ANNALS OF THE FACULTY OF ENGINEERING HUNEDOARA
2005, Tome III, Fascicole 3

THE MU – PUZZLE FORMAL SYSTEM: THEOREM GENERATION AND PROVING

BALAN Mihai, BALAN Mirela Ramona

[1]DENMARK TECHNICAL UNIVERISTY
ANKER ENGELUNDSVEJ 1, BUILDING 101A, 2800, KGS.
LYNGBY, DENMARK

ABSTRACT

This paper introduces a formal system in a form of a little puzzle. "Can you produce MU?" is the puzzle. An initial string of characters: MI is supplied. One has to generate all theorems in the MU – Puzzle starting from the given string. A tree of theorems is created. In case of theorem proving, one has to prove that giving a theorem, an axiom and the MU - Puzzle formal system, the given theorem either belongs to the given formal system or not. The results of the experiments are analyzed and interpreted. Optimized solutions are proposed by jumping out from the M-mode (machine) into the I-mode (intelligent).

1. INTRODUCTION

The purpose of this paper is to analyse and discuss theorem generation and proving for a formal system i.e. *MU Puzzle,* in Artificial Intelligence. Standard ML is used as the implementation language.

The first part of this paper analysis the *Theorem Generation*. All theorems in the MU Puzzle are generated starting from a given axiom (*MI*). An infinite number of theorems can be generated in a formal system. In the *MU Puzzle* this is the case when an axiom starts with *M*. Thus, one needs to use a variable for specifying the maximum number of generated theorems. The *breadth – first search* algorithm is used for generating all possible theorems. Different experiments are performed. The number of generated theorems and the time needed for this process are compared and analyzed. A tree of theorems is created and optimized solutions are proposed.

The second part deals with the *Theorem Proving*. By using a given formal system, an axiom and a theorem, one has to prove that either the given theorem belongs to the given formal system or not. In the case of the *MU Puzzle* formal system, the given axiom is *MI* and the theorem that

has to be proven is: *MU*. The *breadth – first strategy* is used for this purpose. Different experiments are performed. The results are analyzed and several properties of the made theorems are stated.

2. THEORETICAL CONSIDERATIONS

A formal system can explain most of the models and theories used in Artificial Intelligence. It includes a language for specifying formulas and a framework for manipulating these formulas. An *interpretation* that defines the meaning of the formulas in the formal system can be connected to that formal system. The formulas and the interpretation define a model of the problem domain. The three components: the formulas, the formal system, and the interpretation form a *theory* for the problem domain.

Using a formal system as the basis for Artificial Intelligence has several advantages. Knowledge about formal systems makes it much easier to understand logical theories, rewrite systems, production systems, and blackboard systems that are all very important in AI. [1]

Formalization is the act of creating a formal system, in an attempt to capture the essential features of a real world or conceptual system in formal language. [2] A formal system includes a formal grammar used for modelling purposes. A *formal grammar* is an abstract structure that describes a formal language precisely: i.e., a set of rules that mathematically describes a set of finite-length strings over an alphabet. Thus, a formal system consists of *an alphabet, a set of well-formed formulas in the alphabet, a set of axioms, a set of inputs and a set of deduction rules*. The first two elements define the *formal language* (*L*). The set of well-formed formulas can be given as a set of rules for forming formulas. A rule has a left-hand side and a right-hand side separated by a '□'. The left-hand side is often called the *conditional part*. The right hand side of the rule is called *conclusion*. Often the deduction rules have only one conclusion. [1]

A formal system is used for making *deductions* i.e. a process where the input is a formal system and a formula. The purpose of the deduction process is to determine whether the formula can be obtained from the axioms either directly or using the deduction rules. A deduction process consists in several steps. In each step a deduction rule is applied to formulas e.g. axioms or previous obtained formulas (premises P_i , I = 1..n), getting new formulas (conclusions C_i j = 1..m). [1]

$$P_1, P_2, \ldots, P_n \vdash_R C_1, C_2, \ldots, C_m$$

A *proof* in a formal system is a finite sequence of formulas $F_1, F_2, \ldots, F_n$, where F_i is either an axiom or a formula being deduced from the set { $F_1, F_2, \ldots, F_{i-1}$ } by a deduction rule. [1]

