List of changes / Comments
Paper: Towards certified virtual machine-based regular expression parsing

T. Delfino R. Ribeiro

Response to Comments: Reviewer #1

General remarks

This paper describes interesting and promising work on the reduction semantics of regular expression parsing. My comments are in the attached PDF but, in summary, the paper as it is needs some important rewriting to make clear what appears to be (as I understand it) the main aspects of their work: (i) the choice of a VM-based approach, (ii) the use of reduction semantics and (iii) the properties of the automated testing.

Response: (i) The main reason for choosing the VM approach for RE is the lack of formally verified semantics and algorithms for it. We believe that having a correct VM semantics makes possible to get a efficient algorithm by applying developed techniques for optimizing abstract machines for λ-calculus [3]. This is a venue that we intend to pursue in future works.

(ii) Reduction semantics are quite standard in the context of abstract machines for functional languages. Our choice of using reduction semantics is mainly because we intend to adapt results from [3] to the context of RE parsing.

(iii) We apologize for the absence of details about our automated test suite. The main reason for such suscint description is the lack of space. We properly extend our technical report to include more details about the test suite implementation.

◊ ◊ ◊

I believe that much of the implementation details, that are very important indeed, could be left to the technical report (or something like a Jupyter notebook) and open room for much needed higher-level (than code) explanations.

Response: The main reason for including the complete implementation is it's correspondence with the semantic rules. Without this equivalence we would need to justify why our implementation is correct with respect to the proposed small-step semantics. Currently we already have a certified implementation in Coq, which is available in project's on-line repository.

`https://github.com/thalesad/regexvm`

♢ ♢ ♢

Due to these clarifications and that, perhaps, publishing it now could be a bit premature (perhaps with some prototype-level Coq coding?), I will not champion it. I certainly suggest it to be accepted if room.

Response: We believe that our work has enough content to deserve acceptance. We tried to improve presentation while leaving details to the technical report available on-line:

`https://github.com/rodrigogribeiro/regexvm/raw/master/papers/sblp2018/report/report.pdf`

Main comments present in annotated pdf

It is very cool that your approach is fully formal and fully available for experimentation. I managed to clone and execute your code, after upgrading stack.

Response: Thanks! We believe that is very important to make the work reproducible.

♢ ♢ ♢

The main problem for me is that I am not convinced on why I should use your technique and tooling. Would it be because your tests gave us some confidence that it implements the NFA epsilon semantics of RE?

Response: Our long term goal is to investigate the suitability of using VMs for RE parsing. In order to reach such objective, we think that we should provide a correct semantics and our paper reports on the design of such VM and the use of property based testing to check if our definitions make sense. Currently, we are working towards a fully verified implementation of our semantics in Coq proof assistant. We add a paragraph in the introduction about the current status of the mechanization.

♢ ♢ ♢

The section on the properties QuickCheck checks for must be increased with an in depth description, with code if you must, on why correctness is enforced by your tests. Give more examples and details on how the verification process functions.

Response: We have used a short description of tests due to space reasons. We will expand the description of the test suite on the technical report, which is available at:

`https://github.com/rodrigogribeiro/regexvm/raw/master/papers/sblp2018/report/report.pdf`

♢ ♢ ♢

If the use of reduction contexts is a key point in your approach, it should be better explained.

Response: We rewrite some parts of the text in order to explain better the usage of context in the semantics.

♢ ♢ ♢

What about efficiency with respect to other implementations? You mentioned PEG but you treat only RE, correct? (PEG is more expressive than RE, it covers some CFG and some CSG as well.)

Response: Currently we are interested on the correctness of our VM semantics. We are aware that our semantics didn't allow for an efficient implementation of parsing. After having fully mechanized proofs, we intend to adapt the ideas of [3] to the context of RE matching in order to achieve better performance.

♢ ♢ ♢

Isn't your approach greedy as well? This is what you say in your GitHub.

Response: There is a subtle difference between how PEG and RE interprets the Kleene star operator. The semantics of RE Kleene star can be seen as a non-deterministic loop, while PEG's isn't. PEG star operator will match all the possible occurrences of the described pattern. For instance, the RE $a^\star a$ matches the string "aaa" correctly, since the $a^\star$ operator should non-deterministically stop after processing the second 'a' letting the last one to be consumed by the RE a. Observe that, the same expression executed using PEG semantics will never succeed, since it will try to consume all occurrences of character "a" using the star operator then it will fail by trying to match the empty string with the expression 'a'.

♢ ♢ ♢

Why? Only VM based algorithm can do that?

Response: There are approaches that do not use VM for building evidence for parsing as bit-strings. We mention them on related works section.

