MaLARea: a Metasystem for Automated Reasoning in Large Theories

Josef Urban
Dept. of Theoretical Computer Science
Charles University
Malostranske nam. 25, Praha, Czech Republic

Abstract

MaLARea (a Machine Learner for Automated Reasoning) is a simple metasystem iteratively combining deductive Automated Reasoning tools (now the E and the SPASS ATP systems) with a machine learning component (now the SNoW system used in the naive Bayesian learning mode). Its intended use is in large theories, i.e. on a large number of problems which in a consistent fashion use many axioms, lemmas, theorems, definitions and symbols. The system works in cycles of theorem proving followed by machine learning from successful proofs, using the learned information to prune the set of available axioms for the next theorem proving cycle. Although the metasystem is quite simple (ca. 1000 lines of Perl code), its design already now poses quite interesting questions about the nature of thinking, in particular, about how (and if and when) to combine learning from previous experience to attack difficult unsolved problems. The first version of MaLARea has been tested on the more difficult (chainy) division of the MPTP Challenge solving 142 problems out of 252, in comparison to E's 89 and SPASS' 81 solved problems. It also outperforms the SRASS metasystem, which also uses E and SPASS as components, and solves 126 problems.

1 Motivation and Introduction

In the recent years there has been a growing need for Automated Reasoning (AR) in Large Theories (ARLT). First, formal mathematical libraries created by proof assistants like Mizar, Isabelle, HOL, Coq, and others have been growing, often fed by important challenges in the field of formal mathematics (e.g. the Jordan Curve Theorem, the Kepler Conjecture, the Four Color Theorem, and the Prime Number Theorem). At least some of these proof assistants are using some automated deductive methods, and some of these libraries are (at least partially) translatable to first-order format [MP06b, Urb06] (like the TPTP language [SS98]) suitable for Automated Theorem Provers (ATPs). Also, when looking at the way how some of the hard mathematical problems like Fermat's last theorem and Poincare's conjecture were solved, it seems that difficult mathematical problems quite often necessitate the development of large (sometimes even seemingly unrelated) mathematical theories, which are eventually ingeniously combined to produce the required results. An attempt to create a set of notrivial large-theory mathematical problems as an initial benchmark for ARLT systems is the MPTP Challenge[1] (especially its chainy division), consisting of 252 problems (some of them with quite long human-written formal proofs) in a theory with 1234 formulas and 418 symbols which represents a part of the Mizar library.

[1] `http://www.cs.miami.edu/~tptp/MPTPChallenge/`

Today, there also seems to be a growing interest in using Automated Reasoning methods (and translation to TPTP) for sufficiently formal large "nonmathematical" knowledge bases, like SUMO [NP01] and CyC [MJWD06]. It seems that semantic ontologies and semantic tagging and processing e.g. for publications in natural and technical sciences are currently taking off, creating more and more applications for ARLT.

On the other hand, it seems to be quite a common thinking that the resolution-based ATP systems using usually some complete combination of paramodulation and resolution are easily overwhelmed when a large number of irrelevant axioms are added to the problems. Sometimes (actually surprisingly often in the AI and Formal Math communities) this is even used as an argument attempting to demonstrate the futility of using ATPs for anything "serious" in formal mathematics and large theories in general. There have been several answers to this problem (and to the general problem of the fast growing search space) from the ATP community so far. Better (i.e. less prolific while still complete) versions of the original calculi, and their combinations with even more efficient special-purpose decision procedures, have been a constant research topic in the ATP field. More recently, a number of heuristical approaches have appeared and been implemented. These methods include strategy scheduling (competitive or even cooperative, see e.g. [Sut01]), problem classification and learning of optimal strategies for classes of problems, lemmatization (i.e. restarting the problems only with a few most important lemmas found so far, see also [Pud06]), weakening [NW04] (solving a simpler problem, and using the solution to guide the original problem), etc.

The metasystem for ARLT which is described here falls into this second category of heuristical additions governing the basic ATP inference process. It is based on an assumption (often spelled-out by the critics of uniform ATP) that while working on problems in a particular domain, it is generally useful to have the knowledge of how previous problems were solved, and to be able to re-use that particular knowledge in a possibly nonuniform and even generally incomplete way. Again, this idea is not exactly new, and not only in the world of "very-AI-but-very-weak" experimental AR systems implemented in Prolog. At least the very efficient E prover [Sch02] has the optional capability to learn from previous proofs, and to re-use that knowledge for solving "similar" problems. In comparison to this advanced functionality of E, the first version of MaLARea is quite simple, and more suitable for easy experimenting with different systems. The machine learning is done by an external software, and one can imagine any reasonable learning system to take the place of the currently used SNoW [CCRR99] system. The features used for learning should be easy to extend, as well as the whole learning setting. It is quite easy to change the parameters of how and when particular subsystems are called, and to experiment with additional heuristics. In the next section we describe the structure of the metasystem in more detail, and then we show its current performance on the chainy division of the MPTP Challenge.

