Meta-explanation in a Constraint Satisfaction Solver

Jacques Pitrat
LIP6 Université Paris 6
pitrat@lip6.fr

Abstract

Meta-explaining is useful to a human or an artificial researcher in order to learn how to use its knowledge efficiently. For instance, when the system RESEARCHER solves the same problem using two different methods, it can meta-explain why it finds a more elegant solution with one of them. We will describe how this system builds and uses a meta-explanation.

Keywords: explanation, meta-explanation, problem solving, constraints

1. Why meta-explanation is useful

With the solution to a problem, we are receiving an explanation that convinces us it is right; however we are often unsatisfied, because we do not only need to be convinced, we hope to understand how the solution could have been found. Moreover we want to learn to find solutions; as we did not find this solution, we want to know how a clever solver came to the idea and built up its main steps.

Proving there is an infinity of prime numbers shows well this need. The proof is a reductio by absurdum: assume that the set of prime numbers is finite, then some prime number N would be greater than all the other primes. Consider N!+1; if it is a prime, there is a contradiction as it is larger than N. If it is not a prime, it has at least a prime divisor P. As every number less than N does not divide N!+1, P is greater than N and one still has a contradiction. This explanation is crystal clear, and we are convinced there is an infinity of prime numbers. However, we are unsatisfied: certainly most steps of the proof are natural, for instance try a reductio by absurdum, consider both possibilities for N!+1 (it is prime or not), show that each number less than N does not divide N!+1. However the key step is not easy to find: how someone came across the idea to consider N!+1?

In describing how we find the idea to consider some step of a proof, we do not explain but we meta-explain. For instance we indicate the elements that enable us to choose to apply some derivation rule to some arguments, the execution of this rule giving a new result that appears in the explanation. While an explanation bears on the use of knowledge, a meta-explanation bears on the use of meta-knowledge which is here knowledge on efficiently using the derivation rules.

Math students are often disheartened because their teacher only gives an explanation, few teachers offer in addition a meta-explanation which would enable them to learn how to find new proofs. They are convinced that to find a proof is a gift restricted to some mathematicians; they do not believe they will be able to find such proofs because they do not see what kind of method could give them the possibility to consider, among all the possible numbers, a number as useful as N!+1.

Meta-explaining is also useful for AI systems: if we want them finding more easily interesting solutions, we must give them the possibility to understand why, at each step, it is natural to consider the useful derivation rule. It is also important that they could understand, in case of failure, why they have not considered the good step when trying to find some solution or why they have considered it too late, favoring many other rules that could have been either discarded or considered much later.

However, with some systems, there is nothing to meta-explain because they systematically apply all the possible derivation rules with all their possible arguments: the only meta-explanation that can be given for considering a rule is that it was in the set of known rules. One can give an interesting meta-explanation only if the system carefully chooses a rule and its arguments at each step and the meta-explanation describes this choice. Thus, to meta-explain one must access the expertise that controls the utilization of the rules. This is easier for a system if it is given in a declarative form; this is possible although such an expertise contains procedural information, knowledge about using knowledge. Moreover when this procedural meta-knowledge is given in a declarative form, it is also easier for a system to learn it since its elements can be found independently. The interest of giving procedural meta-knowledge in a declarative form is known since a long time in AI [1,2,3,7,8].

I am developing the system RESEARCHER which features some of the activities of a human AI researcher. RESEARCHER includes the subsystem MALICE that solves constraint satisfaction problems using an approach similar to ALICE [4]. It has a set of derivation rules that enables it to decrease the size of the search space. Only when no more rule can be executed, it backtracks. In that way, MALICE finds elegant solutions, developing a very small tree, often smaller than the tree developed by good human problem solvers. MALICE receives declarative meta-knowledge to choose among the rules, I call it a method. I defined such a meta-expertise, called my method, and RESEARCHER generated several other meta-expertises; one of them, called here RESEARCHER's method, is considered in this paper because it is a good compromise between the elegance of the solutions and the time required to find them.
I will describe one of the many modules of RESEARCHER, the one which can understand why elegant solutions found with a problem solving method were not found with another one. It is particularly interesting to compare the results found using my method and RESEARCHER's method. Thus it can meta-explain why a blunder is made when MALICE uses one of these problem solving methods and not when it uses the other one. This paper will describe how this possibility is implemented.

