Proc. Eighth International Conference on Database and Expert Systems
Applications (DEXA'97), Sept. 1-5, 1997, Toulouse, France, pp. 91-100.

# Design and Implementation of TEMPO Fuzzy Triggers

**Tarik Bouaziz[‡], Janne Karvonen,**
**Antti Pesonen and Antoni Wolski**

Technical Research Centre of Finland (VTT)
VTT Information Technology
P.O. Box 1201, FIN-02044 VTT, Finland

`e-mail: <first name>.<last name>@vtt.fi`
`http://www.vtt.fi/tte/projects/tempo/`

**Abstract**

Fuzzy triggers are database triggers incorporating fuzzy concepts. The approach leads to the application of approximate reasoning to trigger-based decision making. In C-fuzzy triggers, fuzzy rules may be specified in the trigger condition part. The C-fuzzy trigger model is presented, and an implementation thereof in the TEMPO Server—a prototype active database system—is described. The performance test results are included.

**Keywords**: Active databases, approximate reasoning, fuzzy triggers.

## 1 Introduction

In various application domains of information technology a phenomenon of data explosion may be observed. For example, in the field of industrial process management, the data explosion effect is caused by the improved data acquisition techniques: more and more process variables are reported at higher and higher frequencies. In a needs survey performed in 1993 [JP93] a peak acquisition rate required for a major power station installation was about 5 000 measurements per second, and the requirements have been growing since then. In the presence of data explosion one needs powerful means to extract useful information from the flooding data. In time-critical environments like manufacturing process management and control, the speed of data evaluation is also of great importance. As we see it, providing the right information at the right time becomes a new challenge of next generation database systems. The goal may be achieved by improving the active capabilities of database systems. In this paper, we present an implementation of a new type of database triggers using fuzzy inference in making the decisions. The results of this work are applicable to other environments suffering from data explosion—ranging from corporate databases to World Wide Web.

The work presented in this paper was originally driven by the requirements of an industrial application: a paper machine drive control system. A paper machine is equipped with tens of high-power electric motors. Process measurements data is stored in a database which is fed by sensors. There is a need to analyze the data and to

[‡] The work has been carried out within the ERCIM Fellowship programme (ERCIM Human Capital and Mobility Programme).

provide useful information to the end user, in a timely and appropriate manner, in order to prevent failures of the drive system. To achieve this, in the TEMPO project, we have proposed a technique of *fuzzy triggers* [BW96]. In this paper, we focus on a concrete implementation of the concept.

To our best knowledge, no attention has been paid until now on integrating imprecision and/or uncertainty within database triggers. There are many approaches through which this integration can take place [BW96, BW97]. The TEMPO approach focuses on a seamless integration of fuzziness within database triggers. There are three design criteria which we strive to satisfy in TEMPO.

The first goal is to enhance the expressive power of triggers to capture the expert knowledge which is imprecise, incomplete or vague. This knowledge is more easily expressed using fuzzy rules which allow fuzziness in the antecedents and/or consequents [Zad84, Zad89]. Indeed, many experts have found that fuzzy rules provide a convenient way to express their domain knowledge. In industrial applications, linguistic terms such as *low*, *medium*, *high*, *large*, *small*, etc. are widely used since they convey more information than crisp values would do [Men95].

The second goal is to extend the (exact) reasoning inherent to triggers and to integrate it with the approximate reasoning more tightly. This makes fuzzy triggers a powerful mechanism to capture both approximate and exact reasoning characterizing real-world problems. Approximate reasoning deals with inference under imprecision and/or uncertainty in which the underlying logic is approximate rather than exact [Zad75, GKB+84]. It should be noted that in our daily life most of the information on which our decisions are based is linguistic rather than numerical in nature. In this perspective, approximate reasoning provides a natural framework for the characterization of human behavior and suitable for decision making.

The third goal, a practical one, is to find an easy way to add a fuzzy trigger capability to an existing active database system since we have previously developed such a system [WKP96]. Also, a possibility to use an existing fuzzy inference engine is highly recommended since it reduces the implementation effort.

This paper is organized as follows: Section 2 illustrates the main features of the model of implemented fuzzy triggers. Section 3 presents selected topics of the implementation of the TEMPO Server prototype. We show performance test results in Section 4. Then, we conclude in Section 5.

