Realization of Prime Proof for the Three Prime Case by ATP

Erman Acar

January 24, 2010

Contents

1 Introduction

This report is about the work that has been done for testing the interactive theorem prover ATP which was written by Tomer Libal. For testing, an instant proof of a proof schematic version of Fürstenberg's proof of the infinitude of primes which has been given in [3], is chosen. This schema had

been used in order to provide a proof analysis showing Euclid's argument is in the kernel of Fürstenberg's proof. The proof schema was based on resolution calculus [4, 1]. Beside that, this work also can be used as guide for any ATP user with its content consisting of realization of a relevant size proof and side remarks.

In the subsequent section, the language of ATP will be introduced. The distinction between the input syntax and output syntax is given and the reader is informed of how to translate the formulas which are desired to work on to ATP prover.

The third section concerns the proof which is the three primes case of the prime proof. In the beginning, it shows how the proof specific encoding is done. So this part is the crucial part for reading the machine generated proof. Next, initialization and real axioms are listed and derivation path is provided. Crucial remarks and revelations emerged from the implementation phase is mentioned on the section which is called remarks.

The fourth section includes a compilation of a bug which has been fixed, information for the reader about technical difficulties encountered and counter measurements has been taken and also some suggestions in extra.

2 About the language of ATP

2.1 Input syntax

Input syntax of ATP is based on prefix notation and is quite simple to learn. Here, it will be explained how to encode logical formulas in ATP. In the following, you see the basic rules about the syntax:

- constants are the strings in which starts with "a-c".
- variables are the strings in which starts with "d-z".
- functions/predicates are the strings starts with "d-z".
- equality symbol is "=" itself.

And it is straight forward as follows when we encode the clause below. So,

$p(a, x), g(x) = f(a) \vdash q(b, y)$

is encoded by

-p(a,x) | -=(g(x),f(a)) | q(b,y).

Obviously, any predicate or function name is used in prefix notation. Moreover, we put "-" to the beginning of the literal in order to make it a negative literal and omit it to make it positive and also used " |" in order to separate them in any place in the whole formula. The point "." must not be forgotten in the end of the formula. This is the well-known TPTP-format.

2.2 Output syntax

Output syntax of the ATP is slightly different than its input one. Once you entered your input data on your input file, it simplifies the formulas by putting a dash sign ":-" (standing for "⊢") to the middle of the negative and positive literals, so the bar sign "-" in the head of the negative literals is not needed anymore and dropped. Also "|" signs turn to "," . Furthermore, only the equality symbol "=" loses its prefix form, so it becomes much easier to read the formulas in that form. For instance, the input

-p(a,x) | -=(g(x),f(a)) | q(b,y).

becomes

p(a,x) , g(x) = f(a) :- q(b,y).

Once you encoded your formulas you wanted to start with and run ATP, in any state of the proof that you are carrying on to the end, you will be going on with the output syntax.

3 About the refutation of the three primes case of Fürstenberg's proof

The proof of three prime case for Fürstenberg's proof on ATP is based on the proof schema presented in [1]. The schema is up to provide a refutation based on resolution calculus in the language of first order arithmetic ([1], [3]). Schema provides a refutation method for arbitrary (finitely) number of the prime existence, so to speak, itself a proof for argument "finitely many prime number exists" and in the case of three primes case instance, proof is based on the refutation of a particular instance argument "there are only three primes". So, this section will be all about this machine proof.

3.1 Encoding the language of the proof

Now it will be introduced how the language of proof schema in [3] is transformed to the language of ATP (in output syntax). It is,

- For relation "<", predicate name "smaller" is used.
- For binary function "+", function name "plus" is used.
- For binary function "*", function name "times" is used.
- For unary functions "s_1"..."s_7", function names "sone"..."sseven" are used respectively.
- For variables "k","l","m", "j","t", "m_0","m_1","t_0","t_1","j_0","j_1","k", "l","m","j","t","mzero","mone","tzero","tone","jzero","jone" are used respectively.
- For primes "p_0", "p_1", "p_2", constant names "cpzero","cpone","cptwo" are used.
- For natural numbers "0","1", constant names 'czero","cone" are used.
- "=" stays as it is.