2 How MaLARea works

2.1 Basic idea

The basic idea of the metasystem is to interleave the ATP runs with learning on successful proofs, and to use the learned knowledge for limiting the set of axioms given to the ATPs in the following runs. In full generality, the goal of the learning could be stated as creating an association of some features (in the machine learning terminology) of the conjecture formulas (or even of the whole problems when speaking generally) with proving methods which turned out to be successful when those particular features were present. This general setting is at the moment mapped to reality in the following way: The features

characterizing formulas are just the symbols appearing in them. The "proving method" is just an ordering of all available axioms. So the goal of our learning is to have a function which when given a set of symbols produces an ordering of axioms, according to their expected relevancy with respect to that set of symbols. One might think of this as the particular set of symbols determining a particular sublanguage (and thus also a subtheory) of a large theory, and the corresponding ordering of all the available axioms as e.g. a frequency of their usage in a book written about that particular subtheory. There are certainly many ways how this setting can be generalized (more term structure and problem structure as features, considering ATP ordering and literal selection strategies as part of the "proving method", etc.). The pragmatic justification for doing things in the first version this way, is that while this setting is sufficiently simple to implement, it also turned out to be quite efficient in the first experiments with theorem proving over the whole translated Mizar library [Urb06, Urb04].

This central "deduce, learn from it, and loop" idea is now implemented using a simple (one might call it "learning-greedy") "growing axiom set" and "growing timelimit" policy. The main loop first tries to solve all the problems "cheaply", i.e., with the minimal allowed number of the most relevant axioms and in the lowest allowed timelimit. Each time there is a success (i.e. a new problem is solved), the learning is immediately performed on the newly available solution, and the axiom limit and timelimit drop to the minimal values (hoping that the learning has brought a new knowledge, which will make some other problem easily solvable). If there is no success with the minimal axiom and time limits, the system first tries with a doubled axiom limit until a certain maximal threshold (currently 128) on the number of axioms is reached. If there is still no success, the time limit is quadrupled, and the axiom limit drops to a small value again, etc. The system is now limited by the minimal and maximal values of the axiom and time limit, and also by a maximal number of iterations (currently 1000). That means that the system stops if either all the problems are solved (unlikely), or the maximal values of the limits are reached without any new solution, or the maximal number of iterations has been reached.

2.2 Detailed implementation

The first version of MaLARea is available at `http://lipa.ms.mff.cuni.cz/~urban/MaLARea/MaLARea0.1.tar.gz`. The distribution contains a main Perl script (TheoryLearner.pl), and a number of ATP and other tools used by the main script. These tools currently are:

- the `E` (version 0.99) prover, and the tools epclextract and eproof needed for getting a detailed TPTP proof from E
- the `SPASS` (version 2.2) prover [Wei01]
- the `SNoW` (version 3.0.3) learning system
- the `tptp4X` utility from the TPTPWorld distribution for transforming the TPTP format to other ATP formats (like DFG)
- the `GetSymbols` utility from the TPTPWorld distribution for extracting symbols and their arities from formulas written in the TPTP language

The set of theorem provers could be larger, the constraints that we use is that

- the provers should be reasonably efficient and reasonably orthogonal

- it has to be possible to run the prover in a "proof mode", getting from it the list of axioms which were actually used in the successful proof (this is necessary for the learning phase, and it is the main reason why Vampire [RV02] with its undocumented and hardly parsable proof output cannot be used)

- a future requirement might be that the provers should be easily parameterizable, making learning over their strategies possible (this is definitiely the case for E)

The input to the system is a set of files (problems in the FOF TPTP format, the system probably would not handle the CNF format now). The following two "large theory" assumptions are made about the input problems:

- that the names of formulas in the files are stable (i.e. one name always denotes the same formula in all files in which it appears)

- and that the symbols are stable (the same symbol in two files has the same intended meaning).

So far, the system has always worked with problems where each conjecture was assumed to be provable from its axioms (assumed SZS status [SZS04] Theorem), however it seems that having some possibly countersatisfiable problems would not hurt the system. For simplicity of the Perl processing, we require that each formula is written on just one line in the input file (this can be achieved by preprocessing with `tptp4X`), and that each file is named after its (exactly one) conjecture, by adding certain prefix and certain suffix (specified as parameters to the main script) to the conjecture's name. As an example (and the main target of the system so far), the 252 problems from the chainy division of the MPTP Challenge are included in the distribution.

Given a set of problems, the system starts by creating a main "specs" file, containing for every conjecture the names of the axioms which are available (in its problem file) for its proving. Similarly, the system creates (using the GetSymbols utility) a "refsyms" file, containing for each formula appearing in some problem the set of symbols appearing in it. These files are actually kept in memory (as hashes) during the whole processing. The formulas and the symbols are disjointly numbered. This later serves for communicating with the SNoW system, which assumes the features over which it learns to be represented as numbers.

