On the Testability of BDI Agent Systems

Michael Winikoff
Stephen Cranefield

The Information Science Discussion Paper Series

Number 2008/03
November 2008
ISSN 1177-455X

On the testability of BDI agent systems

Michael Winikoff   Stephen Cranefield

November 2008

Abstract

Before deploying a software system we need to assure ourselves (and stake-holders) that the system will behave correctly. This assurance is usually done by testing the system. However, it is intuitively obvious that adaptive systems, including agent-based systems, can exhibit complex behaviour, and are thus harder to test. In this paper we examine this "obvious intuition" in the case of Belief-Desire-Intention (BDI) agents. We analyse the size of the behaviour space of BDI agents and show that although the intuition is correct, the factors that influence the size are not what we expected them to be; specifically, we found that the introduction of failure handling had a much larger effect on the size of the behaviour space than we expected. We also discuss the implications of these findings on the testability of BDI agents.

Keywords: Testing, Complexity, Validation, Belief-Desire-Intention (BDI)

1 Introduction

Increasingly we are called upon to develop software systems that operate in dynamic environments, that are robust in the face of failure, that are required to exhibit flexible behaviour, and that operate in open environments. One approach for developing such systems that has demonstrated its effectiveness in a range of domains is the use of the metaphor of *software agents* to conceptualise, design and build software systems [1]. Agent systems have been increasingly finding deployment in a wide range of applications (e.g. [2, 3]).

As agent-based systems are increasingly deployed, the issue of *assurance* rears its head. Before deploying a system, we need to convince those who will rely on the system (or those who will be responsible if it fails) that the system will, in fact, work. Traditionally, this assurance is done through testing. However, it is generally accepted that adaptive systems exhibit a wider and more complex range of behaviours, making testing harder. For example:

> "*...validation through extensive tests was mandatory ...However, the task proved challenging for several reasons. First, agent-based systems explore realms of behaviour outside people's expectations and often yield surprises ...*" [2, Section 3.7.2].

That is, there is an intuition that agent systems exhibit complex behaviour, which makes them harder to test. In this paper we explore this intuition, focusing on the well-known Belief-Desire-Intention (BDI) [4, 5] approach to realising adaptive and flexible agents, which has been demonstrated to be practically applicable, resulting in reduced development cost and increased flexibility [3].

We explore the intuition that "agent systems are harder to test" by analysing both the space of possible behaviours of BDI agents, and the probability of failure. We focus on BDI agents both because they provide a well-defined execution mechanism that can be analysed, but also because we seek to understand the complexities (and testability implications) of adaptive and intelligent behaviour in the absence of parallelism (since the implications of parallelism are already well-known).

Specifically, we aim to answer these questions:

1. How large is the behaviour space for BDI agents?

2. What factors influence the size of the behaviour space?

3. Is it feasible to assure the effectiveness of BDI systems by testing?

As might be expected, we show that the intuition is correct. The contribution of this paper is thus to confirm the intuition by *quantifying* the size of the behaviour space. Additionally, we find some surprising results about what factors influence the size of the behaviour space.

Although there has recently been increasing interest in testing agent systems (see for example [6, 7, 8, 9]), there has been surprisingly little work on determining the feasibility of testing agent systems in the first place. Padgham and Winikoff [10, Pages 17-19] analyse the number of successful executions of a BDI agent's goal-plan tree, but they do not consider failure or failure handling in their analysis, nor do they consider testability implications. Shaw and Bordini [11] have analysed goal-plan trees and shown that checking whether a goal-plan tree has an execution schedule with respect to resource requirements is NP-complete. This is a different problem to the one that we tackle: they are concerned with the allocation of resources between goals, rather than with the behaviour space.

The remainder of the paper is structured as follows. We begin by briefly presenting the BDI execution model (section 2). Section 3 is the core of the paper where we analyse the behaviour space of BDI agents. We then consider how our analysis and its assumptions holds up against a real system (section 4) before discussing the implications for testing and concluding (section 5).

2 The BDI Execution Model

Although the BDI model may be viewed from philosophical [5] and logical [4] perspectives, we are interested here in the *implementation* perspective, as exhibited in a range of architectures and platforms (such as JACK [12], JAM [13], dMARS [14], PRS [15, 16], Jason [17], etc.). For the purposes of our analysis here, a formal and detailed presentation is unnecessary. Those interested in formal semantics for BDI languages are referred to (e.g.) [18, 19, 17].

In the implementation of a BDI agent the key concepts are *beliefs* (or, more generally, data), *events* and *plans*. The reader may find it surprising that goals are not key concepts in BDI systems. The reason is that goals are modelled as events: the acquisition of a new goal is viewed as a "new-goal" event, and the agent responds by selecting and executing a plan that can handle that event[1]. In the remainder of this section, in keeping with established practice, we will describe BDI plans as handling events (not goals).

A BDI plan consists of three parts: an *event pattern* specifying the event(s) it is relevant for, a *context condition* (a Boolean condition) that indicates in what situations the plan can be used, and a *plan body* that is executed. A plan's event pattern and context condition may be terms containing variables, so a matching or unification process (depending on the BDI system used) is used by BDI interpreters to find plans that respond to a given event. In general the plan body can contain arbitrary code in some programming language[2], however for our purposes (following abstract notations such as AgentSpeak(L) [18] and CAN [19]) we assume that a plan body is a sequence of steps, where each step is either an action[3] or posting an event. Note that actions can succeed or fail.

PlanA: **handles event**: *achieve goal* go-home
context condition: train imminent
plan body:
(1) walk to train station
(2) check train running on time
(3) catch train
(4) walk home

PlanB: **handles event**: *achieve goal* go-home
context condition: not raining and have bicycle
plan body:
(1) cycle home

PlanC: **handles event**: *achieve goal* go-home
context condition: true (i.e. always applicable)
plan body:
(1) walk to bus stop
(2) check buses running
(3) catch bus
(4) walk home

Figure 1: Three Simple Plans

[1]Other types of event typically include the addition and removal of beliefs from the agent's belief set.

[2]E.g. for JACK a plan body is written in a language that is a superset of Java.

[3]This includes both traditional actions that affect the agent's environment, and internal actions that invoke code, or that check whether a certain condition follows from the agent's beliefs.

For example, consider the simple plans given in Figure 1. For the first plan, PlanA, the plan is relevant for handling the event "achieve goal go-home", and it is applicable in situations where the agent believes that a train is imminent. The plan body consists of a sequence of four steps (in this case we assume that these are actions, but they could also be modelled as events that are handled by further plans). Figure 2 shows the first plan, as it might be written in the JACK agent language.

```
plan PlanA extends Plan
{
  #handles event GoHome go_home;

  context() {
    nextTrain()-now() < ACCEPTABLE_WAIT; // Train imminent
  }

  body()
  {
    walkTo(trainStation.location());
    checkTrainOnTime(); // If returns false, plan fails
    catchTrain();
    walkTo(home.location());
  }
}
```

Figure 2: Simple Plan in JACK

A key feature of this approach is that each plan encapsulates the conditions under which it is applicable (by defining an event pattern and context condition) [20]. This allows for additional plans for a given event to be added in a modular fashion, since the invoking context (i.e. where the triggering event is posted) does not contain code that selects amongst the available plans, and is a key reason for the flexibility of BDI programming.

A typical BDI execution cycle is an elaboration of the following event-driven process (summarised in Figure 3)[4]:

1. An event occurs (either received from an outside source, or triggered from within the agent)

2. The agent determines the set of plans in its plan library that are declared to handle an event that matches the triggering event. This is the set of *relevant* plan instances.

3. The agent determines a subset of the relevant plans that are *applicable* in the current situation. A plan (instance) is applicable if its context condition is true. If there are no

[4]BDI engines are, in fact, more complicated than this as they can interleave the execution of multiple active plans (or *intentions*) that were triggered by different events.

```
**Boolean function** execute(event)
let relevant-plans = set of plan instances resulting from
    matching all plans' event patterns to event
let tried-plans = $\varnothing$
**while** true **do**
    let applicable-plans = set of plan instances resulting from
        solving the context conditions of relevant-plans
    applicable-plans := applicable-plans $\setminus$ tried-plans
    **if** applicable-plans is empty **then return** false
    select plan p $\in$ applicable-plans
    tried-plans := tried-plans $\cup \{p\}$
    **if** execute(p.body)=true **then return** true
**endwhile**

**Boolean function** execute(plan-body)
**if** plan-body is empty **then return** true
**elseif** execute(first(plan-body)) = false **then return** false
**else** execute(rest(plan-body))
**endif**

**Boolean function** execute(action)
attempt to perform the action
**if** action executed successfully **then return** true **else return** false **endif**
```

Figure 3: BDI Execution Cycle (pseudo code)

applicable plans then the event is deemed to have failed, and if it has been posted from a plan, then that plan fails. Note that a single relevant plan may lead to no applicable plan instances (if the context condition is false), or to more than one applicable plan instance (if the context condition has multiple solutions).

4. One of the applicable plan instances is selected and is executed. The plan's body may create additional events that are handled using this process.

5. If the plan body fails, then failure handling is triggered.

Regarding the final step, there are a few approaches to dealing with failure. Perhaps the most common approach is to select an alternative applicable plan, and only consider an event to have failed when there are no remaining applicable plans. In determining alternative applicable plans one may either consider the existing set of applicable plans, or re-calculate the set of applicable plan instances (ignoring those that have already been tried), as is done in Figure 3. This makes sense because the situation may have changed since the applicable plans were determined.

An alternative failure handling approach, used by Jason [17], is to post a failure event which can be handled by a user-provided plan. Although this is more flexible, since the user can specify what to do upon failure, it does place the burden of specifying failure handling on the user. Note that Jason does provide a "pattern" that allows the traditional BDI failure handling mechanism to be specified [17, pages 171–172].

Given the execution cycle, the three example plans given earlier can give rise to a range of behaviours, including the following:

- Suppose the event "achieve goal go-home" is posted and the agent believes that a train is imminent. It walks to the train station, finds out that the train is running on time, catches the train, and then walks home.
- Suppose that upon arrival at the train station the agent finds out that trains are delayed. Step (2) of PlanA fails, and the agent considers alternative plans. If it is raining at the present time, then PlanB is not applicable, and so PlanC is adopted (to catch the bus).
- Suppose that the agent has decided to catch the bus (because no train is believed to be imminent, and it is raining), and that attempting to execute PlanC fails (e.g. there is a bus strike). The agent will reconsider its plans and if the rain has stopped (and it has a bicycle) it may then use PlanB.

The events and plans can be visualised as a tree (sometimes called a "goal-plan tree") where each event has as children the plan instances that are applicable to it, and each plan instance has as children the events that it posts. The goal-plan tree is an "and-or" tree: each goal is realised by one of its plan instances ("or") and each plan instance needs all of its sub-goals to be achieved ("and"). In order to be consistent with existing practice we shall use the term "goal" rather than "event" in the remainder of this paper.

3 Behaviour Space Size of BDI Agents

We now consider how many possible behaviours there are for a BDI agent that is trying to realise a goal[5] with a given goal-plan tree. Executing a goal-plan tree is highly non-deterministic: we need to choose a plan for each goal in the tree, when we consider failure we need to consider for each action and for each goal whether it fails or not, and if it does, what failure recovery is done, and finally, if there is any parallelism, then we need to select a particular interleaving of the parallel threads.

In this section we seek to answer the question of how large is the behaviour space for BDI agents? We do this by deriving formulae that compute the number of behaviours (both successful, and unsuccessful, i.e. failed) for a given goal-plan tree.

We make the following assumptions which allow us to perform the analysis.

[5]We focus on a single goal in our analysis: multiple goals can be treated as the concurrent interleaving of the individual goals.

- We assume that the tree is uniform in that for a goal at depth d, all of its children are at depth $d-1$ (and similarly for plan instances); and we assume that all plan instances have k (or 0) sub-goals (see Figure 4).

- We do not model the instantiation of plan types to plan instances by matching their event patterns to events. Instead we assume that each goal (uniformly) has j applicable plan instances. This can be the case if each goal has j relevant plans, each of which results in exactly one applicable plan instance, but can also be the case in other ways.

We re-examine these assumptions in section 4 where we consider a (non-uniform) goal-plan tree from an industrial application.

