 1: Operators currently defined in XFL3.

Operator	Type	Default function
and	binary	min(a,b)
or	binary	max(a,b)
implication imp	binary	min(a,b)
also	binary	max(a,b)
not	unary	(1-a)
strongly	unary	$(a)^2$
moreorless	unary	$(a)^{1/2}$
slightly	unary	4*a*(1-a)
defuzzification defuz	defuzzification	center of area

The assigned functions are defined in external files which we name as packages. The format to identify a function is "*package.function*". The package name (*xfl* in Figure 1) could be removed if the package has been imported previously (using the command "*import* package;").

2.2 Types of linguistic variables

An XFL3 type is an object which describes a type of linguistic variable. This means to define its universe of discourse, to name the linguistic labels covering that universe, and to specify the membership function associated to each label. The definition format of a type is as follows (Figure 2):

type identifier **[**min**,** max**;** card**] {**
label membership_function**(**parameter_list**);**
label membership_function**(**parameter_list**);**
.... **}**

```
type input1 [0,100] {
  short xfl.triangle(0,25,50);
  medium xfl.triangle(25,50,75);
  tall xfl.triangle(50,75,100);
 }

type input2 extends input1 {
  very_short xfl.triangle(-10,0,25);
  very_tall xfl.triangle(75,100,110);
 }
```

Fig. 2: Example of variable type definition.

where *min* and *max* are the limits of the universe of discourse and *card* (cardinality) is the number of its discrete elements. If cardinality is not specified, its default value (currently, 256) is assumed. When limits are not explicitly defined, the universe of discourse is taken from 0 to 1.

The format of the linguistic label identifier is similar to the operator identifier, that is, "*package.function*" or simply "*function*" if the package where the user has defined the membership functions has been already imported.

XFL3 supports inheritance mechanisms in the type definitions (like its precursor, XFL). To express inheritance, the heading of the definition is as follows (Figure 2):

type identifier ***extends*** identifier **{**

The types so defined inherit automatically the universe of discourse and the labels of their parents. The labels defined in the body of the type are either added to the parent labels or overwrite them if they have the same name.

2.3 Rule bases

A rule base in XFL3 is an object containing the rules which define the logic relationships among the linguistic variables. Its definition format is as follows (Figure 3):

rulebase identifier **(**input_list **:** output_list**)**
using operatorset **{**
[factor**]** ***if*** **(**antecedent**)** -> consequent_list**;**
[factor**]** ***if*** **(**antecedent**)** -> consequent_list**;**
.... **}**

The definition format of the input and output variables is "*type identifier*", where *type* refers to one of the linguistic variable types previously defined. The operator set selection (systemop in Figure 3) is optional, so that when it is not explicitly defined, the de-

```
rulebase base1(input1 x, input2 y : output z) using systemop
{
   if( x == medium & y == medium) -> z = tall;
   [0.8] if( x<=short |  y != very_tall ) -> z = short;
   if( +(x>tall) & (y ~= medium) ) -> z = tall;
   ............ }
```

Fig. 3: Example of rule base definition.

fault operators are employed. It is also shown in Figure 3 how confidence weights (with default values of 1) can be applied to the rules.

A rule antecedent describes the relationships among the input variables. XFL3 allows to express complex antecedents by combining basic propositions with connectives or linguistic hedges (Table 2 and Figure 4). On the other side, each rule consequent describes the assignation of a linguistic variable to an output variable as "*variable = label*" (Figure 3).

Table 2: Example of fuzzy propositions

Basic propositions	**Description**
variable == label	equal to
variable >= label	equal or greater than (Fig. 3a)
variable <= label	equal or smaller than (Fig. 3b)
variable > label	greater than (Fig. 3c)
variable < label	smaller than (Fig. 3d)
variable != label	not equal to (Fig. 3e)
variable %= label	slightly equal to (Fig. 3f)
variable ~= label	moreorless equal to (Fig. 3g)
variable += label	strongly equal to (Fig. 3h)
Complex propositions	**Description**
proposition & proposition	and operator
proposition | proposition	or operator
! proposition	not operator
% proposition	slightly operator
~ proposition	moreorless operator
+ proposition	strongly operator

2.4 System global behavior

The description of the system global behavior means to define the global input and output variables of the system as well as the rule base hierarchy. This description in XFL3 is as follows (Figure 5):

