Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Technology 6 (2012) 354 – 361

Procedia Technology

**2nd International Conference on Communication, Computing & Security [ICCCS-2012]**

# Evolution of Degree of Purity in Programming Languages

M RajasekharaBabu[a*],Anu Soosan Baby[b],Deepu Raveendran[c], Aswathi Joe[d]

[a]*VIT University, Vellore, Tamil Nadu, India*
[b]*VIT University, Vellore, Tamil Nadu,India*
[c]*VIT University, Vellore, Tamil Nadu, India*
[d]*VIT University, Vellore, Tamil Nadu, India*

**Abstract**

Pure method is a method, which does not have any side effects or the extent of the side effect is limited. Purity Analysis is the technique of determining pure methods. Purity information can be used to perform program analysis, specification, verification and optimization. The extent of side effects in a program can be expressed in terms of Degree of Purity. Degree of purity measures the side effects in different areas such as accessing of files, objects and variables, violation of programming constraints and possible exceptions. This paper proposes a model that measures the degree of purity of programming languages and implements it in scala. This allows us to determine concurrency in a program and execution in multi core machine.



Key words: Degree of Purity; Pure Methods, Semi-Pure Methods; Impure Methods; purity Analysis

## 1. Introduction

The methods which exhibit side effects free behaviour are known as pure methods [1]. Modification in the global state of a program is known as side effects of a method. Side effects occur when a function modifies global variables or by passing arguments by reference. The side effects in a program have negative effect on program comprehension. The development time of programmer can be saved efficiently using a program understanding tool which identifies and labels automatically the side effect free methods [2][3]. Knowing the side effect information in a program is used in various software tests, program analysis, verification, understanding, restructuring and optimization. The trade-off between increased speed and decreased maintenance results in need for purity checking [4]. Side effects occurs in various scenarios such as performing

2212-0173 © 2012 Elsevier Ltd...Selection and/or peer-review under responsibility of the Department of Computer Science & Engineering, National Institute of Technology Rourkela
doi:10.1016/j.protcy.2012.10.042

I/O, modifying global variables, modifying local permanent variables (like static variables in C), modifying an argument passed by reference, modifying a static variable of a function higher up in the function call sequence (usually via a pointer).

This paper proposes a model for purity checking which can automatically identify the number of pure methods and impure methods i.e. methods that have side effects. If it exists then calculate its degree of purity. It identifies number of independent program components that doesn't change the state of the program. The proposed method analyses the side effect free behaviour of a program to evaluate different purity definition that ranges from strong to weak by taking several purity forms that are specific to dynamic execution and to accommodate various constraints imposed by consumer application and memoization.

The function which does not perform any I/O, no global variables modification, no local permanent variables modification, no pass-by-reference parameters modification, no modification to nonlocal/static variables via pointers and any functions that called also satisfies these conditions is said to be pure [5][6] .By loosening the restrictions on pure methods we can give another annotation known as semi pure methods. The semi pure methods can be considered as a subtype of pure methods. Like pure methods semi pure methods also bound to restrictions and compilation rules. A pure method always overrides the semi pure methods. The semi pure method may cause side effect to a program under certain contexts.

The primary goal of this paper is to propose a model which measures the degree of purity in the language and also analyses the type of impurity in the language such as file access, object access, non-local variable access and exception. The type of impurity in turn determines the degree of purity. The degree of purity and purity of a method is inversely proportional. The method which has the above mentioned four types of impurity is treated as highly impure method.

The model also performs various class analysis to determine the class of all objects to which reference variable may point to and side effect analysis on memory location modified by execution of program statement. The proposed model observes the various kinds of side effects that a program can have including Observable states, immediate side effects, and transitive side effect. Any object name that is transitively reachable from variable is included in observable states. The invocation has immediate side effects if method assigns value to static field or method contains object allocation site that correspond to observable object name. Transitive side effects are caused due to indirect side effects that are due to called methods [2] [7].