## 2 TEMPO Fuzzy Triggers

In this section we briefly describe a trigger model incorporating approximate reasoning (fuzzy inference) in the process of the evaluation of the condition part of an ECA trigger. We are calling such triggers *C-fuzzy triggers* (or Condition-fuzzy ECA triggers). For a more detailed presentation of this one and other fuzzy trigger models, see [BW96, BW97]. The C-fuzzy trigger model is based on the concepts of the linguistic variables, the corresponding terms and the rule set function.

## 2.1 Linguistic Variables

A linguistic variable is a variable whose values are words rather than numbers [Zad89]. It has a name and a term set which is a set of linguistic values (terms) defined over the definition domain of the linguistic variable. Each term is defined by its corresponding membership function.

***Example:*** Let us consider the linguistic variable *temperature*. Its term set *T(temperature)* could be *T(temperature) = {low, normal, hot}* where each term is characterized by a fuzzy set in a universe of discourse *U = [0, 300]*. We might interpret "low" as "a temperature below about 100øC," "normal" as "a temperature close to 120øC" and "hot" as "a temperature above about 130øC". These terms can be characterized as fuzzy sets whose membership functions are shown in Figure 1. Each element $u \in U$ belongs to a fuzzy set F with a degree of membership $\mu_F(u) \in [0, 1]$. For example, if the current temperature is 90øC then the membership degree to the fuzzy subset *low* is equal to 0.5.

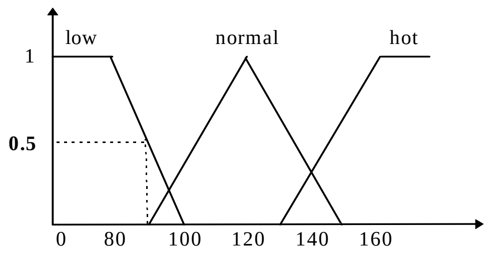

**Figure 1**. Membership functions of the linguistic variable temperature.

The domain of a linguistic variable is defined using a *linguistic type*. For example, we assume that the above linguistic variable is of the linguistic type called *Temperature* which can be defined as follows[1]:

```
CREATE LING TYPE Temperature float (
     low       TRAPEZOID (0, 0, 80, 100),
     normal    TRAPEZOID (90, 120, 120, 150),
     hot       TRAPEZOID (130, 160, 300, 300)
)
```

## 2.2 Rule Set Functions

A fuzzy logic rule takes the form of *if-then* statement such as "if X IS A then Y IS B" where X and Y are linguistic variables, A and B are terms. The *if* part of a fuzzy *if-then* rule is called the *antecedent* (or premise), whereas the *then* part is called the *consequent*. The antecedent part of a fuzzy rule is a conjunction and/or a disjunction of *fuzzy propositions* such as (X IS A) where *X* is a linguistic variable representing a database value, fuzzyfied using the term A.

[1] The syntax of the language RQL/F used in these examples is based on SQL. The full syntax of the language elements can be found in the extended version of this paper in ftp://ftp.vtt.fi/pub/projects/rapid/tempo-design.ps.

A *rule set* is a series of *if-then* statements, grouped in a structure called *a rule set function*. The idea behind it is to utilize the expressive power of fuzzy rules and to take the advantages of the fuzzy inference by incorporating the rule set as a function call in the trigger condition part. Formally, the rule set function is defined as follows:

$$RSF : \{R \times S_i\} \to D$$

where *R* is a set of fuzzy rules in which the consequent linguistic variable should be the same one occurring in all the fuzzy rules in *R*. $S_i$ refers to the current state of the database. The range of RSF, *D*, is a domain (universe of discourse) of the output linguistic variable. Thus, RSF yields a value which can be used in a regular comparison predicate evaluating, in turn, to *true* or *false*.

***Example:*** Let us consider a rule set function which monitors the speed and the temperature of a motor. Each fuzzy rule in the rule set traduces the occurrence of an alarming condition. Let us assume the default alarm value is “a_none” which is considered as an output when none of these rules is fired (no alarm). We are interested in specifying rules leading to other alarm values. The rule set may be defined as follows:

```
CREATE RULE SET ControlAlarm (temperature Temperature, speed Speed)
     Severity DEFAULT a_none(
          IF temperature IS normal AND speed IS high THEN a_low,
          IF temperature IS high AND speed IS very_high THEN a_high,
          …)
```

The rule set `ControlAlarm` monitors the temperature and the speed parameters of a motor. The rule set expresses the conditions when the operator should be alarmed. In order to simplify the syntax, the output linguistic variable does not appear explicitly in the example shown above (only its type is specified as “Severity”).

