Адаптация Lincheck для тестирования многопоточных алгоритмов на C/C++

Гаев Александр Анатольевич

научный руководитель: Коваль Никита Дмитриевич

СПб ВШЭ

10 июня 2021 г.

Проблема и мотивация

- Многопоточность многократно усложняет тестирование и увеличивает вероятность ошибки
- Не существует практичных инструментов для тестирования многопоточного кода на C++
- Многопоточные библиотеки на C++ вынуждены писать много кода для тестирования
- Хотелось бы предоставить удобный и практичный инструмент для тестирования кода на C++
- Для этого было решено адаптировать Lincheck

Текущие решения

- Удобные только Line-Up и Lincheck
- Не существует практичных для C++
  - Есть множество исследовательских проектов

Название	Практичность	Возможности	Доступность	Последнее обновление
JCStress[1] – Java	✗	✗	✓	2021
Line-Up[2] – C#	✓	✓	✗	2010
CONCURRIT[3] – C++	✗	✗	✓	2013
Lincheck[4] – JVM	✓	✓	✓	2021

[1] OpenJDK. (2015). JCStress. https://github.com/openjdk/jcstress

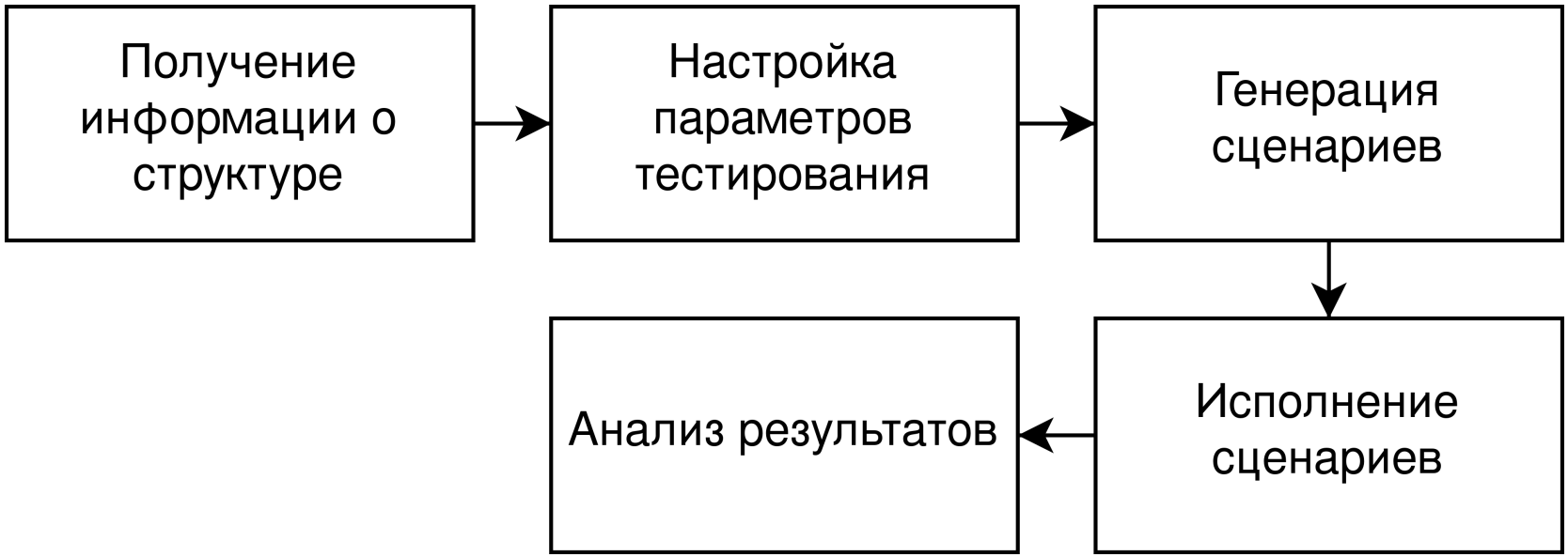
[2] Burckhardt et al. (2010). Line-up. ACM SIGPLAN Notices