A *theorem*, $T \in F_s$, is a formula for which there is a proof, where T is F_n. [1]

The MU Puzzle is an example of a formal system - "*Can you produce MU?"* is the puzzle. [3] The goal of the MU Puzzle is to determine whether

MU is a theorem in the above system, given that *MI* is the only axiom. This paper tries to give an answer to this issue.

The MU Puzzle is defined as a formal system, in the following way:

- an alphabet : {*M, I, U*}
- formulas: any set of characters in the alphabet resulting in a string.
- a set of axioms : {*MI*}
- a set of deduction rules :
  - *Rule 1: xI → xIU* — given a string that ends in *I*, it is possible to add an *U* at the end.
  - *Rule 2: Mx → Mxx* — given a string that begins with *M*, any string that follows M can be duplicated.
  - *Rule 3: xIIIy → xUy* — given a string that contains *III*, a new string is created by replacing *III* with *U*.
  - *Rule 4: xUUy → xy* — given a string that contains *UU*, a new string can be created by deleting *UU* from the string.

where x and *y* represents any character or string (possibly empty) from the alphabet.

None of the rules can be used backwards. Given a particular string e.g. *MUIIU* of elements from the alphabet and only one axiom, *MI*, one can asks whether the given string is a theorem in the system or not. This can be proved by finding a sequence of deduction rules that leads from the axiom to the desired theorem.

$$MI \xrightarrow{R2} MII \xrightarrow{R2} MIIII \xrightarrow{R1} MIIIIU \xrightarrow{R3} MUIU \xrightarrow{R2} MUIUUIU \xrightarrow{R4} MUIIU$$

3. THEOREM GENERATION

A SML computer program can be used for generating all possible theorems starting with the given axiom: *MI*. The algorithm can do that in a very methodical way:

- *Step 1* **-** Apply every applicable rule to the *axiom MI*. This gives two new theorems: *MIU* and *MII*.
- *Step 2* **-** Apply every applicable rule to the theorems produces in *step 1*.
- *Step 3* **-** Apply every applicable rule to the theorems produces in *step 2*, etc …

Thus, the algorithm generates a tree of theorems (fig. 1) where each branch corresponds to using a rule that creates a new theorem from an old one. The *breadth – first search algorithm* is used for generating all possible theorems. The algorithm guarantees that the first solution found is also the shortest one but it requires more working memory than depth - first search.

The formal system used for theorem generation is depicted in fig 2.

In order to generate all theorems, a *theoremGen* function is built. The first argument is the given theorem. This function returns a new list of theorems as a second argument. The rule names and the actual rules are the third parameter and the fourth parameter, respectively. The last parameter is the maximum number of generated theorems. Thus, an infinite number of generated theorems can be avoided. Each obtained theorem contains information about its generation. A deduction map is used to map from a theorem into a parent that is either an axiom or a list of parents.

The theorem generation process is done by calling the *generate* function. A deduction map is returned when either the maximum number of theorem is zero or the list of unused theorems is empty. This function takes an axiom and it returns new theorems that can be derived from the axiom. This function uses the previous *theoremGen* function to generate the theorems. The rule names and the actual rules are the third parameter and the fourth parameter, respectively. The generation process is done in a *first-breadth* way and it stops when the maximum number of theorems is reached or no new theorems can be generated.

