

# Evolutionary Structural Testing of Software with Pointers

Maria Prutkina  
Daimler AG

Group Research & Advanced Engineering  
D-71059 Sindelfingen, Germany  
maria.prutkina@daimler.com

Andreas Windisch

Technische Universität Berlin  
Daimler Center for Automotive IT Innovations  
D-10587 Berlin, Germany  
tu-berlin.windisch@daimler.com

**Motivation** Evolutionary structural testing is an approach for the complete automation of the test data generation process which searches test data providing high structural coverage by means of evolutionary algorithms (EA). Pointer usage complicates the test data generation considerably. An approach for the improvement of evolutionary structural testing is proposed in this paper, allowing effective handling of source code with pointers.

**Generation of Test Data for Pointers** One approach to initialise a given pointer would be to generate an auxiliary variable of the same type and to initialize the pointer with the address of this variable or with nil.

Another opportunity is to consider the address of a newly created variable and the addresses of already existing variables of the same type as possible candidates for the initialization of the pointer. It is expedient if there are only few suitable data assignments available for the variables of the certain data type. By forming a list of the possible candidates need to be distinguished if a pointer or a referred value is constant or not. In order to decide, which candidate is favored, a new variable, a selector, is added. Its value is also optimized by evolutionary algorithms.

If the pointer is initialized with the address of the new generated auxiliary variable the problem of generating test data for a pointer is traced back to the problem of generating test data for a referred variable. For simple data types determination of test data takes place for the newly created auxiliary variable by means of EA [1]. For pointer structures pointer fields of one other type are viewed as pointers to simple data types and are initialised with new generated auxiliary variables. If the auxiliary variable again contains pointers, the sequence is repeated.

Static analysis establishes that a structure contains pointers to the same type. The user selects whether this structure is a list, a tree or a "normal structure". For the latter pointer fields to the same type are initialized with nil. For dynamically linked data structures not only data fields but also the number of necessary nodes is to be defined and a linking

corresponding to the structure should be maintained. During the test object execution the number of input variables is unchangeable. Thus, the maximal length of the list or the maximal depth of the tree should be limited.

The user defines **lists** through its boundary element, specifies a pointer field of the same type as a pointer to nil, an external element of the same type, the next or the previous node. He states whether a list is circular and defines the minimum and maximum length. List length is optimized using EA and takes values between the minimum and the maximum stated length. A list is "built" from its specification; consists of a boundary node and an array of the length ( $max. length - 1$ ). List length, data fields of the node and the array are "read" and linked into a list of the actual length.

A **tree** is declared and defined by the root node. The user specifies the pointer fields to the same type as pointers to nil, an external element, the tree root, the parent, the child, the next or the previous sibling node. He states the minimum and maximum tree depth. To generate a tree two auxiliary variables are added and optimized by EA. One is the tree depth, taking values between the minimum and maximum depth. The second is an array with elements of the tree except the root. Array length is calculated from the number of children and the maximum depth. Tree depth, array and root node are "read" and the linking is set. Since often several nodes are never accessed, no complete tree is needed. To abandon such nodes a auxiliary variable for each child node is added, deciding if this child should be included in the tree and is optimized using EA.

**Conclusions** The suggested approach has been implemented. It was thus possible to prove its applicability for test data generation for software with pointers.

## References

- [1] J. Wegener, A. Baresel, and H. Sthamer. Evolutionary test environment for automatic structural testing. *Information and Software Technology*, 43:841–845, 2001.