[3] Elmas et al. (2013). CONCURRIT. ACM SIGPLAN Notices

[4] Koval et al. (2020). Testing concurrency on the JVM with lincheck. In Proceedings of the 25th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming

Lincheck

Lincheck[5] – фреймворк для тестирования многопоточных структур данных под JVM, предоставляющий возможность декларативного написания тестов



- Проверяет линеаризуемость
  - Ищет эквивалетное последовательное исполнение

[5] https://github.com/Kotlin/kotlinx-lincheck

Kotlin Multiplatform

Kotlin Multiplatform – Исполнение кода на Kotlin на разных платформах:

- Kotlin/JVM – JVM байт-код, Lincheck написан на Kotlin/JVM
- Kotlin/Native – Машинный код, имеет тесную интеграцию с C/C++
- Kotlin/Common – Один код для всех платформ
  - Поддерживается только системой сборки gradle

Если перевести Kotlin/JVM приложение(Lincheck) на Kotlin Multiplatform, то можно получить тесную интеграцию с C++ через Kotlin/Native

Цель и задачи

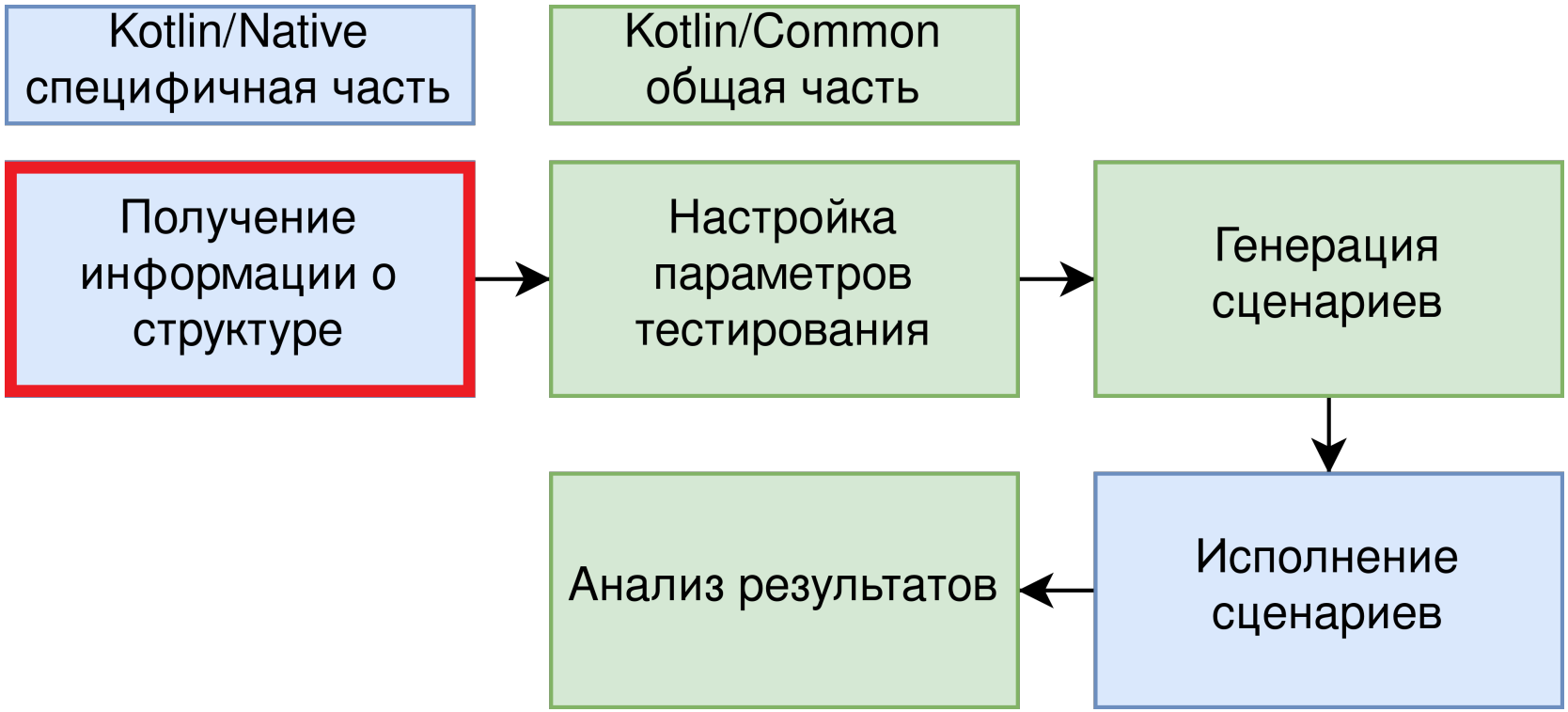
Цель – адаптировать Lincheck для тестирования многопоточных алгоритмов на C++.
Задачи:

- Перевести Lincheck на систему сборки gradle
- Переписать Lincheck на Kotlin/Common и Kotlin/Native
- Разработать удобный интерфейс на C++
- Протестировать популярные библиотеки на C++

Перевести Lincheck на систему сборки gradle

- Lincheck использовал систему сборки maven
- Kotlin Multiplatform поддерживается только на gradle
- Lincheck был переведен на gradle и настроен для Kotlin Multiplatform

Переписать Lincheck на Kotlin/Common и Kotlin/Native



Получение информации о структуре данных

Kotlin/JVM

```kotlin
@StressCTest(iterations = 10, threads = 3, verifier =
    EpsilonVerifier::class)
class StackTest {
    val stack = Stack<Int>()

    @Operation fun pop(): Int = stack.pop()
    @Operation fun push(value: Int) = stack.push(value)

    @Test fun test() = Linchecker.check(this::class)
}
```

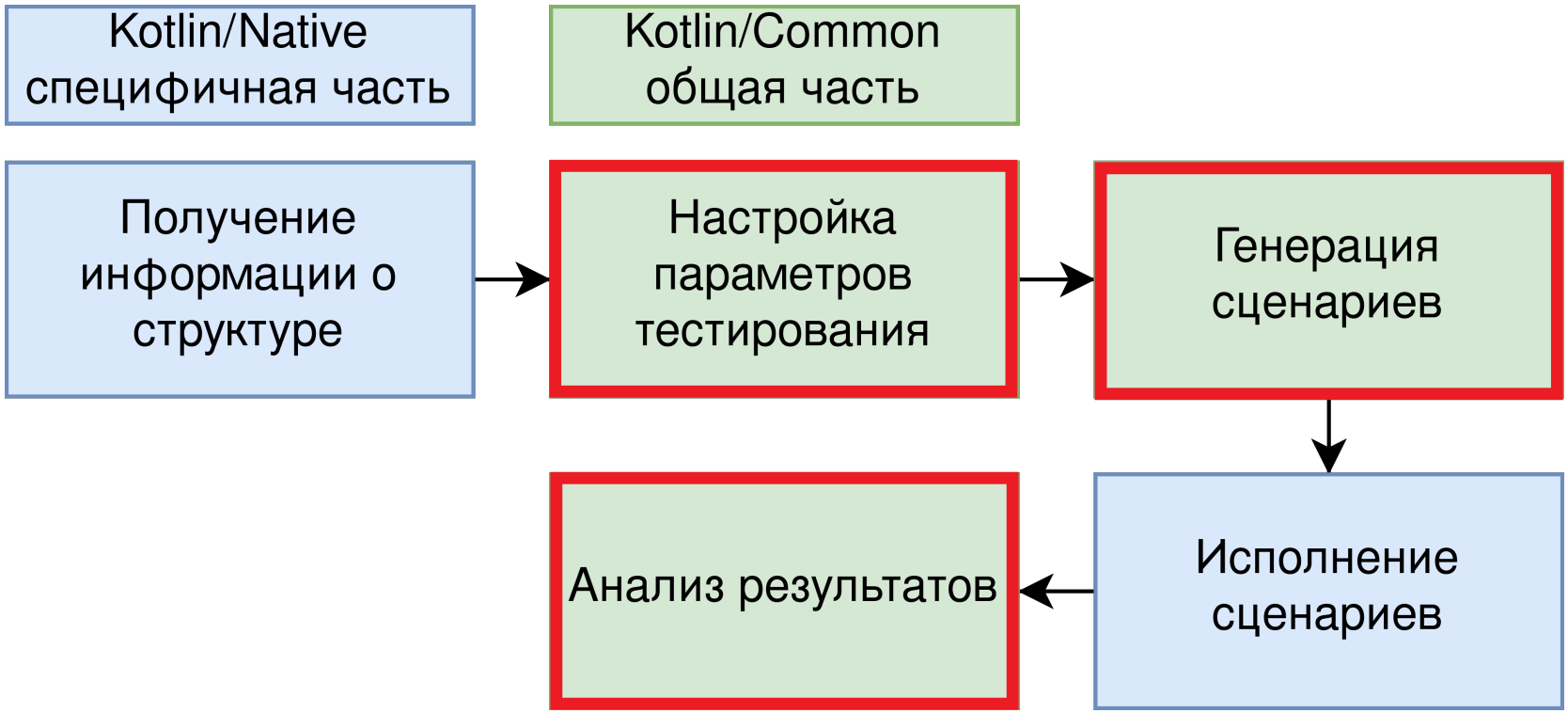
Kotlin/Native

```kotlin
LincheckStressConfiguration<HashMap>().apply {
    iterations(10)
    threads(3)

    initialState { HashMap() }

    operation(IntGen(), IntGen(), HashMap::put)
    operation(IntGen(), HashMap::get)
}.runTest()
```