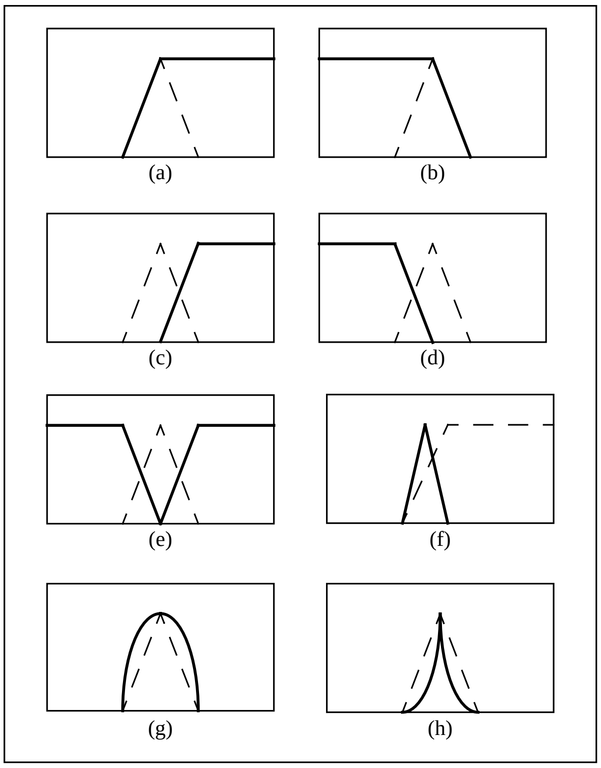


Fig. 5: Illustrating linguistic hedges.

system (input_list : output_list) {
rule_base_identifier**(**inputs **:** outputs**);**
rule_base_identifier**(**inputs **:** outputs**);**
.... **}**

```
system (input1 x, input2 y : output z) {
   rulebase1(x, y : inner1);
   rulebase2(x, y : inner2);
   rulebase3(inner1, inner2 : z);
 }
```

Fig. 4: Example of system behavior definition.

The definition format of the global input and output variables is the same format employed in the definition of the rule bases. The inner variables which may appear establish serial or parallel interconnections among the rule bases. Inner variables must firstly appear as output variables of a rule base before being employed as input variables of other rule bases (Figure 5).

3 Function packages

Four types of functions can be defined in XFL3: binary functions, unary functions, membership functions, and functions associated with defuzzification methods. A great advantage of XFL3 is that these functions can be defined freely by the user in external files (named as packages). The definition format is as follows:

binary identifier **{** blocks **}**
unary identifier **{** blocks **}**
mf identifier **{** blocks **}**
defuz identifier **{** blocks **}**

The blocks that define a function include its name (and possible *alias*), the parameters which specify its behavior as well as the constraints on these parameters, the description of its behavior in the different languages to which it could be compiled (*java, ansi_c* and *cplusplus*, for instance), and even the description of its differential function (if it is employed in gradient-based learning mechanisms). This information is the basis to generate automatically a Java class that incorporates all the function capabilities and can be employed by any XFL3 specification. Regarding defuzzification methods, some of them can be only employed with certain membership functions. To express this dependence, the block *definedfor* indicates those allowed membership functions.

The use of packages allows the designer to define any desired function. The standard package currently used in XFL3 (and named *xfl*) contains the most usual functions, as shown in Table 3.

Table 3: The standard package *xfl*.

Function type	Possible assigned functions
Binary	min, prod, bounded_prod, drastic_prod, max, sum, bounded_sum, drastic_sum, dienes_resher, mizumoto, lukasiewicz, dubois_prade, zadeh, goguen, godel, sharp.
Unary	not, sugeno, yager, pow, parabola.
Membership functions	trapezoid, triangle, isosceles, slope, bell, sigma, rectangle, singleton
Defuzzification methods	CenterOfArea, FirstOfMaxima, LastOfMaxima, MeanOfMaxima, FuzzyMean, WeightedFuzzyMean, Quality, GammaQuality.

4 Example of an XFL specification

One of the main features of XFL3 is the inclusion of linguistic hedges and hierarchical structures on the definition of fuzzy systems. The modular division of the system description allows the designer to confront the development of complex systems. In this sense, linguistic hedges can be used to decrease the number of linguistic labels employed and to express logic rules more compactly [10].

