Automated translation tools are used to translate applications from one programming language to others with the help of API mapping relations. This paper proposes an approach of how to automatically generate test cases to evaluate such translation tools and detect differences in API behavior based on the translation. The approach is evaluated on five translation tools and summarizes the findings and their implications.

>>> Evaluation <<<

Strengths ------------- -

Since one primary goal during translation is to ensure that both applications exhibit the same behavior, understanding differences in behavioral differences in API behavior is important. - The authors describe a novel approach of how to utilize existing test generation approaches in order to automatically test behavioral differences caused by API translations with the help of translation tools.

- The authors provided an empirical comparison on behavioral differences of mapped API elements between the J2SE and .NET frameworks with regard to five translation tools. Based on these results, the authors further analyzed the implications from various perspectives.

-The presented approach of API testing is practically relevant as it can be used to rate translation tools in their effectiveness of translating capabilities with regard to differences the approach is able to find.

Weaknesses -------------

-The evaluation shows that translation tools are quite impefect since even the best translator tool (JCLA) was only able to translate 22.4% of the generated test cases without compilation errors. Of those 22.4% of test cases approximately 50% failed because of behavioral differences. This of course is not a weakness of the paper. However, with this in mind, I am not convinced that this detection rate of behavioral differences is good because of the effectiveness of the TeMAPI approach or just because the translator tools are that bad. In other words, the translators are already so bad that it is no longer clear to me what the 50% error rate entailed. Perhaps any approach on static or dynamic analysis could have found them. There is really no benchmark here - The paper completely ignores the issue of the relationship among different API methods. Now, I do not know how the translators work nor have I really spent time looking at them but I do know that API methods often have to be seen in context of other methods (e.g., a file read only makes sense after a file open). It is not clear whether translators make use of such knowledge for better translating methods. Since your test cases only ever tested single API methods, it is quite possible that the translators were never able to perform well because they lacked contextual information if might have been able to obtain by looking at code around it.

-The substantiation of the conclusion: “To detect behavioral differences, our approach combines random testing with dynamic-symbolic-execution-based testing, and achieves to detect more behavioral differences than with single techniques” seems quite weak to me. I can see a small benefit but when you look at the results in table 8 the detected differences between using a single technique (column P for the Pex only approach) and the combined approach (column T for Pex and Randoop) are very small and in 8 out of 13 listed cases even the same.

Other issues -------------

- In my opinion something that is missing in the general discussion and also in future work is the fact how the knowledge of behavioral differences could be used to generate better API mappings or guidelines of how to use certain APIs if the developers intend is to translate the application at some point to another language.

-An evaluation of real applications/libraries providing their own API would also be very interesting, to get a feel for how translation tools perform there and how many behavioral differences actually are still present/detectable in a custom API with the usage of the APIs evaluated in this paper. Minor issues -------------

- Some information on how long such an evaluation of a translation tool takes would be nice - - I would prefer not to have tinyurl links

Presentation ------------- -

There are a lot of overfull boxes where text reaches into the margin (e.g. page 8 at the bottom: NotImplementException)

- Captions are very close to the figures (e.g. Figure 5), also spaces to the following text seem to little (e.g. Table 4)

- Table 4 is too wide

>- The paper completely ignores the issue of the relationship among different API methods. Now, I do not know how the translators work nor have I really spent time looking at them but I do know that API methods often have to be seen in context of other methods (e.g., a file read only makes sense after a file open). It is not clear whether translators make use of such knowledge for better translating methods. Since your test cases only ever tested single API methods, it is quite possible that the translators were never able to perform well because they lacked contextual information if might have been able to obtain by looking at code around it.Hao: This issue is not addressed yet. Should I add to discussion or threats to vadility?

Tao's comments: I think the comment was rooted from your unclear description of method sequence generation with Pex or Randoop.

>

- The substantiation of the conclusion: “To detect behavioral differences, our approach combines random testing with dynamic-symbolic-execution-based testing, and achieves to detect more behavioral differences than with single techniques” seems quite weak to me. I can see a small benefit but when you look at the results in table 8 the detected differences between using a single technique (column P for the Pex only approach) and the combined approach (column T for Pex and Randoop) are very small and in 8 out of 13 listed cases even the same.