Общая часть


Kotlin/Native
специфичная часть
Kotlin/Common
общая часть
Получение
информации о
структуре
Настройка
параметров
тестирования
Генерация
сценариев
Исполнение
сценариев
Анализ результатов

Общая часть, Java Reflection

- Общая часть была получена из существующего JVM кода с помощью переноса и замены Java Reflection на абстрактные классы
- Lincheck активно использует Java Reflection
  - Для описания структуры с помощью @StressCTest, @Operation, ...
  - Для внутренней передачи информации о структуре

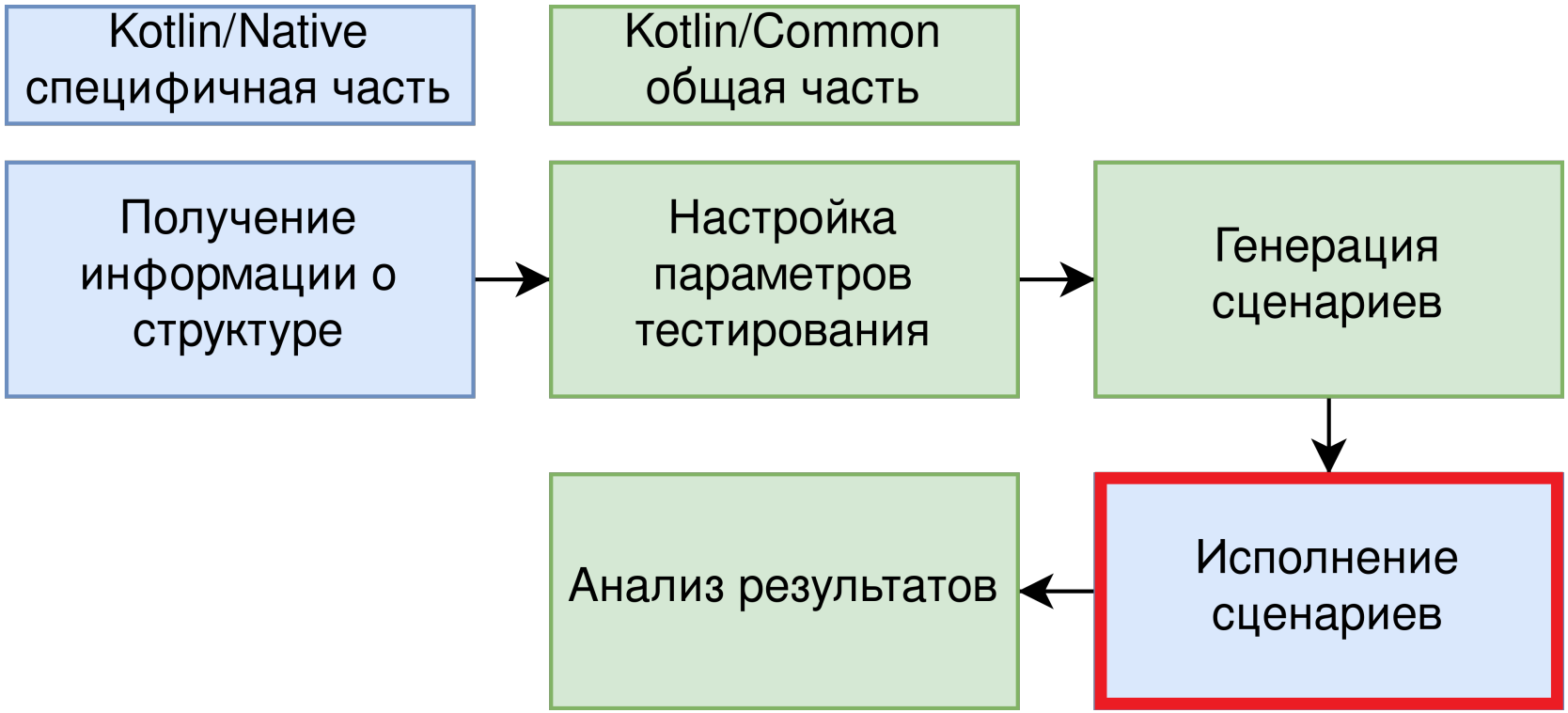
```
1  // Before, Kotlin/JVM
2  class StressCTestConfiguration(
3  testClass: Class<*>,
4  generatorClass: Class<out ExecutionGenerator>,
5  verifierClass: Class<out Verifier>,
6  sequentialSpecification: Class<*>?
7  )
8
9  // After, Kotlin/Common
10 class StressCTestConfiguration(
11 testClass: TestClass,
12 executionGenerator: (conf, ctest) -> ExecutionGenerator,
13 verifierGenerator: (spec) -> Verifier,
14 sequentialSpecification: SequentialSpecification<*>
15 )
```

Общая часть, Java Reflection

- Большинство сущностей удалось перевести в функциональные типы
- Некоторые сущности разделились с помощью expect/actual механизма
  - Для трансформаций кода
  - Для последовательной спецификации

```
1  // Kotlin/Common
2  expect class TestClass {
