**This document explains how we have addressed each of the comments made by JSPE reviewers of the paper “A Bug Reproduction Approach Based on Directed Model Checking and Crash Traces”. We have numbered each comment. Original comments from reviewers are in bold; our responses are in non-bold text.**

# Reviewer 1:

**R1.1 Proving a formal definition of the syntax of traces;**

We added a formal definition of a crash trace generated from uncaught exceptions in Java. AMore formally, we define a Java

crash trace T of size N as a sequence of frames T = f 0 , f 1 , f 2 , ..., f N where each frame represents

either a method call and the location of said method, an exception or wrapped exception. In Figure

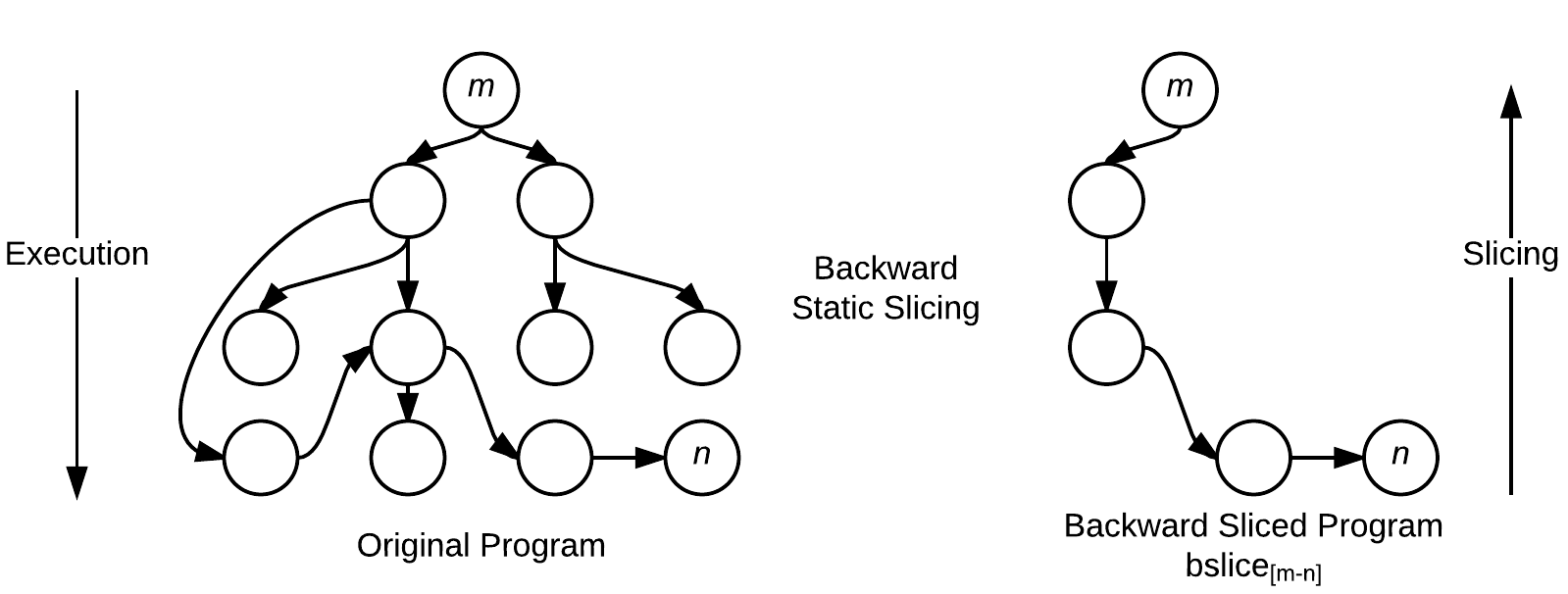
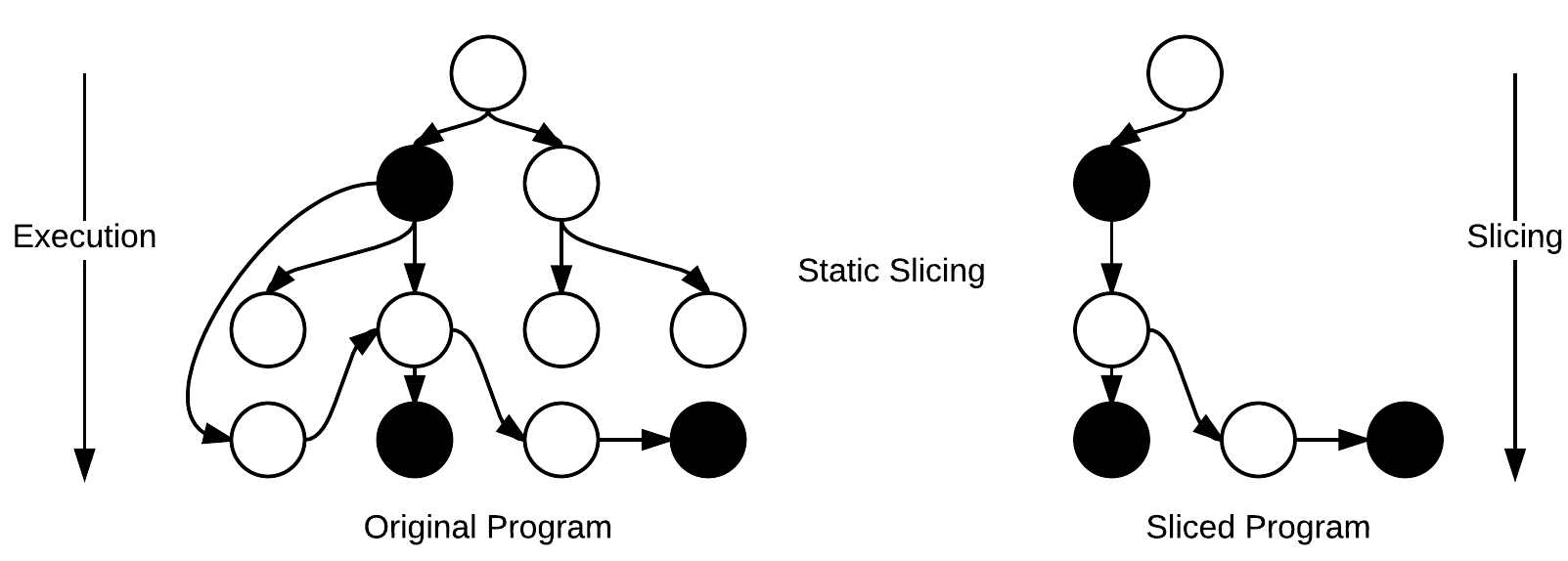
5, the frame f 0 represents an exception, the frame f 7 a method call (jsep.Foo.buggy) in a file

(Foo.java line 17) leading to the exception and f 8 represents a wrapped exception. Please see the first paragraphof section 4.1.