As an example of complex system modelling, we have considered the classic problem of parking a truck in a loading dock [11]. In this problem, only the case of backward driving is generally considered. However, this makes it very difficult for the truck to park when starting at a bad-oriented position as shown in Figure 6. The approximation we have followed is to directly emulate how we will act as drivers, which for us is a two step decision problem: to decide whether moving forwards or backwards and, depending on the case, to select the proper angle of the wheels. This knowledge is, hence, represented by a hierarchical system. In particular, four rule bases are employed, as shown at the bottom of Figure 7. The rule base *direction* emulates our non fuzzy making decision about the direction of movement. On the contrary, the rule bases *forward* and *backward* emulates our fuzzy decision about the wheel angle when driving forward or backward. Finally, the rule base *switch* chooses the proper turn as a function of the movement direction.

This example does not attempt to illustrate an optimum way of solving the truck-dock problem but the efficiency of XFL3 for representing expert linguistic knowledge. In this sense, the type definitions of the variables have been reduced by using the *greater* and *smaller* linguistic hedges, and the rule base definitions have been compacted thanks to combinations of connectives and hedges, as we express linguistically.

Figure 8 shows the results of two simulations. In the first one, the truck starts at the bottom-left corner, with a north-east orientation. The system decides to drive forward approaching the truck to the vertical and, once at that position, leads it backwards to the

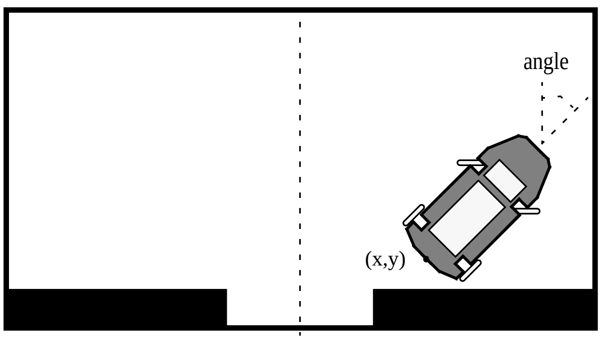
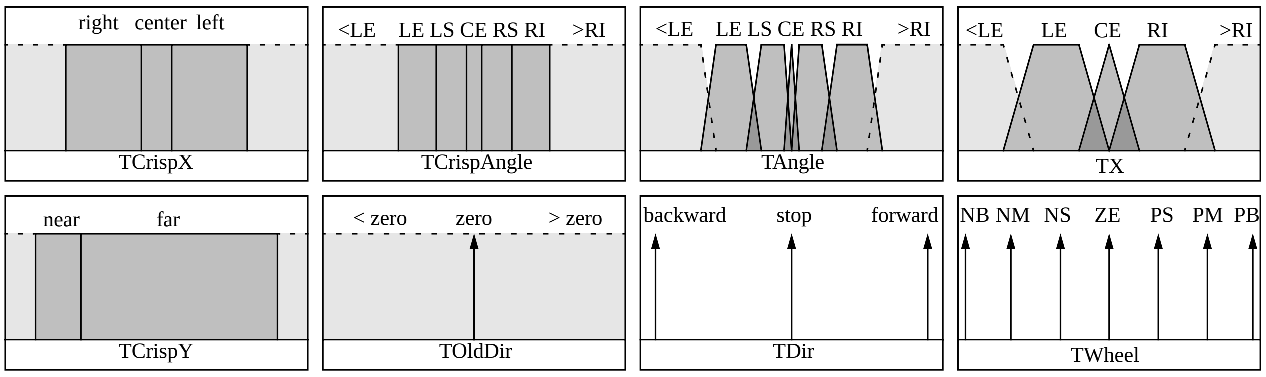