It must be understood that everything is in prefix notation (except "=") and the names starts with "c" is for constants.

3.2 Initializing the proof

Unfortunately, ATP does not have a fixed ordering (yet), so that if you order or modify the clauses in a different way, index number of the clauses will be changed. Hence it is useful to notify the reader, that the derivation path which are going to be shown next section is subject to the following ordering of clauses. As we begin, we order the clauses like below:

(0) :- k = k
(1) :- plus(plus(k,l),m) = plus(k,plus(l,m))
(2) :- plus(k,czero) = k
(3) plus(k,l) = plus(k,m) :- l = m
(4) :- plus(czero,k) = k
(5) plus(k,l) = plus(m,l) :- k = m
(6) k = plus(l,k) :- l = czero
(7) k = plus(k,l) :- l = czero

(8) plus(k,l) = k :- l = czero
(9) plus(k,l) = l :- k = czero
(10) k = l :- plus(m,k) = plus(m,l)
(11) plus(cone,plus(k,cone)) = cone :-
(12) :- times(k,l) = times(l,k)
(13) plus(k,cone) = czero :-
(14) :- times(k,times(l,m)) = times(times(k,l),m)
(15) :- times(times(k,l),m) = times(k,times(l,m))
(16) :- times(k,cone) = k
(17) :- times(cone,k) = k
(18) :- times(k,plus(l,m)) = plus(times(k,l),times(k,m))
(19) :- plus(times(k,l),times(m,l)) = times(plus(k,m),l)
(20) :- times(plus(k,l),m) = plus(times(k,m),times(l,m))
(21) :- plus(times(k,l),times(k,m)) = times(k,plus(m,l))
(22) :- plus(times(k,l),k) = times(k,plus(l,cone))
(23) :- times(m*plus(k,l)) = plus(times(l,m),times(k,m))
(24) smaller(k,l), smaller(k,m), smaller(l,m), plus(k,times(i,m)) = plus(l,times(j,m))
:-
(25) cone = times(k,l) :- k = cone
(26) smaller(k,l), smaller(k,m), smaller(l,m), plus(l,times(i,m)) = plus(k,times(j,m))
:-
(27) cone = times(l,k) :- k = cone
(28) :- smaller(czero,plus(k,cone))
(29) :- k = l, smaller(k,l), smaller(l,k)
(30) smaller(plus(n,cone),m) :- smaller(n,m)
(31) smaller(cone,k), k = cone :-
(32) smaller(cone,k) :- cone = times(l,k)
(33) smaller(czero,cpzero) :- cpzero = plus(sseven(cpzero),cone)
(34) tzero = cpzero :- smaller(cone,tzero)
(35) tzero = cpone :- smaller(cone,tzero)
(36) smaller(czero,cptwo) :- cptwo = plus(sseven(cptwo),cone)
(37) smaller(czero,cpone) :- cpone = plus(sseven(cpone),cone)
(38) :- mzero = cone, times(sone(mzero),sfour(mzero)) = mzero
(39) tzero = cptwo :- smaller(cone,tzero)
(40) :- mzero = cone, sone(mzero) = cpzero, sone(mzero) = cpone, sone(mzero)
= cptwo
(41) :- smaller(cone,plus(plus(w,cone),cone))
(42) :- plus(k,l) = plus(l,k)
(43) :- plus(k,plus(l,m)) = plus(plus(k,l),m)

Most of these clauses are nothing but the axioms from the list AX which

is referred in [3][1] except the reflexivity axiom (first one in the list above), 34th and 35th (from B_i) and 38th (from A) in the characteristic clause set [3].

3.3 Axiom list

Here we give a list of axioms. This list is important by the purely mathematical point of view for answering the question of "what are the relevant axioms to prove the theorem".

