
Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales
Departamento de Computación

# Reescribiendo binarios de test usando C#

Tesis de Licenciatura en Ciencias de la Computación

Ezequiel Dario Gambaccini

Codirector: Juan Pablo Galeotti
Codirector: Diego Garbervetsky
Buenos Aires, Argentina, 2021

**Commented [GU1]:** Diego y yo somos codirectores

**Commented [GU2]:** Galeotti

**Commented [GU3]:** Diego y yo somos codirectores

**Commented [GU4]:** Pone Buenos Aires, no queda claro que sea legal hacerlo remoto :)

**AKCNOWLEDGEMENTS**

To Melina, my wife, for preventing me from throwing everything out the window.

To Juan Pablo, and Diego, my directors, for their guidance and dealing with a full remote thesis in a different time zone during a global pandemic.

To Pantazis Deligiannis, one of the authors from Coyote, for his help to debug some Coyote issues and ping pong some rewrite ideas together.

To Matias Hunicken, my friend, for reviewing this manuscript and giving suggestions, as well as discussing different approaches for rewriting code.

## Table of Contents

## 0. ABSTRACT

Distributed systems and concurrent programming are now the way for designing applications, due to the increase of availability in cloud computing and multi core processors.

However, this programming paradigm is not free of problems, which are very hard to find, test and reproduce, as they become probabilistic in nature due to the intertwined nature of concurrent applications.

Coyote is a tool which allows users to test their applications to find these problems and provides a way to reproduce them. However, in order to do this, users are required to write specific tests using specific Coyote APIs so that the tool can do its job.

In this thesis, I propose a way to do a binary rewrite of existing tests written in C# using the xUnit test framework so that they are able to be executed within Coyote without any extra work from the user by doing an IL level rewrite of existing tests written in the C# programming language.

Initial evaluation shows the approach can soundly handle real applications and could be of help for non-expert developers in finding concurrency bugs.

## 1. INTRODUCTION

When developing a computer program, generally it is expected that it takes full advantage of the hardware resources of the system and that it executes as fast as possible. Maximizing the use of a processor used to be easy, as they used to be single threaded, so just maximizing single thread performance would guarantee that the program ran as fast as possible, with the added bonus that when new processors came to market with higher clock frequencies, the program would also get a performance boost.

Commented [GU5]: no usar contractions

However, in the early 2000s, due to the relation between processor frequency (f), voltage (v) and consumed power ($f \sim V$ and $P \sim C_{dyn} * V^3$ [1]), where a linear increase in frequency would cause a cubic increase in power dissipation, this trend ended, as the requirements for cooling would be prohibitive for most users, and if the heat was not properly handled, it could cause a physical meltdown of the processor. This issue is known as the “Power wall”.

In order to keep increasing the performance of computer processors, manufacturers decided to scale horizontally, that is, add more processor cores to the same package instead of increasing frequency. This brought an issue to programmers, as in order to take advantage of the hardware and increase the performance of their programs, they now had to modify their programs to execute across multiple processor cores, also known as concurrent programming.

Concurrent programming can bring performance increases for computer programs, but with several caveats:

- The maximum performance increase a computer program will always be less than the number of cores, since there are pieces of the program that can’t be parallelized, as well as having multiple execution threads competing for access to some resources. This is known as Ahmdal’s Law [2].
- Concurrency bugs can be very subtle, such as race conditions, where there is no set order for accessing data, which can make them hard to reason about and probabilistic in nature, as the possible number of states in the program increases exponentially with the added number of threads, which leads to having a small number of possible states where the bug can happen.

- It's possible that in order to have a program execute concurrently, a full rewrite of the program is needed.

In synthesis, doing effective concurrent programming is hard.

In order to mitigate concurrency bugs in the C# programming language for programs using asynchronous programming pattern, which makes easier to write concurrent programs as function calls can be represented as tasks which can be executed within a thread pool, Microsoft Research developed a tool called Coyote which provides an API to develop tests and execute them within the Coyote tool so that the tool can find possible concurrency bugs, as well as provide a trace on how to reproduce the bug. This has been a boon for several teams working on different services in the Azure cloud that Microsoft owns.

However, this testing is not free, as programmers are required to take a direct dependency to the Coyote tool APIs, and maintain them on their services, as well as write dedicated tests for this, which makes adoption harder.

The contributions of this thesis are:

- A technique to automatically generate test cases prepared for Coyote using regular test cases as an input.
- A C# implementation to translate xUnit tests into Coyote ready tests.
- An evaluation over test cases in the Coyote tool, and the concurrent collections library in the C# runtime.

In this preliminary evaluation one can observe that the technique is effective to generate tests cases which run in Coyote, allowing programmers to focus on writing tests instead of learning a new tool.

**Commented [DG6]:** Algun ejemplo minimo, para dar una idea (tipo race condiction o deadlock)

**Commented [EDG7R6]:** Eso lo explique en background, agrego aca tambien?

**Commented [DG8]:** no se entiende en que ayuda este pattern

**Commented [EDG9R8]:** ok, agrego aclaracion

**Commented [DG10]:** Contribucion de la tesis:
- Una tecnica para generar automaticamente casos de test preparados para coyote a partir de test standard
- Una implementacion en C# para traducir xUnit en test de coyote
- Una evaluacion sobre blah

Esta evaluacion preliminar se observan indicios de que la tecnica es efectiva para generar casos de test concurrentes, permitiendo que los programadores no tengan que aprender una herramienta nueva (o algo asi)

**Commented [EDG11R10]:** ok!

**Commented [GU12]:** The contribution of this thesis are:

## 2. BACKGROUND

Concurrent programming is a programming paradigm where a programmer tries to maximize the usage of the resources the application has available by distributing computations across different processors, threads, or processes (units of computation). This can possibly increase the performance of the application, depending on how much of the workload can be executed non sequentially, as Amdahl's law states. In cloud computing, the term used for throwing more hardware at a problem is called "scale out".

However, there is a major challenge in concurrent programming, which is concurrency control, that is, ensuring the correct order of interactions between different units of computation.

Commented [GU13]: major

Concurrency control can have issues such as race conditions, deadlock, and resource starvation. Since the appearance of these problems depends on a specific sequence/s of interactions, they become probabilistic in nature, and very hard to reason about. Their complexity also increases as the load becomes more distributed, as a program designed to be distributed across 2 threads might fail if distributed across 4 threads.

Let's take a deeper look at these issues.

A **race condition** happens when 2 or more units of computation try to access a resource at the same time to read it or write it, and the result of the program execution changes depending on the order of the units of computation accessing the resource.

Check this following example:

Commented [GU14]: the

```
class Test
{
    private int x = 0;

    public void Inc()
    {
        this.x++;
        if (this.x > 0)
        {
            Console.WriteLine("Positive!");
        }
    }

    public void Dec()
    {
        this.x--;
        if (this.x < 0)
        {
            Console.WriteLine("Negative!");
        }
    }

    public async Task Operate()
    {
        var t1 = Task.Run(() => this.Inc());
        var t2 = Task.Run(() => this.Dec());

        await Task.WhenAll(t1, t2).ConfigureAwait(false);
    }
}
```

One would expect that a call to the Operate method would print only “Positive!” to screen. However, since there is no control on how read/write to the x field, the result is up in the air.

Here is a sample output after calling Operate 10 times:

Negative!

Positive!

Positive!

Negative!

Positive!

Positive!

Positive!

Positive!

Negative!

Positive!

The variability of the result is due to both operations being executed at the same time. Sometimes the Dec method is executed first, so x would be -1, and when Inc is called, then x goes back to 0, hence only printing “Negative!”.

A **deadlock** happens when the program execution cannot progress due to 2 or more units of computation blocking each other from continuing due to needing the other unit of computation to release a resource or do something and vice versa. The necessary and sufficient conditions are as follow [3]:

1. Mutual exclusion
   The resources involved must be unshareable; otherwise, the processes would not be prevented from using the resource when necessary.
2. Hold and wait or partial allocation
   The processes must hold the resources they have already been allocated while waiting for other (requested) resources. If the process had to release its resources when a new resource or resources were requested, deadlock could not occur because the process would not prevent others from using resources that it controlled.
3. No pre-emption
   The processes must not have resources taken away while that resource is being used. Otherwise, deadlock could not occur since the operating system could simply take enough resources from running processes to enable any process to finish.
4. Resource waiting or circular wait
   A circular chain of processes, with each process holding resources which are currently being requested by the next process in the chain, cannot exist. If it does, the cycle theorem (which states that "a cycle in the resource graph is necessary for deadlock to occur") indicated that deadlock could occur.

**Resource starvation** happens when a unit of computation is perpetually denied access to a resource it needs to continue its work. Possible causes for starvation are due to scheduling algorithms and deadlock.

**Cloud computing** refers to companies which offer services of computing and storage on demand to users, with the benefit of users not having to purchase and maintain hardware. The biggest offerings are AWS, provided by Amazon inc., Azure, provided by Microsoft inc., and Google Cloud Platform, provided by Google inc. With these offerings, users and companies can easily create heavily distributed services and scale them on demand using different software APIs, thus making concurrent programming and distributed systems the prevalent way of programming.

Commented [GU15]: these

This means that having a way to test and validate those concurrent programs don’t have the previously mentioned issues has become a bigger necessity than it ever has been.

**Coyote** [4] is a tool developed by Microsoft Research to find these concurrency issues in programs using the asynchronous programming pattern written in the C# programming language. It finds these issues by executing what they call concurrency unit tests. The source of non-determinism can be declared as part of the tests and can be as simple as 2 threads competing over a resource or performing concurrent operations over an object.

When the test is executed in the Coyote tool, the tool systematically explores a large number of interleavings of concurrent operations as well as non-deterministic choices so that it covers a large set of behaviors in a very short time. The source of non-determinism is as simple as 2 threads competing over a resource or performing concurrent operations over an object. In order to do this, Coyote takes control of the concurrency so that it can manipulate every possible scheduling. With appropriate mocking, Coyote can also do this in “developer” mode on a single laptop with little or no dependence on the bigger production environment.

Appropriate mocking here means mocking APIs for different services, so that the mock behaves the same as the API would, allowing Coyote to take control over its behavior since it would live in memory. For example, using concurrent dictionaries (a type of thread safe hash table), a user could mock a key value database or storage service.

Coyote is not a verification system. It does not use a theorem prover to make correctness guarantees, instead it uses intelligent search strategies to drive systematic testing, based on deep understanding of concurrency primitives that have been used in the code to analyze. This approach has proven to work well for large production teams, including many teams in Microsoft Azure because it has a small barrier to entry with almost immediate benefits for those who adopt it.