Fig. 6: Example of the truck-dock problem.



```
operatorset opset { defuz xfl.FuzzyMean(); and xfl.prod(); }
type TCrispX [-50.0,50.0] { ... }
  ......
type TWheel [-30.0,30.0] { ... }
rulebase backward(TX x, TAngle angle : TWheel wheel) using opset { ... }
rulebase forward(TX x, TAngle angle : TWheel wheel) using opset { ... }
rulebase switch(TWheel bw, TWheel fw, TDir dd : TWheel wheel, TDir dir)
using opset { ... }

rulebase direction(TCrispX x,TCrispY y,TCrispAngle angle,TOldDir olddir : TDir dir)
using opset {
 if(y > far) -> dir = backward;
 if(y == far & olddir <= zero ) -> dir = backward;
 if(y == far & olddir > zero ) -> dir = forward;
 if(y <= near & x != center & angle <= RI & angle >= LE) -> dir = forward;
 if(y <= near & x != center & (angle > RI | angle < LE)) -> dir = backward;
 if(x == center & y == near & angle <= RS & angle >= LS) -> dir = backward;
 if(x == center & y == near & (angle > RI | angle < LE)) -> dir = backward;
 if(x == center & y == near & (angle == LE | angle == RI)) -> dir = forward;
  if(x == center & y < near & angle == CE) -> dir = stop;
 if(x == center & y < near & (angle > RI | angle < LE)) -> dir = backward;
 if(x == center & y < near & angle < CE & angle >= LE) -> dir = forward;
 if(x == center & y < near & angle > CE & angle <= RI) -> dir = forward;
}

system (TX x,TCrispY y,TAngle angle,TOldDir olddir : TWheel wheel,TDir direction){
 direction(x, y, angle, olddir : dd);
 backward( x, angle : bw);
 forward( x, angle : fw);
 switch( bw, fw, dd : wheel, direction);
}
```

angle \ x	<LE	LE	CE	RI	>RI
<LE	PS	PM	PM	PB	PB
LE	NS	PS	PM	PB	PB
LS	NM	NS	PS	PM	PB
CE	NM	NM	ZE	PM	PM
RS	NB	NM	NS	PS	PM
RI	NB	NB	NM	NS	PS
>RI	NB	NB	NM	NM	NS

rulebase backward

angle \ x	<LE	LE	CE	RI	>RI
<LE	NB	NB	NB	NB	NM
LE	NB	NM	NM	NM	NS
LS	NM	NS	NS	NS	PS
CE	NS	NS	ZE	PS	PS
RS	NS	PS	PS	PS	PM
RI	PS	PM	PM	PM	PB
>RI	PM	PB	PB	PB	PB

rulebase forward

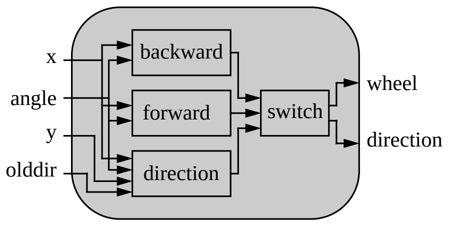
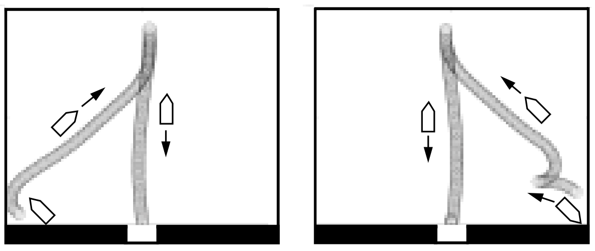


Fig. 7: Summary of the XFL3 specification of a fuzzy control system for the truck-dock problem.



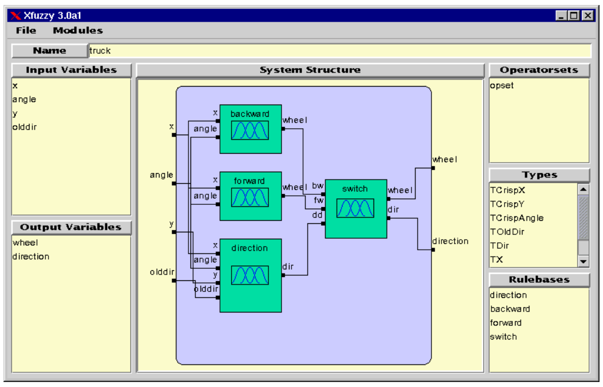
(a) (b)

Fig. 8: Simulation results on the truck-dock problem.

dock. The second simulation begins at the bottom-right corner in a south-west direction. In this case, the truck is driven backwards till an horizontal orientation is reached, then is directed forward to the vertical and finally goes back to the dock.

5 Summary of the XFL3-based tools

The core of any application developed with XFL3 is based on the use of Java classes which contain the whole structure and functionality of the specifications to be worked with. Using these classes and the Java graphic libraries, several tools have been built



Name: truck

Fig. 9: Main window for the **xfedit** tool.

which allow exploiting the XFL3 features through the different development stages of a fuzzy system design.