## 2.3 C-fuzzy Triggers

A C-fuzzy trigger is a regular ECA trigger which includes, in its condition part, a function call to a rule set. The execution semantics of a C-fuzzy trigger is based on the execution model of regular ECA trigger as shown in [WC96], p. 17. In its simple form, the *rule processing algorithm*, which characterize the execution model, repeatedly executes three consecutive calculations performed when an event occurs:

1. detecting an event and finding a relevant trigger,
2. evaluating the condition and
3. executing the action if the condition is true.

Our approach of incorporating fuzzy inference into triggers requires only the modification of the second calculation step of the above rule processing algorithm. The condition contains a rule set function call which takes place during the condition evaluation. The following three calculation steps are included in the evaluation of each rule set function call:

2.1 ***Fuzzification:*** the fuzzification is the process of converting crisp input data to fuzzy sets. The linguistic variables in the antecedent part of the rules are evaluated, i.e. the corresponding source data are mapped into their membership functions and truth values then fed into the rules. We are assuming no fuzzy values need to be stored in the database.

2.2 ***Inference:*** The most commonly used fuzzy inference method is the so-called *Max-Min inference method* [Lee90], in particular in engineering applications [Men95]. The Max-Min inference method is applied to the rule set, producing a fuzzy conclusion.

2.3 ***Defuzzification:*** the result of the fuzzy inference is a fuzzy set. The defuzzyfication step produces a representative crisp value as the final output of the system. There are several defuzzification methods [Lee90, Men95]. The most commonly used is the *Centroid (Center-of-gravity) defuzzifier* which provides a crisp value based on the center-of-gravity of the result (the output fuzzy set). The Center-of-gravity method yields to a crisp function value which is then applied to the comparison predicate.

***Example:*** Let us assume that, when the returned defuzzified value of the rule set function is between two and three, then a medium-level alarm is raised. A corresponding trigger can be defined as follows:

```
CREATE TRIGGER Trig_alarm_medium INSERT ON motor
     WHEN (ControlAlarm(temp, speed) > 2 AND
           ControlAlarm(temp, speed) <= 3)
     (MediumTempAlarm@TempAlarms)
```

If the condition is satisfied, the action specified as `MediumTempAlarm@TempAlarms` is executed. Note that the action part contains a call to an external procedure, i.e. the action of a trigger is executed outside the database. This feature is significant for control room applications of complex industrial processes (e.g. a paper mill or a power plant).

## 3 TEMPO Fuzzy Trigger Implementation

The TEMPO Server has been written in C++ and it runs both in the Unix (HP-UX) and Windows NT environments. Its object model comprises of 147 C++ classes. The *Client/Serve*r and *Server/Action Server* protocols have been implemented using the Socket API. For in-process multithreading, the libraries of Win32 API (for Windows NT) and the DCE-compliant *pthread* package (for Unix) were used. A free demonstration package of TEMPO Server for Windows NT (and Windows 95) is available for downloading from our web site (http://www.vtt.fi/tte/projects/tempo/). The following subsections present selected topics of the implementation.

### 3.1 Architecture Overview

The TEMPO Server is a prototype active database system equipped with C-fuzzy triggers. It has been designed for maintaining rapidly changing temporal data acquired from an industrial process. The requirements of such environments have been analyzed in a previous case study [WKP96] which have led to the development of RapidBase—an active time series database system. TEMPO Server has been built by extending RapidBase. In terms of external characteristics, the RQL language of RapidBase has been extended with constructs to define database objects related to rule set processing: linguistic types and terms, rules and rule sets. The trigger syntax has been modified to include rule set calls in the trigger conditions (predicates). The syntax of the resulting language, RQL/F, is demonstrated in the examples appearing in this paper.

In the true spirit of contemporary database systems, all the objects of newly introduced database types may be maintained dynamically by the user: they can be created, dropped or modified at any time. In the same time, the referential integrity constraints among the objects are maintained by the system.