3      fun createInstance(): Any
4  }
5
6  // Kotlin/JVM
7  actual class TestClass(val clazz: Class<*>) {
8      val name = clazz.name
9
10     actual fun createInstance(): Any = clazz.getDeclaredConstructor().newInstance()
11 }
12
13 // Kotlin/Native
14 actual class TestClass(val create: () -> Any?) {
15     actual fun createInstance(): Any = create()
16 }
```

Исполнение сценариев



Исполнение сценариев

- Была написана подсистема, которая исполняет сценарии
- Kotlin/Native не позволяет многопоточно работать с изменяемым состоянием
  - Был использован экспериментальный небезопасный режим Kotlin/Native, который позволяет это делать
  - Экспериментально было получено значение в 500 запусков каждого сценария, при котором всё стабильно работает
    - Если больше, то непотокобезопасный сборщик мусора ведёт к падению
- Kotlin/Native не позволяет задавать свой деструктор или аналог метода finalize в Java
  - При работе с объектами из C++ необходимо контролировать и освобождать память
  - Был добавлен интерфейс Finalizable и контроль за создаваемыми ресурсами в C++

Интерфейс на C++

- Был сгенерирован интерфейс на C
  - Kotlin/Native принимает параметры через C
  - Оборачивает их в Kotlin объекты
  - Отправляет эти объекты на тестирование
- Была написана удобная обертка на C++

