BDI Agent Testability Revisited

(JAAMAS Extended Abstract)

Michael Winikoff
Department of Information Science, University of Otago
Dunedin, New Zealand
michael.winikoff@otago.ac.nz

ABSTRACT

This paper extends our understanding of BDI agent program testability. It considers this with respect to the all edges test adequacy criterion, comparing with previous work that considered the all paths criterion. Our findings extend the earlier analysis to give a more nuanced understanding of the difficulty of testing BDI agents. In particular, we include analysis comparing BDI programs with procedural programs that allow for an exception handling construct.

Keywords

Verification and validation; agent-based systems

1. INTRODUCTION

When any software system is deployed, it is important to have assurance that it will function as required. Traditionally, this assurance is obtained by testing. However, there is a general intuition that agents exhibit behaviour that is complex. Given this complexity, a key question is whether agent systems are harder, and possibly even infeasible, to assure by testing.

The only work that we are aware of that considers the question of testability is the recent work by Winikoff & Cranefield [7], which investigates the testability of Belief-Desire-Intenton (BDI) agent programs with respect to the all paths test adequacy criterion. They concluded that BDI agent programs do indeed give rise to a very large number of possible paths. They therefore conclude that whole BDI programs are likely to be infeasible to assure via testing. However, they do acknowledge that all edges criterion is optimistic: it is regarded as “the generally accepted minimum” [2]. The contribution of this paper is to extend the previous work to obtain a better understanding of, and a tighter bound on, the testability of BDI agent programs.

2. ALL-EDGE COVERAGE ANALYSIS

Given a program and a test adequacy criterion the testability of a program is the smallest number of test cases that would be required to satisfy the criterion. The all paths criterion is satisfied iff the set of tests in the test suite cover all paths in the program’s control flow graph. The all edges criterion (also referred to as “branch coverage”) is satisfied iff the set of paths in the test suite covers all edges in the control-flow graph [3].

We define a BDI program $P$ using the grammar:

$$P ::= a \mid g^{(P)} \mid P_1; P_2 \mid P_3 \triangleright P_2$$

where $a$ is an action, $g^{(P)}$ is a (sub-)goal with associated applicable plans $P = \{P_1, \ldots, P_n\}$, $P_1; P_2$ is a sequence, and $P_3 \triangleright P_2$ represents a “backup plan” (used to model failure handling): if $P_1$ succeeds, then nothing else is done (i.e. $P_2$ is ignored), but if $P_1$ fails, then $P_2$ is used.

One important feature of BDI programs is that the execution of a BDI program (or sub-program) can either succeed or fail. A failed execution triggers failure handling. We represent this by mapping a program $P$ to a graph that is reachable from the start node $S$, and that has two outgoing edges: to $Y$ (corresponding to a successful execution) and to $N$ (corresponding to a failed execution). Each of $Y$ and $N$ has an edge to the final node $E$.

We have derived equations (see [6]) that calculate the number of paths from $S$ to $E$ required such that all edges appear at least once in the set of paths. In order to do this, it turns out that we need to also capture how many of these paths correspond to successful executions (go via $Y$) and how many go via $N$.

Our analysis found that the number of tests required to satisfy the all edges criterion is not just lower (as expected) but very much lower (see Table 1, comparing “All Paths” and $p(g)$). Indeed, the number of tests required is sufficiently small to be feasible. However, we do need to emphasise that all edges is generally considered to be a minimal requirement, and that there are arguments for why the all paths criterion is more appropriate for history-sensitive systems, such as agent systems.

3. PROCEDURAL PROGRAMS

Following Winikoff & Cranefield [2] we define an abstract procedural program as:

$$Q ::= s \mid Q + Q \mid Q; Q \mid Q \triangleright Q$$

A single test corresponds to a path through the program’s control-flow graph from its starting node to its final node.