Commented [GU16]: prover

The strategies Coyote supports for now are 3:

- Random strategy, which is essentially backtracking over the problem space, exploring the problem space trying different states until a bug is found. [5]
- Probabilistic concurrency testing [6], a randomized algorithm for concurrency testing. PCT randomly schedules the threads of the program during each test run. In contrast to prior randomized testing techniques, PCT uses randomization sparingly and in a disciplined manner. As a result, PCT provides efficient mathematical probability of finding a concurrency bug in each run. Repeated independent runs can increase the probability of finding bugs to any user-desired level of certainty. PCT relies on the crucial observation that bugs in practice are not adversarial. Concurrency bugs typically involve unexpected interactions among a few instructions executed by a small number of threads. If PCT is able to schedule these few instructions correctly, it succeeds in finding the bug irrespective of the numerous ways it can schedule instructions irrelevant to the bug. The following characterization of concurrency bugs captures this intuition precisely. The depth of a concurrency bug is defined as the minimum number of scheduling constraints that are sufficient to find the bug. Intuitively, bugs with a

higher depth exhibit in fewer schedules and are thus inherently harder to find. Any schedule that satisfies this ordering constraint finds the bug irrespective of the ordering of other instructions in the program. On each test run, PCT focuses on probabilistically finding bugs of a particular depth d. PCT is a randomized priority-based scheduler and schedules the runnable thread with the highest priority at each scheduling step. Priorities are determined as follows. On thread creation, PCT assigns a random priority to the created thread, where lower numbers indicate lower priorities. Additionally, PCT changes thread priorities at d − 1 randomly chosen steps during the execution.

During execution, the scheduler schedules a low priority thread only when all higher priority threads are blocked. Only one thread is scheduled to execute in each step. A thread can get blocked if it is waiting for a resource, such as a lock that is currently held by another thread. Threads can change priorities during execution when they pass a priority change point. Each such point is a step in the dynamic execution and has a predetermined priority value associated with it. When the execution reaches a change point, the scheduler changes the priority of the current thread to the priority value associated with the change point.

Given inputs n (number of threads), k (number of instructions), and d (expected depth of the bug), PCT works as follows.

1. Assign the n priority values d, d+1, . . ., d+n randomly to the n threads (we reserve the lower priority values 1, . . .,(d − 1) for change points).
2. Pick d − 1 random priority change points k1, . . . , kd−1 in the range [1, k]. Each $k_i$ has an associated priority value of i.
3. Schedule the threads by honoring their priorities. When a thread reaches the i-th change point (that is, when it executes the $k_{i\text{-th}}$ step of the execution), change the priority of that thread to i.

This randomized scheduler provides the following guarantee. Given a program that creates at most n threads and executes at most k instructions, PCT finds a bug of depth d with probability at least 1/nkd−1.

- Reinforced learning, using the classical Q-learning (QL) algorithm [7]. The general reinforcement learning (RL) scenario comprises an agent interacting with its environment. In the beginning, the agent has no knowledge about the environment. At each step, the agent can only partially observe the state of the environment and invoke an action based on some policy. As a result of this, the environment transitions to a new state, and provides a reward signal to the agent. The objective of the agent is to select a sequence of actions that maximizes the expected reward. For coyote, the RL agent is the controlled concurrency testing (CCT) scheduler, and the RL environment is the program: the scheduler (agent) decides the action in the form of which worker to execute next, and the program (environment) executes the action by running the worker for one step, and then passing control back to the scheduler to choose the next action.

  For each state-action pair, QL associates a (real-valued) quality metric called q-value. When QL observes that the program is in state $s$, it picks an action $a$ from the set of all enabled actions with probability that is governed by the q-value of ($s$, $a$). After one run of the program finishes, QL updates the q-values of each state-

action pair that it observed during the run, according to the rewards that it received, and uses the updated q-values for choosing actions in the next run. The goal of QL is to explore the program state space, covering as many diverse set of executions as it can. In other words, the scheduler should maximize coverage. For this purpose, the reward was always a fixed negative value (−1), which has the effect of disincentivizing the scheduler from visiting states that it has seen before. In other words, the more times a state $s$ has been visited before, the higher is the likelihood of QL to stay away from $s$ in future runs. The choice remains probabilistic and the probabilities never go to 0. This is essential because the scheduler only gets to observe the program state partially.

Coyote does not require that a team starts from scratch and rebuilds their system. Coyote uses binary rewriting during test time to take control of the concurrency in your unmodified code. This rewrite affects only Task objects and related concurrency types from the .NET Task Parallel Library. A Task object [8] (Task<T> for operations that return a value of type T) represents an asynchronous operation which typically gets executed asynchronously on a thread pool thread. A lambda expression or method is passed as part of its constructor in order to be executed asynchronously later.

For advanced users, Coyote also provides a powerful in-memory actor and state machine programming model that allows the programmer to build reliable concurrent systems from the ground up. This programming model allows you to program at a high-level of abstraction. Coyote actors are built using asynchronous C# APIs, supported by a lightweight runtime, making it easy to program efficient non-blocking code.

Coyote actors are Coyote's implementation of the Actor model, a mathematical model of concurrent computation where the actor is the universal primitive of concurrent computation. Actors can create more actors, make local decisions, and send more messages. Actors can only modify their private state, and can only affect each other indirectly through messaging.

Even though Coyote offers binary rewriting for different concurrency types in .NET, it still requires that the programmer writes tests and configurations specific to Coyote.

xUnit [9] is unit testing framework for .Net languages, specifically C#. It allows users to easily write unit tests, using the constructor of the test class as setup, and having the user implement the IDisposable interface if a teardown is needed. It also allows users to create specific classes for common behavior across different tests. xUnit has 2 types of tests available, Facts and Theories [10]. Per their website, Facts are tests which are always true and test invariant conditions. Theories are tests which are only true for a particular set of data. Due to this, Theory tests need to have input data. For example, a Theory would be a function IsEven(n), which tests if a given number is even or not. This function is only true for the subset of numbers which are even in the set of natural numbers. Theories can be used to create parameterized tests.

**Commented [GU17]:** Poner un ejemplo de Fact y de Theory

An example of a very simple Fact test is as follows:

```
[Fact]
public void TwoPlusTwoIsFour()
{
    var a = 2;
    var b = 2;
    var sum = this.Sum(a, b);

    Assert.Equal(4, sum);
}

public int Sum(int a, int b) => a + b;
```

An example of a very simple Theory test is as follows:

```
[Theory]
[InlineData(2)]
[InlineData(4)]
[InlineData(6)]
[InlineData(8)]
public void IsEven(int n)
{
    var isEven = (n % 2) == 0;

    Assert.True(isEven);
}
```

Using a theory to create a parameterized test:

```
[Theory]
[InlineData(2, 2, 4)]
[InlineData(4, 2, 6)]
[InlineData(6, 2, 8)]
[InlineData(8, 2, 10)]
public void SumTest(int a, int b, int expected)
{
    var sum = this.Sum(a, b);

    Assert.Equal(expected, sum);
}

public int Sum(int a, int b) => a + b;
```

**Cecil** [11] is a C# library to generate and inspect programs and libraries in the ECMA CIL form (C# bytecode).

It can be used to analyze .NET binaries, as well as modify them by altering the IL code.

## 3. TECHNIQUE

In order to reduce the burden on the programmer when writing tests, and for existing codebases where rewriting tests to support Coyote would be very expensive in man-hours and resources, I propose a way to automatically rewrite tests written in xUnit.

This way, the only thing the programmer must do is execute the rewrite tool, then the coyote rewrite command, and then execute tests as usual, simple steps which can be automated into a powershell script.

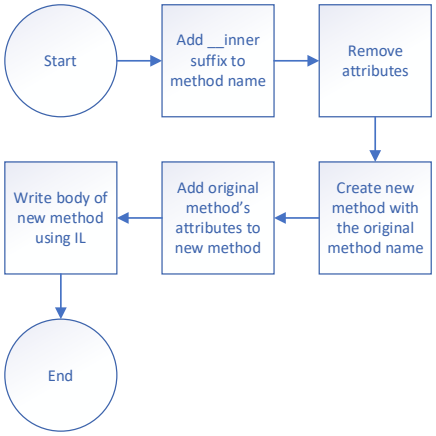
The idea is to use C#, Cecil and Coyote to write a tool to modify test library files (dll).

At a high level, the steps are as follows:

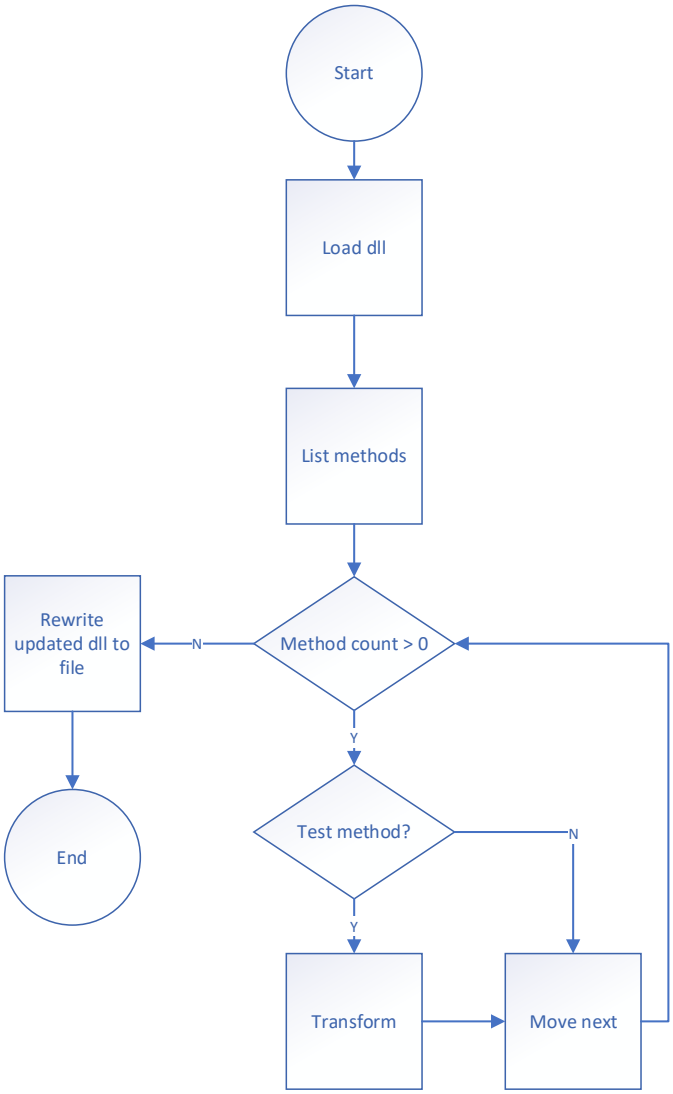
1. Load dll into the tool memory.
2. Search for xUnit test methods using Cecil.
    - This is simple as xUnit test methods have attributes identifying them.
3. Once test methods are found, apply a transformation according to their type.
4. This transformation will do the following:
    - Rename the test method by adding the "__inner" suffix.
    - Remove all its attributes.
    - Create a new method named as the original method, using the original method's attributes. The body is generated in IL using Cecil.
5. Write the modified code back to the dll.
6. Copy a dll with helper methods to the same folder as the modified test dll.