1. $n + 1 < m \vdash n < m$
2. $\vdash 0 + k = k$
3. $1 < k, k = 1 \vdash$
4. $\vdash k < l, k < m, l < m, k + (i * m) = l + (j * m)$
5. $\vdash k * l = l * k$
6. $\vdash k * (l * m) = (k * l) * m$
7. $\vdash (k * l) * m = k * (l * m)$
8. $\vdash (k + l) * m = (k * m) + (l * m)$
9. $\vdash k * (l + m) = (k * l) + (k * m)$
10. $\vdash 1 * k = k$
11. $\vdash k * 1 = k$
12. $\vdash k + (l + m) = (k + l) + m$
13. $1 < (w + 1) + 1$
14. $\vdash (k + l) * m = (l * m) + (k * m)$
15. $\vdash (k + l) * m = (k * m) + (l * m)$
16. $\vdash k * (l * m) = (k * l) * m$

Once the proof has been done, the derivation sequence tells us which axioms in the sense of initial clauses has been used. Even though the reflexivity axiom $\vdash k = k$, B_i and A clauses were used in initial input, we did not cover them in our list of axioms above.

[1] Axiom list can be found at http://www.logic.at/ceres/examples/primeproof/prime3-ce.pdf

3.4 Derivation Sequence for the three prime case

In this section, a whole derivation will be given to the user of ATP. Before doing this , a simple notation will be introduced . Let us denote the clauses by their numbers in the resolution (or paramodulation) as following:

ATP(28,4)

then let it denote, resolution or paramodulation of the clauses indexed with 28 and 4 and from the generated clauses to show the one with index 1 is chosen is shown by our notation:

C[1].

Enter tX for automated X steps, sX for displaying all clauses containing X or
Please choose two clauses to process: 28 4
Please choose one clause for insertion (0 for inserting nothing):
(1) :- smaller(czero,cone)
(2) :- plus(czero,smaller(czero,plus(k_2,cone)))
(3) :- smaller(plus(czero,czero),plus(k_2,cone))
(4) :- smaller(czero,plus(czero,plus(k_2,cone)))
(5) :- smaller(czero,plus(k_2,plus(czero,cone)))
1

So the "ATP(28,4) with C[1]" denotes the action which is shown above in ATP interface, hence it is pressed by the user on the last line.

So the derivation expression proceeds in that fashion. We are at the point right after the initial clauses are read by ATP (so the last clause is number 43).[2]

(44) ATP(28,4) with C[1].$0 < 1$
(45) ATP(34,0) with C[5]. III_0
(46) ATP(30,4) with C[2].
(47) ATP(46,45) with C[1].II_0
(48) ATP(35,0) with C[5].III_1
(49) ATP(46,48) with C[1].II_1
(50) ATP(39,0) with C[5]. III_2
(51) ATP(50,46) with C[1].II_2
(52) ATP(38,31) with C[6].
(53) ATP(40,31) with C[4].

[2]Some clauses has comments to it pointing out which clauses are the in the schema [3]

(54) ATP(52,53) with C[8].
(55) ATP(52,54) with C[8].
(56) ATP(52,55) with C[4].D_2
(57) ATP(24,4) with C[4].
(58) ATP(57,12) with C[1].
(59) ATP(58,12) with C[2].V
(60) ATP(59,14) with C[2].V^*
(61) ATP(60,15) with C[2].
(62) ATP(61,56) with C[32].VI
(63) ATP(62,44) with C[1].
(64) ATP(63,45) with C[1].
(65) ATP(64,47) with C[1].F_0
(66) ATP(65,12) with C[4].
(67) ATP(66,15) with C[2].
(68) ATP(67,59) with C[1].
(69) ATP(68,44) with C[1].
(70) ATP(69,49) with C[1].
(71) ATP(70,48) with C[1].
(72) ATP(71,14) with C[3].
(73) ATP(72,12) with C[9].
(74) ATP(73,59) with C[1].
(75) ATP(74,44) with C[1].
(76) ATP(75,51) with C[1].
(77) ATP(76,50) with C[1].F_2
(78) ATP(33,47) with C[2].by B_0
(79) ATP(37,49) with C[2].by B_1
(80) ATP(36,51) with C[2].by B_2
(81) ATP(77,78) with C[1].
(82) ATP(81,20) with C[1].
(83) ATP(79,82) with C[2].
(84) ATP(83,20) with C[1].
(85) ATP(84,18) with C[1].
(86) ATP(85,17) with C[1].
(87) ATP(86,17) with C[1].
(88) ATP(87,17) with C[3].
(89) ATP(88,80) with C[2].
(90) ATP(89,18) with C[1].
(91) ATP(90,16) with C[1].
(92) ATP(91,80) with C[3].
(93) ATP(93,17) with C[1].
(94) ATP(94,17) with C[3].