First we will describe the structure of the problem solver that can use various problem solving methods and the structure of the meta-knowledge that controls the use of the rules. We will show the difference between the trace that is useful for explaining and the meta-trace that is useful for meta-explaining. Then we will show how the system gives meta-explanation such “Why has the system found, with some method, a more elegant solution than with another one?” and we will give two examples of such meta-explanations Finally, we will see that it is possible to implement other kinds of meta-explanations.

2. A general problem solving system

As with ALICE [4], for solving a problem one must find one or several correspondences between one or several unknown sets; each correspondence has a departure and an arrival set. The goal is to find certain links between the elements of these two sets that satisfy all the constraints. At the start, all the elements of both sets are linked with possible links. Given derivation rules can eliminate some possible links, define certain links, create new constraints, find that there is a contradiction, and so on. An “unknown” is an element of the departure set that has not yet a certain link toward an element of the corresponding arrival set. If F is a correspondence and if its departure set is the integers from 0 to 9, F(2), F(7) and F(9) are examples of unknowns. The subsystem that solves problems in RESEARCHER is called MALICE because it uses methods similar to those of ALICE.
A set of derivation rules has been given to RESEARCHER; with these rules MALICE restricts the search space. When it has no rule to apply, it chooses an unknown, gives it one of its possible values and executes the rules that can now be applied. Backtracking is made only in the last resort, MALICE tries to solve problems with trees as small as possible; an elegant solution is one with a very small tree.
As one gives a set of rules, knowledge is given in a declarative way; meta-knowledge must be added, so that MALICE knows how and when to use these rules. RESEARCHER learns a declarative form of this procedural meta-knowledge: although how to use rules is procedural, it is possible to define it declaratively.
Let us sketch the behavior of MALICE: several kinds of events are defined and, for each of them, a set of rule triggers indicates which rules may be applied when such an event occurs and when MALICE has to execute these rules. Executing a rule may create new events, thus MALICE has new rules to apply and it carries on till it finds a contradiction, a solution or nothing to do; in this last case, it chooses an unknown, gives it one of its possible values and resumes the process. This is complemented with a monitoring expertise that restricts the use of some rules when their excessive utilization is shown to be harmful.

2.1. The rules

The main parts of a rule are its conditions and its actions. If all its conditions are true, one may execute its actions. The conditions are expressions such as F(5)=4 which is true if the element 5 of the departure set of correspondence F has a certain link toward the element 4 of the arrival set of this correspondence. Naturally, there are much more complex conditions which use the logical and algebraic usual connectives. Let us see three of these rules, which are given to RESEARCHER in a more rigorous formalism:

Rule R8: If C1 and C2 are equality constraints and if some unknown U is in both constraints, one has a new constraint when one substitutes in C1 the value of U given by C2.

Rule R39: If constraint C contains only two unknowns U1 and U2, and if, for some value V of U1, constraint C is false for all the possible values of U2, then one can remove V from the possible values of U1.

Rule R85: If constraint C is:

$$X_1+X_2+...+X_i+...+X_n = Y$$

where the X and Y are expressions, then one has the new constraint:

$$X_i \leq \max(Y) - \min(X_1+...+X_{i-1}+X_{i+1}+...+X_n)$$

The functions max et min give an upper and a lower bound of their argument.

2.2. The events

Ten kinds of events can prompt the system to apply rules. For instance the event LINK is the creation of a certain link between an element of a departure set and an element of an arrival set while the event UNLINK removes a possible link between two elements; it appears with rule R39. The event CONSTRAINT is the creation of a new constraint; this is the event of rules R8 and R85. The event START is triggered at the beginning of a resolution.

We can notice that finding a solution or a contradiction are not events in the preceding meaning: they stop the expansion of a path of the tree, but they do not prompt the execution of some rules.

2.3. The rule triggers

A set of rule triggers is linked to each kind of event. A rule trigger includes a set of conditions (possibly empty), a priority and the name of a rule. If one of the conditions is false, the associated rule will not be considered. If all the conditions are true, the rule will be considered depending on its priority. The priority may be a constant, but it may be a more complicated expression. For instance, the priority of the rules associated with the event CONSTRAINT often depends on the number of unknowns in this constraint or on the nature of its main connective: equality constraints are usually interesting as are those with few unknowns. When a rule has been accepted, it is stored with its priority and those with the highest priority are considered first. Two special priorities are ATONCE and EXCLUDED; in the first case the rule is immediately executed and, in the second one, it is not stored.

