Verification and Validation of Bioinformatics Software

Joshua W.K. Ho, joshua@it.usyd.edu.au

Supervisor: Dr. Michael A. Charleston
School of Information Technologies and NICTA

1. Software verification and validation

- **Software verification:** To check that the algorithm is correctly implemented in the source code. That is, *did we build the software right?*
- **Software validation:** To check that software performs what it is intended to perform. That is, *did we build the right software?*

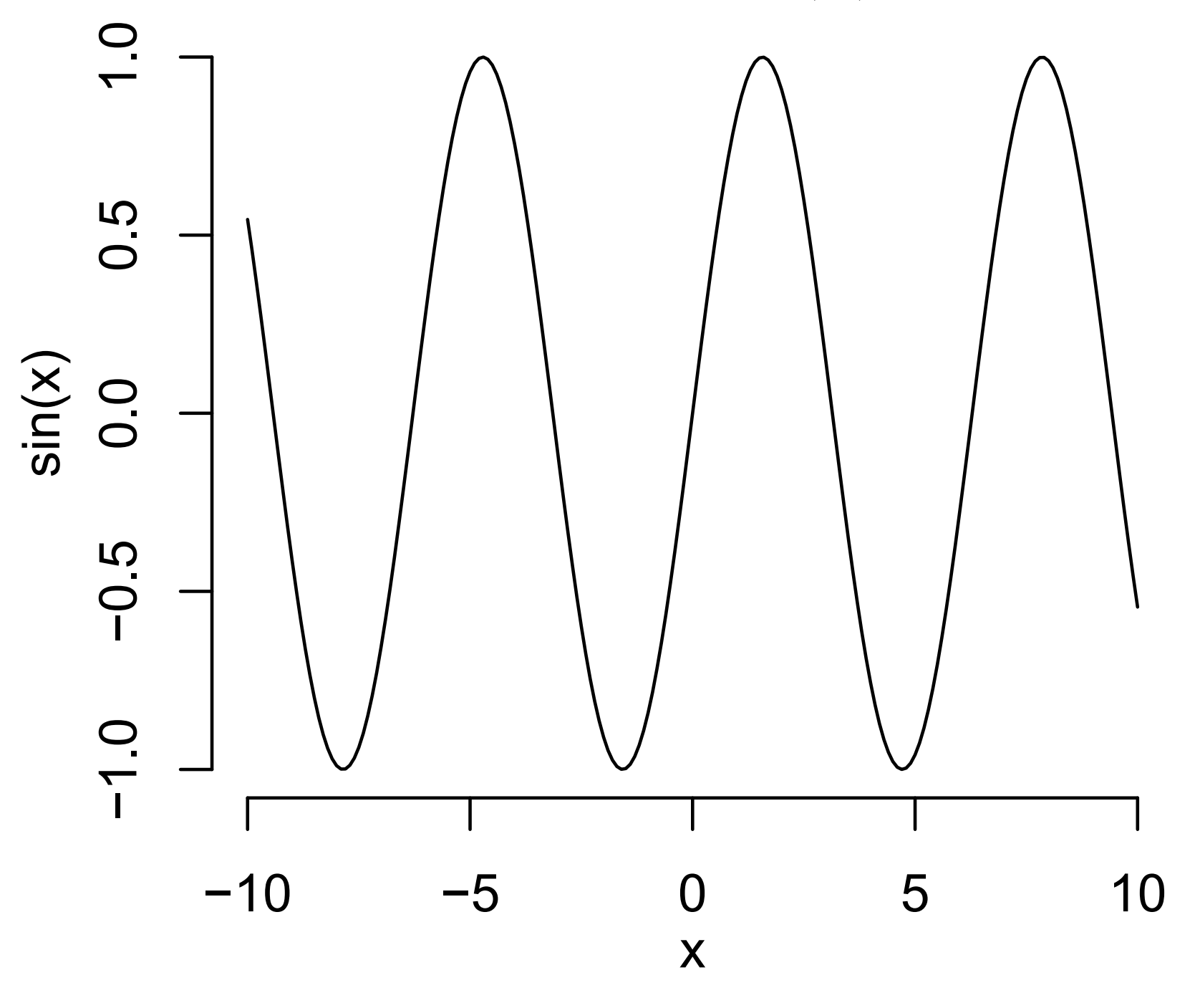
Functional software testing is the most common approach to perform software verification and validation.

2. How to test `sin(x)` = $sin(x)$?

Aim: verify the correctness of a computer program, `sin(x)`, that computes the mathematical function $sin(x)$ given any real value x.