After this, the user shall execute tests as usual, with the difference that now they will be executed within Coyote.

Transformation flow:




High level flow for the technique:




Rewrite example

Given the following class:

```
public class Test
{
    private int x = 0;

    private bool printPositive = false;

    public void Inc()
    {
        this.x++;
        if (this.x > 0)
        {
            this.printPositive = true;
        }
    }

    public void Dec()
    {
        this.x--;
        if (this.x < 0)
        {
            this.printPositive = false;
        }
    }

    public async Task<bool> Operate()
    {
        var t1 = Task.Run(() => this.Inc());
        var t2 = Task.Run(() => this.Dec());

        await Task.WhenAll(t1, t2).ConfigureAwait(false);

        Console.WriteLine(this.printPositive ? "Positive!" : "Negative!");

        return this.printPositive;
    }
}
```

C# Task methods explanation:

Task.Run(method/lambda function): This method executes the given method/lambda in the CLR threadpool, and returns a Task object to track the progress of the operation.

Task.WhenAll(t1, t2, …, tn): This method returns a Task object that will complete when every t1, …, tn task objects complete. It essentially means that it will asynchronously wait until every given task is completed.

Task.ConfigureAwait(false): It allows the Task object to be executed in the threadpool, and not necessarily in the same thread the Task object was created.

This is done to prevent performance issues as well as possible deadlocks.

And the following test method:

```
[Fact]
public async Task OperateTest()
{
    var result = await new Test().Operate().ConfigureAwait(false);

    Assert.True(result, "Result is not true");
}
```

After a rewrite, the following test methods would be in the dll:

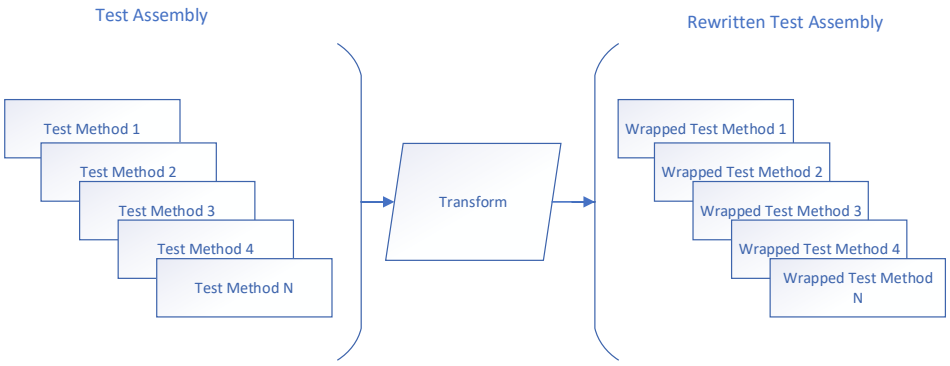
```
[XunitCoyoteRewritten]
public async Task OperateTest__inner()
{
    bool result = await ControlledTask<bool>.ConfigureAwait(new Test().Operate(), false);
    Assert.True(result, "Result is not true");
}

[Fact]
[XunitCoyoteRewritten]
public void OperateTest()
{
    using (XUnitTestWrapper xunitTestWrapper = new XUnitTestWrapper(
        _: true,
        instance: (object) this,
        methodName: "OperateTest__inner",
        methodArgs: (object[]) null))
    XUnitTestTemplates.RunTestInCoyote(new Func<Task>(xunitTestWrapper.Invoke));
}
```

OperateTest now has our wrapper implementation in its body, and the original test case is now called OperateTest__inner.

## 4. IMPLEMENTATION

The idea that's behind all the approaches is to wrap all existing tests in each test assembly in a way such that Coyote can execute the existing test method without having the user modify the original code.



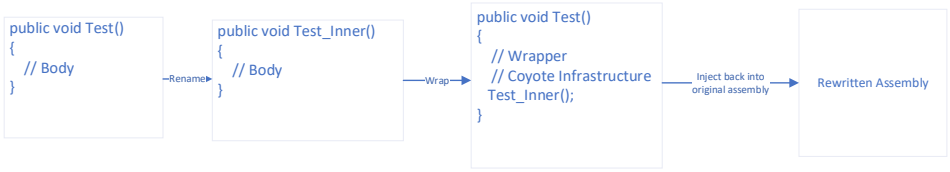

All the attempted approaches had to be able to handle xUnit's possible test types and combinations (Fact, Theory, void test, async Task, async generic, etc).

Possible xUnit test cases type method signatures, as well as their description and the difficulty involved with their rewrite:

| Type | Description | Difficulty |
|---|---|---|
| static async Task test() | Static no parameters test | Easiest case. |
| static async Task test(T1 p1, T2 p2, …) | Static test with parameters | Slightly more complex. |
| async Task test() | No parameters instance test | Compared to the static case, this version gets more complex as I need to figure out a way to create or reference containing object. |
| async Task test() with setup and teardown | No parameters instance test with setup and teardown | More complex than previous case, as creating and disposing the object correctly becomes a necessity in order to make each iteration execution in Coyote independent. |
| async Task test (T1 p1, T2 p2, …) | Instance test with parameters | Same difficulties as no parameters instance test, |

| | | |
|---|---|---|
| | | with the extra complexity of handling arguments. |
| async Task test (T1 p1, T2 p2, …) with setup and teardown | Instance test with parameters, setup and teardown | Same difficulties as no parameters version, with the extra complexity of handling arguments. |
| static async Task test<T1, T2>(T1 p1, T2 p2, …) | Generic static test with parameters | Need to figure out how handle generics in the rewrite super complex. |
| async Task test<T1, T2>(T1 p1, T2 p2, …) | Generic instance test with parameters | More complicated than no generics version due to generics. |
| async Task test<T1, T2>(T1 p1, T2 p2, …) with setup and teardown | Generic instance test with parameters, setup and teardown | More complicated than no generics version due to generics. |

The first approach attempted was trying to modify existing test methods so that their bodies would point to a method within the transform tool, and this method would then execute the original test code. This was attempted in order to avoid writing custom IL (intermediate language, CLR's bytecode) to perform the rewrite.




This brought to light a set of challenges, such as having to reimport all the types within the wrapper method, which is a hard issue to do with generic method calls, and not very extensible, since this would only work on Facts, as if the test is a Theory and you need to pass along parameters, it wouldn't work.

The second approach was a revision of the first in order to handle Theory scenarios. Using the Roslyn [12] compiler's APIs, the new approach consisted of the following:

1. Create a wrapper class which would hold the test method and its arguments. By create, I mean literally generate new code to generate a new type, which would then be instantiated, and it would hold the method and its arguments.
2. Within this class, expose a method called ToFuncTask, which takes no arguments, and returns a Task object. The idea of this is to translate any static async Task test(T1 p1, T2 p2, …, Tn pn) call into a static async Task ToFuncTask() call, which takes no parameters.

- For Fact type of tests, the only step required is replacing the value of the local Func<Task> variable with a reference to the Fact test method.

3. Using Cecil and IL, replace in the theory template a local Func<Task> variable value with a reference to WrappedType.ToFuncTask method.
4. Inject template into test module.

This was a success for the static scenarios as well as the instance scenarios without setup, teardown, and generics.

The issue happens when this method is applied to a test which depends on setup and teardown. This means that the test will change the environment state with its execution, and further executions in this state will probably be incorrect.

After meditating a lot, I decided to take the learnings from these experiments (using wrappers) and try a new approach, which handles every case except for the generic scenarios. This ended up not being a big issue, as the generic scenarios are possible, but not common at all.

In order to prevent issues with generics and type imports, I decided to use pure IL for the rewrite. Now, the body of a rewritten method will look like this for static methods:

```
using XUnitTestWrapper xunitTestWrapper = new XUnitTestWrapper(
    objectType: typeof(method.DeclaringType),
    methodName: method.Name,
    methodArgs: method.HasParameters ? args : null)
{
    XUnitTestTemplates.RunTestInCoyote(new Func<Task>(xunitTestWrapper.Invoke);
}
```

And like this for instance methods:

```
using XUnitTestWrapper xunitTestWrapper = new XUnitTestWrapper(
    _: true,
    instance: this,
    methodName: method.Name,
    methodArgs: method.HasParameters ? args : null)
{
    XUnitTestTemplates.RunTestInCoyote(new Func<Task>(xunitTestWrapper.Invoke);
}
```

The _ parameter in the instance case is used so that the CLR can use the correct constructor overload, as the instance argument has ‘object’ type, which is the base class for every class in C#, including the ‘Type’ class of the objectType argument in the static scenario.

This code is written by using pure IL. The using statement [13] is used to ensure the correct use of objects which implement the IDisposable [14] interface to appropriately dispose of managed resources. XUnitTestWrapper is a class which implements this interface in order to properly dispose of constructor arguments and other managed

state. The using statement is syntactic sugar which translates to a try-finally clause [15]. This guarantees for most cases that the code executed in the finally will run and properly handle lingering resources.

The idea for the XUnitTestWrapper class is to handle object creation, disposal, and test execution, as well as provide a common function interface. XUnitTestWrapper will essentially translate any $F(x_1, x_2, .., x_n) \rightarrow$ Task to F(void) →Task. It has 2 constructors, one to handle instance tests, and the other to handle static tests.

In the Invoke method, the XUnitTestWrapper instance creates the tested object instance if necessary, then executes the test method, and if necessary, disposes of the object instance. This allows each test run to be independent from the previous one. This detail is important as Coyote runs tests several times to explore the problem space. The Invoke method in the wrapper body is used as an argument in the creation of a new Func<Task> object (essentially a function pointer), and it is then passed to the RunTestInCoyote method in the class XUnitTestTemplates, which has all the required code infrastructure to execute the Invoke Func instance in Coyote.

Both XUnitTestWrapper and XUnitTestTemplates are written in C#, when the wrapper tool rewrites the test code it also copies the dlls for both classes into the folder where the rewritten code lives. Therefore, when XUnit executes the tests, it picks up the required classes and methods from the copied DLLs.

Due to these facts, now the tool has the best of both worlds. The use of IL is minimized to only the rewriting, whereas the actual behavior for test wrapping, execution, and execution within Coyote is all handled by code written in C#.

## 5. EXPERIMENTS

Main question: How robust is this tool?

