Implementing CppInterOp API, Exposing Memory Ownership and Thread Safety

Compiler C|R Research

Google Summer of Code 2026
Midterm Presentation

Mentors: Vassil Vasilev, Aaron Jomy, Vipul Cariappa

Author: Kerem Şahin

Goal of First Term

Compiler C|R Research

- Extracting information about memory ownership semantics from function bodies, detecting allocator-deallocator pairs, and using this information to automate memory management of programs.

General Algorithm

Compiler C|R Research

- Every time API function is called to analyze a function, a traverser instance is created. Traverser instance is consist of two main variable member:
- Variable Map: It is a map used to store whether the variable points to a heap allocated memory, and memory is allocated by which operator/function
- Result: A variable keeps track of returned expressions, and detect possible contradictions between different return expressions

General Algorithm

Compiler C|R Research

- When a VarDecl is seen: This node represents variable declarations. If there is an initialization, it is evaluated to see whether it is an another variable which points heap allocated memory, or it is new/new[]/malloc call. Then variable and it is AllocType value is saved to the varMap
- When a BinaryOperator is seen: This node represents operands that take two arguments and we are interested in '=' operator where value of variable changes. New RHS is evaluated and saved to the varMap
- When a ReturnStmt is seen: This node represents return …; statements. The return expression is evaluated and resolved as an AllocType value, and handled according to the value of ***Result.*** If there is a contradiction between two ReturnStmt, it is handled at this moment.

How If-Else Branches are Handled

Compiler C|R Research

```
int* func(bool b){
    int* ptr = nullptr;
    if(b)
        ptr = new int;
    else
        ptr = (int*)malloc(sizeof(int));
    return ptr;
}
```

- To handle assignments done in a if-else branch, traverser struct has a variable called undoLog, which is a map keeps old values of variables that overwriten in a if-else branch. By this way varMap will not be copied over and over again for every branch.

- To achieve a deterministic result, we have to know 2 things
- 1) What would be value of variables if an ***IfStmt*** works.
- 2) What were the values of variables before this ***IfStmt*** worked.

PRs

Compiler C|R Research

// Detection for builtin memory ownership notations

- ***Merged:*** Add IsAllocator and IsDeallocator API functions #1020
- ***Merged:*** Adds memory allocation analysis functions #1041
- ***Merged:*** Rebase computation under RASTV and support for recursion #1064
- ***Pending:*** FIX: Evaluating assignments on if-else branches #1065
- ***Pending:*** Deallocation: Adds memory deallocation analysis functions #1052

Tests

- We have done tests in both STL and libc.
- In STL we were able to detect allocator functions of containers succesfully.
- In libc we were able to detect strdup, strndup, wcsdup and fopencookie functions.
- But could not detect fdopen ,which is another function that returns heap allocated memory, because of the wrapping on returned pointer.

```
→ CppInterOp git:(alloc-detect-branch) ✗ grep -n "OperatorNew" stl_alloc_types.json
94:  {"name": "std::allocator_traits<std::allocator<char16_t>>::allocate", "result": "OperatorNew"},
99:  {"name": "std::allocator_traits<std::allocator<char32_t>>::allocate", "result": "OperatorNew"},
104:  {"name": "std::allocator_traits<std::allocator<int>>::allocate", "result": "OperatorNew"},
109:  {"name": "std::allocator_traits<std::allocator<std::_List_node<int>>>::allocate", "result": "OperatorNew"},
121:  {"name": "std::allocator_traits<std::allocator<std::__detail::_Hash_node<std::pair<const int, int>, false>>>::allocate", "result": "OperatorNew"},
128:  {"name": "std::allocator_traits<std::allocator<std::__detail::_Hash_node_base *>>::allocate", "result": "OperatorNew"},
133:  {"name": "std::allocator_traits<std::allocator<std::__detail::_Hash_node<int, false>>>::allocate", "result": "OperatorNew"},
140:  {"name": "std::allocator_traits<std::allocator<int *>>::allocate", "result": "OperatorNew"},
145:  {"name": "std::allocator_traits<std::allocator<std::_Fwd_list_node<int>>>::allocate", "result": "OperatorNew"},
152:  {"name": "std::allocator_traits<std::allocator<std::_Rb_tree_node<std::pair<const int, int>>>>::allocate", "result": "OperatorNew"},
159:  {"name": "std::allocator_traits<std::allocator<std::_Rb_tree_node<int>>>::allocate", "result": "OperatorNew"},
937:  {"name": "std::__new_allocator<char16_t>::allocate", "result": "OperatorNew"},
948:  {"name": "std::__new_allocator<char32_t>::allocate", "result": "OperatorNew"},
959:  {"name": "std::__new_allocator<int>::allocate", "result": "OperatorNew"},
970:  {"name": "std::__new_allocator<int *>::allocate", "result": "OperatorNew"},
980:  {"name": "std::__new_allocator<std::_List_node<int>>::allocate", "result": "OperatorNew"},
992:  {"name": "std::__new_allocator<std::_Fwd_list_node<int>>::allocate", "result": "OperatorNew"},
1014:  {"name": "std::__new_allocator<std::_Rb_tree_node<std::pair<const int, int>>>::allocate", "result": "OperatorNew"},
1026:  {"name": "std::__new_allocator<std::_Rb_tree_node<int>>::allocate", "result": "OperatorNew"},
1038:  {"name": "std::__new_allocator<std::__detail::_Hash_node<std::pair<const int, int>, false>>::allocate", "result": "OperatorNew"},
1050:  {"name": "std::__new_allocator<std::__detail::_Hash_node<int, false>>::allocate", "result": "OperatorNew"},
1062:  {"name": "std::__new_allocator<std::__detail::_Hash_node_base *>::allocate", "result": "OperatorNew"},
```

What is not done yet

- I will copy some logical things built in allocation PR to the deallocation part, and will do tests for deallocation.
- For the next term I will be implementing pairing mechanism for allocattors-deallocators, and analysis for thread-safety of functions.

Compiler C|R Research

Thanks for listening :)