Abstractly we can consider the process of executing a goal-plan tree as taking a goal-plan tree, and, by following the BDI execution cycle, progressively making decisions about which plans to use for each goal and executing these plans. This process can be seen as taking a goal-plan tree and yielding a sequence of actions that were executed (see the appendix).

Our analysis uses the following terminology:

- Each goal g has j applicable plan instances $p1 \ldots pj$. Our uniformity assumption makes these plan instances indistinguishable for the purpose of our analysis, so we can omit the indices.

- Each non-leaf plan instance p has k sub-goals $g1 \ldots gk$ (for the moment we ignore actions). Due to our uniformity assumption we can also treat these sub-goals as identical and omit the indices.

- We consider a goal-plan tree to have depth d if there are d goal "layers" (see Figure 4), that is, a goal-plan tree of depth 1 has either a single goal which has j plans as its children (and the plans have no sub-goals), or a single plan with k sub-goals, each of which has j plans. A tree of depth 0 is a plan with no sub-goals. A goal/plan of depth d, where $d > 0$, is written (resp.) as g_d or p_d.

- We use $n^{\checkmark}(x_d)$ to denote the number of *successful* execution paths of a goal-plan tree of depth d rooted at x (where x is either a goal g or a plan p). Where specifying d is not important we will sometimes elide it, writing $n^{\checkmark}(x)$.

- Similarly, we use $n^{\times}(x_d)$ to denote the number of *unsuccessful* execution paths of a goal-plan tree of depth d with root x (either g or p).

- We extend this notation to *plan body segments*, i.e. sequences $x_1;\ldots;x_n$ where each x_i is a goal or action and ';' denotes sequential composition. We abbreviate a sequence of n occurrences of x by x^n.

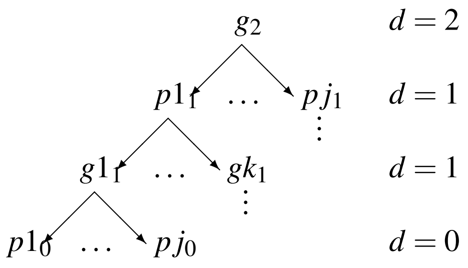


Figure 4: Goal-plan tree

3.1 Base Case: Successful Executions

We begin by calculating the number of *successful* paths through a goal-plan tree in the absence of failure (and of failure handling). This analysis follows that of [10, pages 17–19].

Roughly speaking, for a goal, the number of ways it can be achieved is the *sum* of the number of ways in which its children can be achieved (since the children represent alternatives, i.e. "or"). On the other hand, for a plan, the number of ways it can be achieved is the *product* of the number of ways in which its children can be achieved (since the children must all be achieved, i.e. "and"). More precisely, $n^{\checkmark}(x_1;x_2) = n^{\checkmark}(x_1)\,n^{\checkmark}(x_2)$, that is, the sequence is successful if both x_1 and x_2 are successful.

Given a tree with root g (a goal), assume that each of its j children can be achieved in n different ways[6], then, because we select one of the children[7], the number of ways in which g can be achieved is jn. Similarly, for a tree with root p (a plan), assume that each of its k children can be achieved in n different ways, then, because we execute all of its children, the number of ways in which p can be executed is $n \cdots n$, or n^k. A plan with no children (i.e. at depth $d = 0$) can be executed (successfully) in exactly one way. This yields the following definition:

$$\begin{aligned}
n^{\checkmark}(g_d) &= j\,n^{\checkmark}(p_{d-1}) \\
n^{\checkmark}(p_0) &= 1 \\
n^{\checkmark}(p_d) &= n^{\checkmark}({g_d}^k) = n^{\checkmark}(g_d)^k
\end{aligned}$$

Expanding this definition we obtain

$$\begin{aligned}
n^{\checkmark}(g_1) &= j\,n^{\checkmark}(p_0) = j\,1 = j \\
n^{\checkmark}(g_2) &= j\,n^{\checkmark}(p_1) = j\,(n^{\checkmark}(g_1)^k) = j\,(j^k) = j^{k+1} \\
n^{\checkmark}(g_3) &= j\,n^{\checkmark}(p_2) = j\,(j^{k+1})^k = j^{k^2+k+1} \\
n^{\checkmark}(g_4) &= j\,n^{\checkmark}(p_3) = j\,(j^{k^2+k+1})^k = j^{k^3+k^2+k+1}
\end{aligned}$$

which can be generalised to:

$$n^{\checkmark}(g_d) = j^{\sum_{i=0}^{d-1} k^i}$$

[6]Because the tree is assumed to be uniform, all of the children can be achieved in the same number of ways, and are thus interchangeable in the analysis, allowing us to write $j\,n$ rather than $n_1 + \ldots + n_j$.

[7]This is done in any order: in general there is no assumption that plans are selected in a given order.

If $k > 1$ this can be simplified using the equivalence $k^{i-1} + \ldots + k^2 + k + 1 = (k^i - 1)/(k - 1)$ to give the following closed form definition:

$$n^{\checkmark}(g_d) = j^{(k^d-1)/(k-1)} \quad (1)$$

$$n^{\checkmark}(p_d) = j^{k\,(k^d-1)/(k-1)} \quad (2)$$

If $k = 1$ we have $n^{\checkmark}(g_d) = n^{\checkmark}(p_d) = j^d$.

Note that the equation for $n^{\checkmark}(p_d)$ assumes that sub-goals are achieved sequentially. If they are executed in parallel then the number of options is higher, since we need to consider all possible interleavings of the sub-goals' execution. For example, suppose that a plan p_d has two sub-goals, $g1_d$ and $g2_d$, where each of the sub-goals has $n^{\checkmark}(g_d)$ successful executions, and each execution has l steps (we assume for ease of analysis that both execution paths have the same length). The number of ways of interleaving two parallel executions, each of length l is (see, e.g., [21, Section 3]):

$$\binom{2l}{l} = \frac{(2l)!}{(l!)\,(l!)}$$

and hence the number of ways of executing p_d with parallel execution of subgoals is:

$$n^{\checkmark}(p_d) = n^{\checkmark}(g_d)^2 \binom{2l}{l} = n^{\checkmark}(g_d)^2 \frac{(2l)!}{(l!)\,(l!)}$$

In the remainder of this paper we assume that the sub-goals of a plan are achieved sequentially, since this is the common case, and since it yields a lower figure which, as we shall see, is still large enough to allow for conclusions to be drawn.

3.2 Adding Failure

We now extend the analysis to include failure, and determine the number of *unsuccessful* executions, i.e. executions that result in failure of the top-level goal. For the moment we assume that there is no failure handing (we add failure handling in section 3.3).

In order to determine the number of failed executions we have to know where failure can occur. In BDI systems there are two places where failure occurs: when a goal has no applicable plan instances, and when an action (within an applicable plan instance) fails. However, our uniformity assumption means that we do not address the former case—it is assumed that a goal will always have j instances of applicable plans. Note that this is a *conservative* assumption: relaxing it results in the number of unsuccessful executions being even larger.

In order to model the latter case we need to extend our model of plans to encompass actions. For example, suppose that a plan has a body of the form $a1; ga; a2; gb; a3$ where ai are actions, ga and gb are sub-goals, and ";" denotes sequential execution. Then the plan has the following five cases of *unsuccessful* (i.e. failed) executions:

1. $a1$ fails
2. $a1$ succeeds, but then ga fails

3. $a1$ and ga succeed, but $a2$ fails

4. $a1$, ga, and $a2$ succeed, but then gb fails

5. $a1$, ga, $a2$ and gb succeed, but $a3$ fails

Suppose that ga can be executed *successfully* in $n^{\checkmark}(ga)$ different ways, then the third case corresponds to $n^{\checkmark}(ga)$ different failed executions: for each successful execution of ga, extend it by adding a failed execution of $a2$ (actions can only be executed in one way, i.e. $n^{\checkmark}(a) = 1$ and $n^{\times}(a) = 1$). Similarly, if gb has $n^{\checkmark}(gb)$ successful executions then the fifth case corresponds to $n^{\checkmark}(ga)\, n^{\checkmark}(gb)$ different failed executions. If ga can be *unsuccessfully* executed in $n^{\times}(ga)$ different ways then the second case corresponds to $n^{\times}(ga)$ different executions. Similarly, the fourth case corresponds to $n^{\checkmark}(ga)\, n^{\times}(gb)$ different executions. Putting this together, we have that the total number of *unsuccessful* executions for a plan p with body $a1;ga;a2;gb;a3$ is the sum of the above five cases:

$$1 \; + \; n^{\times}(ga) \; + \; n^{\checkmark}(ga) \; + \; n^{\checkmark}(ga)\, n^{\times}(gb) \; + \; n^{\checkmark}(ga)\, n^{\checkmark}(gb)$$

More formally, $n^{\times}(x_1;x_2) = n^{\times}(x_1) + n^{\checkmark}(x_1)\, n^{\times}(x_2)$, that is, the sequence can fail if either x_1 fails, or if x_1 succeeds but x_2 fails. It follows that $n^{\checkmark}(x^k) = n^{\checkmark}(x)^k$ and $n^{\times}(x^k) = n^{\times}(x)\,(1 + \cdots + n^{\checkmark}(x)^{k-1})$, which can easily be proven by induction.

More generally, we assume there are ℓ actions before, after, and between the sub-goals in a plan, i.e. the above example plan corresponds to $\ell = 1$, and the following plan body corresponds to $\ell = 2$: $a1;a2;g3;a4;a5;g6;a7;a8$. A plan with no sub-goals (i.e. at depth 0) is considered to consist of ℓ actions (which is quite conservative: in particular, when we use $\ell = 1$ we assume that plans at depth 0 consist of only a single action).

The number of *unsuccessful* execution traces of a goal-plan tree can then be defined, based on the analysis above, as follows. First we calculate the numbers of successes and failures of the following repeated section of a plan body: $g_d;a^{\ell}$:

$$\begin{array}{rcl}
n^{\checkmark}(g_d;a^{\ell}) & = & n^{\checkmark}(g_d)\, n^{\checkmark}(a^{\ell}) \\
 & = & n^{\checkmark}(g_d)\, n^{\checkmark}(a)^{\ell} \\
 & = & n^{\checkmark}(g_d)\, 1^{\ell} \\
 & = & n^{\checkmark}(g_d) \\
n^{\times}(g_d;a^{\ell}) & = & n^{\times}(g_d) + n^{\checkmark}(g_d)\, n^{\times}(a^{\ell}) \\
 & = & n^{\times}(g_d) + n^{\checkmark}(g_d)\, n^{\times}(a)\,(1 + \cdots + n^{\checkmark}(a)^{\ell-1}) \\
 & = & n^{\times}(g_d) + n^{\checkmark}(g_d)\, \ell
\end{array}$$

We then have for $d > 0$:

$$\begin{aligned}
n^{\times}(p_d) &= n^{\times}(a^{\ell};(g_d;a^{\ell})^k) \\
&= n^{\times}(a^{\ell}) + n^{\checkmark}(a^{\ell})\, n^{\times}((g_d;a^{\ell})^k) \\
&= n^{\times}(a)\,(1+\cdots+n^{\checkmark}(a)^{\ell-1})) + n^{\checkmark}(a)^{\ell}\, n^{\times}((g_d;a^{\ell})^k) \\
&= \ell + 1\, n^{\times}(g_d;a^{\ell})\,(1+\cdots+n^{\checkmark}(g_d;a^{\ell})^{k-1}) \\
&= \ell + (n^{\times}(g_d) + n^{\checkmark}(g_d)\,\ell)\,(1+\cdots+n^{\checkmark}(g_d)^{k-1})) \\
&= \ell + \left(n^{\times}(g_d) + \ell n^{\checkmark}(g_d)\right) \frac{n^{\checkmark}(g_d)^k - 1}{n^{\checkmark}(g_d) - 1} \quad \text{(assuming } n^{\checkmark}(g_d) > 1)
\end{aligned}$$

This yields the following definitions for the number of *unsuccessful* executions of a goal-plan tree, *without* failure handling. The equation for $n^{\times}(g_d)$ is derived using the same reasoning as in the previous section: a single plan is selected and executed, and there are j plans.

$$\begin{aligned}
n^{\times}(g_d) &= j\, n^{\times}(p_{d-1}) \\
n^{\times}(p_0) &= \ell \\
n^{\times}(p_d) &= \ell + \left(n^{\times}(g_d) + \ell n^{\checkmark}(g_d)\right) \frac{n^{\checkmark}(g_d)^k - 1}{n^{\checkmark}(g_d) - 1} \\
& \qquad \text{(for } d > 0 \text{ and } n^{\checkmark}(g_d) > 1)
\end{aligned}$$

Finally, we note that the analysis of the number of successful executions of a goal-plan tree in the absence of failure handling presented in Section 3.1 is unaffected by the addition of actions to plan bodies. This is because there is only one way for a sequence of actions to succeed, so Equations 1 and 2 remain correct.