### 1st Experiment, Coyote tests.

Why: This experiment was performed as a sanity check to ensure that the rewritten code would behave the same as Coyote does when presented with different kinds of concurrency bugs.

Setup: Remove Coyote infrastructure from the tests. The tests in general followed a schema similar to the following:

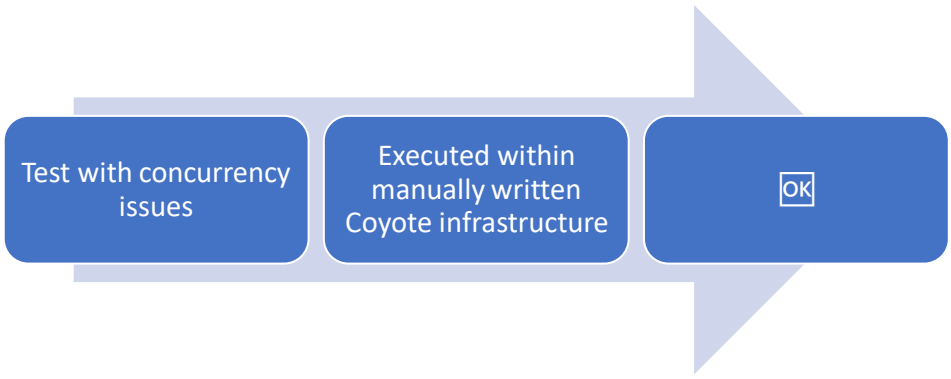
```
public void Test()
{
    this.ExecuteWithinCoyote(
        testMethod: async () => BuggyDoSomething(),
        expectedError: "xxxx",
        configuration: new Configuration(),
        replay: true;
    );
}
```

The ExecuteWithinCoyote method body created all the necessary infrastructure for the testMethod parameter to be executed within the Coyote engine. For our testing, we replace the body of the ExecuteWithinCoyote with only the invocation of the testMethod parameter.

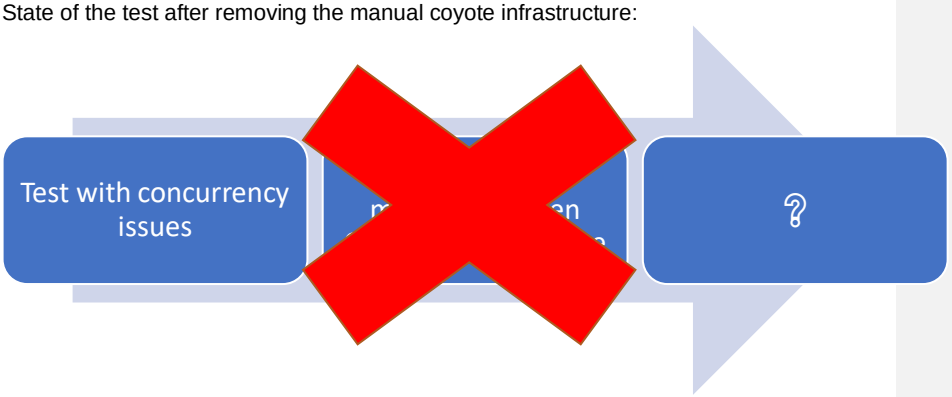
This way, after a rewrite, when executing the test, the testMethod parameter would be executed under our Coyote code injected by the rewrite, allowing us to replicate the original behavior.

Essentially:

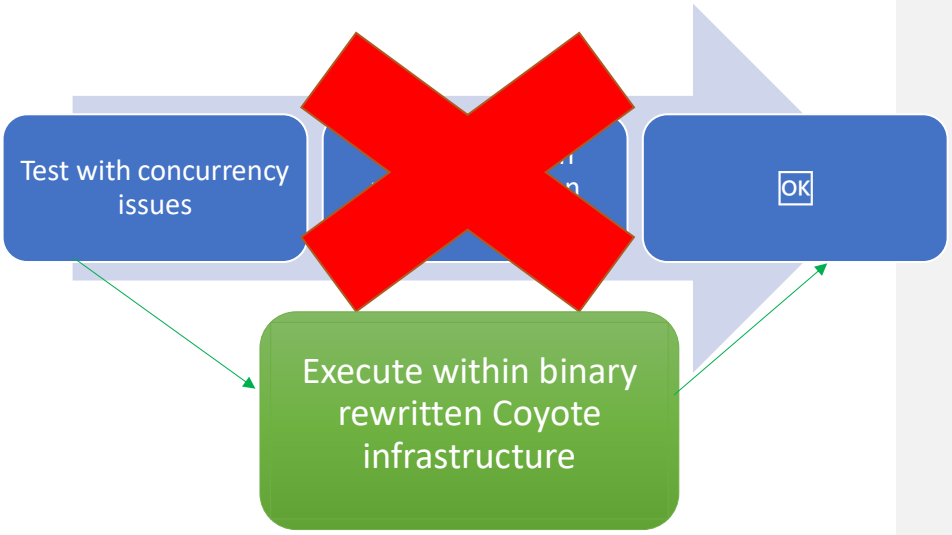
1) Original state of the test.



2) State of the test after removing the manual coyote infrastructure:




3) Test after rewrite:



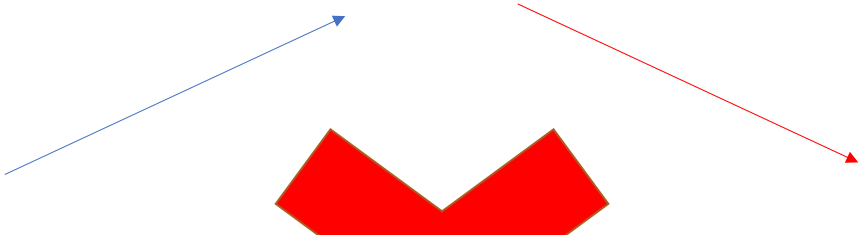

Results: All tests were evaluated and had successful results, as they failed as was expected and the bugs they had were found.

Analysis: From these results, we can conclude that the tool is properly working as expected, at least within the context of validation of coyote itself, and tests are getting executed in Coyote.

Commented [GU18]: conclude

Commented [GU19]: as expected





### 2nd Experiment, .Net Concurrent Collections test (System.Collections.Concurrent namespace).

Why: We performed this experiment to test how our rewriter performed with code on the wild.

Setup: Rewrite code assembly using coyote instrument certain concurrent data structures and emulate others using a combination of active wait and concurrent variables.

Results: Except for some tests that weren't possible to run due to some concurrent primitives not supported by Coyote (volatile read, for example), all of the ones that were able to be executed succeeded. This does not mean that it's impossible for concurrency bugs to exist, only that Coyote wasn't able to find any.

### 3rd Experiment, PowerShell

Why: We performed this experiment to test how our rewriter performed with code on the wild.

Setup: Rewrite code assembly using coyote instrument certain concurrent data structures and emulate others using a combination of active wait and concurrent variables (ManualResetEventSlim for this scenario was emulated).

Results: Except for some tests that weren't possible to run due to some concurrent primitives not supported by Coyote (volatile read, for example), most of the ones that were able to be executed succeeded, except for 2 in the CommandPredictionTests class.

One, PredictInput, is badly designed, it depends on task's delays to validate its results, however that is not deterministic as the scheduler can execute stuff however it prefers.

The other, Feedback, is missing a wait to ensure that the Fast predictor (an object in the test), is done before validating.

This does not mean that it's impossible for concurrency bugs to exist, only that Coyote wasn't able to find any.

## 6. CONCLUSION

In conclusion, the experiments have shown that the rewrites performed by the tool are correct as far as Coyote can be correct, due to the Coyote tests, and they show that the tool can be used with other heavily used and popular libraries such as System.Collections.Concurrent in the C# runtime and projects, such as Powershell, even detecting some issues in the way some tests were written in the latter case.
The user interaction to enable this new functionality is minimal, only running 2 commands over the dll which has the tests, a Coyote rewrite, in order to replace common types used in concurrency with types Coyote can manipulate, and a rewrite by the tool presented in this work so that the regular tests written using the xUnit test framework can be executed within the Coyote tool runtime.
In exchange for executing these commands, the user gets concurrency checks in their code, simplifying and speeding up code development.
As future work, a more in depth analysis could be done to find bugs in live codebases, as in this case the only issues found were 2 badly written tests, which makes sense, as one would expect the concurrent collections library in C# and the Powershell console to be robust.

## 7. APPENDIX, RESULTS

**Coyote Tests:**

| Test | Result | Comment |
|---|---|---|
| Microsoft.Coyote.SystematicTesting.Tests.Threading.LockStatementTests.TestSimpleLock | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Threading.LockStatementTests.TestWaitPulse | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Threading.LockStatementTests.TestMonitorWithLockTaken | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Threading.MonitorTests.TestSimpleMonitor | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Threading.MonitorTests.TestMonitorWithReentrancy1 | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Threading.MonitorTests.TestMonitorWithReentrancy2 | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Threading.MonitorTests.TestMonitorWithReentrancy3 | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Threading.MonitorTests.TestMonitorWithInvalidSyncObject | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Threading.MonitorTests. | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Threading.MonitorTests.TestMonitorWithInvalidPulseState | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Threading.MonitorTests.TestMonitorWithInvalidPulseAllState | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Threading.MonitorTests.TestMonitorWithInvalidUsage | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Threading.MonitorTests.TestComplexMonitor | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Threading.ThreadPoolTests.TestQueueUserWorkItem | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Threading.ThreadPoolTests.TestQueueUserWorkItemAsync | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Threading.ThreadPoolTests.TestQueueUserWorkItemWithException | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Threading.ThreadPoolTests.TestQueueUserWorkItemWithAsyncException | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Threading.ThreadPoolTests.TestUnsafeQueueUserWorkItem | TRUE | |