In that way, we have a declarative method for giving procedural meta-knowledge. As there is a set of rule triggers, this does not give an order for their use and it is easy to add new triggers. In order to explain why a rule was not executed after an event, there are three possibilities: this rule is not in a rule trigger of this event, one of the conditions of the trigger is false or the value of the priority is EXCLUDED. This will be used for generating meta-explanations.

A group of sets of rule triggers is the group of all the sets associated with all the events; I call such a group of rule triggers a “method”. With different methods MALICE has very different performances. It can use several methods: I defined one of them after many experiments while RESEARCHER defines two methods: an initial method built only from a close examination of the rules and a better method it learned from the explanation of the results of its initial method. When RESEARCHER asks MALICE to solve a problem, it indicates which method it must use. We will see later that it is interesting to understand the differences of MALICE's behavior with different methods.

2.4. Monitoring MALICE

Some rules, such as R8, are explosive: one can apply them to their result and that generates new constraints which can also be arguments to R8. Therefore, RESEARCHER includes a monitoring module that controls MALICE. For each constraint, the system defines an interest and, if it is lower than a threshold, the constraint is not kept. If there are too many constraints, the monitoring module will increase the value of the threshold and it also removes all the constraints which had been kept but whose interest is now less than the new threshold. Therefore there is a new possibility for not applying a rule to a constraint: the constraint was eliminated because its interest was too low. When MALICE solves the same problem with two different methods, the value of the threshold is not always the same because one method can generate more constraints. This may explain why a good solution was found with one method and not with another one: in the poor solution, the useful constraint was removed because the threshold was too high.

3. Trace and meta-trace

While MALICE solves a problem, it generates a trace and a meta-trace which will be used by RESEARCHER to explain and to meta-explain.
First, MALICE keeps all the events and, for each event, the rule, with its arguments, that produces it. With such a trace, RESEARCHER is able to explain a solution: it keeps all the events that are a contradiction, the elimination of a possible link or the creation of a certain link. If the rule tied to one of these events used a constraint, RESEARCHER includes in the explanation the event where this rule was generated and it recursively uses the same method for this event. Naturally, in the trace and in the explanation, it also keeps the indication that there was a backtrack: some unknown received the value V. With this explanation, a human being can be convinced that a solution is correct. RESEARCHER also uses its explanation, it finds out executing a rule is useful when that appears in the explanation. The other uses of a rule, which are in the trace and not in the explanation, are useless. Thus it can learn from these useful and useless uses of the rules when it is better or not to consider them.
MALICE simultaneously creates a meta-trace which includes all the elements of the trace, but also why it decides to consider a rule: it keeps this decision with the event that triggers it and with its priority. It also keeps the decision on not keeping a new constraint. From the event, RESEARCHER can scan the conditions and the expression defining the priority, so it can have a full information on the reasons for applying a rule and on its timing. For generating a meta-explanation, RESEARCHER can find, as for an explanation, all the steps that are useful, this time with their priority, but it also keeps the steps where it chooses to consider a useful event and it gives the information on the trigger that decides it. As most elements in a meta-explanation are evident, the system can make conspicuous for a human user the steps with a low priority, there are usually those that are surprising for us. This information could also be used by RESEARCHER for learning better trigger rules: if the priority of a useful derivation is low, it would be better to increase it and if the priority of a useless derivation is high, it would be better to decrease it.
We will now see another use of the meta-trace, understanding why an elegant solution has been found with one method and not found with another one.

4. Why an elegant solution has not been found?

RESEARCHER can ask MALICE to solve a problem with one method among the several at its disposal; usually, with two different methods, MALICE generates different solutions. Naturally, it finds the same number of solutions, but the size of the generated trees may be very different. A solution is more elegant if its tree has less leaves: MALICE had to make less backtracks or there were less values to consider when it was backtracking. As RESEARCHER can find out why a solution is more elegant than another, a human or an artificial researcher may be able to modify some rule triggers in the "poor" method so that MALICE will generate more elegant solutions.

4.1. Finding a difference between two solutions

When both trees have the same number of leaves, RESEARCHER does not try to improve the corresponding methods. On the other hand, when one tree has less leaves than the other, there are certainly at least one inefficiency in the method that leads to the larger tree. It would be possible to find all of them but, for the present time, RESEARCHER tries to understand the reasons for only one of these inefficiencies; it happens that, when there are several inefficiencies, most of them usually have the same cause.
This difference between the sizes of the trees may be due to a different choice of the unknown for some backtrack: a poor choice for an unknown can significantly increase the size of the tree. Thus, RESEARCHER begins with looking whether the unknown at the first backtrack was the same in both trees. If it is not, MALICE solves again the problem with the presumed poor method with the commitment to choose the

same unknown as in the good tree for its first backtrack. If the new tree has the same number of leaves, RESEARCHER has understood the reason for the poor solution, this is due to a poor choice of an unknown when backtracking; moreover it knows a good choice and a bad choice for this unknown, this information would be very useful for a program learning to choose unknowns for backtracking. If the number of leaves is always different, it will proceed comparing the good tree and the new poor tree.