Attempt 1: special value testing. Simply check certain special values of x, such as $x = 0, \frac{\pi}{6}, \frac{\pi}{4}, \frac{\pi}{2}...$ where we expect to get $sin(x) = 0, \frac{1}{2}, \frac{1}{\sqrt{2}}, 1$ respectively. Problem: only a very small portion of all possible x in the input domain is tested

Attempt 2: plot the graph for $sin(x)$.



Problem: only a qualitative verification, not quantitative

Attempt 3: compare the results with multiple programs that implement $sin(x)$. Problem: what happens when an inconsistency is detected?

Attempt 4: compare the results with a reference sine function table Problem: Not scalable to test the large range of legal x values.

Since it is hard to check the correctness of an arbitrary output of `sin(x)`, this program is said to lack a tangible *oracle*. In other words, testing of this program suffers from the *oracle problem*.

3. Bioinformatics programs are hard to test

Modern bioinformatics software is characterized by two features:

1. **Data intensive.** *e.g.,* microarray data, and DNA sequencing data which can be as large as 1 TB in size.
2. **Sophisticated algorithmic procedures.** *e.g.,* machine learning algorithms, combinatorial optimization algorithm or heuristics, approximation algorithms, simulation algorithms, and randomized algorithms.

Many bioinformatics programs naturally suffer from this **oracle problem**: programs for biological network simulation, sequence alignment, phylogenetic tree reconstruction, and microarray analysis.

