Probabilistic Analysis for Component Reliability Composition *

Dave Mason
Ryerson University
350 Victoria Street
Toronto, Ontario, Canada
dmason@sarg.ryerson.ca

ABSTRACT

One of the desirable properties of predictable assembly is reliability. Given reliability and transformation functions for components, it is possible to accurately compose reliabilities. Currently the transformations are limited in their domain of applicability, but we are working to extend their domain.

Categories and Subject Descriptors

D.2 [**Software**]: Software Engineering

1. INTRODUCTION

Predictable assembly is the activity of predicting properties of assemblies of components prior to actually acquiring the components. One of the many properties of interest is reliability. That is, given a set of components with known reliability, predict the reliability of the assembly.

The first problem is to accurately characterize the reliability of the components of which the assembly will be composed. Unfortunately this cannot be treated as a simple number, but must be a function of the component input.

The second problem is a calculus for composing the component reliabilities so as to calculate the assembly reliability.

Both parts of the problem are computationally expensive, but are amenable to parallelism, so have the potential to be practical despite the cost. A larger problem with the technique is that it is only currently applicable to a limited problem domain; we are working to extend the application domain.

2. COMPONENT RELIABILITY

Reliability has long been considered as a number between 0 and 1 assigned to the execution of a program in a particular environment for a given amount of time. While this may be acceptable in measuring a complete system, it clearly is not applicable for a component that is intended for incorporation into an assembly where the environment is unknown.

To talk about reliability in this context, a component's reliability must be expressed as a function from an input distribution or operational profile to a number between 0 and 1, as described in [2]. This allows the reliability to be determined however the component is used in an assembly.

Generating the paths and reliabilities requires access to the original programs, however once they have been generated they can be distributed to component users without providing access to the program. This allows the users to characterize the reliability of the components in their particular environment (with their operational profile) and thereby determine the reliability of their system before having access to the component.

2.1 Path Generation

Our approach is to generate the paths through the component. Since all of the elements of the input that are selected by the conditionals that define the path will execute exactly the same sequence of instructions. If they are further partitioned by the specification, the resulting subdomains will have a common specification and a common implementation, and thus form a continuous domain of reliability. This is not to say that all points in the domain will have identical reliability, but that there is a continuity whereby a statistical sampling of the domain will produce a meaningful measure of reliability. This is related to [4], but with the path domains further restricted so as to guarantee that all points in the same domain are executed by exactly the same sequence of instructions and that therefore sampling of points within that domain is a statistically valid approach.

The next step is to identify paths that lead to failures such as divide-by-zero, arithmetic overflow, etc. and designate their domain as failing. Of those that are not obviously failing, the next step is to sample the implementation and specification so as to assign a failure factor. There are several special cases that can allow direct determination of the reliability for some domains.

The result of this process is a set of domains associated with paths and their associated failure factor. Then, given an input distribution, the reliability of the component with that distribution is easily calculated.

2.2 Infinite Numbers of Paths

Of course, there may be a huge, or even infinite, number of paths for a given component. However, with the use of

*

CBSE Orlando, FL, USA

Probability Density Functions (see section 3), it is possible to generate the paths in frequency order based on the input distribution. This means that the paths that have the largest contribution to the reliability are generated first. By assuming that all the paths that have not yet been generated are failing, a conservative approximation to the reliability will be available from very early on in the process of generating the paths. As more paths are generated, the approximation will asymptotically approach the true reliability.

Furthermore, the input distribution used to generate the paths can change over time. Once a path is generated, there is no need to generate it again. So the component developer can start with an arbitrary distribution, and then modify it as time goes on to reflect the actual usage in customers' systems. The more accurate the initial distribution, the faster the meaningful set of paths will be produced.

3. PROBABILITY DENSITY FUNCTIONS

Simply having a reliability function would be acceptable if there was only a single component in our assembly for which we needed to determine a reliability. In practice, of course, each component transforms its input (proportional to its distribution) and produces outputs with their own distributions, which become the input distribution for the components that follow.

In [2] the input to the reliability function is described as a histogram. Unfortunately as each component transforms its input, the resulting distribution is not a histogram except in the extreme expression as a point-wise histogram, which is computationally intractable. [3] presents a preliminary treatment of the math required to work with the input distribution as Probability Density Functions.

A Probability Density Function is a positive function with an integral of 1. The input distribution to the component will be expressed as a Probability Density Function for each of the parameters to the component. For our purposes the range of each Probability Density Function will be the set of all possible values for that parameter. This is the maximum number of independent variables for the component, but it is possible that the number will be smaller, as some variables may be functions of others.

It is possible to define Probability Density Functions for each operation of the programming language. The PDF for addition is:

$$P_{x+y}(z) = \int P_{x,y}(z-y, y)\, dy.$$

The form for addition can be extended to any dyadic left-invertible function (such that: $f^{-1}(f(x,y),y) = x$) as:

$$P_{f(x,y)}(z) = \int P_{x,y}(f^{-1}(z,y), y) \left| \frac{d}{dz} f^{-1}(z,y) \right| dy.$$

Similarly any relation R has a PDF:

$$P_{xRy}(true) = \int \left\{ \begin{array}{ll} P_{x,y}(u,v), & \text{when } uRv; \\ 0, & \text{otherwise.} \end{array} \right\} du\, dv,$$
$$P_{xRy}(false) = 1 - P_{xRy}(true).$$

The relations are used to determine the probability of following any particular path through the component, based on the values of the variables.