3.3 Adding Failure Handling

We now consider how the introduction of a failure handling mechanism affects the analysis. A common means of dealing with failure in BDI systems is to respond to the failure of a plan by trying an alternative applicable plan for the event that triggered that plan. For example, suppose that a goal g (e.g. "achieve goal go-home") has three applicable plans pa, pb and pc; that pa is selected, and that it fails. Then the failure handling mechanism will respond by selecting pb or pc and executing it. Assume that pc is selected, then if pc fails, the last remaining plan (pb) is used, and if it too fails, then the goal is deemed to have failed.

The result of this is that, as we might hope, it is *harder* to fail: the only way a goal execution can fail is if *all* of the applicable plans are tried and *each* of them fails[8].

The number of executions can then be computed as follows: if the goal has j applicable plan instances $p1 \ldots pj$ and each of the plans has $n_i = n^{\times}(pi)$ unsuccessful executions, then we have

[8]In fact, this is actually an under-estimate: it is also possible for the goal to fail because none of the untried relevant plans are applicable in the resulting situation. As noted earlier, we assume in our analysis that goals cannot fail due to there not being applicable plan instances. This is a conservative assumption: relaxing it results in the number of behaviours being even larger.

$n_1 \cdots n_j$ unsuccessful executions of all of the plans. Since the plans can be selected in any order we multiply this by $j!$ yielding $n^{\times}(g_d) = j!\,n^{\times}(p_{d-1})^j$.

The number of ways in which a plan can fail is still defined by the same equation — because failure handling happens at the level of goals — but where $n^{\times}(g)$ refers to the new definition:

$$\begin{aligned}
n^{\times}(g_d) &= j!\,n^{\times}(p_{d-1})^j && (3)\\
n^{\times}(p_0) &= \ell && (4)\\
n^{\times}(p_d) &= \ell + \left(n^{\times}(g_d) + \ell\, n^{\checkmark}(g_d)\right) \frac{n^{\checkmark}(g_d)^k - 1}{n^{\checkmark}(g_d) - 1} && (5)\\
&\quad \text{(for } d > 0 \text{ and } n^{\checkmark}(g_d) > 1)
\end{aligned}$$

Turning now to the number of *successful* executions (i.e. $n^{\checkmark}(x)$) we observe that the effect of adding failure handling is to *convert failures to successes*, i.e. an execution that would otherwise be unsuccessful is extended into a longer execution that may succeed.

Consider a simple case: a depth 1 tree consisting of a goal g (e.g. “achieve goal go-home”) with three children: pa, pb, pc. Previously the successful executions corresponded to each of the pi (i.e. select a pi and execute it). However, with failure handling, we now have the following additional successful executions (as well as additional cases corresponding to different orderings of the plans, e.g. pb failing and then pa being successfully executed):

- pa fails, pb is then executed and it succeeds
- pa fails, pb is then executed and it fails, pc is then executed and it succeeds

This leads to a definition of the form

$$n^{\checkmark}(g) = n^{\checkmark}(pa) + n^{\times}(pa)\,n^{\checkmark}(pb) + n^{\times}(pa)\,n^{\times}(pb)\,n^{\checkmark}(pc)$$

We need to account for different orderings of the plans. For instance, the case where the first selected plan succeeds (corresponding to the first term, $n^{\checkmark}(pa)$) in fact applies for each of the j plans, so the first term, including different orderings, is $j\,n^{\checkmark}(p)$.

Similarly, the second term ($n^{\times}(pa)\,n^{\checkmark}(pb)$), corresponding to the case where the initially selected plan fails but the next plan selected succeeds, in fact applies for j initial plans, and then for $j-1$ next plans, yielding $j\,(j-1)\,n^{\times}(p)\,n^{\checkmark}(p)$.

Continuing this process (for $j = 3$) yields the following formulae

$$n^{\checkmark}(g) = 3\,n^{\checkmark}(p) + 3\,2\,n^{\times}(p)\,n^{\checkmark}(p) + 3!\,n^{\times}(p)^2\,n^{\checkmark}(p)$$

which generalises to

$$n^{\checkmark}(g) = j\,n^{\checkmark}(p) + j\,(j-1)\,n^{\times}(p)\,n^{\checkmark}(p) + \ldots + j!\,n^{\times}(p)^{j-1}\,n^{\checkmark}(p)$$

Parameters			Number of		No failure handling (secs 3.1 and 3.2)		With failure handling (section 3.3)	
j	k	d	goals	actions	$n^{\checkmark}(g)$	$n^{\times}(g)$	$n^{\checkmark}(g)$	$n^{\times}(g)$
2	2	3	21	62 (13)	128	614	$\approx 6.33 \times 10^{12}$	$\approx 1.82 \times 10^{13}$
3	3	3	91	363 (25)	1,594,323	6,337,425	$\approx 1.02 \times 10^{107}$	$\approx 2.56 \times 10^{107}$
2	3	4	259	776 (79)	1,099,511,627,776	6,523,509,472,174	$\approx 1.82 \times 10^{157}$	$\approx 7.23 \times 10^{157}$
3	4	3	157	627 (41)	10,460,353,203	41,754,963,603	$\approx 3.13 \times 10^{184}$	$\approx 7.82 \times 10^{184}$

Table 1: Illustrative values for $n^{\checkmark}(g)$ and $n^{\times}(g)$

resulting in the following equations (again, since failure handling is done at the goal level, the equation for plans is the same as in section 3.1):

$$\begin{aligned} n^{\checkmark}(g_d) &= \sum_{i=1}^{j} n^{\checkmark}(p_{d-1})\, n^{\times}(p_{d-1})^{i-1} \frac{j!}{(j-i)!} && (6)\\ n^{\checkmark}(p_0) &= 1 && (7)\\ n^{\checkmark}(p_d) &= n^{\checkmark}(g_d)^k \ \text{ (for } d > 0 \text{)} && (8) \end{aligned}$$

We have used the "standard" BDI failure handling mechanism of trying alternative applicable plans. Now let us briefly consider an alternative failure handling mechanism that simply re-posts the event, without tracking which plans have already been attempted. It is fairly easy to see that this, in fact, creates an *infinite* number of behaviours: suppose that a goal g can be achieved by pa or pb, then pa could be selected, executed resulting in failure, and then pa could be selected again, fail again, etc. This suggests that the "standard" BDI failure handling mechanism is, in fact, more appropriate, in that it avoids an infinite behaviour space, and the possibility of an infinite loop.

Table 1 makes the various equations developed so far concrete by showing illustrative values for $n^{\times}$ and $n^{\checkmark}$ for a range of reasonable (and fairly low) values for j, k and d and using $\ell = 1$. The "Number of" columns show the number of goals in the tree, and the number of actions in the tree. The number of actions in brackets is how many actions are executed in a single (successful) execution with no failure handling.

3.4 Recurrence Relations

The equations in the previous sections define the functions $n^{\checkmark}$ and $n^{\times}$ as a mutual recurrence on the depth d of the goal-plan tree. The effect of increasing the parameters k and ℓ is evident at each level of the recursion, but it is not so clear what the effect is of increasing the number of applicable plan instances for a goal. To better understand this, in this section we derive a set of recurrence relations for $n^{\checkmark}$ and $n^{\times}$ in the presence of failure handling as a recurrence on the depth d and the number of applicable plan instances n for a given goal node in the goal-plan tree, treating j (for other goal nodes), k and ℓ as constant.

We begin by defining the generalised notation $n^{✗}(g_d,n)$ and $n^{✓}(g_d,n)$ where n is the number of plans that are applicable for the goal g_d. In the previous discussion we have assumed that this value is equal to the constant j for all goal nodes in the goal-plan tree (i.e. $n^{✗}(g_d) = n^{✗}(g_d, j)$ and $n^{✓}(g_d) = n^{✓}(g_d, j)$). Now we isolate the effects of varying the number of applicable plans for a given goal node in the tree, while retaining the regularity assumption for all other goal nodes. We rewrite Equations 3 and 6 using this notation and expressing the right hand sides as functions of $n^{✗}(p_{d-1})$ and (for the case of Equation 6) $n^{✓}(p_{d-1})$:

$$\begin{aligned} n^{✗}(g_d,n) &= f^{✗}(n, n^{✗}(p_{d-1})) \\ n^{✓}(g_d,n) &= f^{✓}(n, n^{✗}(p_{d-1}), n^{✓}(p_{d-1})) \end{aligned}$$

where

$$\begin{aligned} f^{✗}(n,a) &= n!\,a^n \\ f^{✓}(n,a,b) &= \sum_{i=1}^{n} b\,a^{i-1}\,\frac{n!}{(n-i)!} = \sum_{i=0}^{n-1} \binom{n}{i}\, i!\,a^i\,(n-i)\,b \end{aligned} \tag{9}$$

The equality on the right of the last line above will be useful in the discussion below, and corresponds to the following combinatorial analysis of $f^{✓}$. For a goal g_d, each successful execution will involve a sequence of i plan executions that fail (for some i, $0 \leq i \leq n-1$) followed by one plan execution that succeeds. There are $\binom{n}{i}$ ways of choosing the failed plans, which can be ordered in $i!$ ways, and each plan has $a = n^{✗}(p_{d-1})$ ways to fail. There are then $n-i$ ways of choosing the final successful plan, which has $b = n^{✓}(p_{d-1})$ ways to succeed.

Our goal is now to find an explicit characterisation of the incremental effect of adding an extra plan on $n^{✗}(g_d,n)$ and $n^{✓}(g_d,n)$ by finding definitions of $f^{✗}$ and $f^{✓}$ as recurrence relations in terms of the parameter n. Deriving the recurrence relation for $f^{✗}$ is straightforward:

$$\begin{aligned} f^{✗}(n,a) &= n!\,a^n \\ &= (n\,(n-1)\ldots 2\,1)\;\;(\underbrace{a\,a\ldots a\,a}_{n \text{ times}}) \\ &= (n\,a)\,((n-1)\,a)\,\ldots\,(2\,a)\,(1\,a) \end{aligned}$$

This shows that $f^{✗}(0,a) = 1$ and $f^{✗}(n+1,a) = (n+1)\,a\,f^{✗}(n,a)$

However, the derivation of a recurrence relation for $f^{✓}$ is not as simple. Here we use the technique of first finding an *exponential generating function* (e.g.f.) [22] for the sequence $\{f^{✓}(n,a,b)\}_{n=0}^{\infty}$, and then using that to derive a recurrence relation.

The e.g.f. $F(x)$ of the sequence $\{f^{✓}(n,a,b)\}_{n=0}^{\infty}$ is the function defined by the following power series:

$$\begin{aligned} F(x) &= \sum_{n=0}^{\infty} f^{✓}(n,a,b)\frac{x^n}{n!} \qquad (10) \\ &= \sum_{n=0}^{\infty}\left(\sum_{i=0}^{n-1}\binom{n}{i} i!a^i(n-i)b\right)\frac{x^n}{n!} = \sum_{n=0}^{\infty}\left(\sum_{i=0}^{\infty}\binom{n}{i} i!a^i(n-i)b\right)\frac{x^n}{n!} \end{aligned}$$

On the right hand side above we have changed the upper limit of the inner sum to ∞ based on the generalised definition of $\binom{n}{i}$ as $n(n-1)(n-2)...(n-i+1)/i!$, which is valid for all complex numbers n and non-zero integers i [22] and gives $\binom{n}{i} = 0$ for $i > n$.