**R1.2. Proving a formal definition of a frame and a slice and a better discussion ("reasoning") of the algorithm;**

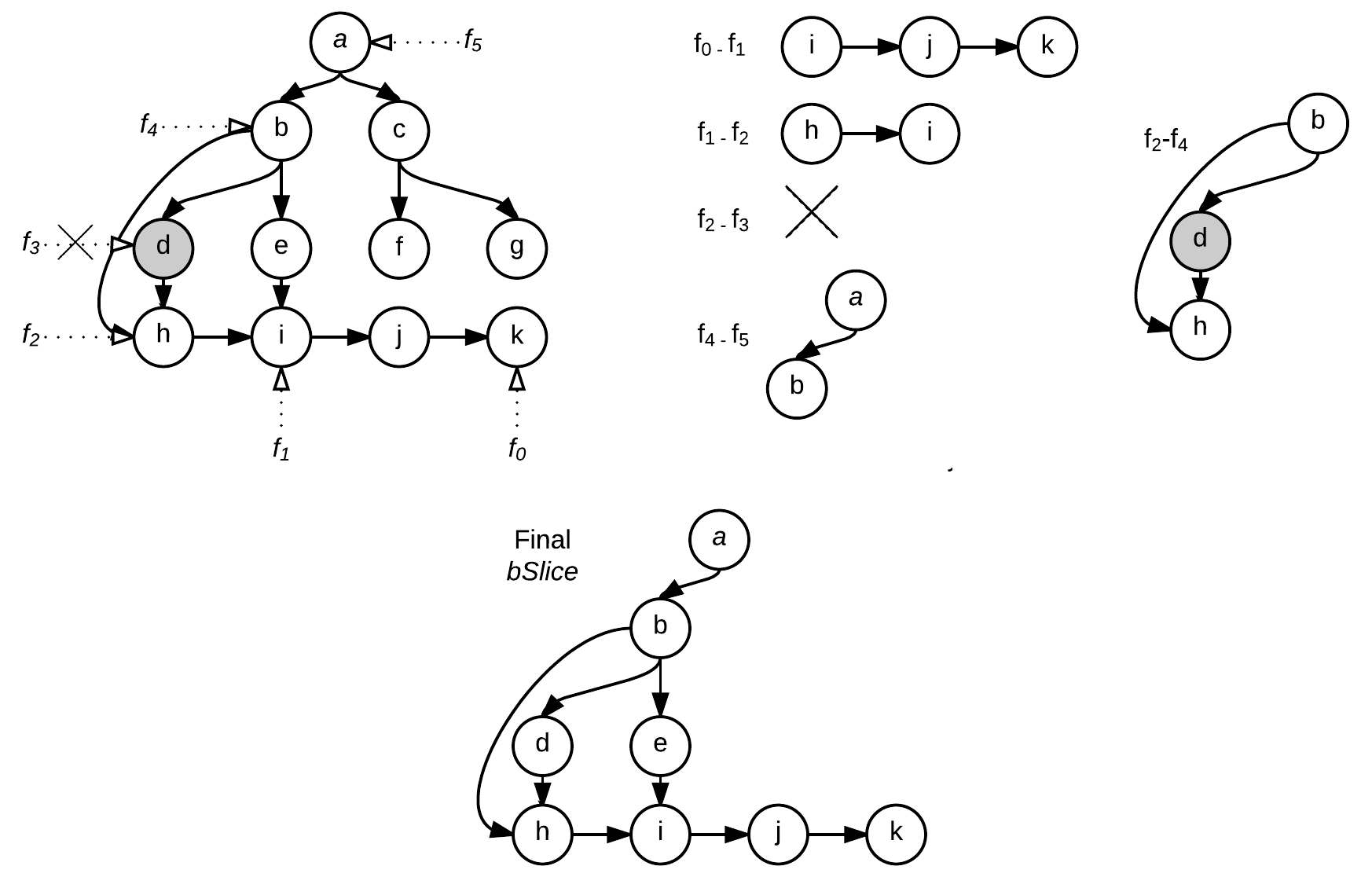
**The whole section 4.3 has been reworked. The first two paragraphs provide a formal definition of a slice and a frame. We also added two figures (6 and 7) to illustrate the definition.**

Static analysis of programs consists of analyzing programs without executing them nor making assumptions about the inputs of the program. There exists many techniques to perform static analysis (data flow analysis, control flow analysis, theorem proving, ect. . . ) that can be used to improve programs in terms of performances, understandability or to debug them. One of the product of static analysis is the static slice which takes a program and slicing properties in order to create a smaller program with respect to the slicing properties. this is particularly interesting for JCHARMING as the sliced program, being smaller, will have a smaller state space. In Figure 6, black states are states that are impacted by the slicing properties. After the slicing, the static slice only contains the black states and the states mandatory to reach them.



Regarding the algorithm, we have completed the previous discussion with more details and added two new paragraphs and a figure representing a step by step execution of our algorithm (figure 10).

[Wahab] Is there a definition of the static slice in the literature? If so, we may need to use it and add the citation. It is good to define the frame as part of the previous comment.



an hypothetical program composed of eleven states ( a...k ) crashes in k . The produced stack traces is composed of five frames T = f 0 , f 1 , f 2 , f 3 , f 4 , f 5 . The frames represents k , i , h , d , b and a , respectively. However, f 3 has been tempered with and no longer match a location inside the SUT. In such program, Algorithm 1 will begin by computing the backward static slice between f 0 ( k ) and f 1 ( i ), then it will compute the backward static slice between f 1 ( i ) and f 2 ( h ). At this point, we passed throught the f or loop (line 5 to 12) two times and both times, the backward static slice was not empty. Consequently, the if statement was equal to true and we combined both backward static slice in the bSlice variable. bSlice is equal to {k , j , i , h} . Then, we want to compute the backward static slice between f 2 ( h ) and f 3 ( d ). Unfortunately, f 3 is corrupted and do not point towards a valide location in the SUT. The corruption can be the result of a copy-pasting error or a deliberate intervention of the reported as shown in Section 6. Consequently, the backward static slice between f 2 ( h ) and f 3 ( d ) will be empty, and we will go to the else statement (line 10). Here, we simply increment of f set by one in order to compute the backward static slice from f 2 ( h ) and f 4 ( b ) instead of f 2 ( h ) and f 3 ( d ). f 4 is not corrupted and the backward static slice from f 2 ( h ) and f 4 ( b ) can be computed and merged to bSlice . Finally, we compute the last slice between f 4 ( b ) and f 5 ( a ). The final backward static slice is k , i , h , d , b and a .

**R1.3 Extending or modifying the evaluation section by a complete characterization of the partially and not reproduced bugs in the 30-x cases (table)**

[Wahab]: I am not sure what the reviewer wants here. I will come back to this after we finish the review.

**R1.4. identifying, and making explicit, the limitations of JCHARMING**

[Wahab]: Can you give me a list of limitations of JCHARMING. We need to add this to the Conclusion Section. I see a limitation of the crash trace is corrupt. What are the other limitations?