Knowing which all methods have side effects has a variety of application in programming environment. This includes software maintenance and in Reverse engineering technology. Here the reverse engineer has to design from existing code. Purity representation is also essential in UML representation of software. In UML an attribute isQuery for operations which specifies whether the specified operation leaves the state of program or state of the system unchanged or whether any side effects has occurred. The default value of isQuery operation is false.

## 2. Literature Survey

The purity analysis for programming languages starts in 1970 and was mainly focused on FORTRAN language. The analysis used was the inter procedural data flow analysis which summarizes the semantic effect associated with sub routine call. In the analysis each program statement was annotated with MOD and REF [8].

Banning represented the MOD problem as data flow problem on programmers call multigraph which can be solved by global data flow analysis. Inorder to perform the data flow analysis Banning decomposed the problem into Alias analysis and side effect computation. Initially Banning computed GMOD (p) for every procedure p representing the generalized mod of p. Then he computed the DMOD(s) for every statement

representing the direct mod of S. GMOD is applied to procedures or functions whereas DMOD is applied to set of statements. MOD is computed using combination of above two sets.

Cooper and Kennedy modified the flow insensitive summary of information by dividing the problem into two sub problems: one for computation of global parameters and another for call by reference formal parameters. They use binding multigraph as data structure to achieve linear time complexity. Burke and Cytron proposed inter procedural side effect analysis. The current inter procedural side effect analysis assumes that when an array element was modified by a procedure call, the entire array could be modified. Studies were conducted to compute side effect of method calls on subscripted reference by implementing inter procedural data flow analysis, and to integrate subscript analysis with aliasing [7].

Ryder and Caroll put forward an incremental interval algorithm for MOD analysis for computing updated side effects to change in system instead of recalculating in its entirety. Pointer analysis is used to determine efficiently the side effect analysis of a program. Salcianu and Rinard presented a combined pointer and escape analysis and generate regular expression to characterize externally mutated heap locations [9].

Haskell and D are the two programming languages that verify the purity at compile time. To use side effects or externally visible variables in Haskell we should explicitly indicate it by using monads in your function signature otherwise we can't. But D allows using side effects and variables in your functions by default and if you need you can explicitly mark them as pure [10].

Razafimahefa implemented an inter procedural side effect analysis for Java based on point to point analysis. This algorithm derives from Sternwards points to information on side effect information. They develop object sensitive points to and side effect analysis, demonstrating the impact of program information can have on side effect data. Dallmeier et al have examined dynamic purity analysis for Java programs which uses ASM byte code manipulation framework to create program traces and to identify impure methods. It also provides a means to compare static and dynamic purity information [11].

Artziet.al. examined "the dynamic analysis of parameter mutability for java programs". JPure is another system for purity analysis in java. This system uses three annotations @pure, @local and @fresh. Analyzing purity using individual methods will not lead to precise purity analysis. It gives performance in precise. When combining these annotations it can perform a precise purity analysis. This algorithm produces annotations without the need for an expensive and costly interprocedural analysis [9].

## 3. Design

This approach for pure method analysis uses a probabilistic model for classifying methods based on their side effects. In general, the methods are classified into Pure, Semi-Pure and Impure methods based on their degree of purity. For determining the degree of purity four parameters are used with predefined priority based on its impact in the program. The 4 parameters with pre-defined priority value are:-

Table 1Predefined Priority Values for various parameters in program

| Parameter | Priority | Reason |
|---|---|---|
| Non-local variable | 4 | Highly affect global state (there will be a definite change in state of a program). |
| Object Reference | 3 | State of the program changes depending on the type of object reference. |
| File Access | 2 | Program state changes depending on type of file access and type of file operations. |
| Exception | 1 | Small variation occurs in the program state since there is an error in the program. |

In this paper, it uses a Bayesian probabilistic model to calculate the degree of purity. The model first determine the important parameters in methods that cause side effects such as number of non-local variable

access, object reference, file access and exceptions occurring in the program. These parameter values are multiplied with predefined priority value. The value thus obtained will give contribution of side effect due to that particular parameter. The model determines the overall contribution of side effects by each parameter.