The right hand side has the form of a product of exponential generating functions [22, Rule 3', Section 2.3]:

$$\sum_{n=0}^{\infty} \left(\sum_{i=0}^{\infty} \binom{n}{i} \alpha(i)\beta(n-i) \right) \frac{x^n}{n!} = \left(\sum_{n=0}^{\infty} \alpha(n) \frac{x^n}{n!} \right) \left(\sum_{n=0}^{\infty} \beta(n) \frac{x^n}{n!} \right)$$

where, for our case, $\alpha(n) = n!\,a^n$ and $\beta(n) = nb$. Therefore, we can write:

$$F(x) = \left(\sum_{n=0}^{\infty} n! \frac{(ax)^n}{n!} \right) \left(\sum_{n=0}^{\infty} nb \frac{x^n}{n!} \right)$$

The left hand sum is $G(ax)$ where $G(y) = \sum_n y^n = \frac{1}{1-y}$ [22, Equation 2.5.1]. The right hand sum is equal to $bx\frac{d}{dx}\left(\sum \frac{x^n}{n!}\right)$ [22, Rule $2'$, Section 2.3] $= bx\frac{d}{dx}e^x$ [22, Equation 2.5.3] $= bxe^x$. Thus we have:

$$F(x) = \frac{1}{1-ax} bxe^x = \frac{bxe^x}{1-ax}$$

Therefore, $f^{\checkmark}(0,a,b)$ is the constant term in the power series $\sum_{n=0}^{\infty} f^{\checkmark}(n,a,b)\frac{x^n}{n!}$, which is $F(0) = 0$. To find a recurrence relation defining $f^{\checkmark}(n+1,a,b)$ we equate the original definition of $F(x)$ in Equation 10 with our closed form of this function, differentiate each side (to give us a power series with the $f^{\checkmark}(n,a,b)$ values shifted one position to the left), and multiply by the denominator of the closed form, giving us:

$$\begin{aligned}
& (1-ax)\frac{d}{dx}\left(\sum_{n=0}^{\infty} f^{\checkmark}(n,a,b)\frac{x^n}{n!} \right) = (1-ax)\frac{d}{dx}\left(\frac{bxe^x}{1-ax} \right) \\
\Longrightarrow\quad & (1-ax)\sum_{n=0}^{\infty} f^{\checkmark}(n,a,b)n\frac{x^{n-1}}{n!} = (1-ax)\left(\frac{b(x+1)e^x}{1-ax} + \frac{abxe^x}{(1-ax)^2} \right) \\
\Longrightarrow\quad & \sum_{n=0}^{\infty} f^{\checkmark}(n,a,b)n\frac{x^{n-1}}{n!} - \sum_{n=0}^{\infty} af^{\checkmark}(n,a,b)n\frac{x^n}{n!} = b(x+1)e^x + a\frac{bxe^x}{1-ax} \\
\Longrightarrow\quad & \sum_{n=0}^{\infty} f^{\checkmark}(n+1,a,b)(n+1)\frac{x^n}{(n+1)!} - \sum_{n=0}^{\infty} anf^{\checkmark}(n,a,b)\frac{x^n}{n!} = bxe^x + be^x + a\sum_{n=0}^{\infty} f^{\checkmark}(n,a,b)\frac{x^n}{n!} \\
& \qquad = b\sum_{n=0}^{\infty} n\frac{x^n}{n!} + b\sum_{n=0}^{\infty} \frac{x^n}{n!} + a\sum_{n=0}^{\infty} f^{\checkmark}(n,a,b)\frac{x^n}{n!}
\end{aligned}$$

Equating the coefficients of $\frac{x^n}{n!}$ we get:

$$\begin{aligned}
& f^{\checkmark}(n+1,a,b) - anf^{\checkmark}(n,a,b) = bn + b + af^{\checkmark}(n,a,b) \\
\Longrightarrow\quad & f^{\checkmark}(n+1,a,b) = anf^{\checkmark}(n,a,b) + b(n+1) + af^{\checkmark}(n,a,b) \\
& \qquad = (n+1)(af^{\checkmark}(n,a,b) + b)
\end{aligned} \tag{11}$$

Equation 11 gives us the recurrence relation for the sequence $\{f^{\checkmark}(n,a,b)\}_{n=0}^{\infty}$ that we have been seeking[9]. Figure 5 brings together the equations we have so far for the failure handling case (including those from the previous section defining $n^{\checkmark}(p_d)$ and $n^{\times}(p_d)$.

$$\begin{aligned}
n^{\checkmark}(g_d,n) &= f^{\checkmark}(n, n^{\times}(p_{d-1}), n^{\checkmark}(p_{d-1})) \\
n^{\times}(g_d,n) &= f^{\times}(n, n^{\times}(p_{d-1})) \\
\\
f^{\checkmark}(0,a,b) &= 0 \\
f^{\checkmark}(n+1,a,b) &= (n+1)\,(a\,f^{\checkmark}(n,a,b)+b) \\
f^{\times}(0,a) &= 1 \\
f^{\times}(n+1,a) &= (n+1)\,a\,f^{\times}(n,a) \\
\\
n^{\checkmark}(p_0) &= 1 \\
n^{\times}(p_0) &= \ell \\
n^{\checkmark}(p_d) &= n^{\checkmark}(g_d,j)^k, \text{ for } d>1 \\
n^{\times}(p_d) &= \ell + \left(n^{\times}(g_d,j)+\ell\, n^{\checkmark}(g_d,j)\right)\frac{n^{\checkmark}(g_d,j)^k-1}{n^{\checkmark}(g_d,j)-1}, \text{ for } d>0
\end{aligned}$$

Figure 5: Recurrence relations for the numbers of failures and successes of a goal plan tree in the presence of failure handling

This formulation allows us to more easily see how the behaviour space grows as the number of applicable plans, n, for a goal grows. In particular, the number of successes of a goal for $n+1$ plans is just $n^{\checkmark}(g,d,n+1) = f^{\checkmark}(n+1, n^{\times}(p,d-1,n+1), n^{\checkmark}(p,d-1,n+1))$, which is just $(n+1)\,(a\,f^{\checkmark}(n,a,b)+b)$ where $a \equiv n^{\times}(p,d-1,n+1)$ and $b \equiv n^{\checkmark}(p,d-1,n+1)$. However, note that, with a uniform goal-plan tree, any change to n also applies to sub-trees, i.e. to a and b above.

3.5 The Probability of Failing

In section 3.3 we said that introducing failure handling makes it harder to fail. However, table 1 appears at first glance to contradict this, in that there are many more ways of failing with failure handling than there are without failure handling.

The key to understanding the apparent discrepancy is to consider the *probability* of failing: table 1 merely counts the number of possible execution paths, without considering the likelihood of a particular path being taken. Working out the probability of failing (as we do below) shows

[9] In the simple case when $a = b = 1$ this is listed as sequence A007526 in the On-Line Encyclopedia of Integer Sequences [23]: "the number of permutations of nonempty subsets of $\{1,\cdots,n\}$".

that although there are many more ways of failing (and also of succeeding), the probability of failing is, indeed, much lower.

Let us denote the probability of an execution of a goal-plan tree with root x and depth d failing as $p^{\text{✗}}(x_d)$, and the probability of it succeeding as $p^{\text{✓}}(x_d) = 1 - p^{\text{✗}}(x_d)$.

We assume that the probability of an action failing is ${\varepsilon_a}^{10}$. Then the probability of a given plan's actions all succeeding is simply $(1-\varepsilon_a)^x$ where x is the number of actions. Hence the probability of a plan failing due to the failure of (one of) its actions is simply $1-(1-\varepsilon_a)^x$, i.e. for a plan at depth 0 the probability of failure is:

$$\varepsilon_0 = 1-(1-\varepsilon_a)^{\ell}$$

and for a plan at depth greater than 0 the probability of failure due to actions is:

$$\varepsilon = 1-(1-\varepsilon_a)^{\ell(k+1)}$$

(recall that such a plan has ℓ actions before, after, and between, each of its k sub-goals). Considering not only the actions but also the sub-goals $g_1,\ldots,g_k$ of a plan p, we have that for the plan to succeed, all of the sub-goals must succeed, and additionally, the plan's actions must succeed giving $p^{\text{✓}}(p_d) = (1-\varepsilon)\,p^{\text{✓}}(g_d)^k$. We can easily derive from this an equation for $p^{\text{✗}}(p_d)$ (given below). Note that the same reasoning applies to a plan regardless of whether there is failure handling, because failure handling is done at the goal level.

In the absence of failure handling, for a goal g with possible plans $p_1,\ldots,p_j$ to succeed we select one plan and execute it, so the probability of success is the probability of that plan succeeding, i.e. $p^{\text{✓}}(g_d) = p^{\text{✓}}(p_{d-1})$. We ignore for the moment the possibility of a goal failing due to there being no applicable plans. This assumption is relaxed later on.

Formally, then, we have for the case without failure handling:

$$\begin{array}{rcl} p^{\text{✗}}(g_d) & = & p^{\text{✗}}(p_{d-1}) \\ p^{\text{✗}}(p_0) & = & \varepsilon_0 \\ p^{\text{✗}}(p_d) & = & 1-[(1-\varepsilon)\,(1-p^{\text{✗}}(g_d))^k] \end{array}$$

Now consider what happens when failure handling is added. In this case, in order for a goal to fail, *all* of the plans must fail, i.e. $p^{\text{✗}}(g_d) = p^{\text{✗}}(p_{d-1})^j$. Since failure handling is at the goal level, the equation for plans is unchanged, giving:

$$\begin{array}{rcl} p^{\text{✗}}(g_d) & = & p^{\text{✗}}(p_{d-1})^j \\ p^{\text{✗}}(p_0) & = & \varepsilon_0 \\ p^{\text{✗}}(p_d) & = & 1-[(1-\varepsilon)\,(1-p^{\text{✗}}(g_d))^k] \end{array}$$

It is not easy to see from the equations what the patterns of probabilities actually are, and so, for illustration purposes, the following table shows what the probability of failure is, both with

[10]For simplicity, we assume that the failure of an action in a plan is independent of the failure of other actions in the plan.

and without failure handling, for two scenarios. These values are computed using $j = k = 3$ (i.e. a relatively small branching factor) and with $\ell = 1$. We consider two cases: where $\varepsilon_a = 0.05$ and hence $\varepsilon \approx 0.185$ (which is rather high); and where $\varepsilon_a = 0.01$ and hence $\varepsilon \approx 0.04$.

As can be seen, without failure handling, failure is *magnified*: the larger the goal-plan tree is, the more actions are involved, and hence the greater the chance of an action somewhere failing, leading to the failure of the top-level goal (since there is no failure handling). On the other hand, with failure handling, the probability of failure is both low, and doesn't appear to grow significantly as the goal-plan tree grows.

ε_a	d	No failure handling	With failure handling
0.05	2	30%	0.64%
	3	72%	0.81%
	4	98%	0.86%
0.01	2	7%	0.006%
	3	22%	0.006%
	4	55%	0.006%

We now relax the assumption that a goal cannot fail due to plans not being applicable, i.e. that a goal will only fail once all plans have been tried. Unfortunately, relaxing this assumption complicates the analysis. This is because we need to consider the possibility that none of the remaining plans are applicable at *each point* where failure handling attempts to recover.

Let us begin by reconsidering the case where there is no failure handling. We use ε_g to denote the probability of a goal failing due to none of the remaining plans being applicable. We assume, for analysis purposes, that this probability is constant, and in particular, that it does not depend on which plans have already been tried nor on the number of relevant plans remaining.

Then the probability of a goal failing is $p^{\text{✗}}(g_d) = \varepsilon_g + (1 - \varepsilon_g)\, p^{\text{✗}}(p_{d-1})$, i.e. the goal fails either because no plans are applicable or because there are applicable plans and the selected plan fails. As before, the equation for plans is unchanged, since failure handling is done at the goal level. Collecting this gives the following equations for the case without failure handling:

$$\begin{array}{rcl} p^{\text{✗}}(g_d) & = & \varepsilon_g + (1-\varepsilon_g)\, p^{\text{✗}}(p_{d-1}) \\ p^{\text{✗}}(p_0) & = & \varepsilon_0 \\ p^{\text{✗}}(p_d) & = & 1 - [(1-\varepsilon)\,(1-p^{\text{✗}}(g_d))^k] \end{array}$$

Observe that setting $\varepsilon_g = 0$ yields the equations derived earlier, where we assumed that a goal cannot fail due to inapplicable plans.