Multi-threading & Networks related bug.

Acquiring the source code or bytecode of the SUT

Accessing the bug report repository

**R1.5. The review instructions also ask for possibilities for condensing (or amplifying) the text. In my opinion, the prose in section 6 could be shortened.**

[Wahab]: Let’s deal with this after we finish the paper.

**R1.6. Page 2: l.16 "In [7]": poor style (writing rules say that the sentence must be complete if you leave out the reference), also see l.40**

We modified the paper to adopt the proper style. We thank the reviewer for pointing this out.

**R1.7. Page2: l.21 "uses a list of function outputs": every output comes from a function -> be more specific. Also, make clear that JCHARMING additionally uses the source code.**

We meant that JCHARMING takes the crash trace as input. The crash trace represents the list of function calls that are output when an uncaught Java exception occurs. We modified the text to avoid the confusion. Please see Page XX, Paragraph YY.

### R1.8 Page 2: l.28 "control flow graph" seems a bit misleading since there is no 1-1 correspondence between statements in the source and program states.

We agree with the reviewer. We meant that the technique builds a graph where each node represents one state of the program and the set of properties that needs to be verified in each state. Ultimately this graph can be mapped to the control flow, but we agree that there is no correspondence between statements and program states. We modified the text by changing “control flow graph” to “graph”.

### R1.9. Page 2: l.38 "Hadoop and, " -> Hadoop, and

Fixed. Thanks for this.

### R1.10. Page 2: l.43 systems are "relatively complex": you contradict yourself a bit here, since, at least for complex bugs, you write in Section 7 that the quality of the trace is what matters.

Yes, the quality of trace is really what matter. This said, crash traces generated from bugs in complex and large systems are expected to be harder to reproduce because they may involve many components. Since software engineers usually copy crash traces in bug descriptions, they may be reluctant to copy large traces. We modified the sentence to become: “This is explained by the fact that the new systems used in this study are relatively larger compared to those [...]”.

# R1.11. Page 3: l.29 "NP-complex" does not exist as technical term IMO. A problem can be "NP-complete" or in the "complexity class@ NP.

We replaced NP-complex by NP-complete throughout the whole paper. Thanks for this.

### R1.12. Page 3: l.35-39: don't you mean that it is impossible to reproduce the bug?

[Wahab] I don’t see this in the paper. You wrote “Not impossible, one could create an approach that re-create a network packet based on the crash characteristics.” Please point me to the paragraph so I could check.

I think it refers to *Without this input, it may be challenging to reproduce the bug.* Last sentence before 2.1

### R1.13. Page 3: What does "specific data" refer to---simply the input?

[Wahab] Same here

All kinds of possible inputs (networks, disks access, GUI interactions, …)

# R1.14. Page 4 : .08 "minimal"... what?

[Mathieu] The sentence sounds clear to me. Could you precise what you don't understand here ?

[Wahab] I agree with the reviewer. This whole paragraph is not clear: “The approach records the execution of the program on the client side and compresses the generated data to the minimal required size that ensures that the reproduction is still achievable. This compression is also performed on the client side. Moreover, the approach keeps traces of all accessed documents in the operating system and also compresses/reduces them to the minimal.” What is the minimal data here? Why is this compression so important. Can you please rephrase or provide specific examples.

The approach records the execution of the program on the client side and compresses the generated data. Moreover, the approach keeps traces of all accessed documents in the operating system and also compresses.

Overall, the approach is able to reproduce on-field bugs using a file that are 70kb in average.

### R1.15. Page 4: l.39 typo in "JRapture"

Fixed.

### R1.16. Page 4 l.42 "and on the replay": grammar

Fixed.

## R1.17. Page 5:

## l.03 "code-level" delete hyphen

### l.03 ", there exist": put in a new sentence

### l.28 "stack" -> stacks

### l.33 "prior to the work ... begins": grammar

### l.34 add a period after "i.e."

Fixed. Thanks for the detailed revision.

## R1.18. Page 6: l.42 "P is the set of properties that each state satisfies": incorrect or at least misleading scope of "the set." You mean "P associates with each state the set of properties that it satisfies."

Yes. This is what we mean. We rewrote the sentence to read: “where S is the set of states, $T \subseteq S \* S$ represents the transitions between the states and $P$ associates with each state the set of properties that it satisfies.. The SUT is said to satisfy a set of properties p when there exists a sequence of states x leading towards these properties.”

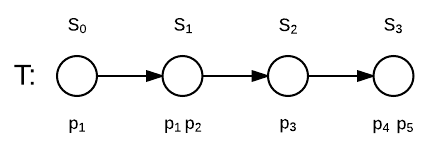
### R1.19. Page 6: l.47 what exactly is "x": a state or a \_sequence\_ of states (path)? I am familiar with the former, i.e., the situation where one asks the model checker whether property p holds in state x of model M, and then might get back the trace that constitutes a counter example. If "x" really is a path, then, for example, "x \in S" (Eq. 1) wouldn't hold, since $S$ is a set of states. Also, in Eq. (3), you would quantify over paths rather than states.

[Wahab] Mathieu, you need to address this very thoroughly. If you say that it is a sate then you need to explain what the path is. Make sure the equations are all correct. You need to provide a good discussion. If we don’t address these comments, he will reject the paper.

All the equation of section 3 (Preliminaries) have been reworked in order to adopt more standard definition of model checking. It now reads

SU T =< S, S 0 , T, L >

where S is a set of states, S 0 the set of initial states, T the transitions relations between states and L the labeling function which labels a state with a set of atomic properties. Figure 1 presents a system with four steps S 0 to S 3 which have five atomic properties p 1 to p 5 . The labeling function L gives us the properties that are true in a state: L(S 0 ) = {p 1 }, L(S 1 ) = {p 1 , p 2 }, L(S 2 ) = {p 3 }, L(S 3 ) = {p 4 , p 5 } .



The SUT is said to satisfy a property p at a given time where there exist a sequence of states x leading to a state where p holds. This can be written as:

(SU T, x) |= p

For the SUT of Figure 1, we can write (SU T, {S 0 , S 1 , S 2 }) |= p 3 because the sequence of states {S 0 , S 1 , S 2 } will lead to a state S 2 where p 3 holds. However, (SU T, {S 0 , S 1 , S 2 }) |= p 3 only ensures that ∃x such that p is reached at some point in the execution of the program and not that p 3 holds for ∀x. [...] As we are interested in verifying the absence of

unhandled exceptions in the SUT, we aim to verify that for all possible combinations of states and

transitions there is no path leading towards a crash. That is:

∀x.(SU T, x) |= ¬c

If there exists contradicting path (i.e., ∃x such that (SU T, x) |= c ) then the model checker engine

will output the path x (known as the counter-example), which can then be executed. The resulting

Java exception crash trace is compared with the original crash trace to assess if the bug is reproduced. While being accurate and exhaustive in finding counter-examples, model checking suffers from the state explosion problem, which hinders its applicability to large software systems.