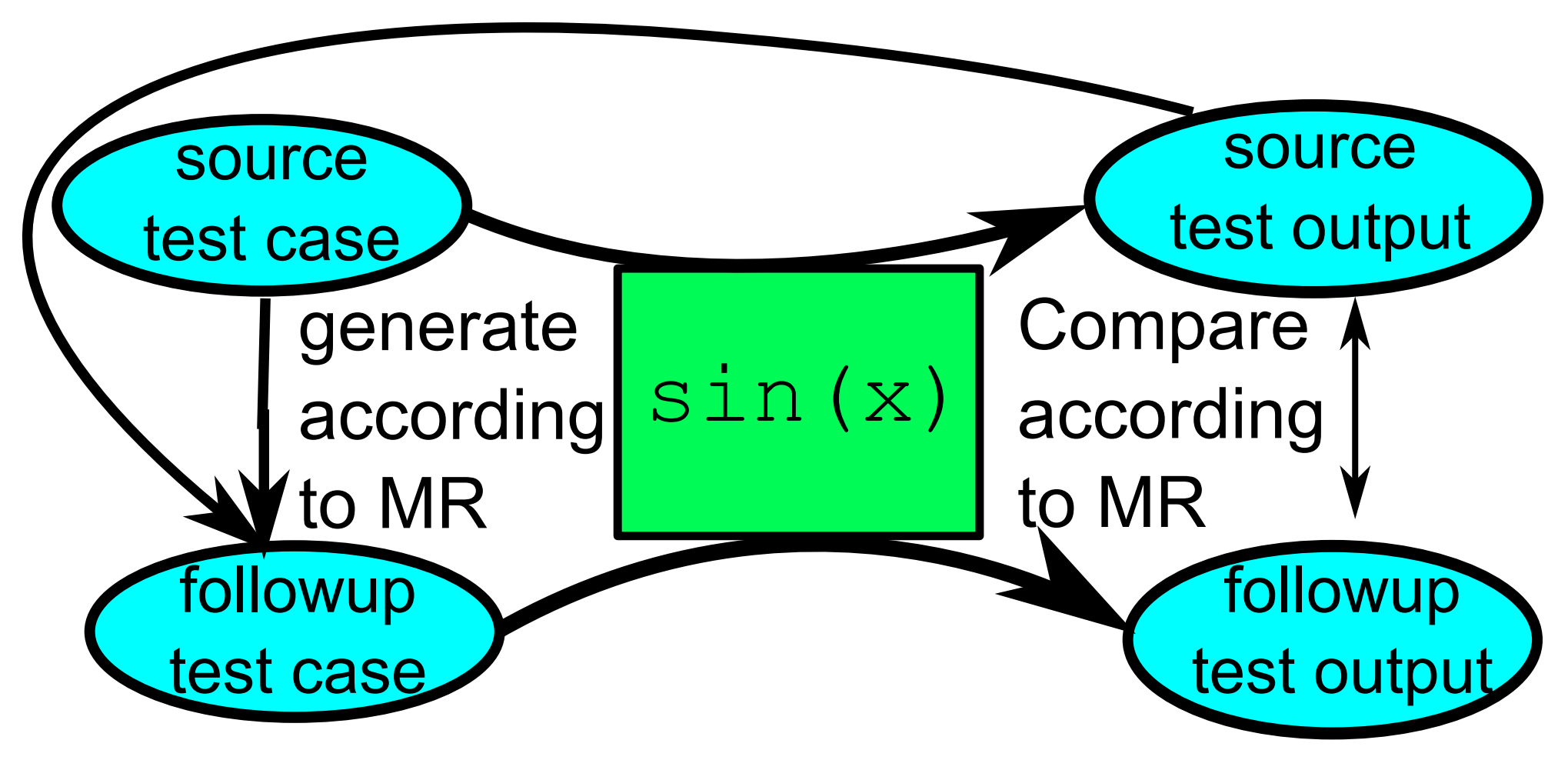
4. Project aim

Apply a recently invented software testing strategy, called **Metamorphic Testing**, to verify and/or validate a diverse range of bioinformatics programs that are traditionally very difficult to test.

5. Metamorphic testing

Chen *et. al.* [2] developed a strategy called **Metamorphic Testing (MT)** to enable generation and verification of large numbers of diverse test cases for software that suffer from the oracle problem. The process of MT is:

1. **Identify a list of Metamorphic Relations (MRs)** that the program should hold with respect to the input data. For example, possible MRs for the `sin(x)` program include
   - $sin(x) = sin(x + 2\pi)$
   - $sin(x) = -sin(-x)$
   - $sin(x) = -sin(x + \pi)$
   - $sin^2(x) + sin^2(\frac{\pi}{2} - x) = 1$
2. **Given a source test case, generate the follow-up test cases and verify outputs based on the MRs.**



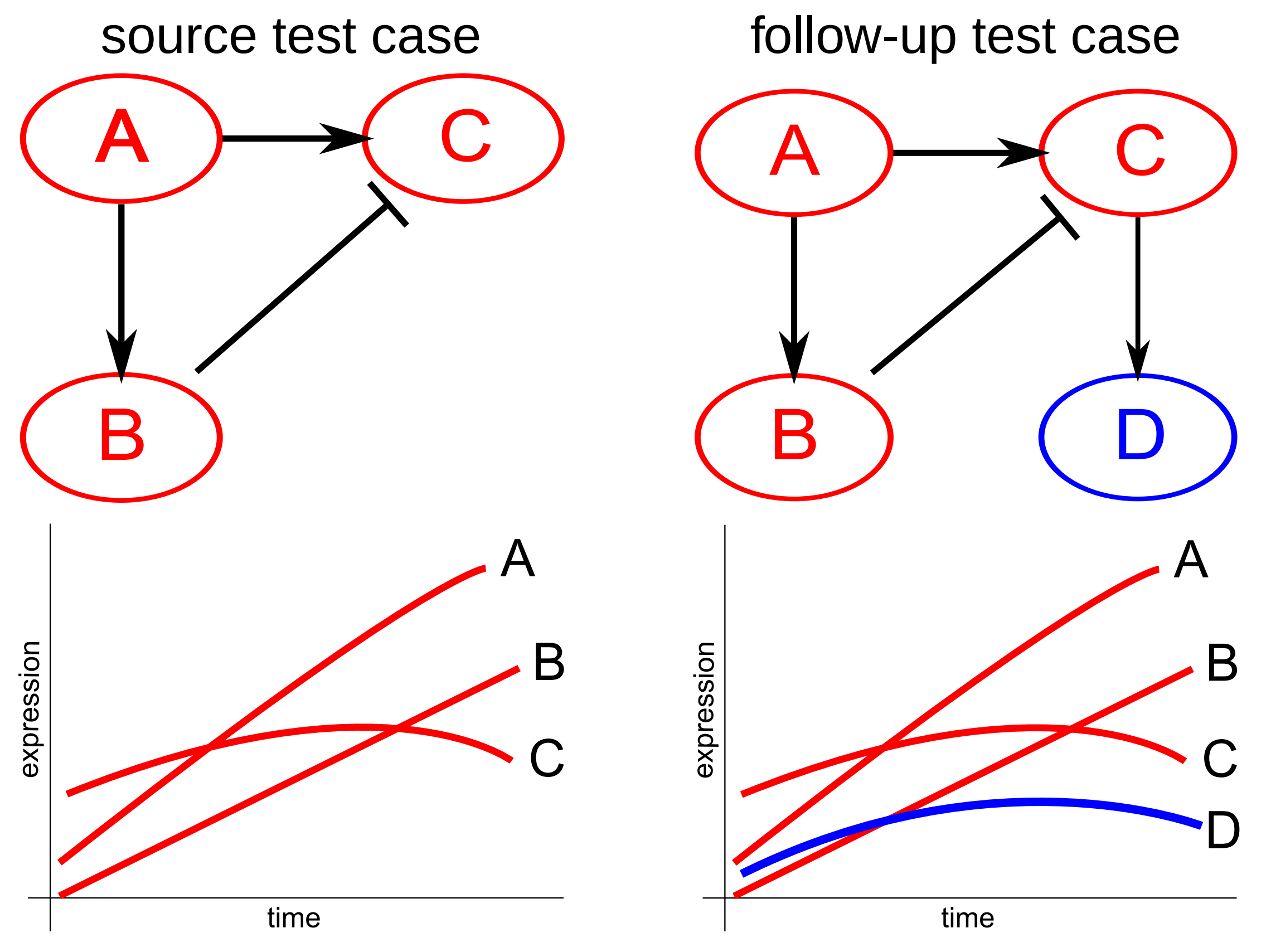
MT alleviates the oracle problem by testing necessary properties of the program, instead of testing for the correctness of specific test cases. It enables a diverse range of test cases to be generated and verified.