(95) ATP(95,43) with C[2].
(96) ATP(96,43) with C[2].
(97) ATP(97,41) with C[1].
(98) ATP(98,78) with C[1].
(99) ATP(99,20 with C[1].
(100) ATP(100,79) with C[2].
(101) ATP(101,20) with C[1].
(102) ATP(102,18) with C[1].
(103) ATP(103,17) with C[8].
(104) ATP(104,17) with C[1].
(105) ATP(105,17) with C[3].
(106) ATP(106,80) with C[2].
(107) ATP(107,18) with C[1].
(108) ATP(108,16) with C[1].
(109) ATP(109,80) with C[3].
(110) ATP(110,17) with C[1].
(111) ATP(111,17) with C[3].
(112) ATP(112,43) with C[2].
(113) ATP(113,43) with C[2].
(114) ATP(114,17) with C[8].
(115) ATP(115,17) with C[2].
(116) ATP(116,17) with C[1].
(117) ATP(117,80) with C[2].
(118) ATP(118,18) with C[1].
(119) ATP(119,16) with C[1].
(120) ATP(120,80) with C[3].
(121) ATP(121,17) with C[1].
(122) ATP(122,17) with C[1].
(123) ATP(123,43) with C[2].
(124) ATP(124,43) with C[2].
(125) ATP(125,98) with C[1].
(126) ATP(126,78) with C[1].
(127) ATP(127,20) with C[1].
(128) ATP(128,79) with C[2].
(129) ATP(129,20) with C[1].
(130) ATP(130,18) with C[1].
(131) ATP(131,17) with C[8].
(132) ATP(132,17) with C[1].
(133) ATP(133,17) with C[1].
(134) ATP(134,78) with C[2].
(135) ATP(135,23) with C[1].

(136) ATP(136,17) with C[1].
(137) ATP(137,80) with C[3].
(138) ATP(138,12) with C[3].
(139) ATP(139,14) with C[1].
(140) ATP(140,80) with C[3].
(141) ATP(141,43) with C[2].
(142) ATP(142,43) with C[1].
(143) ATP(143,98) □

A prooftool readable[3] output.xml file is generated after the refutation has been done, and it is totally informative, so the interested reader can check the above proof steps form prooftool on output.xml file frankly.

3.5 Remarks

Because of the interplay between theoretical stipulations and practical issues shows itself in following the schema which is presented in [3] with a program, it can make the user encountering some little surprises sometimes. As this work is based on main paper [3], implementing the proof revealed some small facts about the work itself. Despite the things have been found are not carrying any level of contradiction to the emphasized ideas lying behind the work [3] but they might stand as interesting small side details. However they can become non-trivial problems if one decides to implement the schema instance with a theorem prover software.

The first one does come from the subsumption. Now we have to point out the reason why the subsumption is disabled. As you go through the schema with the subsumption deletion you won't be able to generate some clauses in the schema. Many of them are caused by multiple associativity following distributivity operations. As an example we can give the derivation sequence in page 12 of [3] between lines 5 and 13. Its not possible at all to derive from clause V to E_r with the subsumption deletion, because after V' clauses that you are looking for will not be generated by ATP and you will be informed by it that it is not added to clauses because of its subsumption by a previous clause. Thus, it does not take the opportunity away from reaching resolution refutation since its still complete theoretically but user has two choices at that point; either still trying to follow scheme by trying to figure it out with adaptation of what he has, to the schema. If one is lucky enough, he can turn back to the schema later on with less efforts. Another choice which can be chosen is to try to modify the schema itself, and making it shorter and more

[3]Reader may find enough information about it on [2]

preferable. One another choice which is the easiest obviously is deactivating the subsumption and going on with the schema which is believed to be true.[4]

Another remark with the proof is about its very last part. On page 13 line 29 of [3], its been said that $\vdash 1 < (k+1)+1$ can be derived from the axioms and for sure once you get it you do the last resolution step and getting resolution refutation finally. Some persisting efforts which was spent by me, has revealed that actually that is not the case, so that the clause has been itself imported as an axiom to the proof.