We thank to reviewer for pointing out these imprecisions on the first version.

### R1.20. Page 6: l.52 "specification" is not a good term here (and has a different meaning in formal methods), do you mean additional assumption?

We agree. We changed the sentence to: “In JCHARMING, we assume that SUTs must not crash…”

### R1.21. Page 6: l.52 just as a remark, "fairness" has a special meaning in (LTL) model checking.

Good point. We agree with this. We change the term “fair environment” to “ typical environment” and defined what we mean by typical environment by adding the following sentence: “In the framework of this study, we consider a typical environment as any environment where the transitions between the states represent the functionalities offered by the program.”

## R1.22.

## Page 7: l.17 "depicts" -> depict

## Page 8: l.51 "are" -> is

Fixed. Thanks.

## R1.23. Page 9: I don't understand Figure 5. Why is there a call to Foo.bar (even though in Figure 1 there is no edge from Bar.foo to Foo.bar)? What is "looptimes" - in Figure 1, the bound is called "loopCount"?

In Figure 1, the entry point of the program contains edges toward Foo.Bar and Bar.Foo.

The Java language provides the InvalidActivityException class which is a specialization of the Exception class. The Exception class have a String constructor that allows developers to define a comment to be displayed of the exception is thrown. In Figure 5, a statement throw new InvalidActivityException("loopTimes"); will have been inserted after a if(loopCount > 2).

[Wahab] Mathieu, can you explain more?

### R1.24. Page 9: Where is the call to Bar.foo the "crash trace contains" (l.23).

It is included in "and 4 more" frame as explained in the Java documentation (http://docs.oracle.com/javase/6/docs/api/java/lang/Throwable.html) : “This line indicates that the remainder of the stack trace for this exception matches the indicated number of frames from the bottom of the stack trace of the exception that was caused by this exception (the ``enclosing exception''). This shorthand can greatly reduce the length of the output in the common case where a wrapped exception is thrown from the same method as the ``causative exception'' is caught.”

### R1.25. Page 9: Generally, I strongly recommend to include a grammar of a trace; this would also help answering some of my later questions (see below).

The grammar for Java stack trace is both very verbose. This is why we did not include it in the paper. We added a reference to an ANTLR (<http://www.antlr.org/>) V3 grammar for Java Stack traces written by Luca Dall'Olio, Christian Cassano and Gabriele Contini. This grammar is used extensively in software engineering.

### R1.26. Page 9: l.38 typo in the caption; I would also reformat the caption to avoid the lonely "2."

Fixed. Thanks.

### R1.27. Page 9: l.42 what exactly is a frame? Put differently, what is the difference between a line number in a trace and a frame? Does f\_0 have to contain a property (in terms of a programming language: an expression), as Figure 5 and p. 12 suggest, or does it have to contain an exception?

As shown in Figure 5, we can see that the first line is the called frame 0 (referred to as frame f\_0), then subsequently the next line is called frame f\_1 , etc.

[Wahab] You need to explain what a frame is in more detail. This comment can be fixed if you fixe well R1.2. When addressing R1.2, it is worth adding a paragraph of what a frame is with some examples.

A Java crash trace T of size N as an ordered sequence of frames T = f 0 , f 1 , f 2 , ..., f N . A frame represents either a method call preceded with its location in the source code, an exception, or a wrapped exception.

In Figure 6, the frame f 0 represents an exception, the frame f 7 a method call (jsep.Foo.buggy)

in a file (Foo.java line 17) leading to the exception, and f 6 represents a wrapped exception.

It is common in Java to have crash traces that contain wrapped exceptions. Such crash traces are

incomplete in the sense that they do not show all the method calls that are invoked from the entry

point of the program to the crash point. According to the Java documentation [30], line 8 of Figure

6 should be interpreted as follows: “This line indicates that the remainder of the stack trace for this

exception matches the indicated number of frames from the bottom of the stack trace of the exception

that was caused by this exception (the “enclosing exception”). This shorthand can greatly reduce

the length of the output in the common case where a wrapped exception is thrown from the same

method as the “causative exception” is caught.”

We are likely to find shortened traces in bug repositories as they are what the user sees without

any possibility to expand their content.

### R1.28. Page 9: l.46 "buggy ": delete the extra blank.

Fixed. We also proof read the paper over and over again to avoid these typos.

**R1.30. Page 10**

### l.17 delete the extra "the."

### l.21 "out of control" -> beyond control

Fixed.

**R1.31. Page 10: l.48 I am not sure whether I completely understand what a slice is in your context: does a slice consist of statements and declarations from the program source? And do frames constitute slicing criteria, or (since this wouldn't work) phrased more precisely: the method invocation they contain? Please provide a formal definition of a slice.**

[Wahab] You need to explain here what a slice is with a simple example that the reviewer can understand. You need to add a small paragraph in the paper that describes what a slice is.

Fix it in 1.2

### R1.32. Page 10: l.54 typo in f\_{n 1}

Fixed.

**R1.33. Page 10: l.55 in my understanding of a slice, the order of statements is essential, especially if the slice has to remain executable. Rather than forming the union of slides, you should therefore, e.g., concatenate them. (Perhaps the implementation is based on a union that happens to preserve the order, so that you never faced the difference between a set and a sequence). Algorithm 1 needs to be changed accordingly.**

[Wahab] You need to explain very well to the reviewer why you are doing a union. It is clear that he does not understand the concept of slicing. You should help this reviewer understand.

Also in 1.2 (Figure 10).

## R1.34. Page 11: l.09 "allow to reach": grammar

Fixed.

### R1.35. Page 11: Figure 6: what is "entry"? What is the purpose of the two self-loops in z0 and z1?

By entry, we mean the entry point of the program, i.e. the main in Java. We just put the self-loops there because they can occur. It was just part of the example. We could have picked another example.

### R1.36. Page 11: l.31 as a comment, the "worst case scenario" is the best-case scenario from the perspective of state space explosion...

We agree that we can at look at it this way, but what we are really interested in is the worst case scenario of our approach.

### R1.37. Page 11: l.44 how can a slice be empty? Since there is an underlying execution between frame f\_i and f\_{i+1}, there have to be program statements, no? Again, a definition of a slice would have helped me (perhaps). I also didn't understand why "offset" is reset to 1 in Algorithm 1.

The slice could be empty if the entire crash trace is corrupt. We simply can collect other parts of the program that will be used to guide the model checker. We added the definition of a slice when addressing comment R1.3. I hope this is all clear now. [Wahab] Please make sure that all the comments connect to each other at the end.

### R1.38. Page 11: l.48 "would": is colloquial

[Wahab] He means that is informal. You can change it to something else, like “will”. Don’t forget to address the comment too.

### R1.39. Page 11: l.55 comma after WALA[31]

Fixed.

## R1.40. Page 12: l.6-l.14 what is the point in this paragraph? "At first sight, it may appear that static slicing alone be used..."... OK, and what at second sight?