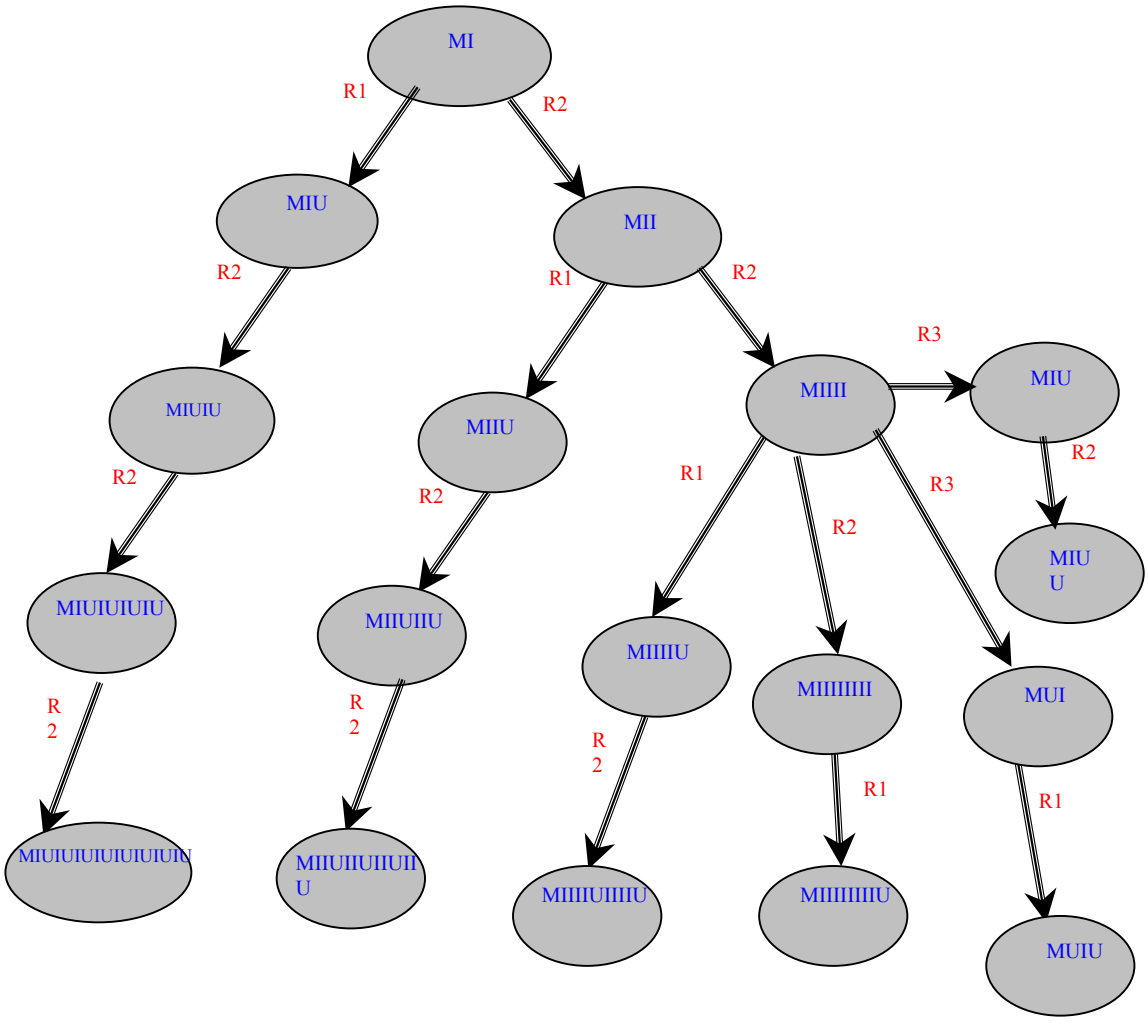

Fig. 1. Theorem Generation Tree

```
datatype F_rule_name      = R1 | R2 | R3 | R4                    (* 2.i *)
datatype Variable_symbol  = x | y                                (* 2.r *)
datatype Constant_name    = M | I | U
datatype Functor_name     = List                               (* dummy *)
datatype Term             = Fct of (Functor_name * Term list) |
                            Const of (Constant_name)|
                            F_v of (Variable_symbol) |
                            T_v of (Variable_symbol)
type     Rule_term        = Term
type     Formula_schema   = Rule_term list                      (* 2.m *)
type     Left_side        = Formula_schema                      (* 2.k *)
type     Right_side       = Formula_schema                      (* 2.l *)
datatype F_rule_def       = F_rule of (Left_side * Right_side)  (* 2.j *)
type     F_rules          = (F_rule_name * F_rule_def) list     (* 1.d *)

type     Formula          = Term list                           (* 1.c *)
type     Axiom            = Formula                             (* 1.b *)
datatype Formal_system    = FS of (Axiom * F_rules)             (* 1.a *)
```

Fig. 2. Formal System declaration for MU Puzzle problem

Several experiments are performed in this case. Different test cases are defined. Each test case is contained in a separate file (i.e. "*TGTestx.sml*", where x = 0 … 7). A different number of maximum calls is defined for each test case. The test case file skeleton is shown in fig. 3. The tests are performed on a Sun System running Solaris 9 operating system and Moscow ML (2.0).

The algorithm for the test case files can be described as follows: load the *Time*, *Timer* and *Int* libraries; start the counter; generate all theorems; get the number of milliseconds needed for the theorem generation and convert it to string; write the generated theorems and the corresponding time for this task in the corresponding result test case file (i.e. *TGx_test.txt*); close the result test case file.

The actual process of theorems generation can be displayed in a tree (fig. 1).