One of the main data structures which the system maintains is a hash (`%gresults`) which for each conjecture (problem) keeps the list of all proof attempts conducted so far, and their results. The data which are exactly kept for one proof attempt are following:

```
[SZSStatus, NumberOfReferences, CPULimit,
 ListOfReferences, ListOfNeededReferences]
```

Where the `SZSStatus` tells the result of the proof attempt, `CPULimit` is the limit with which the proof attempt was run, `ListOfReferences` are the axioms used for that particular proof attempt, `NumberOfReferences` is their number, and in case of a successful proof (SZSStatus Theorem), the `ListOfNeededReferences` tells which of the axioms were actually used in the proof.

Before we start explaining how the ATP proof attempts and machine learnings are organized, we first note what is exactly meant by the term "running ATPs on a problem" in the rest of this paper, and what exactly is meant by machine learning in the current version of the system.

2.2.1 Running ATPs

All the available ATPs (now just E and SPASS) are run on a problem in order of their expected performance (that now means E first, SPASS second) with the given timelimit. They are run only in the fast "assurance mode", i.e. not with the slowdown caused by doing the additional bookkeeping necessary for printing the proof. As soon as one of the provers solves the given problem (this means that it determines that the problem's status is either Theorem or CounterSatisfiable), the "assurance mode" processing stops. If the result status is Theorem, the successful system is re-run in a "proof mode" with a timelimit of 300 seconds. The reason for raising the timelimit so much is that at this point we know that the system is capable of solving the problem, and we want to know the solution so that we can learn from it for solving the rest of the problems. This procedure could be in the future modified e.g. by quite standard strategy scheduling, i.e. running only those provers (with those strategies, and possibly appropriately modified timelimits) which seem (again from previous experience) to be most likely to succeed on the problem. Another useful extension (suggested by Geoff Sutcliffe) motivated by the experience with the SRASS system [SY07] would be addition of the Paradox [CS03] system to the chain of ATP systems, especially in the proof attempts when the number of axioms is decreased to minimum. That's because in these very incomplete specifications the frequency of the CounterSatisfiable result is quite high, and Paradox seems to be currently the strongest system for determining CounterSatisfiability. After the procedure stops (either successfully, or unsuccessfully, with all ATPs resulting in timeout), the result is recorded in the results hash (`%gresults`) in the format described above (that means in case of successful proof also recording the names of the axioms which were exactly needed for the proof). The format now does not keep the information about particular ATPs (and possibly particular strategies). This is not needed as long as some smarter machine learning for selection of ATPs and strategies is not done, however the information about which ATP solved which system can still be retrieved from the metasystem's standard output.

2.2.2 Machine Learning in MaLARea and its use for selection of axioms

As noted above, the SNoW system is used in the naive bayes mode for all learnings, mainly because of its speed and relatively good previous experience on the whole Mizar library (thousands of symbols and tens of thousands of formulas). After each ATP run on all (unsolved) problems, the information about all successful proofs found so far is collected in a format suitable for the SNoW system. As explained above, our goal is to learn the association of symbol sets to axiom orderings. This in practice means that one training example contains all the symbols of a solved conjecture, together with the names of axioms needed for its proof (the symbols are in the machine learning terminology the "input features", while the names of the axioms are the "output (or target) features", i.e. those features that a trained classifier will try to assign to a test example consisting of the input features). Then the classifier (a bayes network) is trained on this set of examples. This procedure is very fast with the SNoW system (seconds or less for the number of features and examples available in the MPTP Challenge problems). It is technically possible just to add the new examples to an existing bayes network trained on the results of the previous ATP runs, however until really large time-consuming learnings (this means really large theories like the whole Mizar library) are needed, this is not necessary. The trained classifier is then used to prune the axiom sets for the next runs. It means that we take all the unsolved conjectures, and create a testing example from each by taking all its symbols. The classifier is run on this set of test examples, printing for each example the

list of target features (axiom names) in order of their likelihood to be useful (as judged from the training examples, i.e. previous proofs). This order of axioms is then used to select the required number of axioms for the next run on the unsolved problems. I.e., provided that the next run will limit the number of axioms to n, we are looking for each problem for n axioms from the problem's specification whose ranking by the classifier is highest.

2.2.3 Initial proof attempts in MaLARea

Before the system enters the main loop it first does two special proving attempts, in order to generate the "initial knowledge". The motivation for both of them is quite heuristical, and both have been subject to experimenting.

The first proof attempt is quite expensive: it tries to solve all the problems in their original form (that means without restricting the axioms in any way) with the maximum timelimit (now 64 seconds). The motivation for these settings is to allow the ATPs to consider the full axiom space for each problem, giving a chance to ATPs internal "large problem" strategies. The heuristical justification for the high timelmit is that any proof found at this point is a proof considering all the available axioms, which is not true for the rest of the proof attempts done by the metasystem. Therefore the information obtained from this proof attempt is in a certain way fresh and unbiased by solutions already found. It can be compared to the mode of work of some mathematicians, who when entering a new field, first try to think about the field themselves, possibly creating fresh insights, and only after that consult other experts in the field and the available literature. It should be noted that this approach probably would not be feasible for an order of magnitude larger theories (e.g. for the ca. 40000 theorems and definitions in the whole Mizar library). On the other hand, there seems to have been progress in this capability of ATP systems in the recent years: several years ago one might have claimed that already 1000 axiom gives to any resolution-based ATP system no hope.