The second sight is “The result will be a dynamic slice. This assumes that we know in advance which input we need to provide to the program in order to reach the crash, which defeats the purpose of bug reproduction in the first place.”

### R1.41. Page 12: l.20 "frame" -> frames

We could not find this typo.

### R1.42. Page 12: l.24 shouldn't "i+offset" be frames[i+offset]?Algorithm 1: shouldn't the slicing criterion change in each iteration of the loop?

[Wahab] You should explain that the slicing criterion does not change. Explain also why not frames[i+offset]. A way to address this comment is to explain why you used i+offset. The idea is to explain even if you feel that the comment is not clear.

Yes, it should be frames[i+offset]. Thanks for pointing this. Line 6 of Algorithm now reads : BSlice currentBSlice ← backward slice from frames[i] to frames[i + offset];

The slicing criterion do change between each iteration. At each iteration, we ask WALA to perform a new backward static slice between two different point of the SUT. The slice is static as we don't assume any knowledge about the SUT nor the root cause of the crash. Consequently, WALA only needs SUT locations to perform the slicing.

### R1.43. Page 12: l.40, Eq.(6): I need some clarification on the premise of \models: assuming "x" is a path, what does "x." mean? Also, SUT and the union of slides are quite different types, why is it legal for both to write "x."?Further, what is the first subset relation good for (the one without the "x"-qualification)? Please introduce the notation and explain the different terms formally.

[Wahab] Please explain.

We thank the reviewer for pointing out this equation. We reworked the whole passage in order to provide a better understanding.

Equation 6 has been slitted in two (equation 6 and 7) and follows the same notation as equations 1, 2 and 3.

Using backward slicing, the search space of the model checker that processes the example of

Figures 2 to 4, where a crash happens when i > 2 , is given by the following expression:

Sliced\_{SUT} =

\begin{pmatrix}

\bigcup\_{i=0}^{entry} bslice\_{[f\_{i+1} \leftarrow f\_i]} \subseteq SUT, \\

S\_0, \\

T.\bigcup\_{i=0}^{entry} bslice\_{[f\_{i+1} \leftarrow f\_i]} \subseteq T.SUT, \\

L

\end{pmatrix}

Where i=0 bslice [f i+1 ←f i ] ⊆ SU T is the subset of states that can be reached in the computed

entry backward slice, S 0 the set of initial states, T. i=0 bslice [f i+1 ←f i ] the subset of transitions relations between states that exist in the computed backward slice and L the labeling function which labels a state with a set of atomic properties. Then, in the sliced SUT, we try to find:

(Sliced SU T , x) |= p i>2

That is, there exists a sequence of state transitions x that satisfies p i>2 .

### R1.44. Page 12: l.45 What do you mean by "c\_{i>1}" needs to be included"? Is this an extra step or does the slicer find that "c\_{i>1}" automatically? And why can one include expressions anyway? Doesn't the slice contain statements?

This paragraph was particularly unclear. What we meant is *The only frame that needs to be untouched for the backward static slice to be meaningful is f 0 . In Figure 6, f 0 is at F oo.bar(F oo.java : 10) . If this line of the crash trace is corrupt then JCHARMING cannot perform the slicing because it does not know where to start the static backward slicing from. The result is a non-directed model checking which is likely to fail.*

We added this after equation 7.

### R1.45. Page 12: l.46 "needs to be untouched": I can't follow

If the first line of the stack trace, i.e., frame 0, is corrupt then JCHARMING cannot perform the slicing because it does not know where to go. The result is a non-directed model checking which is likely to fail. We added a sentence to make this clear (please see paragraph last paragraph before 4.4). [Wahab] Please add this in the paper.

## R1.46. Page 13

### l.05 delete comma after "each"

### l.10 delete "entry"

### l.11 typo in s\_i+1

### l.16 "As shown before": where?

The typos have been fixed. For comment I.16, we added a reference to Figure 6.

# R1.47. Page 14: l.48 "an hypothetical" -> a hypothetical

Fixed.

### R1.48. Page 15: What are the limitations of the unlowering steps? Which parts can you not reconstruct? Please make explicit.

[Wahab] You need to address this. What does he mean by unlowering?

I don't know :x

## R1.49. Page 15: Figure 8: faillure -> failure (3x)

Fixed

### R1.50. Page 15: l.49 "Kilo Line": I've never seen KLoC spelled out like this.

It is spell KloC in many of our references. Which spelling would you prefer ?

### R1.51. Page 15: l.52 "make files": shouldn't it be Makefiles?

It now reads Makefiles.

## R1.52. Page 16

### l.22 "player in" -> player among

### l.55 "built on the top" -> built on top

### l.56 "that" -> , which

### l.08, l.09: delete blanks before the setTarget... method (line 3 and 4 in trace, also in the trace on page 22)

### l.38: add comma after "class"

Fixed.

### R1.53. Page 16: Removal of lines in the preprocessing step could also mean that variables relevant for control- or data dependence are not discovered, correct? This could then cause problem in the construction of test cases, correct?

In fact, the removed lines belong to software that are not part of the SUT such as the Java primitives or embedded libraries. If the bug lies there (outside the SUT) we can't reproduce it because JCHARMING will not be able to find the point of entry to compute the static slice. The preprocessing step does not affect the SUT. In other words, we do not remove anything that is part of the SUT. In other words, the removed variables and lines won’t affect the construction of test cases since the test cases are designed to exercise the SUT (which calls Java libraries). The test cases are not designed to exercise Java libraries directly.

## R1.53. Page 21: Both cases in Section 6.2 I don't find particularly enlightening. Surely, not match-able segments of traces present problems, but that is hardly surprising. Besides, in the second case, JCHARMING could employ a different definition of equality and in the first case include a heuristic. Instead of discussing what is more or less obvious, it would be much more interesting to count the frequency of those cases. Or to cluster resp. classify them, or to discuss how JCHARMING could improve its precision. Asked differently, what characterizes the "quality of ... crash trace" you refer to on page 23?

A good crash trace is a trace where all the frames are intact. In other words, there is no corrupt frame. This may happen but not always. This is why we introduced the partially reproduced bugs. We agree with the reviewer that a study that focuses on understanding what causes the poor quality of traces will be worth conducting. This is however beyond the scope of our paper. The results of such a study can help develop tools that would allow the generation and the reporting of good quality traces. We intend in the future to undertake this study.

### R1.54. Page 21: Several bug reports contain spelling mistakes (the two reports on page 21 contain 2 resp. 1 typo, the one on p.22f. another one). Are those in the original reports? (If so I'm not sure if one is allowed to correct them. lease inform yourself and get back to me :-)

Yes, these are the original reports. We did not want to modify them. We added a sentence in the paper to say that the bug descriptions are reported in the paper as submitted by the developers. [Wahab] Please add this sentence in the paper.

Last two sentences before 6.1

## R1.55. Page 22

### l.42-46 I can't follow your argumentation: why did JCHARMING not find the crash location? What is different in the case of Struts?