During the generating process several properties of the theorems are noticed:

- *All theorems begin with M*. Each new theorem inherits its first letter from an earlier theorem. Thus all theorems' first letter can be traced back to the first letter of the *MI* axiom.
- An *infinite number of theorems* can be generated unless a maximum number of theorems is specified.
- *Rule 2* generates an infinite number of theorems by using any axiom that starts with *M*.
- If the *axiom* doesn't start with *I* or *U* than the *theoremGen* stops.
- The number of generated theorems depends on the chosen axiom.

Details about the maximum number of calls, number of generated theorems and the time needed for that, are listed in *Table 1*

```
(* ------------- Test Case no. 2 -> Theorem Generation ------------- *)

(load "Time");
(load "Timer");
(load "Int");

val cycles = Timer.startCPUTimer();

toString(
let val ax = [Const(M), Const(I)]
    val rules = [(R1, F_rule([F_v(x), Const(I)], [F_v(x), Const(I),
Const(U)])),
                 (R2, F_rule([Const(M), F_v(x)], [Const(M), F_v(x), F_v(x)])),
                 (R3, F_rule([F_v(x), Const(I), Const(I), Const(I), F_v(y)],
                             [F_v(x), Const(U), F_v(y)])),
                 (R4, F_rule([F_v(x), Const(U), Const(U), F_v(y)],
                             [F_v(x), F_v(y)]))
                ]
    val rule_names = dom(rules)
    in generate([(ax, [])], [], rule_names, rules, 50)
    end
);


let
   val timp = (Time.toMilliseconds)(#usr ((Timer.checkCPUTimer)(cycles)))
   val timpStr =Int.toString(timp)
   val myoutstream = TextIO.openOut "TG2_test.txt"
   val afara = TextIO.output(myoutstream, "Test Case no.2 Theorem Generation
for max 50 calls\n ")
   val afara1 = TextIO.output(myoutstream, it)
   val afara2 = TextIO.output(myoutstream, "\n\n Time needed for theorem
generation is: ")
   val afara3 = TextIO.output(myoutstream, timpStr)
in
   TextIO.closeOut(myoutstream)
end;
```

Fig. 3. Theorem Generation for Test Case no. 2

Table 1

Test Case No.	Maximum No. of Calls	No. Of Generated Theorems	Time [ms]
0	5	3	20
1	10	4	20
2	50	18	20
3	100	37	30
4	500	231	180
5	1000	391	870
6	2000	700	3600
7	5000	1952	43010