The second proof attempt is cheap: it uses the minimal timelimit (1 second), and a purely symbol based similarity measure to cut the number of axioms to the maximal axiom limit (now 128). The symbol based measure is now just for simplicity achieved again through the SNoW's bayesian classifier: For each formula (all axioms and conjectures) one training example for SNoW is created from the list of the formula's symbols, and from the formula's name (this can be explained by saying that each formula is useful for proving itself, or more precisely, for proving something with a similar set of symbols). The classifier is trained on these examples, and then evaluated on the symbols of each conjecture formula, providing for each conjecture the ordering of axioms according to their symbol overlap with the conjecture. A possible future extension could be to employ more elaborate similarity measures at this point, possibly also with a higher timelimit. One reason why we are not here as aggressive with the timelimit in comparison to the previous pass, is that the symbol based measure is taken into account in all the following passes, i.e. for all the learnings on successful proofs explained in Section 2.2.2, we also add the training examples saying that each formula can be proved by itself.

2.2.4 The main loop

After the two initial proving passes the system performs first learning (Section 2.2.2) on the successful proofs, and enters the main loop with the initial timelimit set to minimum and axiom limit set to maximum (this is quite arbitrary, it might as well be the minimal axiom limit). The main loop is now limited to a certain number (default 1000) of iterations (passes) (this is probably a bit redundant, but good for fast testing), and it will also stop

when the maximal time and axiom limits are reached without finding any new proof. Obviously, as is the case for the MPTP Challenge, it can also be stopped by a user or operating system after a certain overall timelimit (252 problems times 300 seconds for the Challenge, i.e. 21 hours). The loop starts by an ATP run (see Section 2.2.1) on all unsolved problems with the given axiom and time limits. Then the loop branches.

If no problem was solved by the ATPs, no new learning is done, the axiom limit is doubled (if it is smaller than the maximal axiom limit), and for each unsolved conjecture a new specification is created using the last learning results and the new axiom limit. In case the axiom limit is already maximal, we instead quadruple the timelimit, and set the axiom limit to the double of the minimal value (the minimal value seemed a bit too useless when running with higher timelimits). If both the axiom and time limit are maximal, we stop.

If a problem was solved during the last ATP run, learning immediately follows (in order to take advantage of the newly available knowledge). The time and axiom limits are reset to the minimal values (hoping that the new knowledge will allow us to solve some more problems quickly), and the loop continues.

2.2.5 Usage of previous results

As described above, the system keeps the data about all previous proof attempts. This is mainly used to avoid the proof attempts which do not make sense in the light of the previous results. The current implementation recognizes three such situations for a given unsolved problem:

- the suggested set of axioms is a subset of a previously tried set of axioms, whose result was CounterSatisfiable
- the suggested set of axioms is equal to a previously tried set of axioms, whose result was ResourceOut, and the suggested timelimit is less or equal to the timelimit of the previous attempt (note that this practically interprets ResourceOut as running out of time, which is the vast majority of cases especially with the low timelimits, however the implementation could be improved in this respect)
- all ATPs have previously unexpectedly failed (status Unknown) on the suggested set of axioms (regardless of timelimit)

Note that for checking the last two conditions it would be good to keep already now the detailed information about each ATP system's result, not just one summarized version for all ATPs as is being done right now. In the current implementation this is temporarily worked around by using the Unknown status only if it was the result of all (both) ATPs. If at least one system ended with the ResourceOut status, this is the status recorded in the result datastructure. Another practical problem is that the status Unknown is currently used also if an ATP does not obey the timelimit (specified to it as a parameter) with which it is run, and has to be killed by using the operating system's limit. Given the three policies described above, it seems better to use ResourceOut in such situations, which would make it possible to re-run the system with higher timelimit later. Fortunately, it happens only very rarely that both ATPs have to be killed by the operating system.

Especially the first condition is fulfilled quite often when the axiom limit is in its lower values (the minimum is four axioms). Since the goal of the system is to try as many reasonable axiom subsets as quickly as possible, we try to repair the "subsumed" axiom specifications by adding additional axioms (again according to their rating by the last learning) when the timelimit is minimal (1 second), and the additional check is thus

cheap. So in such cases, the advertised system's axiom limit is not observed (though the correct data are obviously kept in the results datastructure). It could be argued that this should also be done for the higher timelimits. This is quite hard to decide, and hopefully not much relevant to the overall performance of the metasystem. The current heuristical reason for not doing it, is that we are happy to "shake abundantly" the set of axioms when it is cheap (i.e. low timelimit), while we are trying to limit the higher timelimit runs only to the combinations of axioms which make most sense. On the other hand, since we grow the timelimit exponentially, it could be argued that the notion of cheapness applies to all but the highest timelimits.

