GPSQL Miner: SQL-Grammar Genetic Programming in Data Mining

Celso Y. Ishida, Aurora T. R. Pozo

Computer Science Department - Federal University of Paraná
PO Box: 19081, Centro Politécnico - Jardim das Américas
81531-990 Curitiba, Parana, Brazil
cishida@inf.ufpr.br, aurora@inf.ufpr.br
FONE: +55 (41)-267-5244 FAX: +55 (41)-361-3205

Abstract - The present work describes GPSQL Miner, a Genetic Programming system for mining relational databases. This system uses Grammar Genetic Programming for classification's task and one of its main features is the representation of the classifiers. The system uses SQL grammar, which facilitates the evaluation process, once the data are in relational databases. The tool was tested with some databases and the results were compared with other algorithms. These first experiments had shown promising results for the classification task.

I. INTRODUCTION

The current information age is characterized by an extraordinary expansion of data that is being generated and stored about all kinds of human endeavors. An increasing proportion of this data is recorded in the form of computer database. The availability of very large volumes of such data has created a problem of how to extract from them useful, task-oriented knowledge [1].

Researches from different areas recognize that a new generation of intelligent tools for automated data mining is needed to deal with large databases [2]. This work presents GPSQL Miner, a system that allows mining relational databases using a Genetic Programming (GP) approach and SQL features. A Database Management System (DBMS) does all memory and data managements.

This work deals with the classification task performed in Data Mining. Given a set of classified examples, the goal of the classification task is to find a logical description that correctly classifies new cases. In that sense, this logical description found can be considered a classifier. For a better visualization of the classification task, Table I and (1) show an example of the input received by a classification system and the group of rules generated by it [3].

In this paper, we explore the Grammar Oriented GP for the induction of classifiers. GP represents a classifier as an individual and its goal is to find the best classifier through the evolution process. Some advantages of GP favor the mining of large databases, since GP looks for the most promising solutions by doing a better scanning of the fetching space [4].

TABLE I
INPUT RECEIVED BY A CLASSIFICATION SYSTEM

Sex	Country	Age	Buy (Goal)
M	France	25	Yes
M	England	21	Yes
F	France	23	Yes
F	England	34	Yes
F	France	30	No
M	Germany	21	No
M	Germany	20	No
F	Germany	18	No
F	France	34	No
M	France	55	No

IF (Country = "Germany") THEN (Buy = "no")
IF (Country = "England") THEN (Buy = "yes")
IF (Country = "France" and Age ≤ 25) THEN (Buy = "yes")
IF (Country = "France" and Age > 25) THEN (Buy = "no") (1)

Other interesting features of GP are: the possibility of the system to work with invalid or inexact data, commonly found in the real world; it is easy to customize it for different applications by modifying operators, creating new operators or simply by changing the parameters.

However, Traditional Genetic Programming can generate invalid solutions and, the search for the best classifier is prevented or delayed. With the use of a Grammar, GP is able to generate only valid classifiers through the definition of the search space.

GPSQL Miner constructs classifiers using the GP approach and the SQL grammar. The GP technique was integrated with DBMS to make possible the manipulation of large volumes of data. This brings innovations with respect to previous works in the area, such as LOGENPRO [4] that uses Grammar GP to mine flat files.

The paper is organized as follows: in Section II we present related works on this area. Section III explains the Grammar Genetic Programming approach. Section IV shows the main features of GPSQL Miner. Section V illustrates the structure of the classifiers induced. Section VI describes the experiments and the results analysis. Finally, Section VII concludes the paper.

0-7803-7282-4/02/$10.00 ©2002 IEEE

II. RELATED WORK

Data mining and Grammar GP approaches are new research areas. Only a few works were developed integrating both. One of them is LOGENPRO (The LOgic to grammar based GENetic PROgramming system) [4]

LOGENPRO is a system for data mining developed to match the power of implicit parallel search of GP and the power of first order logic. The main features relating to the learning of rules are:

1) The system used an adjusted grammar to specify the structure of the rule. Each rule is an individual represented by its derivation tree and it has the following form: *if* ***antecedents*** *then* ***consequent***. In general, the ***antecedent*** is a set of attributes descriptors and the ***consequent*** is the goal attribute.

