Hao Zhong

发件人: Harald Gall and Nenad Medvidovic [icse2011-papers-chairs@borbala.com]
发送时间: 2010年11月18日星期四 5:01
收件人: zhonghao@nfs.iscas.ac.cn; sthumma@ncsu.edu; txie@ncsu.edu
抄送: icse2011-papers-chairs@borbala.com; icse2011-papers-webadmin@borbala.com

主题: ICSE 2011 Paper Notification [120]

Dear Hao, Suresh and Tao,

Thank you for your submission to ICSE 2011. The program committee met on November 12-13, 2010, to consider the submissions to the Research Paper track. We regret to inform you that your paper,

"Automated Testing of API Mapping Relations"

has not been accepted for inclusion in the conference program. The competition was very strong: only 62 of the 441 submissions were accepted, giving an acceptance rate of 14%.

We enclose below the reviewers' comments to your submission, which we hope you will find useful for your future research work.

Nevertheless, we do hope that you will be able to attend the conference, which will be held May 21-28, 2011, in Waikiki, Honolulu, Hawaii.

Please note that submission to some of the satellite events is still open (see http://2011.icse-conferences.org/content/submissions)

Sincerely,

Nenad Medvidovic and Harald Gall
Program Committee Co-Chairs
ICSE 2011

=--==--=*=--=*=--=*=--=*=--=*=--=*=--=*=--=*=--=*=--=*=--=*=--=*=--=*=--=*=

First reviewer's review:

>>> Summary of the submission <<<

The paper considers the problem of translating applications from one language to another. In that setting, one challenge is mapping calls to library methods in the source language to calls in libraries in the target language. The authors propose two techniques to help. First the generate wrapper calls which insulate the application from the details of the language-specific library calls. These wrappers then permit a kind of differential testing between translated library calls to ensure that they are behaviorally equivalent, and to warn users if they are not. The authors describe their API call mapping support and how it can be integrated with multiple translators between Java and C#.

>>> Evaluation <<<

The problem of migrating applications from deprecated languages to more modern languages has been studied for decades. This paper cites one of the foundational papers in the area, Water's TSE'88 paper (number [22]). That paper identifies two basic strategies: transliteration, which is appropriate when the source and target language are a close match and abstraction and reimplementation, which is appropriate when the source and target language have fundamental differences.

In this submission, the authors appear to restrict themselves to just transliteration approaches, Java and C# are very similar languages, and in this setting it does seem as if API mapping is a thorny issue. One could, of course, view the standard libraries as a language extension in which case Java and C# would be "more different" and perhaps amenable to

abstraction and reimplementation. Unfortunately, the authors do not provide an in depth discussion of the foundational issues of translation and where API mapping fits into those issues.

Instead, the paper takes as a given that people want to do transliteration and look to remedy deficiencies in existing transliteration tools. They apply existing and well-understood testing techniques to perform differential testing of proposed API translations and they find lots of errors. I doubt that the authors of any transliteration tool claims complete/correct translation, so this isn't much of a surprise.

Given this, one is left wondering about the technical contribution of the work.
The novel aspect is the use of wrappers to enable cross-language differential testing, but that is not the primary focus of the work. In the end, I do not see any broadly applicable lessons, principles or techniques developed in the work that would advance program translation as a field. Instead, there is a minor engineering advance that might help make transliteration tools between Java and C# better - I don't regard that as significant or valuable research contribution.

I also found the evaluation to be strange in the following respect. In section 4.1, the data show that translators for Java and C# do a very poor job - at most 22% of the API calls are translated to a form without compilation errors.
For most tools the number of calls that are translated to a compilable form is below 2%. This is strong evidence that the utility of translators is severely limited. Why then does the evaluation go on to consider behavioral equivalence via differential testing? Yes, you show that existing translators are less useful than they appear, but the data from 4.1 already shows they are nearly useless without significant manual intervention. Perhaps your approach should focus on providing semi-automated translation assistance to users of such tools. I know you could use your tools in that way, but currently your paper does not make that an emphasis and an element that is part of your evaluation.

=--==--=*=--=*=--=*=--=*=--=*=--=*=--=*=--=*=--=*=--=*=--=*=--=*=--=*=--=*

Second reviewer's review:

>>> Summary of the submission <<<

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

=--==--=*=--=*=--=*=--=*=--=*=--=*=--=*=--=*=--=*=--=*=--=*=--=*=--=*=--=*

Third reviewer's review:

>>> Summary of the submission <<<

The paper's title is a bit misleading, because the paper is really mostly concerned with an empirical evaluation of several automatic tools that translate Java programs into C#, and vice versa. The paper presents an approach in which wrapper methods are generated that typically call one API method in a Java or C# program, which is then run through an existing translation tool. The output of the translation is then analyzed to find API mappings, i.e., corresponding functionality in multiple languages. In practice, the translation tools can only do a very partial job---they are unable to translate certain methods, and generate code that doesn't compile in other cases. The main interest of this paper appears to be in dealing with the cases that remain (i.e., situations where translation tools generate code that does compile) to detect where behavioral differences exist. This is done by automatically generating tests using existing techniques (dynamic symbolic execution and random test generation). The paper presents an analysis of these results, both quantitative and qualitative.

>>> Evaluation <<<

This paper takes on a very hard problem: automatic translation between different languages. Unfortunately, it has significant problems:

1. It is not clear to me what the contribution is. Clearly, the automatic translation tools under consideration can only do a very partial job as is evidenced by the many compilation errors and behavioral differences, and leave the bulk of the work for the programmer. It is not clear to me how valuable it is to expose

behavioral differences in the cases where a syntactically valid
translation exists. After all, this still leaves the major problem that
many code fragments cannot be translated in the first place. Furthermore,
I'm very unsure about how generally applicable this work is---it
seems to be very focused on Java and C#, which happen to be the
two most similar languages one can imagine for a project of this
kind.

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.
There are also numerous grammatical problems (too many to list),
and I strongly encourage the authors to get some help with the writing.

3. In several places, the authors use cryptic URLs such as
"http://tinyurl.com/2dsgftv" to refer to web pages with
additional information such as Java and C# APIS. In at least some of
these cases, the appropriate thing to do would be to include a summary
of this material in the paper (e.g., the API of ByteArrayInputStream).
I consider this to be a form of "cheating" to have more space. It is
also highly unfriendly to reviewers to require them to type in these
URLs.

Detailed comments:

- page 1: the "recent study" cited here is from 1998

- page 3: "shares the same interface". It is really unclear what is
meant here as the original and translated test methods have
signatures with different types (byte[] vs sbyte[])

- page 3: I'm confused about what kinds of test cases Pex is generating.
More detail is needed, e.g. a full example that shows the original
wrapper in Java, the translated wrapper in C#, the test inputs generated
with Pex, and the output of both versions.

- it is not clear to me how generated tests from Pex are "translated back"
into Java.

- page 5 mentions java.util is one of the translated packages. However,
the majority of classes in java.util are generic, and the authors
say elsewhere that their technique doesn't work on generic classes.
How do I reconcile this?

>>> Points in favour or against <<<

- technical approach seems ad-hoc
- contribution is unclear
- presentation is quite poor

=--==--=*=--=*=--=*=--=*=--=*=--=*=--=*=--=*=--=*=--=*=--=*=--=*=--=*=--=*