2.2.6 More questions on MaLARea policies

The previous paragraph actually shows some of the hard (and interesting) heuristical choices which one has to make when experimenting even with such a simple kind of combined deductive/inductive reasoning system. Why do we (after the first two passes) "learn greedily", and always prefer learning and low timelimit to more ATP with higher timelimits? Would not the "tabula rasa mathematician" argument used for the initial expensive pass also justify a less greedy approach to learning (i.e. let the system learn something, but not all others' inventions at once, the ATPs might still come up with something relatively new)? Or wouldn't it pay to have even much more of the fast "reasonable axiom shaking" attempts instead of the later and expensive higher timelimit attempts? Why do we double the axiom size, and quadruple the timelimits, and why are their minima and maxima set to their current values? Some explanation of this is the experience with the (super)exponentially behaving ATPs that are used, however one might conjecture that the system should be relatively robust to small changes of these values and policies. Even more of such interesting questions are likely to appear if new components (lemmatization, weakening, conjecturing, defining, etc.) are added, and if the learning component becomes more sophisticated. Even now, simple as the whole setting is, it sometimes gives a strange impression of conducting a bit of exploratory Artificial Intelligence.

3 Results

As noted above, the system's main target so far has been the chainy division of the MPTP Challenge. This is a set of 252 related mathematical problems, translated by the MPTP system from the Mizar library. The conjectures of the problems are Mizar theorems, which were recursively needed for the Mizar proof of one half (one of two implications) of the general topological Bolzano-Weierstrass theorem. The whole problem set contains 1234 formulas and 418 symbols. Unlike in the "less AI" bushy division of the Challenge, where the goal is just to reprove the Mizar theorems from their explicit Mizar references (and some background formulas used implicitly by Mizar), the problems in the chainy division intentionally contain all the "previous knowledge" as axioms. This results in an average problem size of ca. 400 formulas. The Challenge allows an overall timelimit policy, i.e., instead of being forced to solve the problems one-at-a-time with a fixed timelimit of 300 seconds, it is allowed to use the overall timelimit of 21 hours in an arbitrary way for solving the problems.

The system was run on this set of problems in five differently parameterized instances, on a cluster of 3056MHz Pentium Xeons each with 1GB memory (the memory limit for all the ATP runs was always 800MB). Before these instances were run, E version 0.99 and SPASS version 2.2 were tested on the cluster in the standard 300 seconds timelimit setting. E has solved 89 problems, and SPASS has solved 81. This is quite similar to the MPTP

Challenge measurements[2] on Geoff Sutcliffe's cluster, which claim 36% (91) problems solved by E 0.99, and 31% (78) problems solved by SPASS 2.2 (the relative differences might be caused e.g. by different memory limits). The total number of problems solved by either E or SPASS is 104.

All the five instances of MaLARea shared the minimal timelimit set to 1 second, and the minimal axiom limit set to 4 axioms. The instances differed in the values for the maximal timelimit, and maximal axiom limit, which were as follows:

- 128_4s: maximal axiom limit set to 128, maximal timelimit to 4 seconds
- 128_16s: maximal axiom limit set to 128, maximal timelimit to 16 seconds
- 128_64s: maximal axiom limit set to 128, maximal timelimit to 64 seconds
- 64_4s: maximal axiom limit set to 64, maximal timelimit to 4 seconds
- 64_64s: maximal axiom limit set to 64, maximal timelimit to 64 seconds

The last instance unfortunately crashed (for unknown, probably cluster-related issues) after 18 hours. The 128_64s version was let to run even beyond the timelimit of 21 hours for a total of 30 hours (when it was stopped by the operating system), to see if there is any improvement in the later stages (which was not the case). The 4 second and 16 second instances have stoped themselves before the timelimit of 21 hours, because they reached their maximal axiom and time limits. The reason for running the very low (4 seconds) maximal timelimit instances was to find out how important is the long initial pass, and how the system performs in a "shallow thinking only" mode.

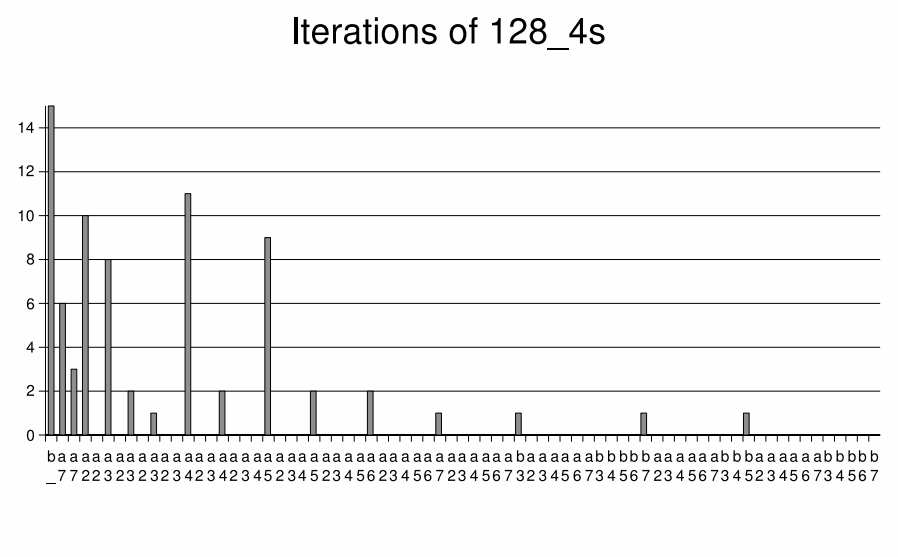
The following Table 1 summarizes the main results (the times are in minutes, last successful iteration is the last iteration in which a problem was solved). The Figures 1