This is because the error belongs to Struts which is another software application used by Log4J, the SUT. We did not have access to Struts.

### R1.56. Page 22

### l.47 capitalize log4j

### l.50 add comma after 46721

Fixed.

### R1.57. Page 22: l.50/51 "We believe that JCHARMING could have successfully..": why don't you just try it? Is it so complicated to rerun JCHARMING?

The generated exception does not belong to Struts system. Indeed, the first frame is from the Struts program, the rest of the stack trace belongs to log4j. As we were using Log4J as SUT, JCHARMING could not perform the backward static slicing and fall back to undirected moel checking of Log4j as a whole. This undirected model checking will fail has for bugs 11570, 40212, 41186, 45335, 46271, 47912 and 47957 (Table II).

[Wahab] Mathieu, your answer is not clear. Please rephrase.

## R1.58: Page 23: l.27 add an "a" before "program's behavior"

### Fixed.

### R1.59. Page 23: l.50-54: correct, certain failures are "hardly reproducible." What to conclude, though?

[Wahab] I will look into that later.

### R1.60. References (plenty of problems):

**Capitalization inconsistent in conference titles (e.g., [1,2,3,4]). Capitalization wrong in first names, tools, etc (e.g, [8,17,19]). Problems with non-ASCII letters (e.g., [15,25]). Incomplete reference (e.g., [14,17])**

[Wahab] Mathieu, please go ref by ref and fix the problem.

# Reviewer: 2

1. **R2.1. The information obtained in the backward slicing operation serves as a way to guide the model checking activity. The approach is interesting, and reasonably well explained, but the formal aspects need to be improved. Whenever formal notations are used, I got quite confused by the inconsistencies that are currently in the text (see my detailed comments). I also would have liked to learn more about the way properties are expressed. This is not explained at all in the current text.**

All the equations have been reworked and their description adapted to improved their formal aspects.

For the properties, we added two paragraphs explaining how JPF handles properties.

*The JPF model checker can execute all the byte code instructions through a custom JVM — known as JVM JPF . Furthermore, JPF is an explicit state model checker, very much like SPIN [33]. This is contrasted with a symbolic model checker based on binary decision diagrams [34]. JPF designers have opted for a depth-first traversal with backtracking because of its ability to check for temporal liveness properties.*

*More specifically, JPF’s core checks for defects that can be checked without defining any property. These defects are called non functional properties in JPF and cover deadlock, unhandled exceptions and assert expression. In JCHARMING, we leverage the non functional properties of JPF as we want to compare the crash trace produced by unhandled exceptions in order to compare them to the bug at hand. Consequently, we do not need to define any property ourselves. However, in JPF, one can define its own properties by implementing listerners — very much like what we did for Section 4.6 — that can monitor all actions taken by JPF. which enables the verification of temporal properties for sequential and concurrency Java programs. One of the popular listener of JPF is jpf-ltl. This listener supports the verification of method invocations or local and global program variables. jpf-ltl can verify temporal properties of method call sequences, linear relations between program variables and the combination of both. In a near future, we might use jpf-ltl and the LTL logic to check multi-threaded related crashes, as described in Section 8.*

1. [Wahab] We need to address this comment by indicating places in the paper where improvements have been made and by referencing the comments below.
2. **R2.2. Regarding the directly related work, I think the authors give a nice overview, but with respect to "directed model checking", the authors do not explain its origins at all (again, see detailed comments).**
3. [Wahab] I agree. I suggest that you need the reference that the reviewer suggested in Revie R2.27.

Model checking, on the other hand, explores each and every state of the program (Figure 3), which

makes it complete, but impractical for real-world and large systems. To overcome the state explosion

problem of model checking, directed (or guided) model checking has been introduced [28, 29].

Directed model checking uses insights – generally heuristics – about the SUT in order to reduce

the number of states that need to be examined.

1. **R2.3. The main benefit of model checking over testing is its ability to determine that a system is absolutely correct. By starting with a bug and guiding the checking to it, you are essentially not model checking anymore, but bug hunting. Of course, directed model checking in a way depends on there being a bug, otherwise is devolves into traditional model checking. A discussion along those lines should be added, I think, to make this clear to the reader, since "model checking" usually has this completeness characteristic of being able to show that a system is correct.**

A small discussion have been added before section 4.

*As powerful as directed model checking is, it loses the advantage of model checking over testing: completeness.*

*Indeed, model checking ensures that a system is absolutely correct.*

*With directed model checking, we are not ensuring the absolute correctness of the program, but hunting for a bug inside a subset of the said program.*

1. [Wahab] We should add the discussion the reviewer is suggesting.
2. **R2.4. Can you say anything regarding the length of your counter-example traces? For understandability, it is crucial that counter-examples are short. It would be interesting to know whether JCHARMING tends to produce the shortest counter-example or not, and if not, whether you could alter the technique to improve this.**

Thanks for suggesting this. We report the amount length of counter-example in a new column of Table 2. Also, we discuss this column in section 6.

*Finally, we report the number of statements in the produced JUnit test, i.e. the counter example. While*

*reproducing a bug is the first step in understanding the cause of a field crash, the step to reproduced*

*should be as few as possible. Indeed, it is crucial for counter-examples to be short to effectively*

*help the developer craft a fix. In average, JCHARMING counter-example were composed of 5.04*

*Java statements. While it is difficult to estimate if the generated counter-example are the shortest*

*possible without a deep understanding of the system at hand, we are confident that five steps is short*

*enough for the developer to understand the problem.*

1. [Wahab] Again, we need to address this.
2. **R2.5. Can you produce results that show to what extent your approach is optimal? How many states were visited vs. how many absolutely need to be visited to find the counter-example? Experiments like that would give an indication how well JCHARMING is able to find the counter-example based on the backward slice information.**
3. [Wahab] I think we need the data he is suggesting.
4. I would have to re run the experiments. Not an option given the remaining time.
5. **R2.6.**

## Page 1 - piece -> pieces

1. **Page 2 - functions output -> function outputs**

## Page 3 - oversee -> oversees

### Page 3 - a NP-Complex -> an NP-complete

1. We fixed all these typos. We changed NP-complex to NP-complete everywhere in the paper. We thank the reviewer for pointing out these typos.

### R2.7. different SMT (satisfiability modulo theories) solvers: different from what?

1. We meant various SMTs and not ‘different’. We replaced the word different by various.

### R2.8. model checking techniques: SMT is being used for model checking itself,so the "even" remark seems to be odd.

1. We rephrased the sentence to read: “In order to overcome these limitations, some researchers have proposed to use various SMT (satisfiability modulo theories) solvers \cite{Dutertre2006} and model checking techniques \cite{Visser2003}.

### R2.9. For example, the reading of a file that is only present on the hard drive of the customer or the reception of a faulty network packet: This is not a complete sentence. Please complete it or attach it to the previous sentence.