At this step, RESEARCHER has two solutions where the trees have a different number of leaves and where the choice of the unknown at the first backtrack is the same. It finds for each tree a path starting at their root; for the larger tree, it stops when there is a backtrack where V has been chosen as a value for an unknown U while V was removed as a possible value for U in the smaller tree. For this last tree, the path stops when V is removed as a possible value for U. For instance, in the good tree MALICE has found that F(4) is different from 5 while in the poor tree, it backtracks on unknown F(4) and it considers 5 among its possible values. This may happen several times when comparing two trees; however RESEARCHER considers only the first time.

If there are some backtracks in these paths, the branch is chosen so that the same value is taken for each unknown; so for each backtrack RESEARCHER keeps the couple unknown-value which are the same for both trees. It defines a new problem from the old one where it adds as initial data the values of the unknowns in the preceding couples. MALICE solves this new problem with both methods; with the poor one it stops when it must backtrack and with the good one when it finds that unknown U has not the value V. Naturally both meta-traces are kept.

To simplify, I assumed that the difference came from the fact that a value for an unknown was removed in one tree and not in the other. It may also happen that there is a contradiction in the good tree while MALICE backtracks in the poor one. This case is handled in the same way.

4.2. Defining a “Why not?” meta-explanation

When both meta-traces have been generated, RESEARCHER compares them. First, it meta-explains the good step in the good tree: using the method described section 3, it keeps all the elements that were necessary to state that unknown U had not the value V (or that there was a contradiction). Then it seeks the first element of this meta-explanation that is not in the meta-trace generated with the poor method: this gives the reason for one of the inefficiencies that appear in the poor solution. To help a human user, it also prints all the steps made in the meta-explanation of the good solution that are a consequence of this step and that finally lead to the elimination of a possible value or to a contradiction.

5. Two meta-explanations

We will give two examples of “Why not?” meta-explanations that RESEARCHER generated. Both problems are simple because it is easier to describe how the system works. However this is not the only reason, meta-explanations for simple problems are often the most useful. In the same way, when a human researcher tries to understand the inefficiencies of his system, he runs it with simple problems so that he can analyze the results easily; with simple problems RESEARCHER is also able to generate a meta-explanation which may be used to improve its method. MALICE can successfully solve problems with more than 10,000 constraints but meta-explanations for such problems are difficult to generate and difficult to use. It is is easier for RESEARCHER to meta-explain solutions for much simpler problems; moreover these meta-explanations are simpler so it is easier to understand why a better solution has not been found. When an inefficiency has been found in the solution of a simple problem, the method is improved and it will be used for all the problems.

5.1 A crypt-arithmetic problem

A crypt-arithmetic problem is defined by arithmetic operations between numbers where the digits in each number are replaced by letters. To different letters must correspond different digits and the first digit in each number must not be 0. In particular, a crypt-addition is the addition of two or more numbers with their result. We consider here the following crypt-addition:

SQUARE + DANCE = DANCER

Several formalizations of this problem as a set of constraints exist; for our example, I have introduced carries. To simplify, I do not give the constraints in the formalism of MALICE but in the more usual algebraic formalism. Instead of writing F('A') for the unknown representing the value of character 'A' in the wanted correspondence F, I only write A. In the same way, instead of R(3), value of the third carry, I write R3. Instead of a variary connective for representing a sum, I use the binary addition symbol '+'. The problem is defined with the six following constraints:

$$2*E = R+10*R5$$
$$R5+R+C = E+10*R4$$
$$R4+A+N = C+10*R3$$
$$R3+U+A = N+10*R2$$
$$R2+Q+D = A+10*R1$$
$$R1+S = D$$

with two additional constraints S≠0 and D≠0. This problem has only one solution. MALICE developed a tree with 17 leaves with the method I generated and a tree with only 10 leaves with the method generated by RESEARCHER. Thus RESEARCHER tried to understand why it was so good and where were the inefficiencies in the solution found with my method. Comparing both solutions, it found that, in both cases, MALICE begins with backtracking with unknown R4. In the branch where it assumes that R4=1, the good solution proves that the value of R3 must be 0 while the poor one starts another backtrack with unknown R3, considering values 0 and 1 successively.