♢ ♢ ♢

which we conjecture is correct.

Response: We use the word "conjecture" because the paper [1] doesn't present any formal argument that this implementation is correct. In our technical report, we include a informal proof about the correctness of this function.

♢ ♢ ♢

What bit-coded parse trees are, what are they for and why are they important?

Response: We added the following text in the introduction, to explain why bit-coded representation of parse-trees are relevant.

As pointed by Lasse2011, bit-codes are useful because they are not only smaller than the parse tree, but also smaller than the string being parsed and they can be combined with methods for text compression.

♢ ♢ ♢

An example would be very useful here.

Response: We omit examples for all definition from the paper for space reasons. We modify the technical report to include examples of relevant definitions. The report is available at:

`https://github.com/rodrigogribeiro/regexvm/raw/master/papers/sblp2018/report/report.pdf`

♢ ♢ ♢

This section needs to be expanded a lot! Isn't it the major difference between your approach and the state-of-the-art?

Response: We agree that the test suite description is suscint, but it isn't the major contribution of our work. We believe that the small-step semantics is our main contribution. The usage of property-based testing is just a way to check if the designed semantics rules are appropriate. We expand the description of the test suit in the technical report available at:

`https://github.com/rodrigogribeiro/regexvm/raw/master/papers/sblp2018/report/report.pdf`

♢ ♢ ♢

This appears to be a good Soft eng. metric but is not really related with correctness with respect to NFA semantics, correct?

Response: Yes, it isn't. Our test suite check if the proposed semantics is sound and complete with respect to a standard RE semantics.

♢ ♢ ♢

I have not been able to grasp that!

Response: We rewrite our conclusion to reduce the emphasis on evaluation contexts in the text. Extra explanations about each rule and how the evaluation contexts are used in the execution are contained in the technical report available at:

`https://github.com/rodrigogribeiro/regexvm/raw/master/papers/sblp2018/report/report.pdf`

Minor comments present in annotated pdf

Move to conclusion.

Response: We removed the last paragraph of section 4.

♢ ♢ ♢

Where? Fig. 1?

Response: We added references to the relevant figures. Thanks!

♢ ♢ ♢

What does e' mean?

Response:It represents an arbirary RE. We added the following sentence in section 2.1: As usual, meta-variable e will denote an arbitrary RE and it can appear primed or subscripted.

♢ ♢ ♢

encoding?

Response: Ok, fixed.

♢ ♢ ♢

Use mathit around words in math mode.

Response: Fixed. Thanks!

♢ ♢ ♢

Response to Comments: Reviewer #2

Main comments

As it stands, the text is a nice exercise in formal languages, but I am not sure it has enough material to deserve a publication in SBLP. As the main contribution would be a "certified" machine, I think a really formal proof (e.g., in Coq), or at least some concrete work in that direction, would make this work much more valuable. Simply "proof sketches" does not seem enough for a work where the main topic are the proofs themselves, as we have no guaranties these sketches can lead to real proofs. In this case in particular, I suspect they cannot.

Response: Currently, we have a verified interpreter of the semantics in Coq proof assistant and we are working towards the mechanization of the equivalence proof between the small step semantics and the traditional (big-step) RE inductive semantics. We included a sentence about this formalization and added it to project's on-line repository.

♢ ♢ ♢

*A known problem in interpreters for regular-expressions is what the paper calls "problematic REs", that is, a regular expression that contains a Kleene closure over a nullable expression. As far as I could see, the presented implementation would loop forever in failed matches against problematic REs. For instance, a match of "b" against the pattern "(a?)*c" would try infinite ways of matching "(a?)*" against the empty string before 'b', and the program would never be able to conclude that the resulting list is empty. When I saw this problem in the algorithm, I could not understand why the exhaustive tests did not detect the problem. Then I found the explanation: For simplicity and brevity, we only generate REs that do not contain sub-REs of the form e? , where e is nullable. All results can be extended to non-problematic REs in the style of Frisch et. al [13]. Probably the text meant "problematic REs" instead of "non-problematic REs". Nevertheless, Theorem 3 does not hold for all REs, as its hypothesis states, but only for non-problematic ones.*

Response: We will rewrite the text to clearly specify that the current version deals only with non-problematic REs. In our view, dealing with problematic REs will only clutter the presentation without adding any new insight. We added the following paragraph at the end of section 2.1:

We say that a RE e is *problematic* if $e = e'^{\star}$ and $\epsilon \in [\![e']\!]$ [2]. In this work, we limit our attention to non-problematic RE's. Our results can be extended to problematic REs without providing any new insight [4, 2].