1. We modified the sentence as follows: “It is worth mentioning that both categories share a common limitation. It is possible for the required condition to reproduce a crash to be purely external, i.e., the reading of a file that is only present on the hard drive of the customer or the reception of a faulty network packet \cite{Chen2013a, Nayrolles2015}.”

### R2.10. reproduce a crash to be purely external.: so, is it possible to reproduce the crash / bug or not? You seem to indicate both.

1. What we mean is that it is challenging but not impossible. One can always investigate how to simulate scenarios with external input. We do not cover this in the paper. In our view, it would be too restrictive to say that a bug reproduction technique cannot reproduce the bug just because it does not have the input data.

## R2.11.

## Page 4 - file whose size averages 70KB -> files that are 70Kb on average

1. Fixed.

### R2.12. Similarly, private: similar to the approach of Clause et al.? Please mention this

1. [Wahab] I don’t see what we need here.
2. Neither do I.

### R2.13

### clones: remove s

### JRaptrue -> JRapture

### creator -> creators

### it saw -> as

### phase: a proper ending of this sentence is missing, e.g. "are presented"

### Fixed. Thanks.

### R2.14. that can also monitor other software system than the intended ones: why is that a problem?

1. It's violation of the host's privacy. You'll not only monitor your system but every Java systems running on the host.

### R2.15.

### system -> systems

## Page 5 - level, there -> level. There

### Page 5 - lies: relies?

### Page 5 - really begins -> really beginning

## Page 6 - used based -> based

1. Fixed. Thanks.

### R2.16. Except for STAR: so how does JCHARMING compare to STAR?

1. [Wahab] You need to discuss the conceptual similarities and differences. How is JCHARMING similar to or different from STAR. You can then add a paragraph like the one below explaining why you did not compare the two approaches experimentally.
2. We couldn't compare to STAR. STAR's source code is available but do not compile. We will have to re-implement the whole approach to compare both tools. Accounting for programming mistakes and imprecision about the approach in Chen's Ph.D thesis, the comparison wouldn't be accurate.

### R2.17. The system: for model checking, it is crucial to mention that the system is formally defined (has a clear semantics)

1. We agree with the review. We modified the text to: “Model checking (also known as property checking) will, given a formally defined system (that could be software \cite{Visser2003} or hardware based \cite{kropf1999introduction}), check if the system meets a specification by testing exhaustively all the states of the system under test (SUT), which can be represented by a Kripke \cite{Kripke1963} structure:”

### R2.18. P is the set of properties that each state satisfies → P is a set of state predicates, and there should be a function assigning subsets of P to states. Typically, however, P does not hold in its entirety in all reachable states

1. We agree. We modified the text as follows: “where S is the set of states, $T \subseteq S \* S$ represents the transitions between the states and $P$ is a set of state predicates, and there should be a function assigning subsets of P to states (P does not necessarily hold in all reachable states).”

### R2.19. The SUT is said to satisfy a set of properties: you seem to give a definition here of satisfying a property that corresponds with testing. For model checking, a property is said to hold if there exists no trace contradicting it.

1. We agree as well. This is what we meant. We changed the sentence to “The SUT is said to satisfy a set of properties $p$ when there exists no sequence of states $x$ leading to a state contradicting $p$”.

### R2.20. not that p holds nor that ∀x, p is satisfiable: if that is the case, then the system does not satisfy p.

1. This is correct. p refers to a state where the program crashes. Therefore, we don't want the program to satisfy p.

### R2.21. fair environment: is there any relation to fairness in formal verification, as in "Behaviour that is infinitely often enabled is also infinitely often executed"?

### We used the term “fair environment” to mean a typical environment where everything runs normally. To avoid confusion, we changed the word “fair” to “typical” and added a sentence to describe what we mean by typical environment.

### R2.22. properties: How are the properties expressed, using which temporal logics?

1. [Wahab] You need to see Sofiene on this. The reviewer is asking about which logic is used LTL, etc.
2. In Java Pathfinder (the model checking engine we extended), the system is exercised until it crashes. If no crash can be yield, then the property The program must not crash is satisfied. There is only one property that we don't need to define. The plural here is only use in the context of preliminaries about model checking.

## R2.23. Page 7: - (3): this is closer to satisfying a property in a model checking context. Usually, the "for all x" does not need to be added though. In fact, the shorter notation SUT |= p would be more in line with what is common

1. We agree. We modified the equation.

### R2.24. such a path exists -> there exists a path contradicting this

1. We agree with this as well. It makes the paper sound better. We modified the sentence as follows: “If there exists a path contradicting $p$ (i.e., $\exists x$ such that $(SUT, x) \models c$) then the model checker engine will output the path $x$ (known as the counter-example), which can then be executed. The resulting Java exception crash trace is compared with the original crash trace to assess if the bug is reproduced.”

### R2.25. what to look for in order to detect the causes: do you mean intermediately, i.e. along the trace? Because the final goal is known here (i > 2)

1. [Wahab] We need to see what this means.

### R2.26. complete, but impractical: that depends on the search order, which, as you mention later, opens the door for directed model checking

1. [Wahab] We need to see what this means.

### R2.27. directed (or guided) model checking has been introduced [28]: actually, the term "directed model checking" was coined in an earlier paper, namely "Directed Explicit-State Model Checking in the Validation of Communication Protocols", by Edelkamp, Leue, and Lluch-Lafuente. Also, you should refer to "Survey on Directed Model Checking" by Edelkamp, Schuppan, Bosnacki, Wijs, Fehnker and Aljazzar

1. Thanks for these links. We added them in the paper.
2. [Wahab] Mathieu, you can use one of the paper to address Comment R2.2

### R2.28

### use insights -> uses insights

## Page 9: Figure 5: Bar.Goo -> Bar.Foo. Also, it may improve readability if you add f0, f1 etc to the figure, as you refer to these in the text

## Page 10: that led to the: remove the

### are not limited to, -> but not necessarily, to

### also reduce -> reduce

### foo: previously, "foo" was capitalised. Please be consistent

### foo: see previous statement

### Fixed. Thanks.

### R2.29.

## Page 11:

### - equation 5: I have a number of comments regarding (5):

### \* i ranges from 0 to entry, but f\_entry was previously not defined, only just entry

### \* i should range to entry-1, since otherwise there should also be an f\_entry+1, and this appears not to be the case

### \* equation 4 states that the union equals the bslice from f\_0 to entry, here it is a subset. Which one is correct?

1. Both. A backward slice (bslice) is a subset.

### [Wahab] I stopped here.



### Indeed, in Figure 6, the set of states: based on figure 6, this set should be empty, as f\_2 has no outgoing transitions. The other sets you mention (f2 to f1 and f1 to f0) are also empty.

1. The states were inversed. The sentence now reads “Indeed, in Figure \ref{fig:jcharming-slice}, the set of states allowing to reach $f\_2$ from $f\_0$ is greater than the set of states to reach $f\_2$ from $f\_1$ plus set of states to reach $f\_1$ from $f\_0$.”