2) The dropping condition is a tailor-made genetic operator created especially for the task of learning rules. This operator automatically selects one descriptor attribute and changes it to ***any***. For example, the rule:

If sepal_length=0.5 and sepal_width between (1, 1.50) and petal_length = 50 then class = Iris-setosa

After to suffer the action from the operator can be modified to:

If sepal_length=0.5 and sepal_width between (1, 1.50) and ***any*** then class = Iris-setosa.

3) Creation of a specific fitness function to evaluate each rule. The fitness function of LOGENPRO is based on the support-confidence framework [5]. The function considers support (measures the coverage of rule) and confidence factor (is the confidence of the consequent will be true under the antecedents).

Our proposal is quite different from LOGENPRO [4]. We do not use confidence and support measures. GPSQL Miner's language is not based on rule schemata. Instead, the user determines the tables and attributes that must be used. Next, we summarize the main differences between the two approaches:

1) LOGENPRO represents an individual as a rule, while GPSQL Miner represents an individual as a full classifier (set of rules) for all the classes.

2) The grammar of LOGENPRO produces rules with fixed number of attributes. GPSQL Miner produces a grammar for the generation of classifiers formed by rules of any number of attributes.

3) In GPSQL Miner, the grammar is automatically created from the user specification; however, the user can modify it. In order to assist the grammar modification, an analysis file is generated automatically, which facilitates the construction of an adequate grammar for each problem.

Next section discusses the grammar genetic programming approach.

III. GRAMMAR GENETIC PROGRAMMING

Darwin's Natural Selection Theory shows that, in nature, the individuals that better adapt to the environment that surrounds them have a greater chance of survival [8]. They pass their genetic characteristics to their descendents and after several generations, only the best individuals survive.

GP is the application of these concepts in computers, to automatically induce programs. It was introduced by John Koza [9], based on the idea of Genetic Algorithms presented by John Holland [10].

Instead of a population of beings, in GP we have a population of computer programs. And the goal of the GP algorithm is to generate better solutions through natural selection. In order to do that, we start with a random population and, generation after generation, apply the so-called genetic operators to simulate the evolution process. A special heuristic function called fitness is used to guide the algorithm in the process of selecting individuals.

GP can also be seen as a search technique, applied on a universe of all possible computer programs that can be generated in a certain language. The outcome of this search is the program that better solves a given problem.

The purpose of this section is to present basic concepts of grammar-oriented GP. It starts with an overview of the GP Algorithm [9].

A. GP Algorithm Overview

First, an initial population of computer programs is randomly generated (generation 0). After that, the algorithm enters a loop that is executed. It consists of two major tasks:

- Evaluate each program, by the use of a fitness function, that is defined accordingly to the problem.
- Create a new population by the selection of individuals based on their fitness values and applying genetic operators, such as: reproduction, crossover and mutation.

At the end of each cycle, a new generation of computer programs is generated which substitutes the previous one. This process is repeated until a solution is found or until the maximum number of generations is reached.

B. Derivation tree

An abstract syntax tree is the most commonly used structure for representing programs in GP [9]. Other possible structures include graphs [11], linear genomes [12] and grammar-based derivation trees [4] [13].

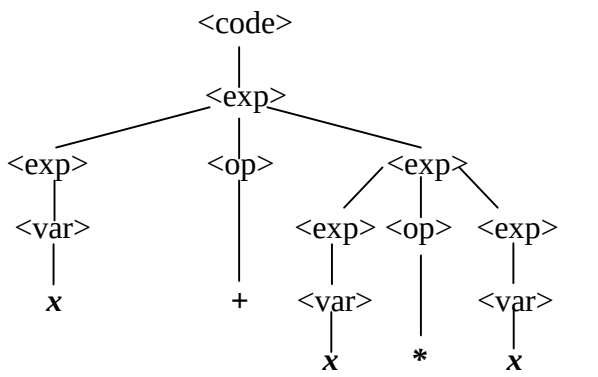
Derivation trees offer two important advantages over other structures. First, it allows the algorithm to be context-free, in the sense that it represents an abstraction from the programming language and the program's format. Second, it restricts the destructive effect of genetic operators over programs, not allowing the creation of syntactically incorrect code.