1. If MRs are derived from the specification, MT performs verification.
2. If MRs are derived from the intended program behaviours, MT performs validation.

6. Testing a network simulator

Testing the correctness of a biological network simulator is a very challenging topic in bioinformatics [1], as there is no simple way to check the simulation result of any non-trivial networks. We tackle this problem by using metamorphic testing [3].

We can define MRs related to addition / removal / modification of edges and nodes in the network and its expected behaviour. The following example demonstrates the MR: "After adding a non-regulator node in the followup test case, its simulated expression values of the network should be identical to those of the source test case except those for the newly added node." An example run of MT using this MR is shown below (addition of non-regulator **D**).



Many other MRs can be defined, and can collectively be used to test the correctness of the program.

7. Testing a supervised classifier

Supervised classification methods (or machine learning methods in general) are commonly used in bioinformatics programs, such as those for microarray analysis and gene finding. Supervised classifiers are hard to test. Given a trained classifier, it is difficult to determine if an incorrect prediction is due to the limitation of the algorithm or a bug in the program. We have successfully applied MT to test two popular supervised classifiers: naïve Bayes classifier and k-Nearest Neighbour classifier [5]. Two example MRs are as follows:

- **MR1: Consistency upon affine transformation:** The prediction results should remain the same if we apply the same arbitrary affine (*i.e.*, linear) transformation function to the training and testing data.
- **MR2: Permutation of the order of the attribute:** The prediction should be identical regardless of the order of the attribute presented in the training and the test data.