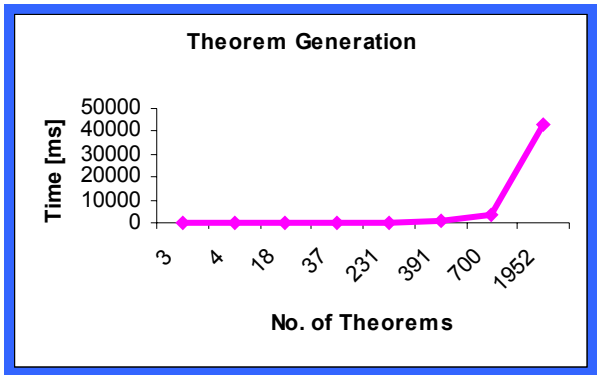

Fig. 4. Theorem Generation Chart Function on Time

4. THEOREM PROVING

4.1. Theorem Proving in the M-mode

Given a formal system, an axiom and a theorem one has to prove that the given theorem belongs to the given formal system. In the MU Puzzle case, the formal system is the MU Puzzle and the axiom is *MI*. This paper tries to prove that either *MU* is theorem in the system or not. One can use either *the first – depth* or *breadth – first* strategy to give the proof.

By using the *first – depth* strategy and *rule no. 2* the theorems can be generated infinitely. One may not find the theorem even if it belongs to the system. Even if it is found the proof could not be minimal.

Considering the search tree in fig. 3, by using the *first-depth strategy* all the theorems will be generated in the left branch of the tree that may have no end. If the needed theorem is *MII* (near the axiom but on the right branch) it might not be possible to find the wanted theorem because theorems are generated only in the left branch. Thus, this solution is not optimal for the problem.

```
(* ------------- Test Used for Proving the MU Theorem ------------- *)

let val ax = [Const(M), Const(I)]
    val rules = [(R1, F_rule([F_v(x), Const(I)], [F_v(x), Const(I),
Const(U)])),
                 (R2, F_rule([Const(M), F_v(x)], [Const(M), F_v(x),
F_v(x)])),
                 (R3, F_rule([F_v(x), Const(I), Const(I), Const(I),
F_v(y)],
                             [F_v(x), Const(U), F_v(y)])),
                 (R4, F_rule([F_v(x), Const(U), Const(U), F_v(y)],
                             [F_v(x), F_v(y)]))
                ]
    val rule_names = dom(rules)
in  proveTheorem([Const M, Const UI ,ax, rule names, rules, 3000)
```

Fig. 5. Tests of Theorem Proving

Bu using the *breadth – first strategy,* the desired theorem can be found during the theorem generation process. This strategy also proves that a minimal number of rules are used for finding the theorem. The complete search tree is displayed in fig. 3.

Several experiments are performed involving the *MU* formula. The code in fig. 5 is used for proving that *MU* formula is either a theorem or not.

The result of running this test is: *P_fail* – the *MU* theorem is not found. This can be interpreted either the formula is not a theorem in the MU Puzzle system, or the upper limit is too small to be found. Different values are chosen for the upper limit and the returned results are the same – the *MU* theorem may not belong to the MU Puzzle system.

4.2. Jumping out from the M-mode into the I-mode

By increasing the upper limit too much the *proof* could not be found. If the *MU* theorem doesn't belong to the system it will never be found no matter how big is the upper limit.

In this case we have to jump out from the *Machine (M) mode* into the *Intelligent (I) mode.* This can be achieved only by humans and not by a machine. Thus, we have to prove that *MU* doesn't belong to the system by finding a common property to all theorems in the system, but not *MU*.

Performing different experiments with different formulae (e.g. *MUU*, *MUUM, MUM*) one comes to notice that any formula that contains only the *M* and *U* letters cannot be a theorem in the MU Puzzle.