| | | |
|---|---|---|
| Microsoft.Coyote.SystematicTesting.Tests.Threading.ThreadPoolTests.TestUnsafeQueueUserWorkItemAsync | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Threading.ThreadPoolTests.TestUnsafeQueueUserWorkItemWithException | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Threading.ThreadPoolTests.TestUnsafeQueueUserWorkItemWithAsyncException | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Threading.ThreadTests.TestThreadStartJoin | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Exceptions.ExceptionFilterRewritingTests. | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Exceptions.ExceptionFilterRewritingTests. | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Exceptions.ExceptionFilterRewritingTests.TestExecutionCanceledExceptionDoubleRethrow | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Exceptions.ExceptionFilterRewritingTests | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Exceptions.ExceptionFilterRewritingTests.TestExecutionCanceledExceptionInDoubleEmptyCatchBlock | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Exceptions.ExceptionFilterRewritingTests. | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Exceptions.ExceptionFilterRewritingTests.TestExecutionCanceledExceptionInNonEmptyCatchBlock | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Exceptions.ExceptionFilterRewritingTests.TestExecutionCanceledExceptionInNonEmptyCatchBlockAsync | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Exceptions.ExceptionFilterRewritingTests. | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Exceptions.TaskExceptionFilterTests.TestAddExceptionFilter | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Exceptions.TaskExceptionFilterTests.TestAddExceptionFilter2 | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Exceptions.TaskExceptionFilterTests.TestAddExceptionFilter3 | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Exceptions.TaskExceptionFilterTests.TestAddExceptionFilter4 | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Exceptions.TaskExceptionFilterTests.TestAddExceptionFilter5 | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Exceptions.TaskExceptionFilterTests.TestAddExceptionFilter6 | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Exceptions.TaskExceptionFilterTests.TestEditComplexFilter2 | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Exceptions.TaskExceptionFilterTests.TestEditComplexFilter3 | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Exceptions.TaskExceptionFilterTests.TestEditComplexFilter4 | FALSE | Expected to fail |
| Microsoft.Coyote.SystematicTesting.Tests.Exceptions.TaskExceptionFilterTests.TestIgnoreRethrowCase | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Exceptions.TaskExceptionFilterTests.TestIgnoreRethrowCase2 | TRUE | |

| | | |
|---|---|---|
| Microsoft.Coyote.SystematicTesting.Tests.Exceptions.TaskExceptionFilterTests.TestConditionalTryCatch | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Exceptions.TaskExceptionFilterTests.TestMultiCatchBlock | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Exceptions.TaskExceptionFilterTests.TestMultiCatchFilter | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Exceptions.TaskExceptionFilterTests.TestMultiCatchBlockWithFilter | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Exceptions.TaskExceptionFilterTests.TestExceptionHandlerInsideLock | FALSE | Expected to fail |
| Microsoft.Coyote.SystematicTesting.Tests.Exceptions.TaskExceptionFilterTests.TestTryUsingTry | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Configuration.ConfigurationTests.TestJsonConfigurationReplacingBinaries | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Configuration.ConfigurationTests. | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.PollingTaskLivenessTests. | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.PollingTaskLivenessTests.TestPollingTaskLivenessPropertyWithDoubleDelay | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.PollingTaskLivenessTests.TestPollingTaskLivenessPropertyFailure | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.AsyncInvocationTests. | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.AsyncInvocationTests.TestCompletedTask | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.CompletedTaskTests.TestCompletedTask | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.CompletedTaskTests.TestCanceledTask | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.CompletedTaskTests.TestCanceledTaskWithResult | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.CompletedTaskTests.TestFailedTask | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.CompletedTaskTests.TestFailedTaskWithResult | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskConfigureAwaitFalseTests.TestAwaitSynchronousTask | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskConfigureAwaitFalseTests.TestAwaitSynchronousTaskFailure | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskConfigureAwaitFalseTests.TestAwaitAsynchronousTask | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskConfigureAwaitFalseTests.TestAwaitAsynchronousTaskFailure | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskConfigureAwaitFalseTests.TestAwaitNestedSynchronousTask | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskConfigureAwaitFalseTests.TestAwaitNestedSynchronousTaskFailure | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskConfigureAwaitFalseTests.TestAwaitNestedAsynchronousTask | TRUE | |

| | | |
|---|---|---|
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskConfigureAwaitFalseTests.TestAwaitNestedAsynchronousTaskFailure | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskConfigureAwaitFalseTests.TestAwaitSynchronousTaskWithResult | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskConfigureAwaitFalseTests.TestAwaitSynchronousTaskWithResultFailure | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskConfigureAwaitFalseTests. | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskConfigureAwaitFalseTests.TestAwaitAsynchronousTaskWithResultFailure | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskConfigureAwaitFalseTests.TestAwaitNestedSynchronousTaskWithResult | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskConfigureAwaitFalseTests.TestAwaitNestedSynchronousTaskWithResultFailure | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskConfigureAwaitFalseTests.TestAwaitNestedAsynchronousTaskWithResult | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskConfigureAwaitFalseTests.TestAwaitNestedAsynchronousTaskWithResultFailure | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskConfigureAwaitTrueTests.TestAwaitSynchronousTask | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskConfigureAwaitTrueTests.TestAwaitSynchronousTaskFailure | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskConfigureAwaitTrueTests.TestAwaitAsynchronousTask | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskConfigureAwaitTrueTests.TestAwaitAsynchronousTaskFailure | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskConfigureAwaitTrueTests.TestAwaitNestedSynchronousTask | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskConfigureAwaitTrueTests.TestAwaitNestedSynchronousTaskFailure | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskConfigureAwaitTrueTests.TestAwaitNestedAsynchronousTask | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskConfigureAwaitTrueTests.TestAwaitNestedAsynchronousTaskFailure | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskConfigureAwaitTrueTests.TestAwaitSynchronousTaskWithResult | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskConfigureAwaitTrueTests.TestAwaitSynchronousTaskWithResultFailure | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskConfigureAwaitTrueTests.TestAwaitAsynchronousTaskWithResult | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskConfigureAwaitTrueTests.TestAwaitAsynchronousTaskWithResultFailure | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskConfigureAwaitTrueTests.TestAwaitNestedSynchronousTaskWithResult | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskConfigureAwaitTrueTests.TestAwaitNestedSynchronousTaskWithResultFailure | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskConfigureAwaitTrueTests.TestAwaitNestedAsynchronousTaskWithResult | TRUE | |

| | | |
|---|---|---|
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskConfigureAwaitTrueTests.TestAwaitNestedAsynchronousTaskWithResultFailure | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRunConfigureAwaitFalseTests.TestRunParallelTask | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRunConfigureAwaitFalseTests.TestRunParallelTaskFailure | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRunConfigureAwaitFalseTests.TestRunParallelSynchronousTask | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRunConfigureAwaitFalseTests.TestRunParallelSynchronousTaskFailure | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRunConfigureAwaitFalseTests.TestRunParallelAsynchronousTask | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRunConfigureAwaitFalseTests.TestRunParallelAsynchronousTaskFailure | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRunConfigureAwaitFalseTests.TestRunNestedParallelSynchronousTask | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRunConfigureAwaitFalseTests.TestAwaitNestedParallelSynchronousTaskFailure | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRunConfigureAwaitFalseTests.TestAwaitNestedParallelAsynchronousTask | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRunConfigureAwaitFalseTests.TestAwaitNestedParallelAsynchronousTaskFailure | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRunConfigureAwaitFalseTests.TestRunParallelTaskWithResult | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRunConfigureAwaitFalseTests.TestRunParallelTaskWithResultFailure | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRunConfigureAwaitFalseTests. | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRunConfigureAwaitFalseTests.TestRunParallelSynchronousTaskWithResultFailure | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRunConfigureAwaitFalseTests.TestRunParallelAsynchronousTaskWithResult | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRunConfigureAwaitFalseTests. | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRunConfigureAwaitFalseTests.TestRunNestedParallelSynchronousTaskWithResult | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRunConfigureAwaitFalseTests.TestRunNestedParallelSynchronousTaskWithResultFailure | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRunConfigureAwaitFalseTests.TestRunNestedParallelAsynchronousTaskWithResult | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRunConfigureAwaitFalseTests.TestRunNestedParallelAsynchronousTaskWithResultFailure | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRunConfigureAwaitTrueTests.TestRunParallelTask | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRunConfigureAwaitTrueTests.TestRunParallelTaskFailure | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRunConfigureAwaitTrueTests.TestRunParallelSynchronousTask | TRUE | |

| | | |
|---|---|---|
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRunConfigureAwaitTrueTests.TestRunParallelSynchronousTaskFailure | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRunConfigureAwaitTrueTests.TestRunParallelAsynchronousTask | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRunConfigureAwaitTrueTests.TestRunParallelAsynchronousTaskFailure | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRunConfigureAwaitTrueTests.TestRunNestedParallelSynchronousTask | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRunConfigureAwaitTrueTests.TestAwaitNestedParallelSynchronousTaskFailure | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRunConfigureAwaitTrueTests.TestAwaitNestedParallelAsynchronousTask | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRunConfigureAwaitTrueTests.TestAwaitNestedParallelAsynchronousTaskFailure | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRunConfigureAwaitTrueTests.TestRunParallelTaskWithResult | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRunConfigureAwaitTrueTests.TestRunParallelTaskWithResultFailure | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRunConfigureAwaitTrueTests.TestRunParallelSynchronousTaskWithResult | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRunConfigureAwaitTrueTests.TestRunParallelSynchronousTaskWithResultFailure | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRunConfigureAwaitTrueTests. | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRunConfigureAwaitTrueTests.TestRunParallelAsynchronousTaskWithResultFailure | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRunConfigureAwaitTrueTests.TestRunNestedParallelSynchronousTaskWithResult | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRunConfigureAwaitTrueTests.TestRunNestedParallelSynchronousTaskWithResultFailure | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRunConfigureAwaitTrueTests.TestRunNestedParallelAsynchronousTaskWithResult | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRunConfigureAwaitTrueTests.TestRunNestedParallelAsynchronousTaskWithResultFailure | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.ExtremeProgrammingChallenge14Tests.TestChallenge14WithDeadlock | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.ExtremeProgrammingChallenge14Tests.TestChallenge14WithFix | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskResultTests.TestParallelTaskResultBeforeWrite | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskResultTests. | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskResultTests.TestParallelTaskResultWithSynchronousInvocationAfterWrite | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskResultTests.TestParallelTaskResultWithAsynchronousInvocationAfterWrite | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskResultTests.TestTaskResultWithException | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskResultTests. | TRUE | |

| | | |
|---|---|---|
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskWaitAllTests.TestWaitAllWithTwoSynchronousTasks | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskWaitAllTests.TestWaitAllWithTwoAsynchronousTasks | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskWaitAllTests. | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskWaitAllTests.TestWaitAllWithTwoSynchronousTaskWithResults | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskWaitAllTests.TestWaitAllWithTwoAsynchronousTaskWithResults | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskWaitAllTests.TestWaitAllWithTwoParallelSynchronousTaskWithResults | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskWaitAllTests.TestWaitAllWithTwoParallelAsynchronousTaskWithResults | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskWaitAllTests.TestWaitAllDeadlock | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskWaitAllTests.TestWaitAllWithResultsAndDeadlock | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskWaitAnyTests.TestWaitAnyWithTwoSynchronousTasks | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskWaitAnyTests.TestWaitAnyWithTwoAsynchronousTasks | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskWaitAnyTests. | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskWaitAnyTests.TestWaitAnyWithTwoSynchronousTaskWithResults | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskWaitAnyTests.TestWaitAnyWithTwoAsynchronousTaskWithResults | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskWaitAnyTests.TestWaitAnyWithTwoParallelSynchronousTaskWithResults | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskWaitAnyTests.TestWaitAnyWithTwoParallelAsynchronousTaskWithResults | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskWaitAnyTests.TestWaitAnyWithIncompleteTask | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskWaitAnyTests.TestWaitAnyWithIncompleteGenericTask | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskWaitTests.TestWaitParallelTaskBeforeWrite | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskWaitTests.TestWaitParallelTaskAfterWrite | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskWaitTests.TestWaitParallelTaskWithSynchronousInvocationAfterWrite | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskWaitTests.TestWaitParallelTaskWithAsynchronousInvocationAfterWrite | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskWaitTests.TestWaitTaskWithTimeout | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskWaitTests.TestWaitTaskWithCancellationToken | TRUE | |