description	solved	iterations	last successful iter.	time to stop	time to solve last
128_4s	131	73	62	300	270
128_16s	141	137	121	930	810
128_64s	142	127	108	1800	1160
64_4s	130	44	35	240	210
64_64s	136	77	62	1080	900

Table 1: Statistics for the five instances of MaLARea fighting the MPTP Challenge

and 2 show the iterations for all five instances and the gains in terms of solved problems. To make the scale readable on these figures, the timelimit is encoded as a letter (a,b,c,d), corresponding to the exponentially grown timelimits (1,4,16,64). The numbers (2,3,4,5,6,7) are the powers of 2 that should be used to get the axiom threshold (i.e. 4,8,16,32,64,128). The first pass in each figure uses an underscore instead of the threshold exponent, which means that the axioms were not limited in that pass. Instead of scaling the Y axis logarithmically, the value of the first most successful pass is cut on the figures, and given in their captions. Also note that the second and third passes are not the same, even though they have the same time and axiom limits. The second pass is the "symbol similarity only" pass, while the third one is the first in the main loop, i.e. the first which uses learning on previous successful proofs.

[2] `http://www.cs.miami.edu/~tptp/MPTPChallenge/Results/SVGResults.html`, `http://www.cs.miami.edu/~tptp/MPTPChallenge/Results/ChainyResults.data`