Considering the 3rd rule (*xIIIy* □ *xUy*) it can be proved that any formula in which the number of *Is* is a multiple of 3 is not a theorem. This is because of *Rule 3* that transforms three consecutive *Is* in a *U*. Thus, the following property can be formulated and proved inductively:

- *The numbers of Is in any theorem in the MU Puzzle is not a multiple of 3*

In order to prove it, one can consider that T is a theorem in the MU Puzzle system and T_i, i = 1 … n, are the corresponding proofs. We have to prove that the number of I*s* in T_i is not a multiple of 3. Let T_1 be the axiom – it contains only one I → not a multiple of 3.

Let us suppose that the number of I*s* is not a multiple of 3 for T_{i-1} and T_{i-1} is not the next step if T_i is not the axiom. Conform the four rules of the MU Puzzle system we have:

- *Rule 1:* The number of I*s* in T_{i-1} is the same as in T_i if T_i follows from T_{i-1}.
- *Rule 2:* The number of I*s* in T_i is the double of the number of the I*s* in T_{i-1} if T_i follows from T_{i-1}. All the time the number of I*s* in T_{i-1} or T_{i-1} is 1 or 2. The same for T_i.

- *Rule 3:* If T_i follows from T_{i-1} the number of I*s* in T_{i-1} is the number of I*s* in T_i minus 3. Thus the number of I*s* in T_{i-1} or T_i modulo 3 is the same.
- *Rule 4:* If T_i follows from T_{i-1} the number of I*s* in T_{i-1} is the same as in T_i.

Thus we prove that the number of I*s* in T_i and also T_n is not a multiple of 3. Because the only way to remove all I*s* from a theorem is to replace 3 I*s* with an U, and the number of I*s* in a theorem is never equal to 3, gives that MU is not a theorem for the MU Puzzle system.

It is an inherent property of intelligence that it can jump out of the task which it is performing, and survey what it has done; it is always looking for, and often finding, patterns. The jumping out can be very simple or very complex. [3]

5. CONCLUSION

A well-chosen representation may considerably reduce the amount of search needed to solve a problem. In case of a bad-chosen representation the problem becomes virtually impossible

In a first attempt to solve the MU Puzzle a theorem is simply represented by its string of letters. Four rules are used to expand nodes of the search tree. Each node in the search tree represents a theorem.

Thus, the MU Puzzle problem is a tree-search problem and consists in finding a path of rules that applied to the *MI* axiom gives the *MU theorem*. A *breadth-first search algorithm* is used.

Instead of concentrating on the selection of the best possible search algorithm, one can try to optimize the chosen representation. For the MU Puzzle, a better representation involves an extra descriptor of knowledge per theorem.

This is a Boolean item (*HasTripleI*) that indicates whether the total number of I*s* in a theorem is a *multiple of three*. We can now verify that each of the four rules creates new theorems with *HasTripleI'*s value equal to that of the theorem it is created from.

The observation that *MI (false)* and *MU (true)* have unequal *HasTripleI* values is sufficient to prove that *MU* is not a theorem.

Choosing a knowledge representation in problem solving is mostly domain-specific. General techniques, e.g. abstraction, proof successful by understanding the domain under investigation.

It is important to select a search algorithm that will find a solution, if it exists, in an efficient manner: programming time, calculation time and the required amount of working memory.

It is an inherent property of intelligence that it can jump out of the task which it is performing *(M-mode),* and survey what it has done *(I-mode*); it is always looking for, and often finding, patterns [3].

REFERENCES

[1.] ØSTERBY, T., *Artificial Intelligence*, Denmark Technical University, 2003.
[2.] http://www.wordiq.com
[3.] http://www.norreg.dk/tok/math01.htm
[4.] www.wikipedia.com
[5.] http://www.rdegraaf.nl/index.asp?sND_ID=350452
[6.] http://www.cse.buffalo.edu/~rapaport/510/formalsystems.html
[7.] http://www.esuc.ucla.edu/
[8.] http://home.planet.nl/~faase009/MIUpuzzle.html