Integrating Debugging with Continuous Testing

Malcolm Stone
University of Glasgow
2250593s@student.gla.ac.uk

Roly Perera
University of Glasgow
roly.perera@glasgow.ac.uk

ACM Reference Format:
Malcolm Stone and Roly Perera. 2017. Integrating Debugging with Continuous Testing. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

Live programming for verification. In prior work [4–6] we explored a form of continuous automated testing [3, 7] for a simple concurrent object-oriented language. The approach, which we called *live programming for verification*, explores the state space of the program in the background as the user writes code, automatically detecting state-related communication errors. Bad runtime configurations are reported to the user as errors in their source code.

```
obj seller
buyer1▸price-request(title)
buyer1◂price-response(15)
buyer1▸{
   agree
      buyer2▸payment(balance)•
   quit•
}

obj buyer1
main▸buy(title)
   seller◂price-request(title)
   seller▸price-response(price)
   math◂div(price, 2)
   math▸val(offer)
   buyer2◂quote(offer)
   buyer2▸{
      agree
         math◂minus(price, offer)
         math▸val(balance)
         seller◂agree
         seller◂payment(balance)
         math◂close•
      quit
         seller◂quit
         math◂close•
   }
```


Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

In the first figure (below left), the coloured underlining under the • (indicating a terminal state) and the identifier `seller` indicates the existence of a state where the object `buyer1` needs to send the `payment` message to `seller` but the latter has terminated and is unable to accept any messages.

The present work extends the implementation with a debugger that allows the programmer to manually explore the state space and also to jump to the runtime state associated with any reported error. In the left figure, the debugger is active with the `seller` thread blocked (indicated by the orange colour) and the `buyer1` thread able to proceed by accepting the `buy` message from `main` (highlighted in green).

The figure below illustrates the "jump to error state" feature. The programmer can click on the underlining associated with an error to step the debugger to the point in the execution where the communication error was detected. Here, the `buyer1` thread has advanced to the point where it is trying to communicate with `seller`, whose execution has now reached the terminal state •.

```
obj seller
buyer1▸price-request(title)
buyer1◂price-response(15)
buyer1▸{
   agree
      buyer2▸payment(balance)•
   quit•
}

obj buyer1
main▸buy(title)
   seller◂price-request(title)
   seller▸price-response(price)
   math◂div(price, 2)
   math▸val(offer)
   buyer2◂quote(offer)
   buyer2▸{
      agree
         math◂minus(price, offer)
         math▸val(balance)
         seller◂agree
         seller◂payment(balance)
         math◂close•
      quit
         seller◂quit
         math◂close•
   }
```

Future work. To turn "jump to error state" into a complete feature we plan to add *causal-consistent* reversibility to the debugger, inspired by work on reversible process calculi [1, 2]. This will allow the programmer to work backwards from error states to diagnose the cause of the error.

References

[1] V. Danos and J. Krivine. Reversible communicating systems. In P. Gardner and N. Yoshida, editors, *Concurrency Theory, 15th International Conference, CONCUR '04*, volume 3170 of *LNCS*, pages 292–307. Springer, 2004.

[2] I. Lanese, C. A. Mezzina, and J.-B. Stefani. Reversing higher-order pi. In *Concurrency Theory, 21st International Conference, CONCUR '10*, pages 478–493. Springer-Verlag, 2010.

[3] L. Madeyski and M. Kawalerowicz. Continuous test-driven development - A novel agile software development practice and supporting tool. In *ENASE 2013 - Proceedings of the 8th International Conference on Evaluation of Novel Approaches to Software Engineering*, pages 260–267, 2013.

[4] R. Perera and S. J. Gay. Behavioural prototypes (extended abstract). In *0th Workshop on New Object-Oriented Languages (NOOL) 2015*, 2016.

[5] R. Perera and S. J. Gay. Liveness for verification. In *Second Workshop on Live Programming Systems (LIVE '16)*, 2016.

[6] R. Perera, J. Lange, and S. J. Gay. Multiparty compatibility for concurrent objects. In D. Orchard and N. Yoshida, editors, *Proceedings of the Ninth workshop on Programming Language Approaches to Concurrency and Communication-Centric Software (PLACES '16)*, volume 211 of *Electronic Proceedings in Theoretical Computer Science*, pages 73–82. Open Publishing Association, 2016.

[7] D. Saff and M. D. Ernst. Continuous testing in Eclipse. In *2nd Eclipse Technology Exchange Workshop (eTX)*, Barcelona, Spain, 2004.