| | | |
|---|---|---|
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskWaitTests.TestWaitTaskWithTimeoutAndCancellationToken | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskWaitTests.TestWaitTaskWithException | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskWaitTests.TestWaitAsyncTaskWithException | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskWhenAllTests.TestWhenAllWithTwoSynchronousTasks | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskWhenAllTests.TestWhenAllWithTwoAsynchronousTasks | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskWhenAllTests. | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskWhenAllTests.TestWhenAllWithTwoSynchronousTaskWithResults | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskWhenAllTests.TestWhenAllWithTwoAsynchronousTaskWithResults | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskWhenAllTests.TestWhenAllWithTwoParallelSynchronousTaskWithResults | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskWhenAllTests.TestWhenAllWithTwoParallelAsynchronousTaskWithResults | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskWhenAllTests.TestWhenAllWithAsyncCaller | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskWhenAllTests.TestWhenAllWithResultAndAsyncCaller | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskWhenAllTests.TestWhenAllWithException | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskWhenAllTests.TestWhenAllWithResultsAndException | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskWhenAllTests.TestWhenAllDeadlock | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskWhenAllTests.TestWhenAllWithResultsAndDeadlock | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskWhenAnyTests.TestWhenAnyWithTwoSynchronousTasks | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskWhenAnyTests.TestWhenAnyWithTwoAsynchronousTasks | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskWhenAnyTests. | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskWhenAnyTests.TestWhenAnyWithTwoSynchronousTaskWithResults | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskWhenAnyTests.TestWhenAnyWithTwoAsynchronousTaskWithResults | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskWhenAnyTests.TestWhenAnyWithTwoParallelSynchronousTaskWithResults | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskWhenAnyTests.TestWhenAnyWithTwoParallelAsynchronousTaskWithResults | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskWhenAnyTests.TestWhenAnyWithAsyncCaller | TRUE | |

| | | |
|---|---|---|
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskWhenAnyTests.TestWhenAnyWithResultAndAsyncCaller | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskWhenAnyTests.TestWhenAnyWithException | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskWhenAnyTests.TestWhenAnyWithResultsAndException | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskWhenAnyTests.TestWhenAnyWithIncompleteTask | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskWhenAnyTests.TestWhenAnyWithIncompleteGenericTask | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.CustomTaskLogTests.TestCustomLogger | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.CustomTaskLogTests.TestCustomTaskRuntimeLog | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRandomBooleanTests.TestRandomBooleanInSynchronousTask | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRandomBooleanTests.TestRandomBooleanInAsynchronousTask | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRandomBooleanTests.TestRandomBooleanInParallelTask | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRandomBooleanTests.TestRandomBooleanInParallelSynchronousTask | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRandomBooleanTests.TestRandomBooleanInParallelAsynchronousTask | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRandomBooleanTests.TestRandomBooleanInNestedParallelSynchronousTask | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRandomIntegerTests.TestRandomIntegerInSynchronousTask | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRandomIntegerTests.TestRandomIntegerInAsynchronousTask | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRandomIntegerTests.TestRandomIntegerInParallelTask | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRandomIntegerTests.TestRandomIntegerInParallelSynchronousTask | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRandomIntegerTests.TestRandomIntegerInParallelAsynchronousTask | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRandomIntegerTests.TestRandomIntegerInNestedParallelSynchronousTask | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.AsyncCallStackTests.TestExpectedAsyncCallStackSize | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskCancellationTests.TestAlreadyCanceledParallelTask | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskCancellationTests.TestAlreadyCanceledAsynchronousTask | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskCancellationTests.TestAlreadyCanceledParallelTaskWithResult | TRUE | |

| | | |
|---|---|---|
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskCancellationTests.TestAlreadyCanceledAsynchronousTaskWithResult | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskCancellationTests.TestCancelParallelTask | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskCancellationTests. | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskCancellationTests.TestCancelParallelTaskWithResult | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskCancellationTests.TestCancelAsynchronousTaskWithResult | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskCancellationTests.TestAwaitNestedAsynchronousTask | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskCancellationTests. | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskCompletionSourceTests.TestSetResult | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskCompletionSourceTests.TestTrySetResult | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskCompletionSourceTests.TestAsynchronousSetResult | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskCompletionSourceTests.TestAsynchronousSetResultTask | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskCompletionSourceTests.TestAsynchronousSetResultWithTwoAwaiters | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskCompletionSourceTests.TestSetCanceled | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskCompletionSourceTests.TestTrySetCanceled | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskCompletionSourceTests.TestAsynchronousSetCanceled | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskCompletionSourceTests.TestSetException | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskCompletionSourceTests.TestTrySetException | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskCompletionSourceTests.TestAsynchronousSetException | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskCompletionSourceTests.TestInvalidSetResult | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskCompletionSourceTests. | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskCompletionSourceTests.TestInvalidSetException | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskCompletionSourceTests.TestIsCompleted | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskDelayTests.TestInterleavingsInLoopWithSynchronousDelays | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskDelayTests.TestInterleavingsInLoopWithAsynchronousDelays | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskDelayTests. | TRUE | |

| | | |
|---|---|---|
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskDelayTests.TestInterleavingsInNestedAsynchronousDelay | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskDelayTests.TestInterleavingsInNestedSynchronousDelays | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskDelayTests.TestInterleavingsInNestedAsynchronousDelays | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskDelayTests.TestInterleavingsInRepeatedNestedSynchronousDelays | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskDelayTests.TestInterleavingsInRepeatedNestedAsynchronousDelays | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskDelayTests.TestInterleavingsInLambdaSynchronousDelays | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskDelayTests.TestInterleavingsInLambdaAsynchronousDelays | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskDelayTests. | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskDelayTests.TestInterleavingsInLocalFunctionAsynchronousDelays | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskDelayTests.TestParallelInterleavingsInNestedSynchronousDelays | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskDelayTests.TestParallelInterleavingsInNestedAsynchronousDelays | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskExceptionTests.TestNoSynchronousTaskExceptionStatus | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskExceptionTests. | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskExceptionTests.TestNoParallelSynchronousTaskExceptionStatus | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskExceptionTests. | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskExceptionTests. | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskExceptionTests.TestSynchronousTaskExceptionStatus | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskExceptionTests.TestAsynchronousTaskExceptionStatus | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskExceptionTests.TestParallelSynchronousTaskExceptionStatus | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskExceptionTests.TestParallelAsynchronousTaskExceptionStatus | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskExceptionTests.TestParallelFuncTaskExceptionStatus | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskFactoryTests.TestStartNewTask | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskFactoryTests.TestStartNewTaskWithSynchronousAwait | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskFactoryTests.TestStartNewTaskWithAsynchronousAwait | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskFactoryTests.TestStartNewNestedTaskWithSynchronousAwait | TRUE | |

| | | |
|---|---|---|
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskFactoryTests.TestStartNewNestedTaskWithAsynchronousAwait | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskFactoryTests.TestStartNewTaskWithResult | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskFactoryTests.TestStartNewTaskWithSynchronousResult | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskFactoryTests.TestStartNewTaskWithAsynchronousResult | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskFactoryTests.TestStartNewNestedTaskWithSynchronousResult | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskFactoryTests.TestStartNewNestedTaskWithAsynchronousResult | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskFactoryTests.TestGenericStartNewTaskWithResult | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskFactoryTests.TestGenericStartNewTaskWithSynchronousResult | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskFactoryTests.TestGenericStartNewTaskWithAsynchronousResult | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskFactoryTests.TestGenericStartNewNestedTaskWithSynchronousResult | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskFactoryTests.TestGenericStartNewNestedTaskWithAsynchronousResult | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskFactoryTests.TestStartNewCanceledTask | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskFactoryTests.TestStartNewTaskCancelation | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskFactoryTests.TestStartNewCanceledTaskWithAsynchronousAwait | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskFactoryTests.TestStartNewTaskCancelationWithAsynchronousAwait | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskIdTests.TestExpectedIdInTaskWithAction | SKIPPED | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskIdTests.TestExpectedIdInTaskWithFunction | SKIPPED | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskIdTests. | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskIdTests.TestExpectedIdInTaskWithGenericAsynchronousFunction | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskInterfaceTests.TestAsyncInterfaceMethodCall | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskInterleavingsTests.TestInterleavingsWithOneSynchronousTask | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskInterleavingsTests.TestInterleavingsWithOneAsynchronousTask | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskInterleavingsTests.TestInterleavingsWithOneParallelTask | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskInterleavingsTests.TestInterleavingsWithTwoSynchronousTasks | TRUE | |

| | | |
|---|---|---|
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskInterleavingsTests. | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskInterleavingsTests.TestInterleavingsWithTwoParallelTasks | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskInterleavingsTests.TestInterleavingsWithNestedParallelTasks | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskInterleavingsTests.TestExploreAllInterleavings | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskParallelTests.TestParallelFor | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskParallelTests.TestParallelForEach | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRunTests. | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRunTests.TestRunParallelTaskFailure | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRunTests.TestRunParallelSynchronousTask | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRunTests.TestRunParallelSynchronousTaskFailure | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRunTests.TestRunParallelAsynchronousTask | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRunTests.TestRunParallelAsynchronousTaskFailure | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRunTests.TestRunNestedParallelSynchronousTask | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRunTests.TestAwaitNestedParallelSynchronousTaskFailure | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRunTests.TestAwaitNestedParallelAsynchronousTask | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRunTests.TestAwaitNestedParallelAsynchronousTaskFailure | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRunTests.TestRunParallelTaskWithResult | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRunTests.TestRunParallelTaskWithResultFailure | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRunTests.TestRunParallelSynchronousTaskWithResult | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRunTests.TestRunParallelSynchronousTaskWithResultFailure | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRunTests.TestRunParallelAsynchronousTaskWithResult | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRunTests.TestRunParallelAsynchronousTaskWithResultFailure | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRunTests.TestRunNestedParallelSynchronousTaskWithResult | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRunTests.TestRunNestedParallelSynchronousTaskWithResultFailure | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRunTests.TestRunNestedParallelAsynchronousTaskWithResult | TRUE | |

| | | |
|---|---|---|
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskRunTests.TestRunNestedParallelAsynchronousTaskWithResultFailure | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskYieldTests.TestTaskYield | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskYieldTests. | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskYieldTests.TestParallelTaskYield | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskYieldTests. | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskYieldTests.TestTwoParallelTasksWriteWithYield | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskYieldTests.TestTwoParallelTasksWriteWithYieldFail | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskYieldTests.TestTwoAsynchronousTasksWriteWithYieldFail | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskYieldTests.TestInterleavingsInNestedYield | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskYieldTests.TestInterleavingsInNestedYields | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskYieldTests.TestInterleavingsInLambdaYields | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskYieldTests.TestInterleavingsInLocalFunctionYields | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.TaskYieldTests.TestInterleavingsInNestedParallelYields | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.Generics.GenericTests.TestGenericClass | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.Generics.GenericTests.TestGenericNestedClass | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.Generics.GenericTests.TestGenericNestedMethod | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Tasks.Generics.GenericTests.TestGenericResult | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Invocations.NotSupportedInvocationsTests.TestNotSupportedContinueWithTaskInvocation | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Invocations.NotSupportedInvocationsTests.TestNotSupportedValueTaskInvocation | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Invocations.NotSupportedInvocationsTests.TestNotSupportedTimerInvocation | TRUE | |
| Microsoft.Coyote.SystematicTesting.Tests.Invocations.UncontrolledTaskTests.TestDetectedUncontrolledDelay | TRUE | |