In this approach, the derivation trees are constructed based on a context-free grammar that is written in BNF (Backus-Naur form), as shown in (2).

```
<code>      ::= <exp>
<exp>       ::= <exp> <op>  <exp>
<exp>       ::= <var>
<op>        ::= + | - | * | /
<var>       ::= x
```
(2)

Each node of the derivation tree can either be a terminal, such as the variable "x", or a non-terminal "<exp>" (2). Unlike terminals, which have an independent value, a non-terminal depends on the evaluation of its components.

To generate a valid computer program from a given grammar, non-terminals have to be randomly chosen and substituted for their values until a point where all leaf-nodes on the derivation tree are terminals. These operations are called productions. Figure (3) shows an example of a derivation tree created with the grammar in (2).



(3)

C. *Applying the Genetic Operators*

Through the evolution process, genetic operators recombine programs by making modifications directly on their derivation trees.

In reproduction no change is made: the individual is simply replicated to the next generation. It is equivalent to the asexual reproduction of beings.

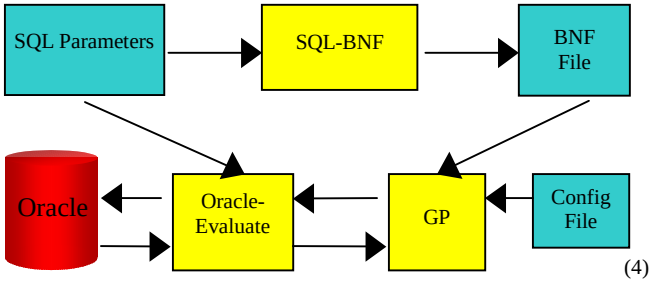
Mutation is the addition of a new segment of code (a new tree) to a randomly selected point of the program. This operation helps to maintain diversity in the population, which seems to be important to avoid early convergence of the algorithm to a single solution. However, there are studies showing that mutation can be ignored [9].

Crossover operation takes two parents to generate two offspring. A random point of crossover is selected on each parent and the sub-trees below these points are exchanged. It is equivalent to the sexual reproduction of beings. When grammars are used, the crossover operator is restricted and only allows the exchange of tree branches that have been generated using the same production rule.

Other operators can be defined, such as inversion (permutation), editing and encapsulation [9]. The next section discusses how these aspects were used in GPSQL Miner.

IV. GPSQL MINER

The name GPSQL Miner is a consequence of its main features: ***GP*** indicates the use of the Grammar Genetic Programming paradigm; ***SQL*** point out that the system use SQL Commands to represent the classifier. ***Miner*** recalls the central application of the system: data mining.



(4)

The aim of GPSQL Miner is the mining of databases by exploiting GP techniques, leaving DBMS responsible for data manipulation, searching performance and computer main memory management.

To accomplish the goals mentioned above, SQL-Miner has three central modules: SQL-BNF, Genetic Programming and Oracle-Evaluate, their functions and relationship are explained below (4).

- SQL-BNF: The tool initially reads two configuration files: config.gpm and SQL Parameters (*.par). The first file sets the main parameters in the Genetic Programming module.
  The SQL Parameter is essential because it defines the objective of the classification task and the input knowledge. This input language allows users declare what tables and attributes will be used in the searching, as well as, the root, non-terminals and terminals for constructing the rules. Using the SQL Parameter file, the system reads information of each column in the database and creates a BNF file.
- The GP module will use the BNF file to create the individuals and to apply the genetic operators. First, the GP module creates the initial population with N individuals (N is defined in Config.gpm file). For this first generation. After that, the module selects the

individuals with best fitness. If the best individual is the solution, the system displays it. If it is not, it makes a new population until the solution is found or the maximum number of generations is reached.

- Oracle-Evaluate: This module is responsible for fitness evaluation. The fitness is based on accuracy, that is, the percentage of the test set correctly classified, and it is computed by accessing the DBMS. The fitness evaluation is a simpler process, once the individual is based on SQL commands.

V. CLASSIFIER

One of the most important innovations introduced on this work is the evolution of SQL commands, as shown in the classifier (5). The character **S** is an abbreviation of SELECT and the **W** is an abbreviation of WHERE.

```
S  Iris-viginica
W I.sepal-length < 6.20
    AND I.sepal-width between 1.65 and 3.10
S  Iris-versicolor
W I.sepal-length < 6.95
S  Iris-viginica                                    (5)
```

In (5) we have 3 rules. After the **S** it has only the goal value (i.e. Iris-viginica and Iris-versicolor) and it is specified before the antecedent part (i.e. I.sepal-length < 6.20...).