Another remark that must be mentioned before starting the proof, is that subsumption must be disabled in order to follow the proof path which will be given.It can be done by the setup.xml file of ATP. We don't want subsumption because we do not want to deal with any surprise or complication which may occur during walkthrough with the proof schema in parallel given in [3].[5]. Also the axiom which was added later (41th) which stands for $1 < (w+1)+1$.[6]

Another issue related to ATP's inner design which is making the proof longer is no factorization is being done on the resolvent. As a result of this, actual D_2 in ATP becomes the following:

$$1 < k, 1 < k, 1 < k, 1 < k \vdash p_0 * s_4(k) = k, p_1 * s_4(k) = k, p_2 * s_4(k) = k$$

So as it seems, on the further evolution of this clause k's on the left hand side will be modified concurrently till F_r and it does not creates a real problem. So your actual F_r is

$$1 < t_3, 1 < t_3, 1 < t_3, 1 < t_3 \vdash$$

but after it with the sequences of distributivity, commutativity and associativity operations has to be repeated four times on it on the very last part of the proof.[7] It makes almost half of the proof. So its a serious lack of proof efficiency on length.[8]

[4]After both of these ways have been tried, it has been realized that these little modifications caused the change propagated more and more on long rest of the proof, and the rest also had to be redesigned. So personally I took the decision to deactivate the subsumption, obviously the easiest choice.

[5]In particular cases, there are subsumptions between different clauses on schema.

[6]The reason will be explained in the next section dedicated to problems and fixes.

[7]According to the proof schema on [3].

[8]Despite this is the case here, on particular cases it might improve the efficiency.

4 Difficulties engaged, modifications, bugs and fixes

During the time resolution refutation was tried to be captured, ATP had its evolution in many levels from bug fixing to new designs aiming better usability, triggered by the problems faced up with. In this section, the problems and difficulties encountered and counter changes will be reported.
Lets start with the impracticalities related to its early design. In the beginning, ATP's interactive part was working like below, let us use the same short example used previously and see what is actually going on:

```
Enter tX for automated X steps, sX for displaying all clauses containing X or
Please choose two clauses to process: 28 4
(44) :- smaller(czero,cone)
(45) :- plus(czero,smaller(czero,plus(k_2,cone)))
(46) :- smaller(plus(czero,czero),plus(k_2,cone))
(47) :- smaller(czero,plus(czero,plus(k_2,cone)))
(48) :- smaller(czero,plus(k_2,plus(czero,cone)))
```

For reminding, this resolution step was us to reach clause number 44 in which stands for $\vdash 0 < 1$. Lets call it goal clause. Now as you might recognized first, there is no indentation, so the user has to improve his reading skills in order to find the goal clause easily, it both consumes time and energy of the user when one consider some resolutions generating eighty new clauses occupying many lines. Beside this, another disadvantage is without asking which clause do we wanted, it just added whatever it could generate and less probably that we will use the rest of the clauses generated again and in each step we proceed in the console, these clauses will come and come again. Despite the fact that it was complete, it was messy and way too exhausting work to try to go on with it.

A necessary side remark is, search engine of ATP wouldn't also help for the cases that the new variables generated by the machine. So you wouldn't be able to make a search on ATP for the clause that you are looking for but don't know how it looks like! Obviously. Hopefully that had been changed. So in that sense it is obvious that todays ATP is much simpler and quite easier to use.

A bug which has been found was basically an absurdity on difference between the forms of clauses which had to be variants at worst, when each was done by different lengths of derivation steps done based on paramodulation. On example what had been going on will be more obvious, so the

exact samples which had exploited the bug will be given. Assume you have the following clauses: [9]:

1. $1 < k \vdash p_0 * s_4(k) = k, p_1 * s_4(k) = k, p_2 * s_4(k) = k$
2. $0 < l, 0 < m, l < m, m * i = l + m * j_1 * j_2 \vdash$
3. $0 < 1$
4. $0 < p_0$
5. $1 < p_0$
6. $k * (l * m) = (k * l) * m$

So the derivation is:

(7) ATP(2,3).
(8) ATP(3,7).
(9) ATP(4,8).
(10) ATP(5,9).
(11) ATP(1,10).