We now consider the probability of failure *with* failure handling. For a goal with *two* plans we have the following cases:

- The goal can fail due to no plans being applicable (ε_g)
- If there are applicable plans ($(1 - \varepsilon_g) \ldots$) then the goal can fail if the first selected plan fails ($p^{\text{✗}}(p_{d-1}) \ldots$) *and* if failure handling is not successful, which can occur if either there

are no applicable plans (ε_g) or, if there are applicable plans ($(1-\varepsilon_g)\ldots$) and the selected plan fails ($p^{\text{✗}}(p_{d-1})$).

Pulling this together, for a goal with two plans we have:

$$p^{\text{✗}}(g_d) = \varepsilon_g + (1-\varepsilon_g)\, p^{\text{✗}}(p_{d-1})\,(\varepsilon_g + (1-\varepsilon_g)\, p^{\text{✗}}(p_{d-1}))$$

In the general case of j available plans, we have that a goal can fail if:

A. there are no applicable plans at the outset, with probability ε_g; or

B. there are applicable plans $(1-\varepsilon_g)$, but the selected plan fails ($p^{\text{✗}}(p_{d-1})$) and then either there are no further applicable plans (ε_g), or

C. there are applicable plans $(1-\varepsilon_g)$, but the selected plan fails ($p^{\text{✗}}(p_{d-1})$) and then either there are no further applicable plans (ε_g),

D. and so on: the reasoning of B is repeated j times.

This gives a definition of the following form:

$$\underbrace{\varepsilon_g}_{\text{A}} + \underbrace{(1-\varepsilon_g)\, p^{\text{✗}}(p_{d-1})\,(\varepsilon_g +}_{\text{B}} \underbrace{(1-\varepsilon_g)\, p^{\text{✗}}(p_{d-1})\,(\varepsilon_g +}_{\text{C}} \underbrace{\ldots}_{\text{D}}))$$

This can be defined in terms of an auxiliary function $p^{\text{✗}}(g_d, i)$ which defines the probability of failure for goal g at depth d where there are i remaining relevant plans instances that may (or may not) yield any applicable plan instances:

$$\begin{array}{rcl}
p^{\text{✗}}(g_d) & = & p^{\text{✗}}(g_d, j) \\
p^{\text{✗}}(g_d, 1) & = & \varepsilon_g + (1-\varepsilon_g)\, p^{\text{✗}}(p_{d-1}) \\
p^{\text{✗}}(g_d, i+1) & = & \varepsilon_g + (1-\varepsilon_g)\, p^{\text{✗}}(p_{d-1})\, p^{\text{✗}}(g_d, i) \\
p^{\text{✗}}(p_0) & = & \varepsilon_0 \\
p^{\text{✗}}(p_d) & = & 1 - [(1-\varepsilon)\,(1 - p^{\text{✗}}(g_d))^k]
\end{array}$$

Observe that setting $\varepsilon_g = 0$ reduces this to the definition derived earlier, since $\varepsilon_g + (1-\varepsilon_g)X$ simplifies to X, and hence $p^{\text{✗}}(g_d, i) = p^{\text{✗}}(p_{d-1})^i$.

As before, it is not immediately clear from the formulae what the actual patterns of probability are. Considering illustrative examples (see the table below) shows that (a) the overall behaviour is the same as before, and (b) if ε_g is assumed to be relatively low compared with the probability of action failure (ε and ε_0), then it doesn't significantly affect the probabilities.

		No failure handling			With failure handling		
ε_a	d	$\varepsilon_g = 0$	$\varepsilon_g = 0.01$	$\varepsilon_g = 0.05$	$\varepsilon_g = 0$	$\varepsilon_g = 0.01$	$\varepsilon_g = 0.05$
0.05	2	30%	33%	43%	0.64%	2.2%	9.4%
	3	72%	76%	86%	0.81%	2.6%	12.8%
	4	98%	99%	100%	0.86%	2.8%	16.5%
ε_a	d	$\varepsilon_g = 0$	$\varepsilon_g = 0.005$	$\varepsilon_g = 0.01$	$\varepsilon_g = 0$	$\varepsilon_g = 0.005$	$\varepsilon_g = 0.01$
0.01	2	7%	9%	10%	0.006%	0.5%	1.1%
	3	22%	27%	32%	0.006%	0.6%	1.1%
	4	55%	63%	70%	0.006%	0.6%	1.1%

3.6 Analysis of the Number of Failures

In this section we briefly examine the relationship between the number of action failures and the number of traces.

We have derived equations that calculate the total number of behaviours (with failure handling). But how many of these behaviours involve many action failures? If we make some sort of assumption about the rate of action failure, and hence the number of failures that will occur in an execution of length ℓ, how does this affect the number of behaviours? Do the large numbers that we have seen reduce significantly?

For instance, considering $j = k = 2$, $\ell = 1$ and $d = 2$, there are 1,922 possible executions that result in failure. How many of these involve many failures of actions, and how many involve only a small number of failures? Figure 6 contains (cumulative) numbers which were counted by looking at all possible executions in this (small) case.

The question is how to generalise this analysis for larger execution spaces. Clearly, counting all possible executions is not feasible. Instead, we turn to generating functions.

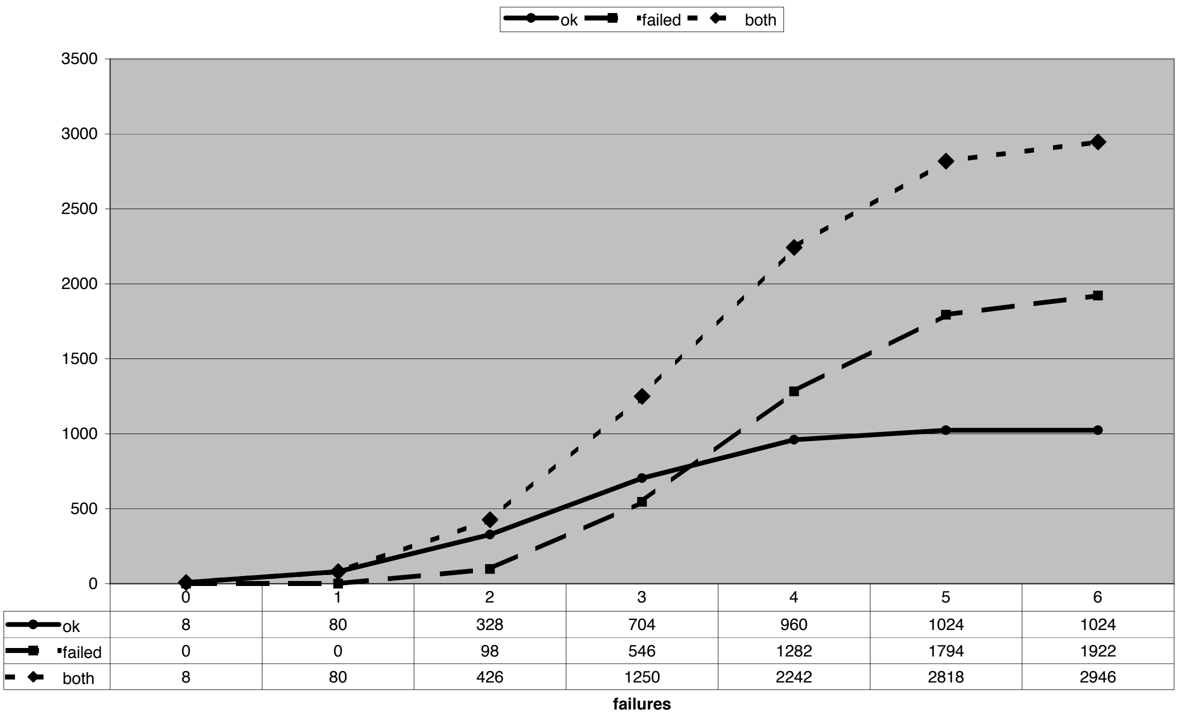
We extend our previous notation by defining $n^{\checkmark}_{af}(s,n)$ and $n^{\times}_{af}(s,n)$ to be the numbers of successful and failed (respectively) executions of a plan body segment s in which exactly n action failures (hence the *af* subscript) have occurred.

For a given segment s (and particularly for $s = g_d$) , we are interested in computing the sequences $\{n^{\checkmark}_{af}(s,n)\}_{n=0}^{\infty}$ and $\{n^{\times}_{af}(s,n)\}_{n=0}^{\infty}$. This can be achived by considering the *ordinary* (rather than exponential) generating functions [22] for these sequences:

$$\begin{aligned} F^{\checkmark}_{af}(s,x) &= \sum_{n=0}^{\infty} n^{\checkmark}_{af}(s,n)\,x^n \\ F^{\times}_{af}(s,x) &= \sum_{n=0}^{\infty} n^{\times}_{af}(s,n)\,x^n \end{aligned}$$

In contrast to exponential generating functions, the terms in these sums do not include a denominator of $n!$.

An action a has one successful execution, which contains no action failures, so $F^{\checkmark}_{af}(a,x) = 1$ (a power series with the coefficient of x^0 being 1 and all other coefficients being 0). Similarly, $F^{\times}_{af}(a,x) = x$ as there is one failed execution, which has one action failure.



	0	1	2	3	4	5	6
ok	8	80	328	704	960	1024	1024
failed	0	0	98	546	1282	1794	1922
both	8	80	426	1250	2242	2818	2946

Figure 6: Number of traces (cumulative) for $j = k = 2$, $\ell = 1$, $d = 2$

We now consider $F^{\checkmark}_{af}(s_1;s_2)$:

$$\begin{aligned}
F^{\checkmark}_{af}(s_1;s_2,x) &= \sum_{n=0}^{\infty} n^{\checkmark}_{af}(s_1;s_2,n)\,x^n \\
&= \sum_{n=0}^{\infty}\left(\sum_{i+j=n} n^{\checkmark}_{af}(s_1,i)\,n^{\checkmark}_{af}(s_2,j)\right)x^n
\end{aligned}$$

The inner sum considers all possible ways of allocating the n action failures to the (necessarily) successful and independent (by assumption) executions of s_1 and s_2. The second line above can be rewritten as a product of ordinary generating functions [22, Rule 3, Section 2.2]:

$$\begin{aligned}
F^{\checkmark}_{af}(s_1;s_2,x) &= \left(\sum_{n=0}^{\infty} n^{\checkmark}_{af}(s_1,n)\,x^n\right)\left(\sum_{n=0}^{\infty} n^{\checkmark}_{af}(s_2,n)\,x^n\right) \\
&= F^{\checkmark}_{af}(s_1,x)\,F^{\checkmark}_{af}(s_2,x)
\end{aligned}$$

Considering $F^{\times}_{af}(s_1;s_2,x)$, we have:

$$\begin{aligned}
F^{\times}_{af}(s_1;s_2,x) &= \sum_{n=0}^{\infty} n^{\times}_{af}(s_1;s_2,n)\,x^n \\
&= \sum_{n=0}^{\infty}\left(n^{\times}_{af}(s_1,n) + \sum_{i+j=n} n^{\checkmark}_{af}(s_1,i)\,n^{\times}_{af}(s_2,j)\right)x^n \\
&= \sum_{n=0}^{\infty} n^{\times}_{af}(s_1,n)\,x^n + \sum_{n=0}^{\infty}\left(\sum_{i+j=n} n^{\checkmark}_{af}(s_1,i)\,n^{\times}_{af}(s_2,j)\right)x^n \\
&= F^{\times}_{af}(s_1,x) + \left(\sum_{n=0}^{\infty} n^{\checkmark}_{af}(s_1,n)\,x^n\right)\left(\sum_{n=0}^{\infty} n^{\times}_{af}(s_2,n)\,x^n\right) \\
&= F^{\times}_{af}(s_1,x) + F^{\checkmark}_{af}(s_1,x)\,F^{\times}_{af}(s_2,x)
\end{aligned}$$

The second line above is based on the observation that each failed execution of $s_1;s_2$ with n action failures is either a failed execution of s_1 with n action failures occurring in that execution, or is a successful execution of s_1 with i failures followed by a failed execution of s_2 with j failures, where $i+j=n$.