Figure 2 depicts the essential components of the TEMPO Server. All the parts shown are comprised within a single operating system process. The server process maintains network connections to client programs submitting RQL/F requests and connections to programs responsible for executing triggers actions (such programs are called *Action Servers*). Following the RapidBase operation model, TEMPO Server supports external trigger actions of arbitrary semantics. The unit of concurrency control and recovery is a single RQL/F command.

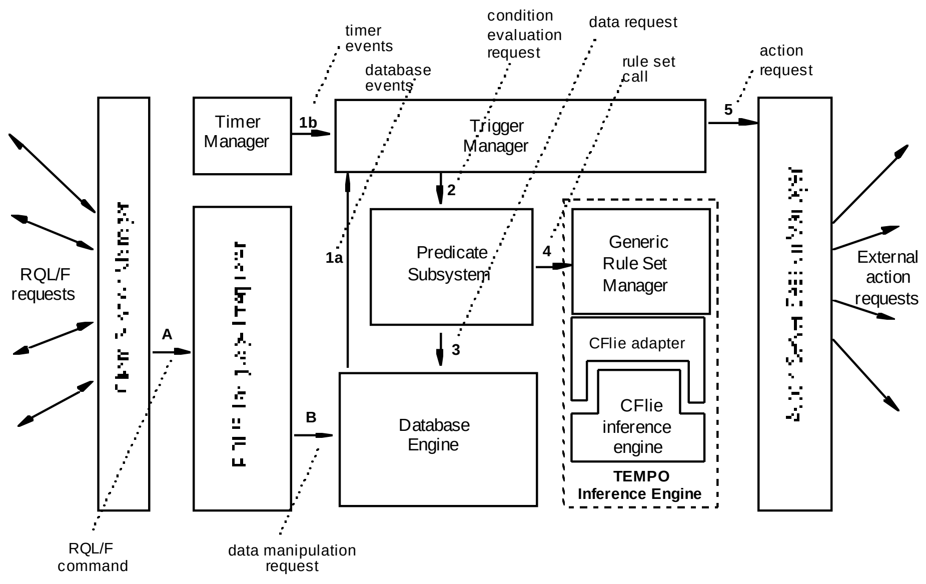

**Figure 2**. General architecture of the TEMPO Server.

Brief descriptions of the TEMPO components are given below:

- The *Managers* are entities responsible for maintaining, persistently, certain objects. The requests to create and modify such objects are directed, to the managers, by the *RQL/F Language Interpreter* (the requests are not shown in the picture).

- The *Database Engine* is a low-level main-memory-based data manager maintaining database table objects and indices.
- The *Predicate Subsystem* provides services to create pre-compiled predicates of the complexity allowed by the RQL/F language. The predicate objects themselves are managed either by the interpreter (if they are related to data manipulation commands) or by the Trigger Manager (if they are a part of a trigger definition).
- The *TEMPO Inference Engine* comprises a generic component and a specific third-party fuzzy inference software (CFlie[2]).

As compared to RapidBase, the Inference Engine is the only entirely new component. Of the other components, only two had to be modified: the Predicate Subsystem—to be able to make use of the rule set objects, and the interpreter—to accept the new language syntax.

## 3.2 Run-time Trigger Processing

The arrows shown in the figure illustrate the advancing of the run-time trigger processing. A data manipulation command arrives from the network and is passed to the interpreter for processing (A) and it, subsequently, results in elementary data requests executed by the Database Engine (B).

A trigger is fired upon detection of a pre-defined event. Regular database events (of type INSERT, DELETE or UPDATE) are detected by the engine (1a). Alternatively, a triggering event may be generated by the Timer Manager (1b), as would be the case of a composite-event based trigger, like a timer trigger [WKP96]. If there is a condition associated with the fired trigger, a stored predicate is invoked (2). The predicate uses the services of the Database Engine (3) to get the current data for the predicate evaluation, including the data for the evaluation of the rule set. If a rule set call is specified, in the predicate, the corresponding rule set is invoked (4) with the necessary input data. The defuzzified rule set call result is returned to the predicate which, in turn, returns the predicate evaluation result to the Trigger Manager. If the trigger condition is satisfied, the trigger action is requested (5).