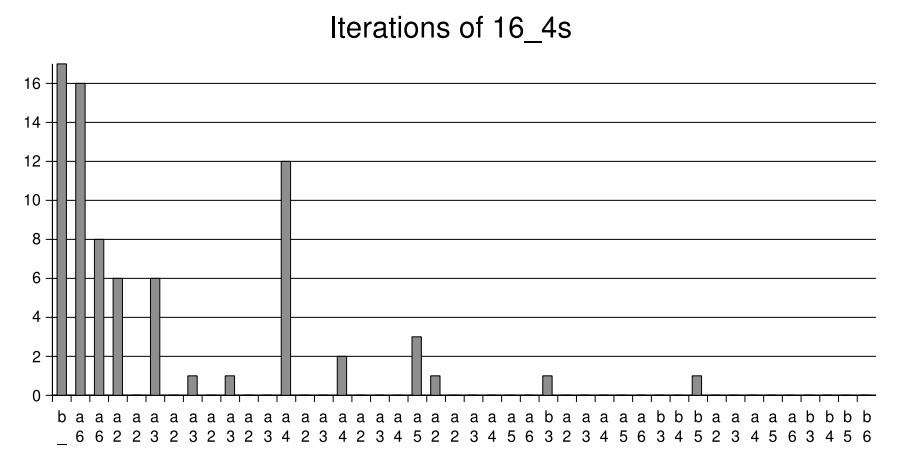


Figure 1: The first pass (cut) value for 128_4s is 71, and 72 for 64_4s

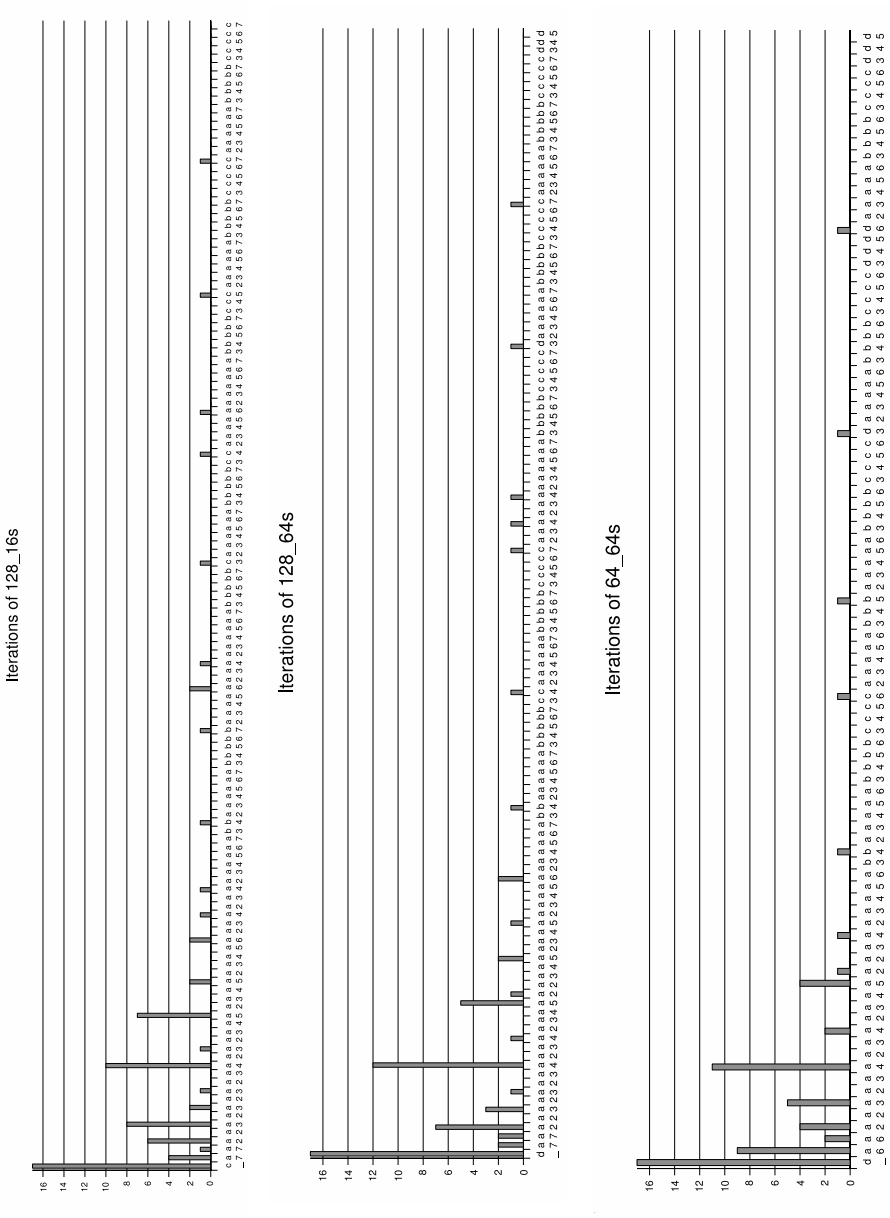


Figure 2: The first pass (cut) value for 128_16s is 85, 96 for 128_64s, and 93 for 64_64s

One easy observation made on the results, is that the combinations of time limit and axiom limit which have not been tried yet usually produce some new proofs. This could be explained by the fact that the relative gain from learning is in those situations much bigger (involving all the previously found solutions) than in the later runs, when only a few new solutions found in the meantime are used to modify the axiom relevancy. On the other hand, it is also interesting how sometimes a solution which was obtained in quite a difficult way and at quite late stage (e.g. the one b7 solution in 128_16s, and the one c7 solution in 128_64s) can make previously difficult problems quite easy to solve (the b7 solution is followed by two a6 and that in turn by one more a4 solutions, i.e. a series of solutions found in 1 second timelimit, similarly for the c7 solution in 128_64s). A bit closer analysis of some of the runs seems to suggest that using smarter learning could make this effect even more frequent. E.g. in some cases it seems that if the classifier knew more about the relationships between some symbols (e.g. that one is a predicate implying the other one, or that they are nearly equivalent predicates or functors), it could draw better analogies and give better advice.

4 Related Work

A very good overview of the field of "machine learning for automated reasoning" is given in the technical report[DFGS99]. The learning capability of E prover mentioned above is today probably the most sophisticated implementation existing in the field. There is quite a lot of related work on symbol-based and structure-based filtering of axioms, a recent one (done for the Isabelle system) is [MP06a], which also cites some more work.

Generally, it is a bit hard for the author to compare related (meta)systems with MaLARea. Quite often that would require further work on those systems, or their reimplementation, which could be criticized as "not being the original system". The point of creating the MPTP Challenge problems in the most standard FOL syntax available today (i.e. TPTP) is to allow everyone to test their system under very clear conditions, and report their results for comparison. It is currently also quite hard to test MaLARea on other than MPTP problems, since it is quite difficult to determine to what extent a given set of large theory problems satisfies the "large theory" criteria needed for MaLARea's machine learning, i.e., consistency of symbol and formula naming. This unfortunately seems to apply also to the set of Isabelle problems included in TPTP and used for evaluation in [MP06a].

5 Future Work and Conclusions

Although MaLARea's current performance is quite encouraging, it is still in a very early stage, and quite a lot of its possible extensions are mentioned above. The machine learning framework could be extended and improved, taking e.g. more relationships among the symbols (and other formula features) into account. Lemmatization could be also quite helpful, and while its addition should not be difficult, it would make the whole theory evolving, not static like so far. The same could be said about defining new useful notions, and possibly reformulating parts of the theory with them. Quite a strong method seems to be weakening, and its extreme version using completely instantiated models. A good database of models for a theory could be also used just as another simple way to classify formulas (adding more features to the learning). Shortly speaking, it seems that with rich theories the AI methods useful for Automated Reasoning can also get quite rich.