Now, assuming that we know $F^{\checkmark}_{af}(g_d,x)$ and $F^{\times}_{af}(g_d,x)$ for some depth d, we can construct the functions $F^{\checkmark}_{af}(p_d,x)$ and $F^{\times}_{af}(p_d,x)$ by applying the results above to expand the right hand sides of the following equations (which simply replace p_d with its plan body):

$$\begin{aligned}
F^{\checkmark}_{af}(p_d,x) &= F^{\checkmark}_{af}(a^{\ell};(g_d;a^{\ell})^k,x) \\
F^{\times}_{af}(p_d,x) &= F^{\times}_{af}(a^{\ell};(g_d;a^{\ell})^k,x)
\end{aligned}$$

It remains to define $F^{\checkmark}_{af}(g_d,x)$ and $F^{\checkmark}_{af}(g_d,x)$ in terms of $F^{\checkmark}_{af}(p_{d-1},x)$ and $F^{\checkmark}_{af}(p_{d-1},x)$. To count the successful executions of g_d with n action failures, we must first choose one of the j applicable plans to be the one that ultimately succeeds. We must then choose between 0 and $j-1$

of the remaining applicable plans that were tried but failed, and consider all possible orderings of these plans. The n action failures must be distributed across the failed and successful plans. This leads us to the following derivation of a procedure to construct $F^{\checkmark}_{af}(g_d,x)$:

$$\begin{aligned}
F^{\checkmark}_{af}(g_d,x) &= \sum_{n=0}^{\infty} n^{\checkmark}_{af}(g_d,n)\,x^n \\
&= \sum_{n=0}^{\infty}\left(j\sum_{m=0}^{j-1}\binom{j-1}{m} m! \sum_{i_0+\cdots+i_m=n} n^{\checkmark}_{af}(p_{d-1},i_0)\,n^{\times}_{af}(p_{d-1},i_1)\cdots n^{\times}_{af}(p_{d-1},i_m)\right) x^n \\
&= j\sum_{m=0}^{j-1}\binom{j-1}{m} m! \sum_{n=0}^{\infty}\left(\sum_{i_0+\cdots+i_m=n} n^{\checkmark}_{af}(p_{d-1},i_0)\,n^{\times}_{af}(p_{d-1},i_1)\cdots n^{\times}_{af}(p_{d-1},i_m)\right) x^n \\
&= j\sum_{m=0}^{j-1}\binom{j-1}{m} m! \left(\sum_{n=0}^{\infty} n^{\checkmark}_{af}(p_{d-1},n)\,x^n\right)\left(\sum_{n=0}^{\infty} n^{\times}_{af}(p_{d-1},n)\,x^n\right)^m \\
&= j\sum_{m=0}^{j-1}\binom{j-1}{m} m!\, F^{\checkmark}_{af}(p_{d-1},x)\,F^{\times}_{af}(p_{d-1},x)^m
\end{aligned}$$

Constructing $F^{\times}_{af}(g_d,x)$ is simpler. A failed execution of a goal involves failed attempts to execute all j applicable plans. All $j!$ orderings of these plans must be considered. This gives us the following construction for $F^{\times}_{af}(g_d,x)$:

$$\begin{aligned}
F^{\times}_{af}(g_d,x) &= \sum_{n=0}^{\infty} n^{\times}_{af}(g_d,n)\,x^n \\
&= \sum_{n=0}^{\infty}\left(j! \sum_{i_1+\cdots+i_j=n} n^{\times}_{af}(p_{d-1},i_1)\cdots n^{\times}_{af}(p_{d-1},i_j)\right) x^n \\
&= j!\left(\sum_{n=0}^{\infty} n^{\times}_{af}(p_{d-1},n)\,x^n\right)^j \\
&= j!\,F^{\times}_{af}(p_{d-1},x)^j
\end{aligned}$$

The equations above define a recursive procedure for computing $F^{\checkmark}_{af}(g_d,x)$ and $F^{\times}_{af}(g_d,x)$ for a given d. Furthermore, given any ordinary generating function $F(x)$ that generates a sequence a_n, the sequence of partial sums of a_n is generated by $F(x)/(1-x)$ [22, Exercise 32, Chapter 2]. Therefore we can generate the numbers of successful and failed executions of g_d with *at most n* action failures using the functions $F^{\checkmark}_{af}(g_d,x)/(1-x)$ and $F^{\times}_{af}(g_d,x)/(1-x)$. The Sage mathematical software system[11] was used to obtain the power series representations of these

[11] http://www.sagemath.org/

functions[12] for the values of d, j, k and l shown in Figures 7 and 8[13].

Examining Figure 7 we can conclude two things. Firstly, the number of traces really explodes for larger numbers of action failures. Secondly, even though making assumptions about the number of failures that can occur reduces the number of traces, the number of traces is still quite large: note the scale on the y-axis – even allowing for at most 5 action failures, there are 1,186,693,266 possible traces. Similar conclusions can be drawn from Figure 8 where, even assuming at most 10 action failures, there are still *many many*[14] traces.

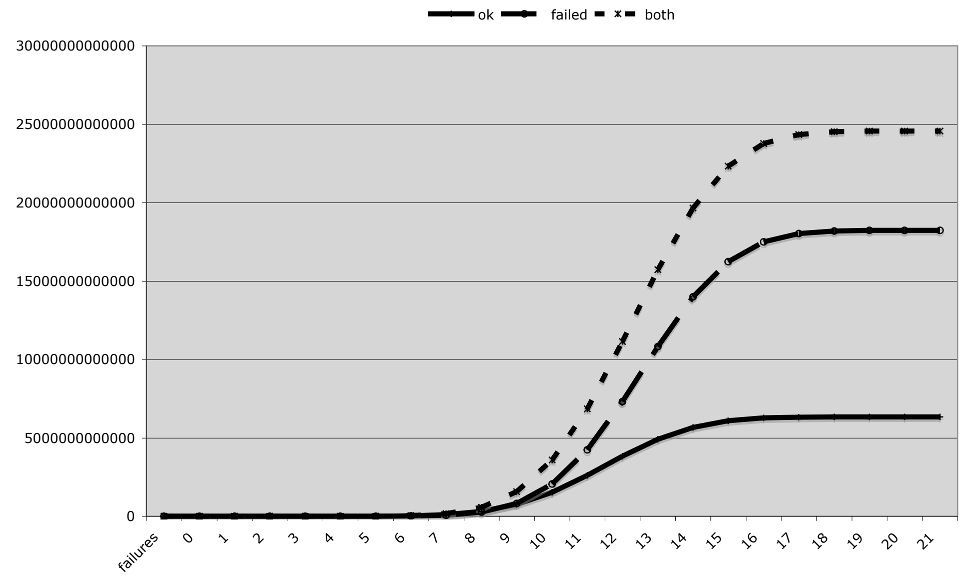


Figure 7: Number of traces (cumulative) for $j = k = 2$, $\ell = 1$, $d = 3$

4 A Reality Check

We now consider to what extent real systems have deep and branching goal-plan trees, and to what extent the large numbers shown in table 1 apply to real applications, rather than to uniform goal-plan trees. As an example of a real application we consider a recent industrial application at Daimler which used BDI agents to realise agile business processes [24].

[12]There are a finite number of actions that can be attempted during any execution of a goal-plan tree, and this defines a bound on the total number of action failures that can occur. Thus $F^{✔}_{af}(g_d, x)$ and $F^{✘}_{af}(g_d, x)$ are polynomial functions of finite order. The functions generating the partial sums have infinite power series, but only a finite number of terms need to be generated as the coefficients increase to a maximum value, which is then repeated infinitely.

[13]The Sage script used for this can be found at http://sage.milnix.org/home/pub/131

[14]4,417,766,935,416,766,347,021,913,820,447,697,963 to be precise.

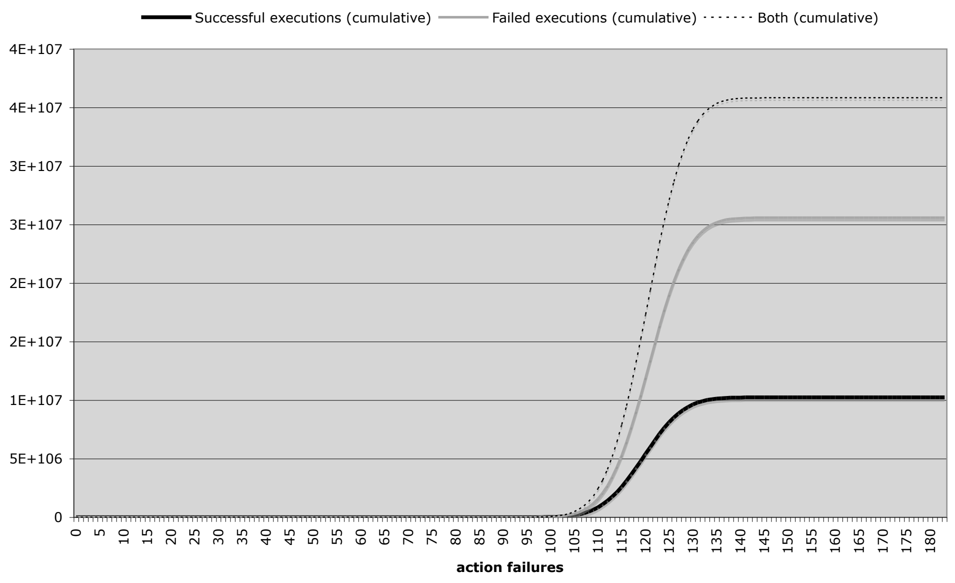


Figure 8: Number of traces (cumulative) for $j = k = d = 3$ and $\ell = 1$

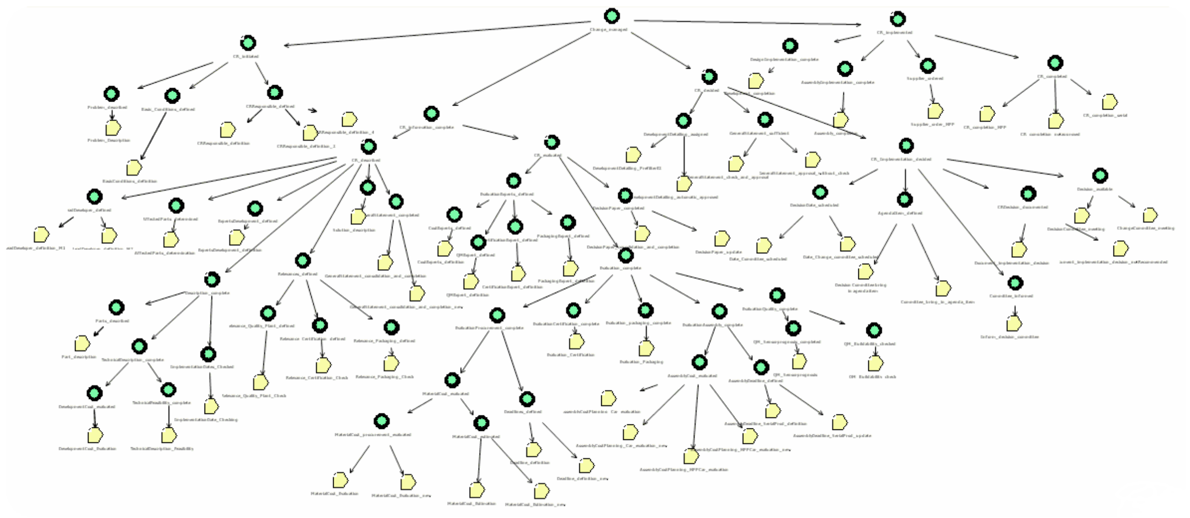


Figure 9: Goal-plan tree from [24, Figure 6]

Figure 9 shows[15] a goal-plan tree from [24] which has "*60 achieve goals in up to 7 levels. 10 maintain goals, 85 plans and about 100 context variables*" [24, Page 41]. Unlike the typical goal-plan trees used in BDI platforms, the tree in Figure 9 consists of layers of "and"-refined goals, with the only "or" refinements being at the leaves (where the plans are). In terms of the analysis presented in this paper we can treat a link from a goal g to a set of goals, say, g_1, g_2, g_3 as being equivalent to the goal g having a single plan p which performs g_1, g_2, g_3 (and has no actions, i.e. $\ell = 0$ for non-leaf plans).

The last row of table 2 gives the various n values for this goal-plan tree, for $\ell = 4$ (top row), $\ell = 2$ (middle row) and $\ell = 1$ (bottom row). Note that these figures are actually *lower bounds* because we assumed that plans at depth 0 are simple linear combinations of ℓ actions, whereas it is clear from [24] that their plans are in fact more complicated, and can contain nested decision making (for example see [24, Figure 4]).