The model calculates the dependency existing between methods and if dependency exists it adds the total contribution of called method to the total contribution of calling method in the dependency graph. The value so obtained will be called degree of purity.

Table 2 Number of various side effect parameters in various methods of the program.

| Function | Non-local | Object reference | File Access | Exception | Total Contribution |
|---|---|---|---|---|---|
| F1 | A | Φ | B | Φ | 4A+2B |
| F2 | Φ | Φ | Φ | Φ | 0 |
| F3 | Φ | Φ | Φ | Φ | 0 |

In the table a, b, c represents constants indicating number of Non-local variable modifications, number of File access, and number of Exceptions respectively. Φ stands for 0 or null.

Next, the proposed model classifies the functions into Pure, Semi-Pure and Impure methods based on their value of degree of purity. For that it calculates the mean of degree of purity of all the methods and assigns that value as the threshold. Semi-pure methods. The proposed model uses this information for side-effect analysis in programming languages.

Step 1: Identify parameters of a method that determines the degree of purity.
Step 2: Assign value to each parameter by following equation:
Pre-defined priority of parameter * Number of modifications of that Parameter.
Step 3: Calculate the sum of value of individual Parameters
Step 4: By dependency analysis add the base method purity value to child class method.
This will be the degree of purity of method.
Step 5: Calculate the threshold value by taking the mean of degree of purity of all functions.
Step 6: Classify methods based on following rule:
If degree of purity=0 then the method is pure. If degree of purity ≤ mean, it is Semi-Pure.
Else the method is Impure

Fig. 1. Algorithm for Purity Analysis

Our proposed method uses Bayesian probabilistic model can be used to predict the class of a newly user defined function based on the obtained value. Using this model we can classify the newly user defined function into Pure, Semi-pure or Impure by finding its probability. Through Bayesian classification we find out the maximum livelihood estimate of probabilities. In Bayesian model we segment the side effect parameter by the class (Pure, Semi-pure or Impure). Since the parameter values are continuous data, a typical assumption is that the continuous values associated with each class are distributed according to a Gaussian distribution.

According to Bayesian Classification, if the new function contains a continuous parameter value x, we will first segment data by its class and then computes mean and variance of x in each class. Let $\mu_c$ be mean of values in x associated with class c, and let $\sigma_c^2$ be the variance of values in x associated with class c.

Then the probability $P(x = v \mid c)$, that is the probability that x belongs to class c can be computed by plugging v into the equation for a Normal Distribution parameterized by $\mu_c$ and $\sigma_c^2$. That is,

$$P(x = v|c) = \frac{1}{\sqrt{2\pi\sigma_c^2}} e^{\frac{-(v-\mu_c)^2}{2\sigma_c^2}} \quad (1)$$

We assign class to each function by finding the probability value for parameter values in each class and multiplying the obtained probability value with highest probability value gives the class of the function.

Let p denotes Non-local variable, q denotes Object Reference, r denotes File Access, s denotes Exception in the current method then the degree of purity of current method can be calculated from the following formula:

$$D = 4p + 3q + 2r + s + c \quad (2)$$

Where, C is the total contribution of degree of purity from dependent parent classes and D is the degree of purity of current method.