## 3.3 Predicate Subsystem

The Predicate Subsystem of the TEMPO Server consists of a set of hierarchical, tree-like *predicate objects*, which are used for evaluating the predicates of RQL/F queries and trigger conditions. Fuzzy inference is fully encapsulated within these objects. Any predicate can contain a rule set call as a sub-expression anywhere in the tree structure. From the point of view of the system, a rule set call is simply an expression that returns a floating-point value. The return value can then be used in other predicate expressions, e.g. in a comparison. The arguments to the rule set call can be any expressions, e.g. column values, constants or even other rule set calls. The

[2] Originally developed as *Flie* at Institute of Robotics, ETH, Zurich, Switzerland; converted to C at Lab. for Concurrent Computing Systems, Swinburne Univ. of Technology, Hawthorn, Australia.

fuzzification of the argument values and the defuzzification of the result value are handled inside the Inference Engine. The Predicate Subsystem only needs to pass the crisp argument values to the Inference Engine and call the proper rule set function.

### 3.4 Inference Engine

TEMPO Inference Engine consists of two main modules: *Generic Rule Set Manager* (GRSM) and the *CFlie inference engine*. GRSM is responsible for storing and maintaining rule set objects by creating, modifying and deleting them. The CFlie inference engine is the software module that implements the inference method. We chose CFlie because of its speed and simplicity. However, CFlie can be easily replaced with some other inference engine package, since the GRSM is implemented in such a way, that it fully hides the underlying engine implementation from other parts of the system.

## 4 Performance

The performance of the fuzzy trigger system was measured using two kind of tests: i) performance of the TEMPO Inference Engine alone and ii) the total throughput of the TEMPO Server in trigger-intensive processing. The selected metrics allow us to asses the applicability of the approach to real life applications. The tests were run in a 133 MHz Pentium PC with 32 MB of main memory. The operating system was Windows NT 4.0. The database size was tailored to the size of available main memory so no page faults were occurring.

In the first test, the speed of the inference process was evaluated by way of full load of inserts to a database table, each causing a fuzzy trigger firing. The performance of the TEMPO Inference Engine was isolated from the overall performance of the system by running the tests with two trigger configurations. In the first case, only one trigger was used. In the second case, two identical triggers were used. No action requests were generated. The Inference Engine performance was calculated by dividing the speed difference of the two cases by the number of inserts. In both cases, five test runs with 10 000 inserts to a database table were used. To find out the non-fuzzy condition evaluation time, similar tests with normal triggers, without fuzzy predicates, were also performed.

With the rule set of eight rules, each consisting of two premise predicates and one conclusion predicate, one fuzzy condition evaluation took approximately 0,45 ms. With 64 rules the corresponding result value was 0,55 ms. With non-fuzzy triggers, one condition evaluation was approximately 0,1 ms (Fig. 3a).

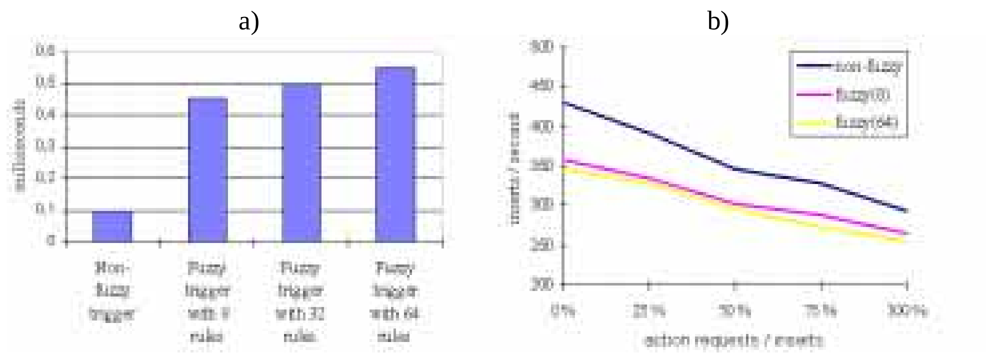

**Figure 3**. Results of a) the TEMPO Inference Engine performance, b) the TEMPO Server throughput performance.

The speed difference between non-fuzzy and fuzzy condition evaluation can be clearly seen. It is caused by a start-up overhead of the rule set processing. However, it should be noted that the number of rules in a rule set has only a slight effect on the performance.

