Omniscient Debugging for Cognitive Agent Programs*

Vincent J. Koeman, Koen V. Hindriks, Catholijn M. Jonker

Delft University of Technology, Mekelweg 4 2628CD Delft, The Netherlands
{v.j.koeman,k.v.hindriks,c.m.jonker}@tudelft.nl

For traditional (cyclic) debugging to work, the program under investigation has to be deterministic [3]. Otherwise, reproducing a bug by rerunning the program is likely to fail as it will not hit the same bug again or even hit different bugs [2]. Real-time programs like multi-agent systems are typically not deterministic. Running the same agent system again more often than not results in a different program run or trace, which complicates the iterative process of debugging.

Omniscient debugging is an approach to tackle these issues. Also known as reverse or back-in-time debugging, omniscient debugging is a technique that originates in the context of object-oriented programming (OOP), allowing a programmer to explore arbitrary moments in a program's run by recording the execution. Such a 'time travelling debugger' is regarded as one of the most powerful debugging tools. However, omniscient debugging is still not widely adopted. An important reason for this is that existing (OOP) implementations have a significant performance impact, with slowdown factors ranging from 2 to 300 times. The fact that *agent-oriented programming* (AOP) is based on a higher level of abstraction compared to most other programming languages provides an opportunity to apply omniscient debugging techniques with a significantly lower overhead. The premise here is that tracing for AOP can be based on capturing only high-level decision making events instead of lower-level computational events. Tracing techniques for AOP would thus need tracing of significantly fewer events while still being able to reconstruct all program states, making omniscient debugging for AOP more feasible in practice than for e.g. OOP.

The main contribution of our work is the design of a tracing mechanism for cognitive agent programs that: (i) has a small impact on runtime performance; we show that our technique only has a 10% overhead instead of the much larger factors known for OOP from the literature; (ii) has virtually no impact on program behaviour; we empirically establish that the same tests succeed and fail with or without our tracing mechanism; (iii) can be effectively used for debugging; we propose a visualization technique tailored to cognitive agents and illustrate its application for fault localization. The key question we thus address is whether it is feasible and practical to apply omniscient debugging techniques to AOP without affecting testability.

There are many ways to trace the execution of a program in AOP specifically. Capturing all events, states, and actions performed in a run, i.e., a *full-trace* as

* The full paper was published in the *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*, pages 265-272, August 2017.

is created in existing solutions for other programming languages, is not feasible in practice for AOP either as it takes 5-20x longer to execute an agent system with such a trace. We therefore started with mechanism that stores a trace that captures as little information as possible but still provides sufficient information for record-replay, i.e., reconstructing any previous state in the programs' run based on replaying the stored *events*. We then extended this minimal trace with information about *state changes* that allows much more efficient reconstruction of a previous state. Finally, we added *source code information* associated with points in a trace to enable a debugger to more effectively explore the trace.

For efficient fault localization, it also needs to be easy for a developer to identify states in a program's execution that are related to the failure under investigation. Moreover, a developer should not get lost in navigating between these states, but always have a sense what point in the execution s/he is evaluating and how the current state affected the execution. Therefore, we adapted the concept of a *space-time view* [1] to cognitive agent programming (see Fig. 1). A space-time view is a table that is structured along space and time dimensions, where the rows in the are composed of the different elements in a state that are traced and the columns entail each step in the agent's execution history.

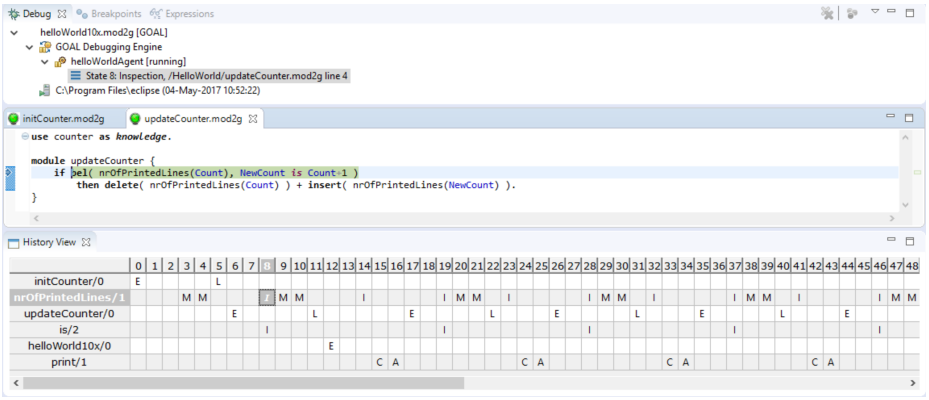

Fig. 1. An example of the space-time view of an execution history.

We believe that our tracing mechanism can provide a starting point for a *history-based explanation mechanism* that can automatically answer questions such as "why did this action (not) happen?"

References

[1] Azadmanesh, M.R., Hauswirth, M.: Space-time views for back-in-time debugging. Tech. Rep. 2015/02, University of Lugano (May 2015)
[2] Engblom, J.: A review of reverse debugging. In: System, Software, SoC and Silicon Debug Conference (S4D), 2012. pp. 1–6 (Sep 2012)
[3] Koeman, V.J., Hindriks, K.V., Jonker, C.M.: Designing a source-level debugger for cognitive agent programs. JAAMAS 31(5), 941–970 (Sep 2017)