Thus RESEARCHER has found two paths; generating a meta-explanation for the good solution and examining the meta-trace for the poor one, it finds that the first element in the meta-explanation that is not in the poor meta-trace is the decision on applying rule R8 so that it substitutes the unknown C in the constraint R+2*C=20 by its value taken from constraint A+N+1=C+10*R3. In the good solution, that gives the new constraint R+2*N+2*A = 18+20*R3. Then MALICE applies to this constraint the rule R85 and gets:

$$20*R3 \le \max(R+2*N+2*A) - 18$$

In both solutions, MALICE has proved that A, N and R are less than or equal to 8; as the values of these three unknowns are different, max (R+2*N+2*A) is equal to (6+2*7+2*8), that is 36. Thus 20*R3≤18 and R3≤0; this removes the value 1 for R3. MALICE was not able to generate the new constraint with my method and this gives one more leaf to the tree.

When one considers the triggers of rule R8 in both methods, one can see that the trigger I defined has one condition which the other trigger has not: the constraint C1, first argument of rule R8, must contain more than 2 unknowns. This explains why the good derivation was not considered since there are only two unknowns in the constraint R+2*C=20. I checked that it was the same reason that forbade MALICE to find elsewhere in this solution the constraint that could enable it to eliminate unnecessary backtracks. I had made a mistake when I put in the trigger the condition on the number of unknowns of constraint C1; I hoped to limit the number of executions of R8 because I wrongly believed that other rules would create the same useful constraints than when applying R8 to a constraint with only 2 unknowns.

5.2. Putting Queens on a board

The problem is to put N Queens on a NxN board so that none of them could capture another one. There are several possible formulations for this problem; here one wants to find a bijection F between the rows and the columns of the board. In that way, there is one and only one Queen on each row and each column. To ensure there is at most one Queen on each diagonal, there are as many constraints as there are couples of squares on the same diagonal. In the case N=6 there are 110 such constraints, for instance:

$$F(5)\neq 2 \text{ OR } F(3)\neq 4$$

which ensures that one cannot simultaneously have a Queen on square (5,2) and another one on square (3,4). Always with N=6, MALICE found with my method, that is with the method I generated, a solution with ten contradictions while, with a method found out by RESEARCHER, there were only 5 contradictions.

RESEARCHER begins with finding a path where there is a difference; in that path, for both solutions MALICE backtracked with F(1)=3 and F(3)=4. Then in the poor solution, it backtracks with F(5) while in the good one it proves F(5)≠1. Then it looks for the first element of the meta-explanation of the good solution that is not in the meta-trace of the poor one. This element is considering rule R39 applied to

$$F(4)\neq 2 \text{ OR } F(5)\neq 1$$

for the possible values of F(5). At this step, MALICE using either method has stated that F (4) cannot take values 3 and 4 (already taken by F(1) and F(3)), 5 ((4,5) is on the same diagonal as (3,4)) and 6 ((4,6) is on the same diagonal as (1,3)). Thus the value of F(4) can only be 1 or 2. Applying R39, MALICE can see the constraint is always false when F(5)=1: if F(4)=2 it cannot have simultaneously F(4)=2 and F(5)=1. Moreover, F(4) cannot take 1 as value since the value of F(5) is assumed to be 1 and F(4) and F

(5) must have different values. Thus F(5)≠1 since for F(5)=1 the constraint is false for all the possible values of F(4); it is no longer necessary to later backtrack with F(5)=1 as in the poor solution.
My method did not considered applying R39 to the OR constraint because one condition of the trigger of rule R39 requires that the main connective of the constraint is not OR. The four other unnecessary leaves in the tree of the poor solution are due to the same reason. Probably, I had wrongly included this condition because there was a family of problems where R39 was too often applied to OR constraints unsuccessfully. Wanting to speed up MALICE, I added more conditions to restrict the use of the rules, but the consequence was that, in some cases, it did not find a more elegant solution.

For both problems, the meta-explanation showed inefficiencies in the method. After correction, this method gives more elegant solutions for many other problems, even for the most difficult ones.