So the resulting clause is:

$$1 < t \vdash p_1 * s_4(t) = t, p_2 * s_4(t) = t$$

where

$$t = 1 + p_0 * (j_1 * j_2)$$

Lets call the clause 11, so if we perform

ATP(6,11)

we have to get one which is

$$1 < t \vdash p_1 * s_4(t) = t, p_2 * s_4(t') = t'$$

where [10]

$$t' = 1 + (p_0 * j_1) * j_2$$

[9] Normal logical notation instead of machine code will make the example easier to understand.

[10] Or necessarily the other right literal t becoming t'

So the problem is we were able to get the clause 12 if we directly perform ATP(6,11) but unable if we make the derivation required above to get the clause 11. Finally this hardly recognizable bug has been fixed.

Another problem which must be told about the use of ATP is the order of input clauses has no unique ordering, so anytime you modify your input file, you have to built new proof paths. An ordering (lexicographic) would make the prover much declarative and simpler in use.

One last deficiency and a difficulty in this project is again with the interface of ATP which is prefix notation. It makes long bulk strings on the screen such as the one below and makes the users job much harder.

(88) times(cone,smaller(cone,plus(cone,plus(times(sseven(cpzero), plus(times(cp one, m_57), times(cpone,m_57))),times(cone,times(plus(sseven(cpone),cone),tim es(plus(cone, cone),m_57))))))), smaller (cone,plus(cone,times(cpzero,times(cpon e,times(plus(cone, cone), m_57))))), smaller(cone,plus(cone, times(cpzero,times(c pone,times(plus(cone,cone),m_57))))), smaller(cone,plus(cone,times(cpzero,time s(cpone,times(plus(cone,cone),m_57))))), plus(cone,times(cpzero,times(cpone,ti mes(plus(cone, cone),m_57)))) = plus(cone,times(cptwo,j_15)) :-

Once you spend much time, certainly you develop much speed and tricks to deal with it because of eye training but that shouldn't be the case. Symbols like "+", "*", "¡" and natural numbers are in common use of almost every logical systems, so the prover parsers must recognize them and print exactly how it is in normal notation, and also term reading can be easier in a way exactly like we do on previous section. [11]

5 Conclusion

The three primes case of the proof schema based on first order formalization of Fürstenberg's proof represented on [3] has been realized in the ATP theorem prover, in order to test it. During the realization phase, different levels of problems have been encountered such as, impracticality of pre-designed interface,a bug in the paramodulation engine and the misleading reference by [3] . Because many modifications of the software, fixing of bug and correction of argument in main work ([3]) took place, improvements obtained and the refutation has been succeeded.

In doing so, formalization of the schema has been transformed to ATP language, axiom list (initialization file) has been redetermined and also a reported bug in paramodulation could be fixed. Issues concerning all this

[11] $1 < t \vdash p_1 * s_4(t) = t, p_2 * s_4(t) = t$ with declaration $t = 1 + p_0 * (j_1 * j_2)$

work; testing and realization also embodied suggestions from the gained experiences on machine based theorem proving. This work must be understood as a small contribution in which is promised in conclusion part of [3] , which is the development of a semi-automated proof analysis method.

6 Acknowledgements

I would like to thank to my supervisor Alexander Leitsch, in this project for his kind help, patience and specially understanding that he showed on the long time completion of this project and also to Tomer Libal, author of the ATP, for being ready to help and give it any time it is needed with so much patience.

References

[1] `http://www.logic.at/ceres`.

[2] `http://www.logic.at/prooftool`.

[3] Matthias Baaz, Stefan Hetzl, Alexander Leitsch, Clemens Richter, and Hendrik Spohr. Ceres: An analysis of fürstenberg's proof of the infinity of primes. *Theor. Comput. Sci.*, 403(2-3):160–175, 2008.

[4] Alexander Leitsch. *The resolution calculus*. Springer-Verlag New York, Inc., New York, NY, USA, 1997.