If p = q = r = s = x, then the degree of purity will be

$$D = 10 * x + c \quad (3)$$

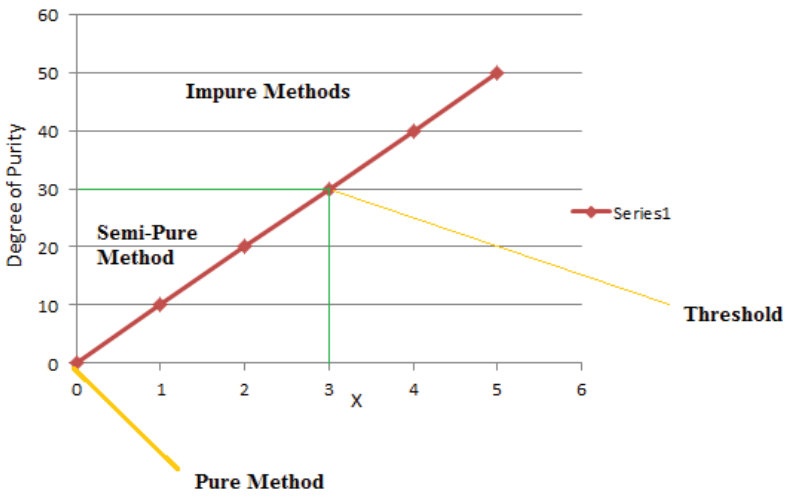


Fig. 2 Classification of methods based on degree of purity

### *3.1 Mathematical Model for Degree of Purity*

**Lemma 1:** Let D (x) is the degree of purity, where x = p = q = r = s then D (x) can be represented as,
D (x) = 10 * x + c
For an independent method C=0, hence the degree of purity will be D (x) = 10*x.
**Proof:**
Mathematical induction can be used to prove the above formula for all values of x, such that threshold
**Basis Step:** Show that the statement holds for x=1.
Consider x = 1, then D (1) = 10
**Inductive Step:**
Assume that the statement is true for x=k.
D ( k) = 10 * k
Then we have to show that, the equation holds for
X = k + 1.
D (k+1) = 10 * k+10
Here by shows that D (k+1) also holds good.

## 4. Implementation

In this paper, the implementation is done in scala version 2.8.1. The main tool used in implementation is Netbeans IDE version 7.0 using scala plugin. Implementation is done in two parts Comparative Analysis (4.1) and Benchmarking (4.2). In Comparative Analysis it classifies methods into various classes according to the degree of purity by taking into consider the various parameter that affects the degree of purity. In Benchmarking it analyzes the performance of purity analyzer by using time and memory as constraints.

### *4.1. Comparative Analysis*

The purity analysis of function Fast Fourier Transform, and Inverse Fast Fourier Transform is given below in the table:

Table 3 Analysis of FFT

| | |
|---|---|
| Non-local Reference | Nil |
| Object Reference | Nil |
| File Access | 3 |
| Exception | 1 |
| **Degree of Purity** | **7** |
| **Class** | **Semi-Pure** |

Function Name: **Fast Fourier Transform (FFT)**

Table 4 Analysis of IFFT

| | |
|---|---|
| Non-local Reference | Nil |
| Object Reference | Nil |

| | |
|---|---|
| File Access | 3 |
| Exception | 1 |
| **Degree of Purity** | **7** |
| **Class** | **Semi-Pure** |

Function Name: **Inverse Fast Fourier Transform (IFFT)**

The distribution of Parameters according to degree of purity is given as follows:

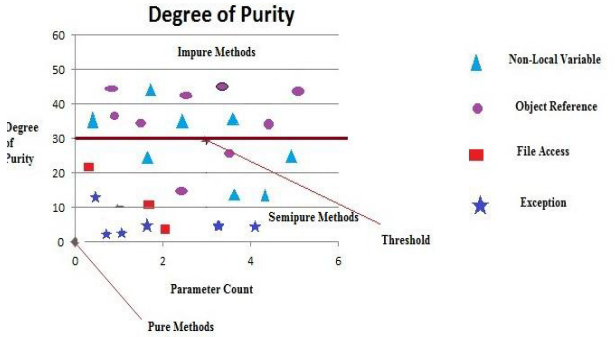


Fig. 3. Degree of Purity Vs. Parameter Count

Pure methods are those methods that have zero parameter count. For semi-pure method there will be more number of file access and exception and very little number of Non-local variable and Object reference. For impure method there will be large number of Non-local variables and Object references.