Тестирование из C++ выглядит так:

```
#include "lincheck.h"

int main() {
    LincheckConfiguration<Counter, Counter> conf;
    conf.iterations(10);
    conf.threads(3);
    // return type, argument type, operation, sequential specification
    conf.operation<int, int, &Counter::add, &Counter::add>("add");
    conf.runTest();
}
```

```
org.jetbrains.kotlinx.lincheck.LincheckAssertionError:
= Invalid execution results =
Parallel part:
| add(-5): -5 | add(9): 9 |
Post part:
[add(-2): -7, add(0): -7, add(-2): -9, add(8): -1, add(5): 4]
```

Тестирование существующих библиотек на C++ с помощью Lincheck

- Lincheck тестирует линеаризуемость
  - Не были протестированы нелинеаризуемые структуры данных
- Протестированы популярные C++ библиотеки

Название	Ошибки	Полностью	Структуры
Libcuckoo[6]	✗	✓	HashMap
Folly/Concurrency[7]	✗	✓	HashMap, [SM]P[SM]C(Un)BoundedQueue
Boost/Lockfree[8]	✗	✓	Queue, Stack, SPSCQueue
Libcds[9]	✗	✗	MSQueue, TreiberStack

- Также протестированы структуры с внедренными ошибками
  - Lincheck находит ошибки

[6] https://github.com/efficient/libcuckoo
[7] https://github.com/facebook/folly
[8] https://github.com/boostorg/lockfree
[9] https://github.com/khizmax/libcds

Результаты

- Lincheck адаптирован для тестирования кода на языках Kotlin/Native и C++
  - Переведен на систему сборки gradle
  - Переписан на Kotlin/Common и Kotlin/Native
- Реализована удобная обертка для использования на языке C++
- Протестированы популярные библиотеки на C++