♢ ♢ ♢

Another problem in the algorithm is its time complexity. The paper says the algorithm was inspired on Thompson's construction. (I could not see the relationship between the two approaches, but that is not relevant here.) However, the *paper fails to describe a key property of the Thompson's construction: Although it constructs a non-deterministic automata, it runs the result as a deterministic one, and therefore it runs in linear time. (Thompson's algorithm represents its [deterministic] state as a list of [non-deterministic] states from the resulting NFA.) Unlike Thompson's construction, however, the presented algorithm seems to have exponential complexity in its worst case.*

Response: We are aware of the time complexity of the algorithm and we intend to investigate how to improve its efficiency after we have formally verified proofs. A direction we want to investigate is to adapt the works on the optimization of abstract abstract machines [3] to our VM.

♢ ♢ ♢

The description of the type Ctx is confusing. First, the paper says (and the name implies) that the type Ctx implements contexts, when in fact contexts are actually lists of Ctx (as the paper explains later).

Response: We rename type Ctx to Hole in order to avoid such confusion.

♢ ♢ ♢

In Theorem 3, it would be simpler to state directly that ...

Response: This was a typesetting error. The correct would be as you stated. Thanks for pointing that.

♢ ♢ ♢

Response to Comments: Reviewer #3

Main comments

Although presenting a relatively small constribution, I find the paper very interesting, with an approapriate presentation of background material and related work. The document is well-written and clear.

Response: Thanks!

♢ ♢ ♢

It would be helpful if you can give some explanation about the correspondence between the rules of the VM and the automata. If I am understanding correctly, it seems that many of the rules correspond to the extra epsilon transitions that are added in the construction of the NFA; a rule for those cases does not advance on the string and simply needs to perform the necessary adjustments on the context and the produced bit-code before entering or after exiting a sub-automata N(e).

Response: Yes, you are right. Sadly, for space reasons, we are not able to give further explanations on how the semantics rules are related to the Thompson NFA construction. We added the relevant explanations in technical report available at

`https://github.com/rodrigogribeiro/regexvm/raw/master/papers/sblp2018/report/report.pdf`

♢ ♢ ♢

In the definition of code (List ts) (Star e) = 0b : codeList ts e, is it correct to always put 0b at the front? This means that the bit-code for a Kleene star is always of the form 0b ... independently it corresponds to an empty or non-empty list of trees. In case a 0b needs to be always at the front of the bit-code for a star, then it should be consumed by the star case of function dec, i.e. this case should be defined this way: dec (Star e) (0b : bs) = do ....

Response: The equation for code function in star case is wrong! Probably an error during the paper writing. Thanks for pointing that! In our source code, we have the following equation:

code (TList ts) (Star e) = codeList ts e

Which has the correct behaviour.

♢ ♢ ♢

Minor comments

LPEG [16] defines a VM an evidence

Response: Fixed. Thanks!

♢ ♢ ♢

is is

Response: Fixed. Thanks!

♢ ♢ ♢

in the introduction of configurations: "(starting by S)" should be a B instead of S

Response: Fixed. Thanks!

♢ ♢ ♢

in Theorem 3 (twice): should be direction F instead of E in the final configuration

Response: Fixed. Thanks!

♢ ♢ ♢

*equations of next deal*s* with*

Response: Fixed. Thanks!

♢ ♢ ♢

closure of the semantics

Response: Fixed. Thanks!

♢ ♢ ♢

Reference [21] is incomplete, is it a technical report, a thesis?

Response: Fixed. We use the citation to the LNCS publication.

♢ ♢ ♢

the proceedings of FLOPS 2014 appear in LNCS.

Response: Fixed. Thanks!

Final remarks

We have modified the paper, technical report and code repository in order to address all considerations from the anonymous reviewers. Finally, we would like to thank you for your careful reading and suggestions that have helped us to improve the presentation of our results.

References

[1] Sebastian Fischer, Frank Huch, and Thomas Wilke. A play on regular expressions. *ACM SIGPLAN Notices*, 45(9):357, 2010.

[2] Alain Frisch and Luca Cardelli. Greedy Regular Expression Matching. *ICALP 2004 - International Colloquium on Automata, Languages and Programming*, 3142:618–629, 2004.

[3] J. Ian Johnson, Nicholas Labich, Matthew Might, and David Van Horn. Optimizing abstract abstract machines. In *Proceedings of the 18th ACM SIGPLAN International Conference on Functional Programming*, ICFP '13, pages 443–454, New York, NY, USA, 2013. ACM.

[4] Lasse Nielsen and Fritz Henglein. Bit-coded regular expression parsing. In Adrian-Horia Dediu, Shunsuke Inenaga, and Carlos Martín-Vide, editors, *Language and Automata Theory and Applications*, pages 402–413, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.