The example uses two columns as part of the antecedent condition: **I.sepal-length** and **I.sepal-width**. It is important to note that the alias of the table's column is specified together with the column in the SQL parameters. Therefore, it is not necessary to have the FROM part of the command in the classifier. This also allows the creation of a smaller individual and facilates the evaluation process (fitness computation).

VI. EXPERIMENTS AND RESULTS

Experiments were conducted to verify how GPSQL Miner compares to other well-known systems that induce rules (concepts), including LOGENPRO. The result of other algorithms was found in [4] and in order to allow comparisons, we used the same methodology.

We employed three databases sets: Iris, Monk1, Monk3, extracted from the UCI Machine Learning Repository [6], that show considerable diversity of size, number of classes, and number and type of attributes. These data sets are in prepositional format and were imported for oracle databases.

The system was executed 25 times for all databases. To each run, the parameters were kept constant. All the parameters used to each database can be found in the appendix.

A. Iris Plant Database.

The Iris Plant Database has 3 classes: Iris-setosa, Iris-versicolor and Iris-viginica and 150 registers. In each run, 67% or 100 registers are randomly chosen to compose the training base and 50 others form the test base.

Results from this experiment are showed in Table II, there we can verify the mean accuracy of some algorithms. If we compared the GPSQL Miner with the similar system, LOGENPRO, our system is a little more precise (4%).

TABLE II
THE CLASSIFICATION ACCURACY OF DIFFERENT APPROACHES ON THE IRIS PLANTS DATABASE

Approach	Accuracy
LOGENPRO	91.04 %
C4.5	93.8 %
ID3	94.2 %
GPSQL Miner	95.04 %
Nearest Neighbor	96.0 %
Neural Net	96.7 %

B. Monk's Problem

Others experiments were performed on the Monk database [7]. The MONK's problems are a collection of three binary classification problems over a six-attribute discrete domain. There are three data sets for this problem [4]: Monk1, Monk2 and Monk3.

The monk1 data set is composed by [4]:

- Training set: 124 with 62 positive examples and 62 negative examples.
- The testing set contains 216 positive and 216 negative examples.
- There are no misclassifications.

The Table III, second column, shows the results of the algorithms. The GPSQL-Miner has 100% of accuracy for the training base, and 99.61% for the test set, a little lower than LOGENPRO.

The monk2 data set is composed by [4]:

- Training set: 169 with 105 positive examples and 64 negative examples.
- The testing set contains 190 positive and 142 negative examples.
- There are no misclassifications.

The third column of Table III shows the results of the algorithms. The system has 71.62% accuracy for the test set, against 60% for it is similar LOGENPRO.

The monk3 data set has [4]:

- Training set: 122 examples with 62 positive and 60 negative examples.
- The testing set contains 204 positive and 228 negative examples.
- There are 5% misclassifications.

Table III, fourth column, shows the results of the algorithms. This time, again, GPSQL Miner is little better than LOGENPRO.

We can see that, in general, GPSQL Miner has better results than LOGENPRO. However, there are some approaches with even better behavior, such as neural nets. In spite of this, the results are very encouraging and show that GPSQL Miner has a good induction behavior.

TABLE III
THE CLASSIFICATION ACCURACY OF DIFFERENT APPROACHES ON THE MONK DATABASE

Approach	Monk1	Monk2	Monk3
AQ15-GA	100 %	86.8 %	100 %
AQ17-DCI	100 %	100 %	94.2%
AQR	95.9 %	79.7 %	87%
Assistant Professional	100 %	81.3 %	100 %
Backpropagation	100 %	100 %	93.1%
CN2	100 %	69.0 %	89.1%
GPSQL Miner	**99.61 %**	**71.62 %**	**96.5 %**
ID3	98.6 %	67.9 %	94.4%
LOGENPRO	**100 %**	**60 %**	**95.4 %**

VII. CONCLUSIONS

In this work, we present GPSQL Miner, a system that uses the Grammar Genetic Programming paradigm for the classification task. We have pointed out some strong points of the tool: the representation of the classifiers in format of SQL commands; the exploitation of DBMS facilities and safety in handling data; and the use of GP to look for the most promising solutions and scanning of the fetching space.