A rough indication of the size of a goal-plan tree is the number of goals. With 57 goals, the tree of Figure 9 has size in between the first two rows of table 2. Comparing the number of possible behaviours of the uniform goal-plan trees against the non-uniform, but real, goal-plan tree, we see that the behaviour space is somewhat smaller in the non-uniform tree, but that it is still quite large, especially in the case with failure handling. However, we do need to remember (a) that the tree of Figure 9 only has plans at the leaves, which reduces its complexity; and (b) that the figures for the tree are a conservative estimate, since we assume that leaf plans have only simple behaviour.

Parameters			Number of		No failure handling (secs 3.1 and 3.2)		With failure handling (section 3.3)	
j	k	d	goals	actions	$n^{\checkmark}(g)$	$n^{\times}(g)$	$n^{\checkmark}(g)$	$n^{\times}(g)$
2	2	3	21	62 (13)	128	614	$\approx 6.33 \times 10^{12}$	$\approx 1.82 \times 10^{13}$
3	3	3	91	363 (25)	1,594,323	6,337,425	$\approx 1.02 \times 10^{107}$	$\approx 2.56 \times 10^{107}$
Workflow with 57 goals(*)					294,912	3,250,604 ($\ell = 4$)	$\approx 2.98 \times 10^{20}$	$\approx 9.69 \times 10^{20}$
(*) The paper says 60 goals,					294,912	1,625,302 ($\ell = 2$)	$\approx 6.28 \times 10^{15}$	$\approx 8.96 \times 10^{15}$
but Figure 9 has 57 goals.					294,912	812,651 ($\ell = 1$)	$\approx 9.66 \times 10^{11}$	$\approx 6.27 \times 10^{11}$

Table 2: Illustrative values for $n^{\checkmark}(g)$ and $n^{\times}(g)$

5 Conclusion

To summarise, our analysis has found that the space of possible behaviours for BDI agents is, indeed, large. As expected, the number of possible behaviours grows as the tree's depth (d) and breadth (j and k) grow. However, somewhat surprisingly, the introduction of failure handling

[15]The details are not meant to be legible: the structure is what matters.

makes a very significant difference to the number of behaviours. For instance, for a uniform goal-plan tree with depth 3 and $j = k = 2$ adding failure handling took the number of successful behaviours from 128 to 6,332,669,231,104.

To put our analysis results in context, we now briefly compare the behaviour space of BDI agents (with no concurrency) with concurrent systems. Consider a uniform goal-plan tree of depth 3 with $k = j = 3$, then we have 1,594,323 successful executions without failure handling, and around 1.02×10^{107} successful executions with failure handling. On the other hand, the number of ways of interleaving n parallel executions, each of length ℓ is (see e.g. [21, Section 3]) $(n\ell)!/(\ell!)^n$ and if we consider the interleavings of two sequential processes with 13 steps[16] then there are 10,400,600 possibilities ($26!/13! \times 13!$). For three processes with 13 steps there are $\approx 8.45 \times 10^{16}$ interleavings and for four processes $\approx 5.36 \times 10^{28}$. In other words, given a certain number of actions, the number of possible behaviours generated by putting these actions into a goal-plan tree (with failure handling) considerably exceeds the number of behaviours obtained by interleaving the concurrent execution of the actions.

What does the analysis in this paper tell us about the testability of BDI agent systems? Before we can answer this question, we need to consider what is being tested. Testing is typically carried out at the levels of individual components (unit testing), collections of components (integration testing) and the system as a whole.

Consider testing of a whole system. The behaviour space sizes depicted in tables 1 and 2 suggest quite strongly that attempting to obtain assurance of a system's correctness by testing the system as a whole is not feasible. In fact, the situation is even worse when we consider not only the *number* of possible executions but also the *probability* of failing: the space of *unsuccessful* executions is particularly hard to test, since there are many unsuccessful executions (more than successful ones), and the probability of an unsuccessful execution is low, making this part of the behaviour space hard to "reach".

Furthermore, as shown in section 3.6, although making assumptions about the possible numbers of action failures that can occur in a given execution reduces the number of possible behaviours, there are still many many behaviours, even for relatively small trees (e.g. $j = k = d = 3$).

So system testing of BDI agents seems to be impractical. What about unit testing and integration testing? Although unit and integration testing are useful, it is not always clear how to apply them usefully to agent systems where the interesting behaviour is complex and possibly emergent. For example, given an ant colony optimisation system [25], testing a single ant doesn't provide much useful information about the correct functioning of the whole system. Similarly, for BDI agents, when testing a sub-goal it can be difficult to ensure that testing covers all the situations in which the goal may be attempted. Likewise, when testing an agent without the rest of the system (including other agents that it interacts with) it can be hard to ensure adequate coverage of different possibilities.

We do need to acknowledge that our analysis is somewhat pessimistic: real BDI systems do not necessarily have deep or heavily branching goal-plan trees. Indeed, the tree described in section 4 has a smaller behaviour space than the abstract goal-plan trees analysed in section 3.

[16] A goal-plan tree of depth 3 with $k = j = 3$ involves executing 13 actions (for $\ell = 1$).

However, even though smaller, it is still quite large, and this did cause problems in validation:

> "*One of the big challenges during the test phase was to keep the model consistent and to define the right context conditions that result in the correct execution for all scenarios. Therefore more support for dependency analysis, automated simulation and testing of the process models is needed*" [24, p42].

However, overall the conclusion seems to be that BDI systems are verifiable through testing to the extent that they *refrain* from exploiting the BDI representation and its features!

Future Work

There is room for extending the analysis of section 3. Firstly, our analysis is for a *single* goal within a *single* agent. Multiple agents that are collaborating to achieve a single high-level goal can be viewed as having a shared goal-plan tree where certain goals and/or plans are allocated to certain agents. Of course, in such a "distributed goal plan tree" there is concurrency. Furthermore, we have only considered achievement goals. It would be interesting to consider other types of goals [26]. Secondly, our analysis has focussed on BDI agents, which are just one particular type of agent. It would be interesting to consider other sorts of agent systems, and, more broadly, other sorts of adaptive systems.

More importantly, having highlighted the infeasibility of verifying BDI agent systems through testing, we need to find other ways of verifying such systems.

An approach that has some promise is the automatic generation of test cases for agent systems [27, 6]. However, the size of the behaviour space suggests that the number of tests cases needed may be very large, and that testing for failed plan execution is difficult. One interesting, and potentially promising, avenue is to use formal techniques to help guide the test generation process (e.g. symbolic execution or specification-guided testing) [28].

Another promising approach that has attracted some interest is model checking[17] of agent systems [29, 30, 31]. However, more work is needed: Raimondi and Lomuscio [31] verify systems where agents are defined abstractly, i.e. not in terms of plans and goals; the MABLE agent programming language [29] is actually an imperative language augmented with certain agent features, not a BDI language; and the work of Bordini *et al.* [30] does not include failure handling.

Acknowledgements

We would like to thank members of the Software Engineering and Collaborative Modelling Laboratory at Otago University[18] for discussions relating to this paper. Work on this paper was done while Winikoff was on sabbatical from RMIT, visiting the University of Otago.

[17]There has also been work on deductive verification, but (based on research into the verification of concurrent systems) this appears to be less likely to result in verification tools that are both (relatively) easy to use and applicable to real systems.

[18]http://secml.otago.ac.nz/

References

[1] Wooldridge, M.: An Introduction to MultiAgent Systems. John Wiley & Sons (Chichester, England) (2002). ISBN 0 47149691X

[2] Munroe, S., Miller, T., Belecheanu, R., Pechoucek, M., McBurney, P., Luck, M.: Crossing the agent technology chasm: Experiences and challenges in commercial applications of agents. Knowledge Engineering Review **21**(4), 345–392 (2006)

[3] Benfield, S.S., Hendrickson, J., Galanti, D.: Making a strong business case for multiagent technology. In: P. Stone, G. Weiss (eds.) Proceedings of the Fifth Conference on Autonomous Agents and Multi-Agent Systems (AAMAS), pp. 10–15. ACM Press (2006)

[4] Rao, A.S., Georgeff, M.P.: Modeling rational agents within a BDI-Architecture. In: J. Allen, R. Fikes, E. Sandewall (eds.) Principles of Knowledge Representation and Reasoning, Proceedings of the Second International Conference, pp. 473–484. Morgan Kaufmann (1991)

[5] Bratman, M.E.: Intentions, Plans, and Practical Reason. Harvard University Press, Cambridge, MA (1987)

[6] Zhang, Z., Thangarajah, J., Padgham, L.: Automated unit testing for agent systems. In: Second International Working Conference on Evaluation of Novel Approaches to Software Engineering (ENASE), pp. 10–18 (2007)

[7] Ekinci, E.E., Tiryaki, A.M., Çetin, Ö.: Goal-oriented agent testing revisited. In: J.J. Gomez-Sanz, M. Luck (eds.) Ninth International Workshop on Agent-Oriented Software Engineering, pp. 85–96 (2008)

[8] Gomez-Sanz, J.J., Botía, J., Serrano, E., Pavón, J.: Testing and debugging of MAS interactions with INGENIAS. In: J.J. Gomez-Sanz, M. Luck (eds.) Ninth International Workshop on Agent-Oriented Software Engineering, pp. 133–144 (2008)

[9] Nguyen, C.D., Perini, A., Tonella, P.: Experimental evaluation of ontology-based test generation for multi-agent systems. In: J.J. Gomez-Sanz, M. Luck (eds.) Ninth International Workshop on Agent-Oriented Software Engineering, pp. 165–176 (2008)

[10] Padgham, L., Winikoff, M.: Developing Intelligent Agent Systems: A Practical Guide. John Wiley and Sons (2004). ISBN 0-470-86120-7

[11] Shaw, P., Farwer, B., Bordini, R.: Theoretical and experimental results on the goal-plan tree problem. In: Autonomous Agents and Multiagent Systems (AAMAS), pp. 1379–1382. IFAAMAS (2008)

[12] Busetta, P., Rönnquist, R., Hodgson, A., Lucas, A.: JACK Intelligent Agents - Components for Intelligent Agents in Java. AgentLink News (2) (1999). URL http://www.agentlink.org/newsletter/2/newsletter2.pdf

[13] Huber, M.J.: JAM: A BDI-theoretic mobile agent architecture. In: Proceedings of the Third International Conference on Autonomous Agents (Agents'99), pp. 236–243. ACM Press (1999)

[14] d'Inverno, M., Kinny, D., Luck, M., Wooldridge, M.: A formal specification of dMARS. In: M. Singh, A. Rao, M. Wooldridge (eds.) Intelligent Agents IV: Proceedings of the Fourth International Workshop on Agent Theories, Architectures, and Languages, pp. 155–176. Springer-Verlag, LNAI 1365 (1998)

[15] Georgeff, M.P., Lansky, A.L.: Procedural knowledge. Proceedings of the IEEE, Special Issue on Knowledge Representation **74**(10), 1383–1398 (1986)

[16] Ingrand, F.F., Georgeff, M.P., Rao, A.S.: An architecture for real-time reasoning and system control. IEEE Expert **7**(6), 33–44 (1992)

[17] Bordini, R.H., Hübner, J.F., Wooldridge, M.: Programming multi-agent systems in AgentSpeak using Jason. Wiley (2007). ISBN 0470029005

[18] Rao, A.S.: AgentSpeak(L): BDI agents speak out in a logical computable language. In: W.V. de Velde, J. Perrame (eds.) Agents Breaking Away: Proceedings of the Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW'96), pp. 42–55. Springer Verlag, LNAI 1038 (1996)

[19] Winikoff, M., Padgham, L., Harland, J., Thangarajah, J.: Declarative & procedural goals in intelligent agent systems. In: Proceedings of the Eighth International Conference on Principles of Knowledge Representation and Reasoning (KR2002), pp. 470–481. Morgan Kaufmann, Toulouse, France (2002)

[20] Georgeff, M.: Service orchestration: The next big challenge. DM Review Special Report (2006). URL http://www.dmreview.com/specialreports/20060613/1056195-1.html. (2006)

[21] Naish, L.: Resource-oriented deadlock analysis. In: V. Dahl, I. Niemelä (eds.) Proceedings of the 23rd International Conference on Logic Programming (ICLP), pp. 302–316. Springer, LNCS 4670 (2007)

[22] Wilf, H.S.: generatingfunctionology, second edn. Academic Press Inc., Boston, MA (1994). URL http://www.math.upenn.edu/~wilf/gfology2.pdf