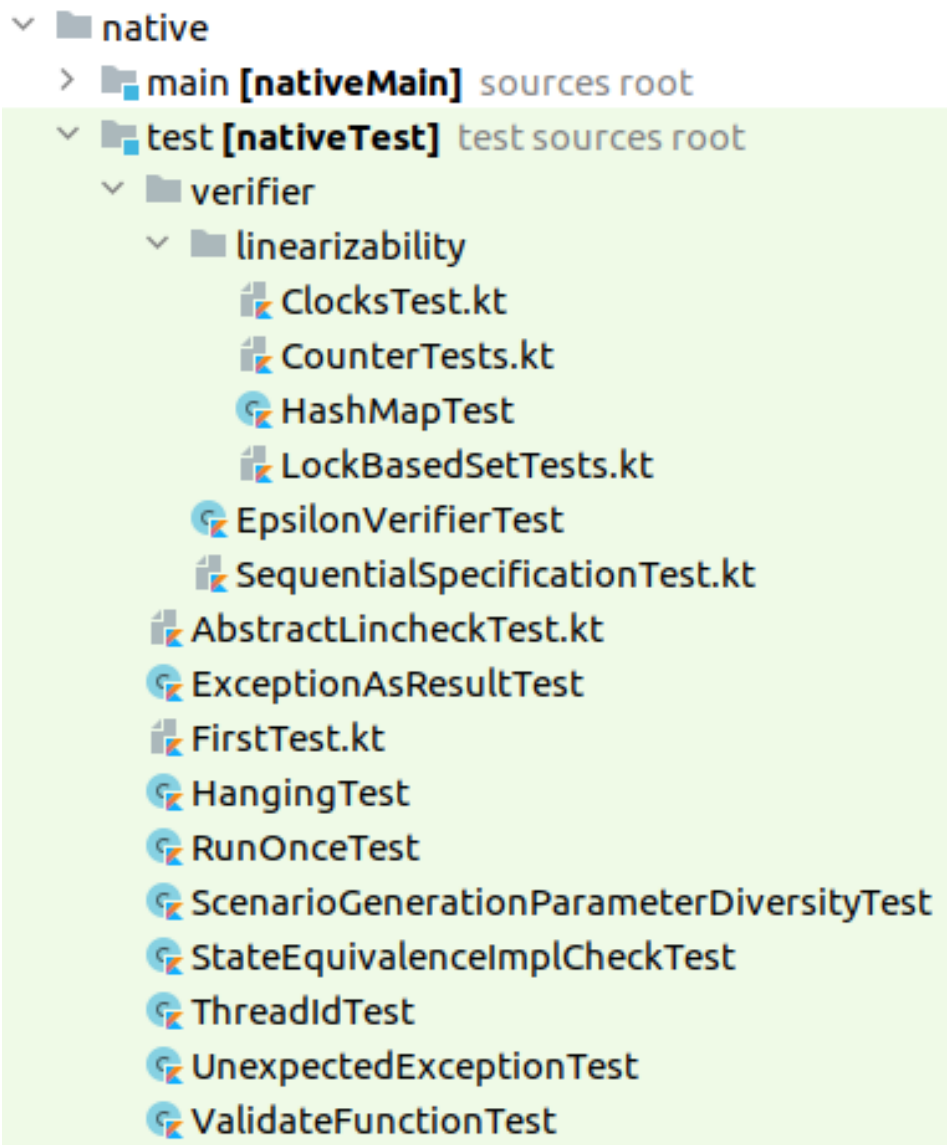
Определения критериев в тексте диплома

- Практичность означает то, ориентировался ли инструмент на практичность при разработке, и является ли он достаточно удобным для использования в промышленных проектах.
- Возможности означает то, способен ли инструмент тестировать структуры на различные сложные критерии корректности, такие как линеаризуемость.
- Доступность означает то, доступен ли инструмент для открытого использования, можно ли применять его и анализировать его код.

Список тестов на C++ и производительность

100 итераций, 500 запусков, 3 потока, 4 операции в потоке	Время работы
LibcdsTest.BadSequentialQueueTest	90ms
LibcdsTest.BadSequentialStackTest	1sec
LibcdsTest.ConcurrentQueueHPTest	2min 13ms
LibcdsTest.ConcurrentQueueDHPTest	2min 13ms
LibcdsTest.ConcurrentTreiberStackHPTest	1min 46ms
LibcdsTest.ConcurrentTreiberStackDHPTest	1min 52ms
LibcuckooTest.BadSequentialMapTest	1sec
LibcuckooTest.HashMapTest	5min 40ms
FollyTest.BadSequentialMapTest	1sec
FollyTest.BadSequentialQueueTest	80ms
FollyTest.HashMapTest	2min 2sec
FollyTest.MPMCDynamicBoundedQueueTest	2min 30sec
FollyTest.MPSCDynamicBoundedQueueTest	5min 3sec
FollyTest.BadMPSCDynamicBoundedQueueTest	50ms
FollyTest.SPMCDynamicBoundedQueueTest	1min 50sec
FollyTest.SPSCDynamicBoundedQueueTest	1min 27sec
FollyTest.BadSPSCDynamicBoundedQueueTest	130ms
FollyTest.MPMCUnboundedQueueTest	2min 7sec
FollyTest.MPSCUnboundedQueueTest	4min 45sec
FollyTest.SPMCUnboundedQueueTest	1min 49sec
FollyTest.SPSCUnboundedQueueTest	1min 25sec
BoostLockfreeTest.BadSequentialQueueTest	120ms
BoostLockfreeTest.BadSequentialStackTest	30ms
BoostLockfreeTest.QueueTest	2min 0sec
BoostLockfreeTest.StackTest	1min 39sec
BoostLockfreeTest.BadSPSCQueueTest	970ms
BoostLockfreeTest.SPSCQueueTest	1min 18sec