<table>
<thead>
<tr>
<th>Parameters</th>
<th>All Paths</th>
<th>All Edges</th>
<th>All Edges with exceptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>j k d</td>
<td>n(^q)</td>
<td>p(g)</td>
<td>q(^q)</td>
</tr>
<tr>
<td>2 2 3</td>
<td>21 42 62 (13)</td>
<td>141</td>
<td>62 1892 1 122</td>
</tr>
<tr>
<td>3 3 3</td>
<td>91 273 363 (25)</td>
<td>6391</td>
<td>363 65,704 4 1801</td>
</tr>
<tr>
<td>2 3 4</td>
<td>259 518 776 (79)</td>
<td>1,585</td>
<td>776 300,701 8 6940</td>
</tr>
<tr>
<td>3 4 3</td>
<td>157 471 627 (41)</td>
<td>10,777</td>
<td>627 196,252 6 4362</td>
</tr>
</tbody>
</table>

Table 1: Comparison of All Paths and All Edges analyses. The first number under “actions” (e.g. 62) is the number of actions in the tree, the second (e.g. 13) is the number of actions in a single execution where no failures occur.

Where the base case is a statement \( s \), and a compound program can be a combination of sub-programs in sequence \( Q_1; Q_2 \), an alternative choice \( (Q_1 + Q_2) \), or an exception handling construct, \( Q_1 \uparrow Q_2 \), where the execution of \( Q_2 \) may throw an exception, and if it does, then \( Q_2 \) is used to handle it. Mapping these programs to control-flow graphs is straightforward, and a program is mapped to a single-entry and single-exit graph.

We have derived equations (see [6]) that calculate how many paths, \( q(Q) \), are required to cover all edges in a procedural program \( Q \). We compare a BDI program \( P \) that has \( N \) actions with a procedural program \( Q \) that has \( N \) statements. If \( P \) does not contain any instances of exception handling (\( q^0 \) in Table 1) then \( q(Q) \leq N \). However, if exception handlers are present (\( q^\infty \) in Table 1) then \( q(Q) \leq 1 + \frac{1}{2}(n(Q)^2 - n(Q)) \). This is a significant change. For example, consider the first row of Table 1 without exceptions, a program with 62 statements can require at most 62 tests to cover all edges. If we allow exceptions then the number becomes \( 1 + \frac{1}{2} \times (62^2 - 62) = 1892 \), which is significantly more than the number of tests required to test the corresponding BDI program.

However, the program \( Q \) that yields this value is pathological: it consists of deeply nested exception handling of single statements! We therefore need to consider what a typical procedural program with exception handling might look like. To answer this question we turn to empirical investigations of programs [1, 4], which shows that adding “even if we allow pathological programs”,

whereas the conclusion lends strength to the earlier result of Winikoff & Cranefield. They found that BDI agent programs were harder to test than equivalently sized procedural programs (with respect to the all paths criterion). We found that this is also the case for the all edges criterion, but somewhat less so.

We also extended their all paths analysis of procedural programs by adding exceptions. We found that even for pathological procedural programs, it was still the case that BDI programs required more tests to cover all paths [3]. This strengthens somewhat the conclusion of Winikoff & Cranefield that BDI programs are harder to test than (equivalently sized) procedural programs by adding “even if we allow pathological programs”.

4. CONCLUSIONS

To summarise, we found a number of (unexpected) results:

- The number of tests required with respect to the all edges criterion was not just smaller than for all paths, but much smaller.
- Unlike the case for all paths, disabling failure handling did not significantly reduce the number of tests required (this result has not been discussed earlier in this extended abstract and is included here for completeness — see [6] for details).
- A BDI program requires more tests than an equivalently sized procedural program with respect to all edges. This conclusion still holds if we allow a realistic number of exceptions, but not for all the cases considered.
- Revisiting the comparison between BDI and procedural programs with respect to the all paths criterion, in the presence of exceptions, finds that BDI programs remain harder to test, even if we permit pathological procedural programs.

Overall the analysis in this paper indicates that Winikoff & Cranefield’s conclusions do generalise to another criterion, which lends additional strength to their conclusion that testing BDI programs is harder than testing procedural programs.

REFERENCES