Currently, there is available a fuzzy system edition tool, named **xfedit**. It allows defining the operator sets, variable types, rule bases, and system structure through a graphical user interface, as shown in Figure 9. Two graphic representation tools, named **xf2dplot** and **xf3dplot**, have been also developed. They allow illustrating the fuzzy system behavior by its output/ input surface in a two or three dimensional space.

A tool for automatically adjusting fuzzy systems described with XFL3 is also available. This tool, named **xfsl**, provides the user with several learning algorithms and allows selecting which system parameters are going to be tuned and which not. Besides, this tool includes two methods of pre- and post-processing for eliminating non significant rules and labels and for clustering the output variables, thus simplifying their associated types.

The final step of any design process is the synthesis step which can lead to a software or hardware system implementation. To ease the software synthesis, three tools have been developed, **xfj**, **xfc** and **xfcpp**, which generate, respectively, the system description in Java, C, and C++ languages.

These and other tools under development attempt to cover all the different stages of a fuzzy system design from its linguistic description to its final implementation (either software or hardware) and will constitute the 3.0 version of the Xfuzzy environment.

6 Conclusions

This paper has introduced the XFL3 language, which has been developed after accumulating experience with the design of the Xfuzzy 2.0 environment. XFL3 eases the description and manipulation of complex fuzzy systems thanks to the use of quite user-defined membership functions, fuzzy operators (including linguistic hedges), and rule bases (admitting hierarchical structures). An illustrative example has been included to show the efficiency of XFL3 to rapidly translate linguistic knowledge. Based on this language, several tools are being developed to constitute the new version of Xfuzzy, which could be executed on any platform containing the Java Runtime Environment.

References:

[1] D. R. López, C. J. Jiménez, I. Baturone, A. Barriga, S. Sánchez-Solano. "Xfuzzy: A Design Environment for Fuzzy Systems", *Proc. 7th IEEE Int. Conf. on Fuzzy Systems* (FUZZIEEE’98), pp. 1060-1065, Anchorage, May 1998.

[2] D. López, F. J. Moreno, A. Barriga, S. Sánchez-Solano. "XFL: A Language for the Definition of Fuzzy Systems", *Proc. 6th IEEE Int. Conf. on Fuzzy Systems* (FUZZIEEE’97), pp. 1585-1591, Barcelona, Jul., 1997.

[3] M. Bonner, S. Mayer, A. Raggl, W. Slany, “FLIP++: A fuzzy logic inference library”, *Fuzzy Logic in Artificial Intelligence: Towards Intelligent Systems*, Martin, T.P., Ralescu, A.L., Eds., Springer-Verlag, 1997.

[4] Z. A. Sosnowski, “FLISP - a language for processing fuzzy data”, *Fuzzy Sets and Systems*, No 37, pp. 23-32, 1990.

[5] O.G.Duarte, G. Pérez. “UNFUZZY: Fuzzy Logic System analysis, design, simulation and implementation software”, *Proc. 1999 Eusflat-Estylf Joint Conf.*, pp. 251-254 Mallorca, 1999.

[6] R. Hartwig, C. Labinsky, S. Nordhoff, B. Landorff, P. Jensch, J. Schwanke. "Free Fuzzy Logic System Design Tool: FOOL", *Proc. 4th European Congress on Intelligent Techniques and Soft Computing* (EUFIT’96), pp. 2274-2277, Aachen, Sep. 1996.

[7] J. Yen, R. Langari, L.A. Zadeh, Eds., *Industrial Applications of Fuzzy Logic and Intelligent Systems*, IEEE Press, 1995.

[8] L.A. Zadeh, “A fuzzy-set-theoretic interpretation of linguistic hedges”, *J. Cybern*, vol. 2, pp. 4-34, 1972.

[9] E.H. Ruspini, P.P. Bonissone, W. Pedrycz, Eds., *Handbook of Fuzzy Computation*, Institute of Physics Pub., 1998.

[10] M. Sugeno, T. Yasukawa, “A fuzzy-logic-based approach to qualitative modeling”, *IEEE Trans. Fuzzy Systems*, vol. 1, no. 1, pp. 7-31, 1993.

[11] S.G. Kong, B. Kosko, “Adaptive Fuzzy Systems for Backing up a Truck-and-Trailer”, *IEEE Trans. Neural Networks*, vol. 3, no. 2, pp.211-223, 1992.