Тестирование Kotlin/Native



Обертка на C++

```
template<typename Ret, typename Arg1, Ret (TestClass::*op)(Arg1), Ret (SequentialSpecification::*seq_spec)(
                Arg1)>
        void operation(const char *operationName, const char *nonParallelGroupName = nullptr, bool useOnce = false) {
            lib->kotlin.root.org.jetbrains.kotlinx.lincheck.NativeAPIStressConfiguration.setupOperation2(
                    configuration,
                    (void *) (void *(*)()) []() -> void * { // arg1_gen_initial_state
                        return new ParameterGenerator<Arg1>();
                    },
                    (void *) (void *(*)(void *)) [](void *gen) -> void * { // arg1_gen_generate
                        auto *obj = (ParameterGenerator<Arg1> *) gen; // add type to void*
                        Arg1 arg = obj->generate(); // invoke generate method
                        return new Arg1(arg); // copy from stack to heap and return
                    },
                    (void *) (void (*)(void *, char *, int)) [](void *arg, char *dest, int destSize) { // arg1_toString
                        strncpy(dest, Lincheck::to_string<Arg1>()(*(Arg1 *) arg).c_str(), destSize);
                    },
                    (void *) (void (*)(void *)) [](void *arg1) { // arg1_destructor
                        Arg1 *obj = (Arg1 *) arg1; // add type to void*
                        delete obj; // destructor
                    },
                    (void *) (void *(*)(void *, void *)) [](void *instance, void *arg1) -> void * { // operation
                        auto *obj = (TestClass *) instance; // add type to void*
                        auto *a1 = (Arg1 *) arg1;
                        Ret res = (obj->*op)(*a1); // invoke op method
                        return new Ret(res); // copy from stack to heap and return
                    },
                    (void *) (void *(*)(void *, void *)) [](void *instance,
                                                            void *arg1) -> void * { // sequential specification
                        auto *obj = (SequentialSpecification *) instance; // add type to void*
                        auto *a1 = (Arg1 *) arg1;
                        Ret res = (obj->*seq_spec)(*a1); // invoke op method
                        return new Ret(res); // copy from stack to heap and return
                    },
                    (void *) (void (*)(void *)) [](void *ret) { // Ret destructor
                        Ret *obj = (Ret *) ret; // add type to void*
                        delete obj; // destructor
                    },
                    (void *) (bool (*)(void *, void *)) [](void *a, void *b) -> bool { // Ret equals
                        Ret *obj_a = (Ret *) a; // add type to void*
                        Ret *obj_b = (Ret *) b; // add type to void*
                        return *obj_a == *obj_b;
                    },
                    (void *) (int (*)(void *)) [](void *ret) -> int { // Ret hashCode
                        return Lincheck::hash<Ret>()(*(Ret *) ret);
                    }
```

Интерфейс на C

```
1   fun setupOperation2(
2       arg1_gen_initial_state: CCreator,
3       arg1_gen_generate: CPointer<CFunction<(CPointer<*>) -> CPointer<*>>>,
4       arg1_toString: ToStringCFunction,
5       arg1_destructor: CDestructor,
6       op: CPointer<CFunction<(CPointer<*>, CPointer<*>) -> CPointer<*>>>,
7       seq_spec: CPointer<CFunction<(CPointer<*>, CPointer<*>) -> CPointer<*>>>,
8       result_destructor: CDestructor,
9       result_equals: EqualsCFunction,
10      result_hashCode: HashCodeCFunction,
11      result_toString: ToStringCFunction,
12      operationName: String,
13      nonParallelGroupName: String? = null,
14      useOnce: Boolean = false,
15  ) = apply {
16      val arg1_paramgen = object : ParameterGenerator<ParameterGeneratorArgument> {
17          val state = arg1_gen_initial_state.invoke()
18          override fun generate(): ParameterGeneratorArgument {
19              return ParameterGeneratorArgument(arg1_gen_generate.invoke(state), arg1_destructor, arg1_toString)
20          }
21      }
22      val actorGenerator = ActorGenerator(
23          function = { instance, arguments ->
24              when (instance) {
25                  is ConcurrentInstance -> {
26                      ObjectWithDestructorAndEqualsAndHashcodeAndToString(op.invoke(instance.obj, (arguments[0] as
                                ParameterGeneratorArgument).arg), result_destructor, result_equals, result_hashCode,
                                result_toString)
27                  }
28                  is SequentialSpecificationInstance -> {
29                      ObjectWithDestructorAndEqualsAndHashcodeAndToString(seq_spec.invoke(instance.obj, (arguments[0] as
                                ParameterGeneratorArgument).arg), result_destructor, result_equals, result_hashCode,
                                result_toString)
30                  }
31                  else -> {
32                      throw RuntimeException("Internal error. Instance has not expected type")
```