**Net Concurrent Collections tests:**

| Test class | Passed | Failed | Fail reason |
|---|---|---|---|

| | | | |
|---|---|---|---|
| ConcurrentDictionaryTests | 58 | 1 | Not supported by coyote |
| ConcurrentDictionary (Includes variations) | 1622 | 1 | Not supported by coyote |
| ConcurrentStackTests.IsEmpty_TrueWhenEmpty_FalseWhenNot | All | | |
| ConcurrentStackTests.Clear_AllElementsRemoved | All | | |
| ConcurrentStackTests.IsEmpty_TrueWhenEmpty_FalseWhenNot | All | | |
| ConcurrentStackTests.PushRange_InvalidArguments_Throws | All | | |
| ConcurrentStackTests.PushRange_NoItems_NothingAdded | All | | |
| ConcurrentStackTests.TryPopRange_InvalidArguments_Throws | All | | |
| ConcurrentStackTests.Add_TakeFromAnotherThread_ExpectedItemsTaken | All | | |
| ConcurrentStackTests.AddTakeWithAtLeastOneElementInCollection_IsEmpty_AlwaysFalse | All | | |
| ConcurrentStackTests.BlockingCollection_WrappingCollection_ExpectedElementsTransferred | All | | |
| ConcurrentQueueTests.BlockingCollection_WrappingCollection_ExpectedElementsTransferred | All | | |
| ConcurrentQueueTests.Clear_CountMatch | All | | |
| ConcurrentQueueTests.Clear_DuringEnumeration_DoesntAffectEnumeration | All | | |
| ConcurrentQueueTests.Concurrent_Clear_NoExceptions | All | | |
| ConcurrentQueueTests.CopyTo_AllItemsCopiedAtCorrectLocation | All | | |
| ConcurrentQueueTests.IEnumerable_GetAllExpectedItems | All | | |
| ConcurrentQueueTests.ManySegments_ConcurrentDequeues_RemainsConsistent | All | | |
| ConcurrentQueueTests.ManySegments_ConcurrentEnqueues_RemainsConsistent | All | | |
| ProducerConsumerCollectionTests.AddFromMultipleThreads_ItemsRemainAfterThreadsGoAway | All | | |
| ProducerConsumerCollectionTests.Ctor_InvalidArgs_Throws | All | | |
| ProducerConsumerCollectionTests.Ctor_NoArg_ItemsAndCountMatch | All | | |
| ProducerConsumerCollectionTests.Ctor_Collection_ItemsAndCountMatch | All | | |
| ProducerConsumerCollectionTests.Ctor_InitializeFromCollection_ContainsExpectedItems | All | | |
| ProducerConsumerCollectionTests.AddSome_ThenInterleaveAddsAndTakes_MatchesOracle | All | | |
| ProducerConsumerCollectionTests.AddTake_RandomChangesMatchOracle | All | | |
| ProducerConsumerCollectionTests.ToArray_AddAllItemsThenEnumerate_ItemsAndCountMatch | All | | |
| ProducerConsumerCollectionTests.ToArray_AddAndTakeItemsIntermixedWithEnumeration_ItemsAndCountMatch | All | | |
| ProducerConsumerCollectionTests.AddFromMultipleThreads_ItemsRemainAfterThreadsGoAway | All | | |

| | | | |
|---|---|---|---|
| ProducerConsumerCollectionTests.AddManyItems_ThenTakeOnSameThread_ItemsOutputInExpectedOrder | All | | |
| ProducerConsumerCollectionTests.TryPeek_SucceedsOnEmptyCollectionThatWasOnceNonEmpty | All | | |
| ProducerConsumerCollectionTests.CopyTo_Empty_NothingCopied | All | | |
| ProducerConsumerCollectionTests.CopyTo_SzArray_ZeroIndex_ExpectedElementsCopied | All | | |
| ProducerConsumerCollectionTests.CopyTo_SzArray_NonZeroIndex_ExpectedElementsCopied | All | | |
| ProducerConsumerCollectionTests.CopyTo_ArrayZeroLowerBound_ZeroIndex_ExpectedElementsCopied | All | | |
| ProducerConsumerCollectionTests.CopyTo_ArrayNonZeroLowerBound_ExpectedElementsCopied | All | | |
| ProducerConsumerCollectionTests.CopyTo_InvalidArgs_Throws | All | | |
| ProducerConsumerCollectionTests.ICollectionCopyTo_InvalidArgs_Throws | All | | |
| ProducerConsumerCollectionTests.GetEnumerator_NonGeneric | All | | |
| ProducerConsumerCollectionTests.GetEnumerator_EnumerationsAreSnapshots | All | | |
| ProducerConsumerCollectionTests.GetEnumerator_Generic_ExpectedElementsYielded | All | | |
| ProducerConsumerCollectionTests.TryAdd_TryTake_ToArray | All | | |
| ProducerConsumerCollectionTests.ICollection_IsSynchronized_SyncRoot | All | | |
| ProducerConsumerCollectionTests.ToArray_ParallelInvocations_Succeed | All | | |
| ProducerConsumerCollectionTests.ToArray_AddTakeSameThread_ExpectedResultsAfterAddsAndTakes | All | | |
| ProducerConsumerCollectionTests.GetEnumerator_ParallelInvocations_Succeed | All | | |
| ProducerConsumerCollectionTests.ManyConcurrentAddsTakes_CollectionRemainsConsistent | All | | |
| ProducerConsumerCollectionTests.ManyConcurrentAddsTakesPeeks_ForceContentionWithOtherThreadsTaking | All | | |
| ProducerConsumerCollectionTests.ManyConcurrentAddsTakesPeeks_ForceContentionWithOtherThreadsPeeking | All | | |
| ProducerConsumerCollectionTests.ManyConcurrentAddsTakes_ForceContentionWithToArray | All | | |
| ProducerConsumerCollectionTests.ManyConcurrentAddsTakes_ForceContentionWithGetEnumerator | All | | |
| ProducerConsumerCollectionTests.DebuggerAttributes_Success | All | | |
| ProducerConsumerCollectionTests.DebuggerTypeProxy_Ctor_NullArgument_Throws | All | | |
| ConcurrentStackTests.PushRange_Concurrent_ConsecutiveItemsInEachRange | | All | Not supported by coyote (StartNew) |
| ConcurrentStackTests.Initialize_ThenTakeOrPeekInParallel_ItemsObtainedAsExpected | | All | Hang, Barrier |

| | | | |
|---|---|---|---|
| ConcurrentStackTests.TryPopRange_Concurrent_PoppedItemsAreConsecutive | | All | Not supported by coyote (StartNew) |
| ConcurrentStackTests.ManyConcurrentAddsTakes_EnsureTrackedCountsMatchResultingCollection | | All | Hang, Barrier |
| ConcurrentQueueTests.Add_TakeFromAnotherThread_ExpectedItemsTaken | | All | Hang, uses volatile |
| ConcurrentQueueTests.AddTakeWithAtLeastOneElementInCollection_IsEmpty_AlwaysFalse | | All | Hang, uses volatile |
| ConcurrentQueueTests.MultipleProducerConsumer_AllItemsTransferred | | All | Not supported by coyote (StartNew) |
| ConcurrentQueueTests.Concurrent_Enqueue_TryDequeue_AllItemsReceived | | All | Hang, uses volatile |
| ConcurrentQueueTests.Concurrent_Enqueue_TryPeek_TryDequeue_AllItemsSeen | | All | Hang, uses volatile |
| ConcurrentQueueTests.ReferenceTypes_NulledAfterDequeue | | All | Not supported by coyote (Manual Reset Event Slim) |
| ProducerConsumerCollectionTests.Add_TakeFromAnotherThread_ExpectedItemsTaken | | All | Hang |
| ProducerConsumerCollectionTests.Initialize_ThenTakeOrPeekInParallel_ItemsObtainedAsExpected | | All | Hang, Barrier |
| ProducerConsumerCollectionTests.AddTakeWithAtLeastOneElementInCollection_IsEmpty_AlwaysFalse | | All | Hang |
| ProducerConsumerCollectionTests.BlockingCollection_WrappingCollection_ExpectedElementsTransferred | | All | Hang |
| ProducerConsumerCollectionTests.ManyConcurrentAddsTakes_EnsureTrackedCountsMatchResultingCollection | | All | Hang, Barrier |