4. COMPONENT PATH ANALYSIS

We are extending a Scheme[1] interpreter to perform abstract interpretation of functions in order to determine the paths.

`+ - / * < = > <> <= >=`
All of these operations are extended to operate on PDF values.

`(pdf-eval` *callback function arguments...*`)`
This performs an abstract interpretation of the function, with the arguments (if provided). As paths are recognized, the callback is called with the frequency of that path (with the provided arguments) and the variant information for that path. That callback can provide the variant to another program to characterize the reliability of the path, add it to a database, etc.

`(pdf-discrete` *start end step*`)`
This generates a discrete PDF to provide as parameters for `pdf-eval` or `pdf-get-coverage`.

`(pdf-get-coverage` *variant arguments...*`)`
This allows the callback function to calculate the frequency of a variant with a different set of arguments.

`(pdf-get-variant` *variant*`)`
This allows the callback function to get a list value representing the path calculation and the predicate.

Here is a very simple example:

```
(pdf-eval (lambda (freq variant)
            (display (cons freq
                           (pdf-get-variant variant)))
            (newline))
          (lambda (x)
            (if (< (/ 20 x) 3)
                x
                3))
          (pdf-discrete (- 2) 6 2))
```

which, when run, produces:

```
(0.200000 (and (= x 0))
          <fault:divide by zero>)
(0.600000 (and (<> x 0) (>= (/ 20 x) 3))
          3)
(0.200000 (and (<> x 0) (< (/ 20 x) 3))
          x)
```

which shows the 3 path domains in this function and for each one the frequency with the provided parameters (in this example: `(pdf-discrete (- 2) 6 2)`), the predicate that identifies the path, and the calculation of the path. Note that fault paths are provided immediately, but that other paths are provided in frequency order.

4.1 Component-Writer's Role

The role of the component-writer is to:

- produce the component: the current implementation is in Scheme, but there is nothing intrinsic about the choice - implementations for other languages could be built, although a safe language would be preferable;
- perform the abstract interpretation: this is fairly automatic, and could be spread over a whole set of machines to run in parallel;

- characterize the reliability of each domain: this means taking the intersection of the specification and the predicate generated for the path and then comparing the specification and the implementation - either structurally or by sampling - this can also be spread over a set of machines.

5. COMPONENT ASSEMBLY

Components are assembled using a calculus somewhat similar to [2], except with the transformation and reliability functions expressed in terms of Probability Density Functions. This means that an abstract interpretation of the system is performed where values may be PDFs as well as the ground values of the language and that in addition to PDF transformations for simple operations, there will be PDF transformations for components.

For each execution of a component, the reliability will be calculated using the reliability function for the component and the total reliability of the program will be the reliability of the components, weighted by the likelihood of each path being executed.

To determine the operational profile of various components in the system, it probably will be necessary to have an implementation of each component. This doesn't necessarily need to be fast, but needs to accurately reflect the operation of the component. Because of the way the paths are generated, a slow version of the function can be created from the variants generated so far:

```
(lambda (x)
  (cond
     ((and (= x 0))
          <fault:divide by zero>)
     ((and (<> x 0) (>= (/ 20 x) 3))
          3)
     ((and (<> x 0) (< (/ 20 x) 3))
          x)
     (else
          <fault:not implemented>)))
```

5.1 Component-User's Role

The role of the component-user is to:

- specify the component;
- farm-out the component creation;
- determine the operational profile for the system;
- build the system glue;
- flow the operational profile through the system to get the operational profile for each component, and then use the reliability function provided by the component-writer to determine the overall system reliability.

6. LIMITATIONS

There are two limitations of the work in its current state: computational complexity, and domain of applicability.

6.1 Computational Complexity

Calculation with Probability Density Functions is exponential in the number of independent variables to each component and in the operational path-length of the component. In the domain of applicability, the number of independent variables is likely to be small, and the operational path-length (the length of the dependency graph of operations) will likely also not be extreme.

This situation is somewhat ameliorated by the fact that the computations are highly amenable to parallelism. A component manufacturer could organize a "path farm" to generate the program paths and to verify the correctness of each discovered path.

6.2 Domain of Applicability

The other problem is that currently the Probability Density Functions are range-based for scalar variables. This is not intrinsic to PDFs, but considerable research will be required to discover PDFs to support arrays, strings, and streams.

7. CONCLUSIONS

One of the most important properties of programs is their reliability or correctness. This will be, if anything, even more important when assembling components into assemblies. This paper outlines an approach to producing predictable reliability from such an assembly, at least in certain contexts.

8. REFERENCES

[1] R. K. Dybvig. *The Scheme Programming Language*. Prentice-Hall, Toronto, Ontario, 2nd edition, 1996.

[2] D. Hamlet, D. Mason, and D. Woit. Theory of software component reliability. In *Proc. 23rd International Conference on Software Engineering (ICSE'2001)*, Toronto, Canada, May 2001.

[3] D. Mason. Probability density functions in program analysis. In *4th ICSE Workshop on Component-Based Software Engineering (CBSE'2001)*, Toronto, Canada, May 2001.

[4] D. J. Richardson and L. A. Clarke. Partition analysis: A method combining testing and verification. *IEEE Transactions on Software Engineering*, 11(12):1477–1490, Dec. 1985.