Chapter 4

Developer Documentation

4.1 CombinationTester< T > Class Template Reference

```
#include <combinationTester.h>
```

Public Member Functions

- CombinationTester (int combinationSize, FunctionPointerMap< T > fpm, InstanceFunctionPointer< T > ifp, CoverageReporter *cr)
- void run ()

4.1.1 Detailed Description

template<typename T>
class CombinationTester< T >

This is the class that connects all others and achieves the main task of TO↩ DO sample code would be the main.cpp file Note: Although functionPointerMap is capable of forwarding passed arguments and returning the result, currently it is discarded. I could not find a straightforward way to store pass variable length and type inputs. Explored options included storing it as std::pair and using std::apply, but that would not resolve variable type parameters. std::invariant could aid in solving this issue.

4.1.2 Constructor & Destructor Documentation

CombinationTester()

```
template<typename T >
CombinationTester< T >::CombinationTester (
            int combinationSize,
            FunctionPointerMap< T > fpm,
            InstanceFunctionPointer< T > ifp,
            CoverageReporter * cr )
```

Collects all the necessary objects constructs a new permutation generator for this test run.

Parameters

combinationSize	maximum length of function call sequences that user wants to test
fpm	see member CombinationTester::functionPointerMap
ifp	see member CombinationTester::getNewInstance
cr	see member CombinationTester::coverageReporter

4.1.3 Member Function Documentation

run()

```
template<typename T >
void CombinationTester< T >::run ( )
```

This function will keep getting new function sequences from permutation generator until it has explored all paths. On each iteration:

1. New permutation of function call sequences is retrieved.
2. Instance of test class is constructed using the getNewInstance function pointer
3. Coverage reporting is initialized with the new permutation
4. Each function in the sequence is called using the functionpointermap. During this step, sanitizerCoverage library functions will insert found pc guards to coverageReporter. If CombinationTester encounters an exception during this step, it blacklists the path, stops and doesn't explore any further paths starting with that sequence, since all possible continuations would be interrupted with that exception and won't provide any new meaningful coverage.
5. Finally, coverageReporter is flushed

entire loop is wrapped in try catch so that no more functions are called after an exception TODO: is this also better for performance? in this implementation this step is not essential since paths are explored in increasing order. So only last call could possibly cause an exception However, if the implementation of permutation↩ Generator is changed later, this guarantee will no longer hold so having the entire loop wrapped in try catch will ensure that testing stops on first exception

4.2 CoverageReporter Class Reference

`#include <coverageReporter.h>`

Public Member Functions

- void startCoverage (std::vector< std::string > functionSequence)
- void addPCForCombination (const std::string &pc)
- void flush ()
- std::set< pc_set > coverage ()
- void **printResults** ()
- void **printResultsToFile** ()
- void **printResultsToFile** (std::string fileName)

Public Attributes

- pc_set currentPC
- std::map< pc_set, std::vector< std::string > > coverageSequences

4.2.1 Detailed Description

Stores reported coverage

4.2.2 Member Function Documentation

addPCForCombination()

```
void CoverageReporter::addPCForCombination (
        const std::string & pc )
```

Parameters

pc	will be added to the current set of collected pcs

coverage()

```
std::set< pc_set > CoverageReporter::coverage ( )
```

get all sets coverered so far

Returns

keys of coverageSequences, set of sets

flush()

```
void CoverageReporter::flush ( )
```

saves current permutation and associated coverage and empties both. if exact same coverage has been found with same or shorter sequence, the coverageSequences won't be updated, if longer one, the sequence for coverage will be replaced. otherwise, the function will check if new coverage contains any of the existing ones as a subset, in which case the old coverage will be removed and replaced with the larger set.

startCoverage()

```
void CoverageReporter::startCoverage (
        std::vector< std::string > functionSequence )
```

saves passed sequence as current one

Parameters

functionSequence	sequence of function names for which the coverage should be recorded

4.2.3 Member Data Documentation

coverageSequences

`std::map<pc_set, std::vector<std::string> > CoverageReporter::coverageSequences`

stores the shortest recorded function sequence for given coverage

currentPC

`pc_set CoverageReporter::currentPC`

set of all coverage points collected for current permutation

4.3 FunctionPointerMap< A > Class Template Reference

`#include <functionPointerMap.hpp>`

Public Member Functions

- template<typename T >
  void insert (std::string functionName, T functionPointer)
- void **insertNonVoid** (std::string functionName, voidFunction< A > functionPointer)
- template<typename T , typename... Args>
  T searchAndCall (A &instance, std::string functionName, Args &&... args)

4.3.1 Detailed Description

template<typename A>

class FunctionPointerMap< A >

Parameters

A	typename that the members will be stored for

4.3.2 Member Function Documentation

insert()

```
    template<typename A >
template<typename T >
void FunctionPointerMap< A >::insert (
            std::string functionName,
            T functionPointer )
```

insert new function to the map casts the function to void ∗(void) and stores the typeid to use for assertion later

Parameters

functionName	key used for looking up the function pointer in the map
functionPointer	pointer to the member function

searchAndCall()

```
    template<typename A>
template<typename T , typename... Args>
T FunctionPointerMap< A >::searchAndCall (
            A & instance,
```

```
std::string functionName,
Args &&... args )
```

This function is capable of passing the arguments to the member function and returning the result of the type T specified in the parameter. Originally, type_↩ index is used to assert that T and Args conform to the function signature. Currently this feature is turned off because of reasons specified in ... TODO: something about rvalue references

Parameters

a	reference to the instance which the function will be called on
functionName	key used for looking up the function pointer in the map
T	return type
args	arguments for function