TEST ORACLES
Formal Definitions and Classifications

Shin Yoo & Mark Harman (UCL)
Muzammil Shahbaz & Phil McMinn (University of Sheffield)
OVERVIEW

• Formal Definitions

• Oracle Literature Timeline & Classification
FORMAL DEFINITIONS

• Few attempts to form an universal framework

• Recent work on formalising testing process (Staats et al. 2011) but with different focus
Of course, one might hope that the software–under–test has been developed with respect to excellent design–for–test principles, so that there might be a detailed, and possibly formal, specification of intended behaviour. One might also hope that the code itself contains pre– and post– conditions that implement well-understood contract–driven development approaches [44]. In these situations, the Oracle Cost problem is ameliorated by the presence of an automatable oracle to which a testing tool can refer to check outputs, free from the need for costly human intervention.

Where there is no full specification of the properties of the system under test, one may hope that it is possible to construct some form of partial oracle, that is able to answer oracle questions for some inputs, relying on other means to answer the oracle question for others. In the literature, such partial oracles are often achieved using a form of metamorphic testing in which testing seeks to exercise known relationships between input and behaviour, where these can be established.

However, for many systems and much of testing as currently practiced in industry, the tester has the luxury of neither formal specification nor assertions, nor even automated partial oracles. The tester must therefore face the potentially daunting task of manually checking the system's behaviour for all test cases generated. In such cases, it is essential that automated software testing approaches address the Human Oracle Cost problem [31, 43].

In order to achieve greater test automation and wider uptake of automated testing, we therefore need a concerted effort to find ways to address the oracle problem and to integrate automated and partially automated oracle solutions into test data generation techniques. This paper seeks to help address this challenge by providing a comprehensive review and analysis of the existing literature on the oracle problem. Hitherto, there has been no such review; research into the oracle problem is still undertaken in a fragmented community of researchers and practitioners. This paper seeks to overcome this by providing the first comprehensive analysis and review of work on the oracle problem on software testing.

2 Definitions

This section sets out some foundational definitions to establish a common lingua franca in which to examine the literature on Oracles. These definitions are formalised to avoid ambiguity, but the reader should find that it is also possible to read the paper using only the informal descriptions that accompany these formal definitions.

Definition 1 (Alphabet) The input to the program under test will be considered to be drawn from a set $I$, while the output will be drawn from a set $O$. 

Definition 1 (Alphabet) The input to the program under test will be considered to be drawn from a set $I$, while the output will be drawn from a set $O$. 


Definition 2 (Test Case) A test case is a pair \((i, o)\) in \(I \times 2^O\) such that \(o\) is non-empty and is singleton in the case that the system, is deterministic for input \(i\).

Definition 3 (Test Instance) A test instance is an element of the set \(I \times O\)
• Define oracle as a set of test cases that establishes acceptable behavioural relationship between input and output

\[ I \times O \rightarrow \{0, 1\} \]
• We want to cater for probabilistic decision on acceptance

\[ I \times O \rightarrow [0, 1] \]
INEXACTNESS

Algorithmic Methodologies for Ultra-efficient Inexact Architectures for Sustaining Technology Scaling
Lingamneni et al., ACM Computing Frontiers 2012

Exact  0.54%  7.58%
• We want to cater for metamorphic relations

• Acceptable behaviour is defined as a relation to other test instances

\[ I \times O \times 2^{I \times O} \rightarrow [0, 1] \]
If certain relation holds between two inputs, you expect a specific relation to hold between corresponding outputs.

Example: 
\[ x' = \pi - x \rightarrow \sin x' = \sin x \]

Traditional examples focus on two instances, but it can be generalised to \( n \) instances.

Example: linearity between input/output requires 3 instances.
• We want to cater for metamorphic relations

• Acceptable behaviour is defined as a relation to other test instances

$$I \times O \times 2^{I\times O} \rightarrow [0, 1]$$
• We want to cater for inferred specification/regression suites

• Acceptable behaviour is defined as a relation to other test instances

\[ I \times O \times 2^{I \times O} \rightarrow [0, 1] \]
Definition 4 (Definite Oracle) A *Definite Oracle* is a function from test instances to \( \{0, 1\} \). That is, a definite oracle is an element of the set \( I \times O \times 2^O \rightarrow \{0, 1\} \).

Definition 5 (Probabilistic Oracle) A *Probabilistic Oracle* is a function from test instances to \([0, 1]\). That is, a probabilistic oracle is an element of the set \( I \times O \times 2^{I \times O} \rightarrow [0, 1]\).
Definition 6 (Completeness)  An Oracle is complete if it is a total function.
Definition 7 (Ground Truth) The Ground Truth, \( \mathcal{G} \) is a definite oracle.

We can now define soundness of an oracle with respect to the Ground Truth, \( \mathcal{G} \).

Definition 8 (Soundness) A Probabilistic Oracle, \( PO \) is sound iff

\[
PO(i, o) \in \begin{cases} 
[0, 0.5) & \text{when } \mathcal{G}(i, o) = 0 \\
(0.5, 1] & \text{when } \mathcal{G}(i, o) = 1
\end{cases}
\]
Definition 9 (Correctness) An oracle is partially correct iff it is sound. An oracle is totally correct iff it is sound and complete.
Classification of Oracles in the literature review

• **Origin:**
  - “test oracle” coined by W. E. Howden (1978)
  - “a program specification, table of examples, or the programmer’s knowledge on how the program should operate”
    (Howden and Eichhorst, Tutorial: Software Testing and Validation Techniques, 1978)

• **Classification:**
  - Specified Oracles
  - Derived Oracles
  - Implicit Oracles
  - No Oracles
Specified Oracles

oracles that are formally specified

- specification based
- algebraic specification
- assertions / design-by-contracts

model based (B, Z, VDM etc.)
transition based (FSM, UML, Statecharts etc.)

languages / formalisms

- model checking
- specification based
- design-by-contracts
- temporal logic
- I/O CO theory
- object-Z
- H Statecharts
- ASML
- Alloy
- RESOLVE
- JML
- OCL
- LE TO

Derived Oracles

oracles that can be derived from the given artefacts

- system executions (traces, log files, invariants, ...)
- documentations (source code, comments, APIs, specs, ...)
- existing knowledge (partial / pseudo oracles, regression, ...)


Pseudo-Oracle  Partial Oracle  Inference  N-versions  Semi-formal docs  Log File Analysis  Regression Testing  Metamorphic Tests  Code Comments  API Docs  Mutant based
Implicit Oracles

oracles which do not require specification

- anomalies (deadlock, livelock)
- errors (divide-by-zero, memory leaks, ...)
- exceptions (crash, ...)

Deadlock/Livelock
Model Checking
Specific problems
JCrasher
Anomaly detection
No Oracles

no way of automatic validation!!!
Testing involves examining the behaviour of a system in order to discover potential faults. The problem of determining the desired correct behaviour for a given input is called the **Oracle Problem**. Since manual testing is expensive and time-consuming there has been a great deal of work on automation and part automation of Software Testing. Unfortunately, it is often impossible to fully automate the process of determining whether the system behaves correctly. This must be performed by a human, and the cost of the effort expended is referred to as the **Human Oracle Cost**.

**RE-COST will develop Search-Based Optimisation techniques to attack the Human Oracle Cost problem quantitatively and qualitatively.**

The RE-COST project seeks to transform the way that researchers and practitioners think about the problem of Software Test Data Generation. This has the potential to provide a breakthrough in Software Testing, dramatically increasing real-world industrial uptake of automated techniques for Software Test Data Generation.