5.3. The results

RESEARCHER tried 200 times to find out why MALICE had found an elegant solution with a method and not with another one. In 65 situations, it saw that the reason was a different choice of the unknown for the first backtrack: when the choice was the same, the trees had the same number of leaves. In 5 situations it was not able to find a reason because there were too many different choices of the unknowns for backtracking. In 130 situations it correctly meta-explained why one method gave a more elegant solution than another one.
The reason is often that the conditions of some trigger are too strong, as we have seen with the preceding examples. It would be easy to remove these restrictions, but that could lead to a harmful consequence: there would be too many rule executions, slowing down the resolution. Moreover, if there are too many constraints, the monitoring module would increase the threshold for keeping the rules and MALICE would wrongly remove some useful rules. This is exactly the other reason for a poor solution: a constraint useful for finding the good solution is removed in the poor one because its interest is rated too low. For both methods, the definition of the interest of a constraint is the same, but one method may generate more constraints than the other; in that case, the monitoring module will raise the threshold for keeping constraints and useful constraints will not be kept. A solution to this difficulty could be to improve the definition of the interest of a constraint. The system has already the information for doing this since it knows, thanks to the meta-explanation, some constraints with an underestimated interest; however this is a further step which has not yet been implemented.
In most of cases, my method gave less elegant solutions than those found with the method generated by RESEARCHER. However, even when MALICE uses my method, it is already a very good problem solver; for instance, there are only two leaves in the tree it generates to solve the crypt-addition

DONALD+GERALD=ROBERT

A good human problem solver is happy if it develops a tree with five leaves to solve this same problem. I wanted to find solutions quickly, so I included many conditions in the triggers, the source of several mistakes RESEARCHER did not make. Thus my method usually finds faster the solutions while the method generated by RESEARCHER finds more elegant solutions, its trees have less leaves

6. Other ways to use meta-explanations

The system can answer Why not? Questions, but it would also be interesting to answer “Why so late?” questions. With some methods, MALICE may find an elegant solution, but it requires too much time: before the good step, it has executed many unnecessary rules. The reason may be that conditions must be added to some triggers which are not enough constrained or that the priorities must be modified, decreasing those of the unnecessary steps and raising those of the good steps. This can be found from executing only the method to improve: it is sufficient to consider all the steps in the meta-trace that does not appear in the meta-explanation. It is easy to find such useful information, but I have to implement a learning module using it.

Meta-explaining has two main interests: it explains to a human being why the useful steps of the solution were chosen so that he can learn to find solutions more quickly. It also gives an artificial system the information enabling it to learn how to use its knowledge efficiently: it may be possible to improve the learning of the conditions and the priorities of the rule triggers. As an explanation can lead to an Explanation Based Learning [5,6], meta-explanation could lead to an Meta-Explanation Based Learning that would enable a system to efficiently proceduralize procedural meta-knowledge given in a declarative form. The main difficulty is that what is good for a family of problems can be bad for another one. Thus, after each modification, a human as well as an artificial researcher must solve many problems to ensure that it is always right to use it. I had poorly made these verifications and that was the reason for my mistakes found by RESEARCHER in section 5. When such a learning component will be completed, the artificial researcher will have a huge advantage over a human one: it can launch and analyze much more verifications.

References

[1] W. Clancey (1983) The epistemology of a rule-based expert system – a framework for explanation. *Artificial Intelligence*, vol. 20, pages 215-251.

[2] R. Davis (1980) Meta-rules: reasoning about control, *Artificial Intelligence*, vol. 15, pages 179-222.

[3] J. Laird, A. Newell and P. Rosenbloom (1987) SOAR: an architecture for general intelligence, *Artificial Intelligence*, vol. 33, pages 1-64.

[4] J.L. Laurière (1978) A language and a program for stating and solving combinatorial problems, *Artificial Intelligence*, vol. 10, pages 29-127.

[5] S. Minton, J. Carbonell, C. Knoblock, D. Kuokka, O. Etzioni, and Y. Gil (1989) Explanation-based learning,: A problem solving perspective, *Artificial Intelligence*, vol. 40, pages 63-118.

[6] J. Pitrat (1976) A program learning to play chess, in *Pattern recognition and artificial intelligence*, Chen ed., Academic Press, pages 399-419

[7] J. Pitrat (1986) MACISTE: a system using meta-knowledge to efficiently use knowledge, *First IPMU*, pages 189-195.

[8] J. Pitrat (1988) Declarative knowledge to use declarative knowledge, in *Sixth International Conference on Mathematical Modeling in Science and Technology*, Rodin and Averla eds., Pergamon Press, pages 408-412.