### *4.2. Benchmarking*

For benchmarking the purity of program we use two parameters:-time and memory.

#### *4.2.1. Time*

The benchmarking based on time is performed on a Pentium 4 @2.8 GHz with 4 GB RAM on scala version 2.8.1 and the analysis time ranges from 4.2-9 seconds. For each analysis, the method, the parent methods, that is invoked transitively and library method used both in child and parent have been analyzed, its corresponding degree of purity have been computed and the method is classified into corresponding class based on its degree of purity. The data structure used for the analysis is the dependency graph for finding dependency between methods.

#### *4.2.2. Memory*

This benchmarking parameter analyzes various memory locations, its scope, type operation etc. Scope of memory includes whether the variable is local or global in the method, the type specifies the type of memory where the variable resides that is cache memory, main-memory (primary memory), secondary memory, and

operation on memory includes reading and writing variables to and from memory. Benchmarking of this parameter is performed on a Pentium 4 machine operating at 2.8 GHz with 4 GB RAM on scala version 2.8.1. The analysis on memory location tells that the impure methods always lead to cache misses which thereby increases memory access time.

## 5. Conclusion and Future Work

Our purity analysis considers various parameters that affect the purity of a program such as Non-local reference, Object reference, File Access, and Exception. We classify our method based on degree of purity into pure, semi pure and impure. Our dynamic purity analysis based on probabilistic approach explores various purity parameters and various benchmarking used for analyzing the purity of program. We aim to continue exploring purity notation for other parameters that affects the purity. Also considers the purity at finer granularities such as loops, basic blocks and individual instructions. Also performs dynamic purity analysis on per input basis which analyses the inputs and identify the bounds on inputs that violate the purity state of program. In future, various methods for handling impurities and avoiding local impurities and exceptions will also be used.

## References

[1] H. Xu, C.J.F. Pickett and C. Verbrugge, “Dynamic purity analysis for java programms,” in Proc. PASTE, ACM, 2007, pp. 75-82

[2] A. Rountev, “Precise identification of side-effect-free methods in java, “in Proc. ICSM. IEEE Computer Society, 2004, pp. 82–91.

[3]Matthew Finifter, Adrian Mettler, Naveen Sastry and David Wagnor “Verifiable Functional Purity in Java”, ACM-2008, Alexandria, Virginia, USA.

[4]J.J. Donaldo, M.harman, M.C Otero “An Empirical Investigation of the Influence of a Type of Side Effects on Program Comprehension”. IEEE Transactions on Software Engineering, Vol.29, No.7, , pp. 665- 670, July 2003

[5]David A Spuler and A.S.M Sajeev, Technical Report on “Compiler Detection of Function Call Side Effects”, James Cook University, Australia, January 1994

[6]Atanas Rountev and Babra G Ryder, “Points to and Side-effect Analysis for Programs Built with Precompiled Libraries”, Department of Computer Science Rutgers University, New Jersey, USA,2004

[7]Anatole Le, O Lhotak, and L J Hendren, “Using inter-procedural side effects information in JIT optimizations” in Proc, CC. Springer 2005, pp. 287-304.

[8]J.D. Choi, M. Burke, and P. Carini,“Efficent flow sensitive interprocedural computation of pointer induced aliases and side effects,” in Proc. POPL, ACM, 1993, pp 232-245.

[9] David J Pearce, “JPure: A modular purity system for Java”, ACM Press 2009.

[10] “Side effect tracking in Huskell and D”, a Blog post on Lambda the Ultimate programming languages weblog, www. http://lambda-the-ultimate.org

[11] Oksana Tkachuk, A thesis on “Adapting Side Effect Analysis for Modular Program Model Checking”, University of Nevada, 1998.