### assuming that z2 is a prerequisite: what is a prerequisite?

1. I prerequisite is a state where you must pass.

### Which state in figure 6 corresponds with entry?

1. F2.

### Why is bslice from f\_0 to entry the given set of states?And the sentence ends prematurely. Which set corresponds with the union of the given bslices? This example is very confusing. Please fix this.

### between each frame -> between each two frames

1. This has been corrected.

### From line 1 to line 5: in Alg. 1, please number the lines, as you use these numbers in the text. Also the lines in the text are wrong. Instead of "line 1 to 5", you should write "line 1 to 6", and instead of "line 6 and ends at line 15" you should write "line 7 and ends at line 15".

1. The lines have been numbered and the line number fixed in the algorithm's description.

### then JCHARMING: remove then

1. The then has been removed

### the possibility to resort to non-directed model checking: and what if the final slice is not empty? How does it help the directed model checking? This is explained later, but it would improve the text if you already give some indication how this works here.

### provides -> provide

1. The typo has been fixed.

## Page 12:

### Frames frames ← extract frames from crash stack;: do not write text in math mode

1. The text is no longer in math mode.

### size of frame: which frame? Multiple were extracted in previous line. Or do you mean frames?

1. Yes, I meant frames. This typo has been corrected.

### equation 6: I have the same issues with this equation as with equation 5. In addition:

### \* are the two cases between the brackets conjunctives? How are they combined?

### \* what is the difference between the two cases? How should I read the second one?

1. A comma was missing in the equation. I believe it's easier to understand now.

### both the transitions and the states are entry elements: how can this be interpreted from the equation?

1. See previous answer.

### That is the -> That is, the

1. A comma has been inserted here.

### only frame that needs to be untouched for the backward static slice to be meaningful is f0.: can you elaborate on this? Why is this the case?

1. F0 is the crash point. If the frame containing the crash point is corrupted or missing, then we can't perform the slicing (as we don't have any direction to point to). If this is the case, the slicing will fail and we will fall back into undirected model checking.

## Page 13:

### choose -> chose

1. The typo has been fixed.

### each, forward -> each forward

1. The typo has been fixed.

### with entry the states: something is wrong here. Please fix

1. I couldn't find this.

### that falls -> that fall

1. The typo has been fixed.

### a set a property -> either "a property" or "a set of properties"

1. The typo has been fixed.

### transitions -> transition

1. The typo has been fixed.

### t is the percentage: do not start a sentence with a mathematical symbol

1. The sentence now reads : “Thus, $t$ is the percentage of identical frames between both crash traces.”

### the exercise of the bug -> execution of the bug, or "the bug to appear"

1. The sentence now reads “To help software developers reproduce the crash in a lab environment we automatically produce the JUnit test cases necessary to run the SUT to cause the bug to appear.” as suggested.

## Page 14

### industrial size -> industrial sized

1. The typo has been fixed.

### listens the -> listens to the

1. The typo has been fixed.

## Page 15:

### a loop from 0 to 3: what do you mean by this?

1. The sentence now reads “Then, the jsme.Bar.foo(int) method will execute a $for-loop$ from $i=0$ until $i=3$ and [...]”

### as shown in Figure 8.: please discuss figure 8

1. Figure 8 as been described. The paragraph now reads : “The generation of the JUnit test itself is based on templates and targets directly the system under test. Templates are excerpts of Java source code with well-defined tags that will be replaced by values. We use templates because the JUnit test cases have common structures. Figure 8 shows our template for generating JUnit test cases. In this figure, each { % % } will be dynamically replaced by the corresponding value when the JUnit test is generated. For example, { % SUT % } will be replaced by Ant if the SUT is Ant. First, we declare four variables that contain the faillure, the threshold above which a given bug is said to be partially reproduced, the differences which count how many lines differ between the original failure and the failure produced by JCHARMING and a StringTokenizer. The StringTokenizer allows to break the original failure into tokens. Second, the test method where { %SUT% } is replaced by the name of the SUT and { %STEPS% } by the steps to make the SUT crash. Then, the stack trace related to the crash is received in catch part of the try-catch block. In the catch part, we compute the number of lines that do not match the original exception and store it into differences † .Finally, the assertTrue call will assert that the stack traces from the induced and the original crash are at least threshold percent similar to each other.”

### study -> studies

1. The typo has been fixed.

### reasonable amount of time: what is a reasonable amount of time?

1. A reasonable amount of time refer to a task that can be done several time a day by a developer, using its own hardware, without jeopardizing its workday. The point of JCHARMING is to speed-up the reproduction process and therefore the creation of a fix. Consequently, we don't want our process to take hours to complete. In the experiments, a reasonable amount of time has been set to 60 minutes.

## Page 16:

### time JfreeChart -> time. JfreeChart

1. The typo has been fixed. The sentence now reads : “JfreeChart [39] is a well-known library that enables the creation of professional charts. Similar to dnsjava, it has been maintained over a very long period of time—JfreeChart was created in 2005— and it is a relatively large application.”

### on the top -> on top

1. I couldn't find this.

### Mahout: in the table, swap the entries for Hadoop and Mahout, to keep it in line with the order in the text

1. Mahout and Hadoop have been swapped in Table I.

## Page 17:

### above t=80%: and below 100%, I assume

1. The sentence now reads : The result is ``Partial'' if the similarity between the crash
2. trace generated by the model checker and the original
3. crash trace is above t=80\% and below t=100\%.

### fill out all -> fill all

1. The typo has been fixed.

## Page 18:

### dispatching:”: end sentence with '.'

1. This a quote from the original bug report where the sentence ends with a “:”. Furthermore, the “:” introduces the stack traces to come.

## Page 19:

### neither -> either

1. The typo has been fixed.

### nor -> or

1. The typo has been fixed.

## Page 20:

### hour -> hours

1. The typo has been fixed.

## Page 21:

### ChecksumException Re-running -> ChecksumException. Re-running

1. A point has been added.

### the attached -> of the attached

1. The typo has been fixed.

### While, -> While

1. The typo has been fixed.

### were -> was

1. The typo has been fixed.

## Page 22:

### of few -> of a few

1. The typo has been fixed.

### Application -> Applications

1. The typo has been fixed.

### We believe that JCHARMING could have successfully reproduced the crash: couldn't you test this?

## Page 23:

### the SUTs analyzed by JCHARMING are the same as the ones used in similar studies: how do the results of JCHARMING actually compare to those obtained in similar studies?

### footnote "described here ... ": make this note a full sentence

1. The footnote now only contains the link.

## Page 24:

### all based on -> all written in

1. The typo has been fixed.

### bug -> bugs

1. The typo has been fixed.

### involves -> involve

1. The typo has been fixed.

### Katoen Jp -> Katoen, J-P.

## Page 26:

### 41. Hadoop A. Hadoop 2011.

### 42. Mahout A. Scalable machine learning and data mining 2012.

### 43. Snyder B, Bosanac D, Davies R. ActiveMQ in Action 2011; :408.: these are not very helpful references. Please add more information