We notice that the grammar GP approach was very important to make the implementation of GPSQL Miner possible. A database environment can have a lot of tables and searching for concepts without a guideline is very difficult. As the user must specify the relationship among the tables, GPSQL Miner is flexible because we can use other attributes to make relations between tables, besides the foreign and primary keys. We have observed that a grammar must be specific for each problem in order to get better results. Creating the BNF file based on the database itself helps to constrain the grammar accordingly.

Even though the system has not been tested with many databases, the experiment results shows that the tool has a very good behavior as an inductive system and that it is adequate for data mining and KDD tasks. Future works include (a) the application of the system in a larger number of domains, (b) exploration of the system parameters such as the crossover and mutation and (c) improvements in the BNF file.

REFERENCES

[1] Michalski, R.S.; Kaufman, K.A. Data Mining and Knowledge Discovery: A Review of Issues and a Multistrategy Approach. 1997.

[2] Mannila, H. Methods and Problems in Data Mining. Proceedings of International. Conference on Database Theory (ICDT'97), Delphi, Greece, January 1997, F. Afrati and P. Kolaitis (ed.), p. 41-55.

[3] Freitas, A; Lavington, S. H. Mining very large databases with parallel processing. Boston: Kluwer Academic, 1998. 208p.

[4] Wong, M. L.; Leung, K. S. Data Mining using Grammar based Genetic Programming and applications. Boston: Kluwer Academic, 2000.

[5] Agrawal, R; Imielinski, T.; Swami, A. Mining Association Rules Between Sets of Items in Large Databases. In Proceedings of the 1993 International Conference on Management of Data (SIGMOD 93), pp. 207-216.

[6] Blake, C. L.; Merz, C.J. UCI Repository of Machine Learning Data Bases. Available in <http://www.ics.uci.edu/~mlearn/MLRepository.html> Irvine, CA: University of California, Departament of Information and Computer Science, 1998.

[7] Thrun, S. B.; et al. The Monk's Problems: A Performance Comparison of Different Learning Algoritmos. Technical Report CMU-CS-91-197, Carnegie Mellon University.

[8] Darwin, C. On the Origin of Species by Means of Natural Selection or the Preservation of Favored Races in the Struggle for Life. Murray, London, UK, 1859.

[9] Koza, J. R. Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, 1992.

[10] Holland, J. H. Adaptation in Natural and Artificial Systems. University of Michigan Press. Second Edition: MIT Press, 1992.

[11] Teller, A. & Veloso, M. Program Evolution for Data Mining. The International Journal of Expert Systems, 8(3):216-236, 1995.

[12] Banzhaf, W., Nordin, P. et al Genetic Programming ~ An Introduction: On the Automatic Evolution of Computer Programs and Its Applications. Morgan Kaufmann Publishers, 1998.

[13] Whigham, P. A. Grammatical Bias for Evolutionary Learning. PhD thesis. School of Computer Science, University of New South Wales, Australian Defense Force Academy, 1996.

0-7803-7282-4/02/$10.00 ©2002 IEEE

APPENDIX – PARAMETERS FOR DATABASES

A. *IRIS Database*

Number of runs	25
Number of generations	50
Generations equal	20
Population size	300
Tournament size	13
Initialization method	2
Minimum tree height	3
Maximum tree height	30
Maximum crossover height	60
Crossover rate	50
Mutation rate	40
Elitist strategy	Yes
Training Set.	67%
Separate Train class	N
BNF file	gpsql.bnf
SQL Parameter file	iris.par
Precision	0.01

B. *Monk Problem*

1) *Monk1 data set*

Number of runs	25
Number of generations	50
Generations equal	20
Population size	300
Tournament size	3
Initialization method	2
Minimum tree height	3
Maximum tree height	30
Maximum crossover height	60
Crossover rate	50
Mutation rate	40
Elitist strategy	Yes
Training Set.	---
Separate Train class	---
BNF file	monk1.bnf
SQL Parameter file	monk1.par
Precision	0.001

2) *Monk2 data set*

Number of runs	25
Number of generations	50
Generations equal	10
Population size	300
Tournament size	3
Initialization method	2
Minimum tree height	3
Maximum tree height	30
Maximum crossover height	60
Crossover rate	50
Mutation rate	40
Elitist strategy	Yes
Training Set.	---
Separate Train class	---
BNF file	monk2.bnf
SQL Parameter file	monk2.par
Precision	0.001