In the second test, the full TEMPO Server throughput was measured (Fig. 3b). In addition to the condition evaluation time of a trigger, also the overhead caused by the insert request processing and action request transmission were taken into account. Three kind of tests were performed: i) with normal, non-fuzzy trigger, ii) with a fuzzy trigger using a rule set of 8 rules and iii) with a fuzzy trigger using a rule set of 64 rules. Five test runs with 10 000 inserts were used with each of the above cases. Among the test cases, the relative amount of inserts causing action requests, was varied.

The difference between non-fuzzy and fuzzy trigger throughput is notable. It can be explained with the speed difference of the condition evaluation of non-fuzzy and fuzzy triggers (see Fig. 3a). Increasing the relative amount of action requests from 0% to 100% of inserts, decreases the maximum insertion speed by approximately 25% in both non-fuzzy and fuzzy trigger tests. It is worth noting that the number of rules in a rule set has only infinitesimal effect on the overall performance.

## 5 Conclusions

We have investigated a seamless approach of integrating fuzzy logic rules with a traditional active database server. We have also demonstrated a feasible implementation of the proposed C-fuzzy trigger model. The performance results enforce our confidence in the approach as a way to alleviate the data explosion problem in data-intensive industrial applications. The benefits and the application domains of C-fuzzy triggers are summarised in [BW96] with some open research problems.

## References

[BW96] Bouaziz T. and Wolski A. *Incorporating Fuzzy Inference into Database Triggers*. Research Report No TTE1-2-96, VTT Information Technology, Espoo, Finland, November 1996.

[BW97] Bouaziz T. and Wolski A. *Applying Fuzzy Events to Approximate Reasoning in Active Databases.* In Proceedings of the IEEE International Conference on Fuzzy Systems, Barcelona, Spain, July 1-5, 1997.

[GKB+84] Gupta M.M, Kandel, A., Bandler, W. and Kiszka, J. (Eds.). *Approximate Reasoning in Expert Systems.* Elsevier Science Publishers, North-Holland, 1984.

[JP93] Jokiniemi, J. and Palomäki, A. *Real-Time Databases: A Needs Survey*. Research Report No. J-15, Lab. for Information Processing, VTT, Helsinki, February 1993, 13 pp.

[KY96] Klir G.J. and Yuan B. (Eds.). *Fuzzy Sets, Fuzzy Logic, and Fuzzy Systems: Selected Papers by Lotfi A. Zadeh.* In Advances In Fuzzy Systems-Applications and Theory, Volume 6, 1996, 821 pages.

[Lee90] Lee C.C. *Fuzzy Logic in Control Systems: Fuzzy Logic Controller-Part I and II.* In IEEE Transaction on Systems, Man, and Cybernetics, Vol. 20, No. 2, March/April 1990, pp. 404-435.

[Men95] Mendel J. M. *Fuzzy Logic Systems for Engineering: A Tutorial*. In Proc. of the IEEE, Special Issues on Engineering Applications of Fuzzy Logic, Vol. 83, No. 3, March 1995, pp. 345 - 377.

[PBW96] Pesonen, A., Bouaziz, T., and Wolski, A. *Case Study: Applying Fuzzy Triggers to a Drive Control System.* Research Report No. J-6/96, VTT Information Technology, Espoo, Finland, August 1996.

[WC96] Widom J. and Ceri S. *Introduction to Active Database Systems*. In Widom J. and, Ceri S. (Eds.), “Active Database Systems: Triggers and Rules For Advanced Database Processing”, Morgan Kaufmann, 1996, pp. 1-41.

[WKP96] Wolski A., Karvonen J., and Puolakka A. *The RAPID Case Study: Requirements for and the Design of a Fast-Response Database System*. Proceedings of the First Workshop on Real-Time Databases (RTDB'96), March 7-8, Newport Beach, CA, USA, pp. 32-39. (also at ftp://ftp.vtt.fi/pub/projects/rapid/case.ps).

[Zad75] Zadeh L.A. *Fuzzy Logic and Approximate Reasoning.* In Synthese, 30, 1975, pp. 407-428. (also in [KY96]).

[Zad84] Zadeh L.A. *The Role of Fuzzy Logic in the Management of Uncertainty in Expert Systems.* In [GKB+84], pp. 3-31.

[Zad89] Zadeh L.A. *Knowledge Representation in Fuzzy Logic*. In IEEE Transactions on Knowledge and Data Engineering, 1(1), 1989, pp. 89-100.