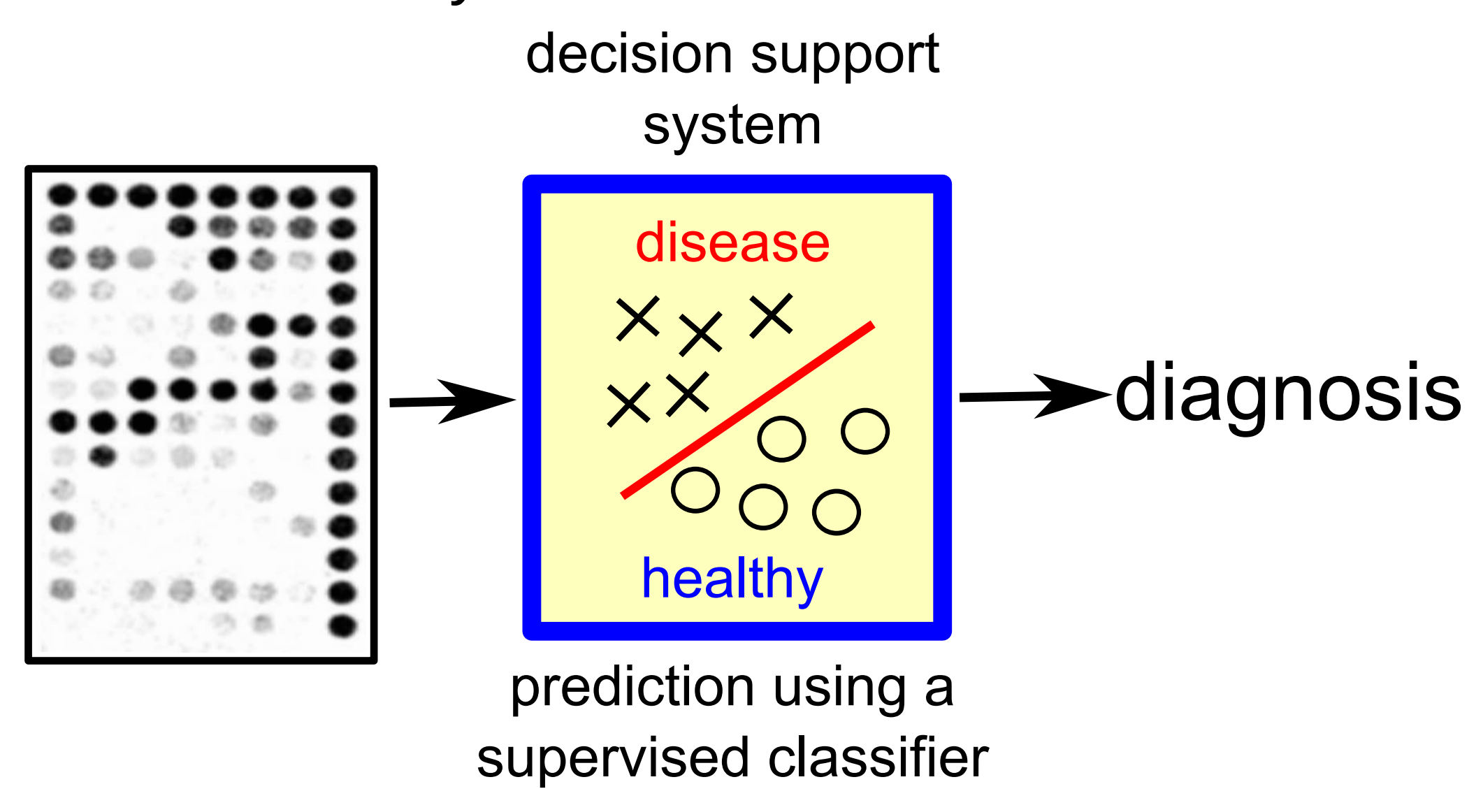
We found that MT can indeed identify some errors, including one error due to the problem of numerical underflow in our target program, which has been undetected in the original package.

8. Implication in scientific discovery

Incorrect computation may lead to incorrect biological conclusions. This is particularly relevant in systems biology research where simulation results are used to guide downstream experiments. Therefore more rigorous software testing is needed to ensure the correctness of these programs.

9. Implication in medical diagnostics

It is increasingly clear that data generated by high-throughput genomic and proteomic arrays (*e.g.,* an antibody microarray) using an individual's blood sample can yield diagnostic and prognostic information [4]. Clearly decision support systems for clinical diagnostics need to be highly reliable, so rigorous verification and validation are necessary.



Acknowledgements

This work is performed in collaboration with Prof. T.Y. Chen from the Centre for Software Analysis and Testing in the Swinburne University of Technology. JWKH is supported by an Australian Postgraduate Award and a NICTA Research Project Award.

References

[1] FT Bergmann and HM Sauro. Comparing simulation results of SBML capable simulators. *Bioinformatics*, 24:1963–1965, 2008.

[2] TY Chen, SC Cheung, and SM Yiu. Metamoorphic testing: a new approach for generating next test cases. Technical report, Tech Rep HKUST-CS98-01. Department of Computer Science, Hong Kong University of Science and Technology, Hong Kong, 1998.

[3] TY Chen, JWK Ho, H Liu, and X Xie. An innovative approach for testing bioinformatics programs using metamorphic testing. *BMC Bioinformatics*, 10:24, 2009.

[4] JWK Ho, MW Lin, S Adelstein, and CG dos Remedios. Customising an antibody leukocyte capture microarray for Systemic Lupus Erythematosus: Beyond biomarker discovery. *Proteomics Clin. Appl.*, (in press), 2009.

[5] X Xie, J Ho, C Murphy, G Kaiser, B Xu, and TY Chen. Application of metamorphic testing to supervised classifiers. In *In Proceedings to International Conference on Quality Software*, 2009 [Winner of the Best Paper Award].

The University of Sydney

NICTA