[23] Sloane, N.J.A.: The on-line encyclopedia of integer sequences. http://www.research.att.com/~njas/sequences/ (2007)

[24] Burmeister, B., Arnold, M., Copaciu, F., Rimassa, G.: BDI-agents for agile goal-oriented business processes. In: Proceedings of the Seventh Conference on Autonomous Agents and Multiagent Systems (AAMAS), industry track., pp. 37–44. IFAAMAS (2008)

[25] Parunak, H.V.D.: “go to the ant”: Engineering principles from natural multi-agent systems. Annals of Operations Research **75**, 69–101 (1997). (Special Issue on Artificial Intelligence and Management Science)

[26] van Riemsdijk, M.B., Dastani, M., Winikoff, M.: Goals in agent systems: A unifying framework. In: Proceedings of the Seventh Conference on Autonomous Agents and Multiagent Systems (AAMAS), pp. 713–720. IFAAMAS (2008)

[27] Nguyen, C.D., Perinirini, A., Tonella, P.: Automated continuous testing of multi-agent systems. In: The Fifth European Workshop on Multi-Agent Systems (EUMAS) (2007)

[28] Dwyer, M.B., Hatcliff, J., Pasareanu, C., Robby, Visser, W.: Formal software analysis: Emerging trends in software model checking. In: International Conference on Software Engineering: Future of Software Engineering, pp. 120–136 (2007)

[29] Wooldridge, M., Fisher, M., Huget, M.P., Parsons, S.: Model checking multi-agent systems with MABLE. In: Proceedings of the First Conference on Autonomous Agents and Multi-Agent Systems (AAMAS), pp. 952–959. ACM Press (2002)

[30] Bordini, R.H., Fisher, M., Pardavila, C., Wooldridge, M.: Model checking AgentSpeak. In: Proceedings of the Second Conference on Autonomous Agents and Multiagent Systems (AAMAS), pp. 409–416. ACM Press (2003)

[31] Raimondi, F., Lomuscio, A.: Automatic verification of multi-agent systems by model checking via ordered binary decision diagrams. J. Applied Logic **5**(2), 235–251 (2007)

A Expanding Goal-Plan Trees

The process of expanding a goal-plan tree into a sequence of actions when using failure handling can be specified precisely by the following Prolog code[19].

To execute a goal we select a plan and execute it. If the plan’s execution fails, then we recover by “re-posting”, i.e. trying the goal again but excluding the plan that has already been tried.

```
exec(goal([]),[]).
exec(goal(Plans), A) :- remove(Plans,P,Rest), exec(P,A1),
    if(failed(A1), recover(Rest,A1,A), eq(A,A1)).
recover(Plans,ActS,R) :- exec(goal(Plans),R1), append(ActS,R1,R).
```

To execute a plan we simply execute its body one step at a time. If any step fails, then we fail, otherwise we continue to execute the remaining steps.

[19] Written in the W-Prolog variant (http://www.winikoff.net/wp)

```
exec(plan([]), []).
exec(plan([S|Ss]), R) :- exec(S,A1),
    if(failed(A1), eq(R,A1), continue(Ss,A1,R)).
continue(Ss,A1,R) :- exec(plan(Ss),A2), append(A1,A2,R).
```

An action can either (non-deterministically) succeed or fail.

```
exec(act(A), [A]).
exec(act(A), [A,fail]).
```

An action sequence is failed if its last element is ”fail”

```
failed(S) :- append(X,[fail],S).
```

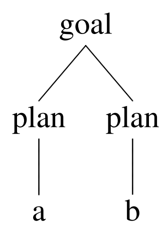
We make use of two library predicates: `append` and `remove`.

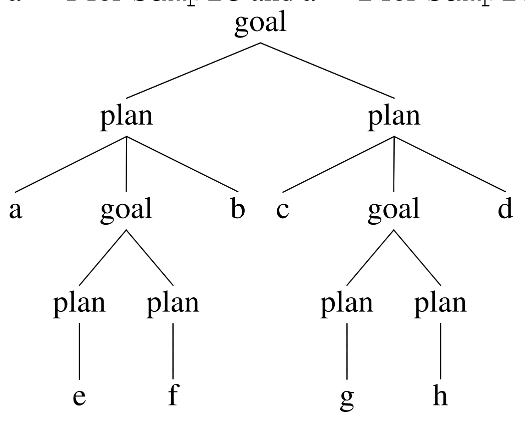
```
append([],X,X).
append([X|Xs],Y,[X|Z]) :- append(Xs,Y,Z).
% remove(A,B,C) iff removing element B from list A leaves list C
remove([X|Xs],X,Xs).
remove([X|Xs],Y,[X|Z]) :- remove(Xs,Y,Z).
```

We collect all possible executions for a given goal-plan tree as follows:

```
go(A) :- sample(P), exec(P,A).
go :- sample(P), exec(P,A), print(A), nl, fail.
go2(A) :- sample2(P), exec(P,A).
go2 :- sample2(P), exec(P,A), print(A), nl, fail.
```

Where the following trees are encoded by `sample` (left) and `sample2` (right). The trees correspond to $j = 2$, $k = \ell = 1$, $d = 1$ for `sample` and $d = 2$ for `sample2`.





The result of `go` is the following output, where a letter indicates the execution of an action, and a ✘ indicates failure. As predicted by our formulae, there are 4 successful executions and 2 unsuccessful executions:

```
a        a ✘ b        a ✘ b ✘
b        b ✘ a        b ✘ a ✘
```

The result of `go2` is the following 162 possibilities (consisting of 64 successful, and 98 unsuccessful executions).

```
a e b
a e b ✘ c g d
a e b ✘ c g d ✘
a e b ✘ c g ✘ h d
a e b ✘ c g ✘ h d ✘
a e b ✘ c g ✘ h ✘
a e b ✘ c h d
a e b ✘ c h d ✘
a e b ✘ c h ✘ g d
a e b ✘ c h ✘ g d ✘
a e b ✘ c h ✘ g ✘
a e b ✘ c ✘
a e ✘ f b
a e ✘ f b ✘ c g d
a e ✘ f b ✘ c g d ✘
a e ✘ f b ✘ c g ✘ h d
a e ✘ f b ✘ c g ✘ h d ✘
a e ✘ f b ✘ c g ✘ h ✘
a e ✘ f b ✘ c h d
a e ✘ f b ✘ c h d ✘
a e ✘ f b ✘ c h ✘ g d
a e ✘ f b ✘ c h ✘ g d ✘
a e ✘ f b ✘ c h ✘ g ✘
a e ✘ f b ✘ c ✘
a e ✘ f ✘ c g d
a e ✘ f ✘ c g d ✘
a e ✘ f ✘ c g ✘ h d
a e ✘ f ✘ c g ✘ h d ✘
a e ✘ f ✘ c g ✘ h ✘
a e ✘ f ✘ c h d
a e ✘ f ✘ c h d ✘
a e ✘ f ✘ c h ✘ g d
a e ✘ f ✘ c h ✘ g d ✘
a e ✘ f ✘ c h ✘ g ✘
a e ✘ f ✘ c ✘
a f b
a f b ✘ c g d

a f b ✘ c h d
a f b ✘ c h d ✘
a f b ✘ c h ✘ g d
a f b ✘ c h ✘ g d ✘
a f b ✘ c h ✘ g ✘
a f b ✘ c ✘
a f ✘ e b
a f ✘ e b ✘ c g d
a f ✘ e b ✘ c g d ✘
a f ✘ e b ✘ c g ✘ h d
a f ✘ e b ✘ c g ✘ h d ✘
a f ✘ e b ✘ c g ✘ h ✘
a f ✘ e b ✘ c h d
a f ✘ e b ✘ c h d ✘
a f ✘ e b ✘ c h ✘ g d
a f ✘ e b ✘ c h ✘ g d ✘
a f ✘ e b ✘ c h ✘ g ✘
a f ✘ e b ✘ c ✘
a f ✘ e ✘ c g d
a f ✘ e ✘ c g d ✘
a f ✘ e ✘ c g ✘ h d
a f ✘ e ✘ c g ✘ h d ✘
a f ✘ e ✘ c g ✘ h ✘
a f ✘ e ✘ c h d
a f ✘ e ✘ c h d ✘
a f ✘ e ✘ c h ✘ g d
a f ✘ e ✘ c h ✘ g d ✘
a f ✘ e ✘ c h ✘ g ✘
a f ✘ e ✘ c ✘
a ✘ c g d
a ✘ c g d ✘
a ✘ c g ✘ h d
a ✘ c g ✘ h d ✘
a ✘ c g ✘ h ✘
a ✘ c h d
a ✘ c h d ✘
a ✘ c h ✘ g d

c g d ✘ a e b
c g d ✘ a e b ✘
c g d ✘ a e ✘ f b
c g d ✘ a e ✘ f b ✘
c g d ✘ a e ✘ f ✘
c g d ✘ a f b
c g d ✘ a f b ✘
c g d ✘ a f ✘ e b
c g d ✘ a f ✘ e b ✘
c g d ✘ a f ✘ e ✘
c g d ✘ a ✘
c g ✘ h d
c g ✘ h d ✘ a e b
c g ✘ h d ✘ a e b ✘
c g ✘ h d ✘ a e ✘ f b
c g ✘ h d ✘ a e ✘ f b ✘
c g ✘ h d ✘ a e ✘ f ✘
c g ✘ h d ✘ a f b
c g ✘ h d ✘ a f b ✘
c g ✘ h d ✘ a f ✘ e b
c g ✘ h d ✘ a f ✘ e b ✘
c g ✘ h d ✘ a f ✘ e ✘
c g ✘ h d ✘ a ✘
c g ✘ h ✘ a e b
c g ✘ h ✘ a e b ✘
c g ✘ h ✘ a e ✘ f b
c g ✘ h ✘ a e ✘ f b ✘
c g ✘ h ✘ a e ✘ f ✘
c g ✘ h ✘ a f b
c g ✘ h ✘ a f b ✘
c g ✘ h ✘ a f ✘ e b
c g ✘ h ✘ a f ✘ e b ✘
c g ✘ h ✘ a f ✘ e ✘
c g ✘ h ✘ a ✘
c h d
c h d ✘ a e b
c h d ✘ a e b ✘

c h d ✘ a f b ✘
c h d ✘ a f ✘ e b
c h d ✘ a f ✘ e b ✘
c h d ✘ a f ✘ e ✘
c h d ✘ a ✘
c h ✘ g d
c h ✘ g d ✘ a e b
c h ✘ g d ✘ a e b ✘
c h ✘ g d ✘ a e ✘ f b
c h ✘ g d ✘ a e ✘ f b ✘
c h ✘ g d ✘ a e ✘ f ✘
c h ✘ g d ✘ a f b
c h ✘ g d ✘ a f b ✘
c h ✘ g d ✘ a f ✘ e b
c h ✘ g d ✘ a f ✘ e b ✘
c h ✘ g d ✘ a f ✘ e ✘
c h ✘ g d ✘ a ✘
c h ✘ g ✘ a e b
c h ✘ g ✘ a e b ✘
c h ✘ g ✘ a e ✘ f b
c h ✘ g ✘ a e ✘ f b ✘
c h ✘ g ✘ a e ✘ f ✘
c h ✘ g ✘ a f b
c h ✘ g ✘ a f b ✘
c h ✘ g ✘ a f ✘ e b
c h ✘ g ✘ a f ✘ e b ✘
c h ✘ g ✘ a f ✘ e ✘
c h ✘ g ✘ a ✘
c ✘ a e b
c ✘ a e b ✘
c ✘ a e ✘ f b
c ✘ a e ✘ f b ✘
c ✘ a e ✘ f ✘
c ✘ a f b
c ✘ a f b ✘
c ✘ a f ✘ e b
c ✘ a f ✘ e b ✘
```

a f b ✘ c g d ✘
a f b ✘ c g ✘ h d
a f b ✘ c g ✘ h d ✘
a f b ✘ c g ✘ h ✘

a ✘ c h ✘ g d ✘
a ✘ c h ✘ g ✘
a ✘ c ✘
c g d

c h d ✘ a e ✘ f b
c h d ✘ a e ✘ f b ✘
c h d ✘ a e ✘ f ✘
c h d ✘ a f b

c ✘ a f ✘ e ✘
c ✘ a ✘