3) *Monk3 data set*

Number of runs	25
Number of generations	50
Generations equal	10
Population size	500
Tournament size	13
Initialization method	2
Minimum tree height	3
Maximum tree height	30
Maximum crossover height	60
Crossover rate	40
Mutation rate	50
Elitist strategy	Yes
Training Set.	---
Separate Train class	---
BNF file	monk3.bnf
SQL Parameter file	monk3.par
Precision	0.01

APPENDIX – BNF FILE

A. *IRIS Database*

#productions
S -> <classifier>
<classifier> -> <r> <r> <r> <r> <r> <r> <r> S <hypotesis>
<r> -> S <hypotesis> W <cond>
<cond> -> <cond> AND <cond>
<hypotesis> -> 'Iris-setosa' | 'Iris-versicolor' | 'Iris-viginica'
<cond> -> I.SEPAL_LENGTH <rel_I.SEPAL_LENGTH>
<rel_I.SEPAL_LENGTH> -> <Noperator> <value_I.SEPAL_LENGTH>
<rel_I.SEPAL_LENGTH> -> between <value_I.SEPAL_LENGTH> and <value_I.SEPAL_LENGTH>
<value_I.SEPAL_LENGTH> -> 5 | 5.1 | 6.3 | 5.7 | 6.7 | 5.5 | 5.8 | 6.4 | 4.9 | 5.6 | 6 | 6.1 | 5.4 | 4.8 | 6.5 | 4.6 | 5.2 | 6.9 | 7.7 | 6.2 | 4.4 | 5.9 | 6.8 | 7.2 | 4.7 | 6.6 | 4.3 | 7.3 | 7.6 | 7.9 | 7.4 | 7.1 | 7 | 5.3 | 4.5
<cond> -> I.PETAL_WIDTH <rel_I.PETAL_WIDTH>
<rel_I.PETAL_WIDTH> -> <Noperator> <value_I.PETAL_WIDTH>
<rel_I.PETAL_WIDTH> -> between <value_I.PETAL_WIDTH> and <value_I.PETAL_WIDTH>
<value_I.PETAL_WIDTH> -> 3 | 2.8 | 3.2 | 3.1 | 3.4 | 2.9 | 2.7 | 2.5 | 3.3 | 3.8 | 3.5 | 2.6 | 2.3 | 2.2 | 3.7 | 2.4 | 3.6 | 3.9 | 2 | 4 | 4.1 | 4.2 | 4.4
<cond> -> I.PETAL_LENGTH <rel_I.PETAL_LENGTH>
<rel_I.PETAL_LENGTH> -> <Noperator> <value_I.PETAL_LENGTH>
<rel_I.PETAL_LENGTH> -> between <value_I.PETAL_LENGTH> and <value_I.PETAL_LENGTH>
<value_I.PETAL_LENGTH> -> 1.5 | 1.4 | 4.5 | 5.1 | 1.3 | 1.6 | 5.6 | 4 | 4.7 | 4.9 | 1.7 | 4.8 | 4.4 | 5 | 4.2 | 3.9 | 4.1 | 5.7 | 6.1 | 5.8 | 5.5 | 4.6 | 1.2 | 3.5 | 3.3 | 6.7 | 6 | 5.9 | 5.4 | 5.3 | 5.2 | 4.3 | 1.9 | 1 | 6.9 | 6.6 | 3.6 | 3.7 | 3.8 | 6.4 | 6.3 | 3 | 1.1
<cond> -> I.PETAL_WIDTH <rel_I.PETAL_WIDTH>
<rel_I.PETAL_WIDTH> -> <Noperator> <value_I.PETAL_WIDTH>
<rel_I.PETAL_WIDTH> -> between <value_I.PETAL_WIDTH> and <value_I.PETAL_WIDTH>
<value_I.PETAL_WIDTH> -> 0.2 | 1.3 | 1.5 | 1.8 | 1.4 | 2.3 | 0.3 | 1 | 0.4 | 0.1 | 2.1 | 2 | 1.2 | 1.9 | 1.6 | 1.1 | 2.5 | 2.2 | 2.4 | 1.7 | 0.5 | 0.6
<Noperator> -> = | != | > | < | >= | <=