Hao: This issue is not addressed yet.

Tao's comments: Do you really want to claim the combination of random testing and DSE testing to be better than individual techniques? If so, your results don't seem to support your claim. You would have to remove such claim or provide convincing empirical results to support the claim.

- In my opinion something that is missing in the general discussion and also in future work is the fact how the knowledge of behavioral differences could be used to generate better API mappings or guidelines of how to use certain APIs if the developers intend is to translate the application at some point to another language.

Hao: I add a paragraph titled as “Improving translation tools and detecting related defects” to Section 5.

Tao's comments: I think in the intro and somewhere else in the paper, you need to discuss how a user would use the reported results by your approach to help better API migration. For example, if a user finds out the different behaviors,

>The paper is well written, and the problem targeted is relevant. I find the presented technique interesting. The authors need to convince more about the importance of this problem. There are several open-source applications such as Lucene.Net and NeoDatis translated from other language versions. Are there any real bugs due to the incorrect translation between APIs?

Hao: I add a paragraph titled as “Improving translation tools and detecting related defects” to Section 5.

Tao's comments: It would be useful if you could get the list of API mapping that your approach detects to have different behaviors, and then see whether translated applications actually use these mapped APIs and then see whether you could find real faults in translated applications.

>Furthermore, the citation of footnote 6 is merely an informal and very short review on the authors' previous work[25], and there is no empirical evidence either. For these reasons, I suggest authors do an empirical investigation on real world multi-language applications to see how common could the incorrectly translated API mappings lead to abnormal software behavior (real bugs).

Hao: I add a paragraph titled as “Improving translation tools and detecting related defects” to Section 5.

Tao's comments: This review comment is related to the previous one. Showing that such wrong API mapping (consistent syntax but wrong semantic) could cause real bugs in translated apps could be useful.

>

It is also possible that an API in language L1 will have a different input domain after being translated from language L2. Thus, the test inputs generated for L2 may not be applicable for the translated API in L1. For such potentially illegal inputs, the different behavior revealed might not truly reflect incorrect API mappings. The authors did not discuss this point through the whole paper.

Hao: Finding 3 is on illegal inputs. I suspect that the reviewer does not read it careful, and other reviewers should not raise the question again, so I did not address this issue.

Tao's Comments: I don't think you should describe the issue to be "illegal inputs". It is just that two API methods from the two languages have two different input domains, as described by the reviewer. You need to discuss on how to add checking code to bridge the different input domains. See the following paper:

Jonathan E. Cook, Jeffrey A. Dage: Highly Reliable Upgrading of Components. ICSE 1999: 203-212

>

The authors use Pex to generate tests for the C# version and then apply those tests to the Java version. As far as I see from the motivating example and the evaluation section, there is no information on how to specify data factories to create non-primitive object instances. That is a critical part for Pex to create high quality test inputs. It is not clear for me that does the TeMAP only handle primitive inputs in the synthesized wrappers? If so, TeMAP may only be able to test even fewer APIs, and will have more limitations in practice.

Hao: In Section, I add an explanation: “ Pex includes some heuristics for generating data factories for non-primitive types12.”

Tao's comments: Although Pex includes these heuristics, as shown by Suresh's FSE 09 paper, Pex's capability of generating good method sequences is poor.

> 2. The presentation is very informal, ad-hoc, and disjointed. In several places, key concepts appear to be undefined (e.g., "safe wrapper methods", "internal techniques"). Furthermore, the generation of wrappers is not presented very precisely, making the work hard to reproduce.

Hao: I replaced “safe wrapper methods” to “safe wrappers”, and the latter is defined in our draft. I replace all “internal techniques” with “TeMAPI’s internal techniques”.

Tao's comments: The key issue seems that you didn't define what is "safe" (if you did define it, that is fine). In the first place when you mentioned "internal techniques", did you explain what they are?