One thing that should be noted about the current version of the system is that it

does not use any Mizar-specific knowledge. There are two reasons for it. One is that the system is intended to be generally useful, not just Mizar-specific. The second reason is that re-using Mizar-specific knowledge requires some additional work on the system. But it is quite possible, that e.g having the standard MPTP algorithm for adding the background (e.g. Mizar type) formulas to the axiom set would sometimes be more useful than relying only on learning. Because particularly type hierarchies are quite likely to appear also in all kinds of non Mizar large theories, it would however be preferable to have a more general (quite likely heuristic, and possibly to some extent also governed by learned previous experience) methods for such "rounding-up" of axiom sets.

6 Acknowledgments

This work was supported by a Marie Curie International Fellowship within the 6^{th} European Community Framework Programme. The resources for the reproving experiments were provided by the Czech METACentrum supercomputing project.

References

[CCRR99] A. J. Carlson, C. M. Cumby, J. L. Rosen, and D. Roth. Snow user's guide. Technical Report UIUC-DCS-R-99-210, UIUC, 1999.

[CS03] Koen Claessen and Niklas Sörensson. New techniques that improve MACE-style model finding. In *Proc. of Workshop on Model Computation (MODEL)*, 2003.

[DFGS99] J. Denzinger, M. Fuchs, C. Goller, and S. Schulz. Learning from Previous Proof Experience. Technical Report AR99-4, Institut für Informatik, Technische Universität München, 1999. (also to be published as a SEKI report).

[MJWD06] C. Matuszek, Cabral J., M. Witbrock, and J. DeOliveira. An Introduction to the Syntax and Content of Cyc. In Baral C., editor, *Proceedings of the 2006 AAAI Spring Symposium on Formalizing and Compiling Background Knowledge and Its Applications to Knowledge Representation and Question Answering*, pages 44–49, 2006.

[MP06a] Jia Meng and L. C. Paulson. Lightweight relevance filtering for machine-generated resolution problems. In Geoff Sutcliffe, Renate Schmidt, and Stephan Schulz, editors, *ESCoR: Empirically Successful Computerized Reasoning*, volume 192 of *CEUR Workshop Proceedings*, pages 53–69. CEUR, 2006.

[MP06b] Jia Meng and L. C. Paulson. Translating higher-order problems to first-order clauses. In Geoff Sutcliffe, Renate Schmidt, and Stephan Schulz, editors, *ESCoR: Empirically Successful Computerized Reasoning*, volume 192 of *CEUR Workshop Proceedings*, pages 70–80. CEUR, 2006.

[NP01] Ian Niles and Adam Pease. Towards a standard upper ontology. In *FOIS '01: Proceedings of the international conference on Formal Ontology in Information Systems*, pages 2–9, New York, NY, USA, 2001. ACM Press.

[NW04] Monty Newborn and Zongyan Wang. Octopus: Combining learning and parallel search. *J. Autom. Reasoning*, 33(2):171–218, 2004.

[Pud06] Petr Pudlák. Search for faster and shorter proofs using machine generated lemmas. In Geoff Sutcliffe, Renate Schmidt, and Stephan Schulz, editors, *ESCoR: Empirically Successful Computerized Reasoning*, volume 192 of *CEUR Workshop Proceedings*, pages 34–52. CEUR, 2006.

[RV02] Alexandre Riazanov and Andrei Voronkov. The design and implementation of VAMPIRE. *Journal of AI Communications*, 15(2-3):91–110, 2002.

[Sch02] S. Schulz. E – a brainiac theorem prover. *Journal of AI Communications*, 15(2-3):111–126, 2002.

[SS98] G. Sutcliffe and C.B. Suttner. The TPTP problem library: CNF release v1.2.1. *Journal of Automated Reasoning*, 21(2):177–203, 1998.

[Sut01] G. Sutcliffe. The Design and Implementation of a Compositional Competition-Cooperation Parallel ATP System. In H. de Nivelle and S. Schulz, editors, *Proceedings of the 2nd International Workshop on the Implementation of Logics*, number MPI-I-2001-2-006 in Max-Planck-Institut für Informatik, Research Report, pages 92–102, 2001.

[SY07] G. Sutcliffe and Puzis Y. SRASS - a semantic relevance axiom selection system. In Pfenning F., editor, *CADE 2007*, Lecture Notes in Artificial Intelligence. Springer, 2007. To appear.

[SZS04] G. Sutcliffe, J. Zimmer, and S. Schulz. TSTP Data-Exchange Formats for Automated Theorem Proving Tools. In V. Sorge and W. Zhang, editors, *Distributed and Multi-Agent Reasoning*, Frontiers in Artificial Intelligence and Applications. IOS Press, 2004.

[Urb04] Josef Urban. MPTP - motivation, implementation, first experiments. *Journal of Automated Reasoning*, 33(3-4):319–339, 2004.

[Urb06] Josef Urban. MPTP 0.2: Design, implementation, and initial experiments. *J. Autom. Reasoning*, 37(1-2):21–43, 2006.

[Wei01] C. Weidenbach. *Handbook of Automated Reasoning*, volume II, chapter SPASS: Combining Superposition, Sorts and Splitting, pages 1965–2013. Elsevier and MIT Press, 2001.