**Powershell tests:**

| Test class | Status | Fail reason |
|---|---|---|
| PSTests.Parallel.NamedPipeTests.TestCustomPipeNameCreation | Stuck | Calls win32 api, not supported by Coyote |
| PSTests.Parallel.NamedPipeTests.TestCustomPipeNameCreationTooLongOnNonWindows | Stuck | Calls win32 api, not supported by Coyote |
| PSTests.Parallel.PSObjectTests.TestCimInstanceProperty | Stuck | Using Concurrent Queue, not supported |
| PSTests.Sequential.CommandPredictionTests.Feedback | Passed | Needs delay with Task.Delay, there is a race condition for fast predictor, a wait needs to be added. |
| PSTests.Sequential.CommandPredictionTests.PredictInput | Failed | The test is badly designed, it's depending on task's delays to validate its results, however that is not deterministic as the scheduler can execute stuff however it prefers. |
| PSTests.Sequential.PowerShellHostingScenario.TestStartJobThrowTerminatingException | Failed | Not supported by Coyote, uses |

| | | |
|---|---|---|
| | | AutoResetEvent which is currently not supported. |
| PSTests.Sequential.PowerShellPolicyTests.PowerShellConfig_GetPowerShellPolicies_BothConfigFilesEmpty | Passed | Coyote rewrite bug issue, class has instance as singleton accessed by static field, Coyote throws. If creating instances for each test, coyote works, and test passes. |
| PSTests.Sequential.PowerShellPolicyTests.PowerShellConfig_GetPowerShellPolicies_BothConfigFilesNotEmpty | Passed | Coyote rewrite bug issue, class has instance as singleton accessed by static field, Coyote throws. If creating instances for each test, coyote works, and test passes. |
| PSTests.Sequential.PowerShellPolicyTests.PowerShellConfig_GetPowerShellPolicies_EmptySystemConfig | Passed | Coyote rewrite bug issue, class has instance as singleton accessed by static field, Coyote throws. If creating instances for each test, coyote |

| | | |
|---|---|---|
| | | works, and test passes. |
| PSTests.Sequential.PowerShellPolicyTests.PowerShellConfig_GetPowerShellPolicies_EmptyUserConfig | Passed | Coyote rewrite bug issue, class has instance as singleton accessed by static field, Coyote throws. If creating instances for each test, coyote works, and test passes. |
| PSTests.Sequential.PowerShellPolicyTests.PowerShellConfig_GetPowerShellPolicies_BothConfigFilesNotExist | Passed | |
| PSTests.Sequential.PowerShellPolicyTests.Utils_GetPolicySetting_BothConfigFilesNotExist | Passed | |
| PSTests.Parallel.FileSystemProviderTests.TestClearContent | Passed | Custom Manual reset event slim fixed |
| PSTests.Parallel.FileSystemProviderTests.TestClearProperty | Passed | Custom Manual reset event slim fixed |
| PSTests.Parallel.FileSystemProviderTests.TestGetContentReader | Passed | Custom Manual reset event slim fixed |
| PSTests.Parallel.FileSystemProviderTests.TestGetContentWriter | Passed | Custom Manual reset event slim fixed |
| PSTests.Parallel.FileSystemProviderTests.TestGetProperty | Passed | Custom Manual reset event slim fixed |
| PSTests.Parallel.FileSystemProviderTests.TestSetProperty | Passed | Custom Manual reset event slim fixed |
| PSTests.Parallel.MshSnapinInfoTests.TestReadCoreEngineSnapIn | Passed | Custom Manual |

| | | |
|---|---|---|
| | | reset event slim fixed |
| PSTests.Parallel.SessionStateTests.TestDrives | Passed | Custom Manual reset event slim fixed |
| PSTests.Parallel.AstDefaultVisitTests.TestAstVisitor | Passed | |
| PSTests.Parallel.AstDefaultVisitTests.TestCustomAstVisitor | Passed | |
| PSTests.Parallel.CryptoUntilsTests.TestSessionKeyExchange | Passed | |
| PSTests.Parallel.FileSystemProviderTests.TestCreateJunctionFails | Passed | |
| PSTests.Parallel.FileSystemProviderTests.TestGetHelpMaml | Passed | |
| PSTests.Parallel.FileSystemProviderTests.TestMode | Passed | |
| PSTests.Parallel.MshSnapinInfoTests.TestReadRegistryInfo | Passed | |
| PSTests.Parallel.PlatformTests.TestIsCoreCLR | Passed | |
| PSTests.Parallel.PSCommandLineParserTests.Test_ARGS_With_Null | Passed | |
| PSTests.Parallel.PSCommandLineParserTests.Test_Throws_On_Reuse | Passed | |
| PSTests.Parallel.PSCommandLineParserTests.TestDefaultParameterIsFileName_Exist | Passed | |
| PSTests.Parallel.PSCommandLineParserTests.TestDefaultParameterIsFileName_Not_Exist | Passed | |
| PSTests.Parallel.PSCommandLineParserTests.TestDefaults | Passed | |
| PSTests.Parallel.PSCommandLineParserTests.TestParameter_Command_No_Value | Passed | |
| PSTests.Parallel.PSCommandLineParserTests.TestParameter_Command_With_Dash_And_ConsoleInputRedirected | Passed | |
| PSTests.Parallel.PSCommandLineParserTests.TestParameter_Command_With_Dash_And_Not_ConsoleInputRedirected | Passed | |
| PSTests.Parallel.PSCommandLineParserTests.TestParameter_Command_With_Dash_And_Tail | Passed | |
| PSTests.Parallel.PSCommandLineParserTests.TestParameter_Command_With_Value | Passed | |
| PSTests.Parallel.PSCommandLineParserTests.TestParameter_ConfigurationName_No_Name | Passed | |
| PSTests.Parallel.PSCommandLineParserTests.TestParameter_ConfigurationName_With_Name | Passed | |
| PSTests.Parallel.PSCommandLineParserTests.TestParameter_CustomPipeName_No_Name | Passed | |
| PSTests.Parallel.PSCommandLineParserTests.TestParameter_CustomPipeName_With_Name | Passed | |
| PSTests.Parallel.PSCommandLineParserTests.TestParameter_EncodedArguments_No_Value | Passed | |
| PSTests.Parallel.PSCommandLineParserTests.TestParameter_EncodedArguments_With_Value | Passed | |
| PSTests.Parallel.PSCommandLineParserTests.TestParameter_EncodedArguments_With_Wrong_Value | Passed | |
| PSTests.Parallel.PSCommandLineParserTests.TestParameter_EncodedCommand_No_Value | Passed | |

| | | |
|---|---|---|
| PSTests.Parallel.PSCommandLineParserTests.TestParameter_EncodedCommand_With_Dash | Passed | |
| PSTests.Parallel.PSCommandLineParserTests.TestParameter_EncodedCommand_With_Dash_And_ConsoleInputRedirected | Passed | |
| PSTests.Parallel.PSCommandLineParserTests.TestParameter_EncodedCommand_With_Dash_And_Tail | Passed | |
| PSTests.Parallel.PSCommandLineParserTests.TestParameter_EncodedCommand_With_Value | Passed | |
| PSTests.Parallel.PSCommandLineParserTests.TestParameter_ExecutionPolicy_No_Value | Passed | |
| PSTests.Parallel.PSCommandLineParserTests.TestParameter_ExecutionPolicy_With_Right_Value | Passed | |
| PSTests.Parallel.PSCommandLineParserTests.TestParameter_Help | Passed | |
| PSTests.Parallel.PSCommandLineParserTests.TestParameter_InputFormat_No_Value | Passed | |
| PSTests.Parallel.PSCommandLineParserTests.TestParameter_InputFormat_With_Right_Value | Passed | |
| PSTests.Parallel.PSCommandLineParserTests.TestParameter_InputFormat_With_Wrong_Value | Passed | |
| PSTests.Parallel.PSCommandLineParserTests.TestParameter_Interactive | Passed | |
| PSTests.Parallel.PSCommandLineParserTests.TestParameter_LastParameterIsFileName_Exist | Passed | |
| PSTests.Parallel.PSCommandLineParserTests.TestParameter_Login | Passed | |
| PSTests.Parallel.PSCommandLineParserTests.TestParameter_MTA_And_STA_Mutually_Exclusive | Passed | |
| PSTests.Parallel.PSCommandLineParserTests.TestParameter_NamedPipeServerMode | Passed | |
| PSTests.Parallel.PSCommandLineParserTests.TestParameter_NoExit | Passed | |
| PSTests.Parallel.PSCommandLineParserTests.TestParameter_NoInteractive | Passed | |
| PSTests.Parallel.PSCommandLineParserTests.TestParameter_NoLogo | Passed | |
| PSTests.Parallel.PSCommandLineParserTests.TestParameter_NoProfile | Passed | |
| PSTests.Parallel.PSCommandLineParserTests.TestParameter_OutputFormat_No_Value | Passed | |
| PSTests.Parallel.PSCommandLineParserTests.TestParameter_OutputFormat_With_Right_Value | Passed | |
| PSTests.Parallel.PSCommandLineParserTests.TestParameter_OutputFormat_With_Wrong_Value | Passed | |
| PSTests.Parallel.PSCommandLineParserTests.TestParameter_ServerMode | Passed | |
| PSTests.Parallel.PSCommandLineParserTests.TestParameter_SettingsFile_Exists | Passed | |
| PSTests.Parallel.PSCommandLineParserTests.TestParameter_SettingsFile_No_Value | Passed | |
| PSTests.Parallel.PSCommandLineParserTests.TestParameter_SettingsFile_Not_Exists | Passed | |
| PSTests.Parallel.PSCommandLineParserTests.TestParameter_SocketServerMode | Passed | |
| PSTests.Parallel.PSCommandLineParserTests.TestParameter_SSHServerMode | Passed | |

| | | |
|---|---|---|
| PSTests.Parallel.PSCommandLineParserTests.TestParameter_STA_And_MTA_Mutually_Exclusive | Passed | |
| PSTests.Parallel.PSCommandLineParserTests.TestParameter_Version | Passed | |
| PSTests.Parallel.PSCommandLineParserTests.TestParameter_WindowsStyle_No_Value | Passed | |
| PSTests.Parallel.PSCommandLineParserTests.TestParameter_WindowsStyle_With_Right_Value | Passed | |
| PSTests.Parallel.PSCommandLineParserTests.TestParameter_WindowsStyle_With_Wrong_Value | Passed | |
| PSTests.Parallel.PSCommandLineParserTests.TestParameter_WorkingDirectory | Passed | |
| PSTests.Parallel.PSCommandLineParserTests.TestParameter_WorkingDirectory_No_Value | Passed | |
| PSTests.Parallel.PSCommandLineParserTests.TestParameter_WorkingDirectory_RemoveTrailingCharacter | Passed | |
| PSTests.Parallel.PSCommandLineParserTests.TestParameterIs_Wrong_Value | Passed | |
| PSTests.Parallel.PSCommandLineParserTests.TestParameterIsFileName_Dash | Passed | |
| PSTests.Parallel.PSEnumerableBinderTests.TestIsStaticTypePossiblyEnumerable | Passed | |
| PSTests.Parallel.PSObjectTests.TestAdaptedMember | Passed | |
| PSTests.Parallel.PSObjectTests.TestEmptyObjectHasNoProperty | Passed | |
| PSTests.Parallel.PSObjectTests.TestMemberSet | Passed | |
| PSTests.Parallel.PSObjectTests.TestMemberSetIsNotProperty | Passed | |
| PSTests.Parallel.PSObjectTests.TestShadowedMember | Passed | |
| PSTests.Parallel.PSObjectTests.TestWrappedDateTimeHasReflectedMember | Passed | |
| PSTests.Parallel.PSObjectTests.TextXmlAttributeMember | Passed | |
| PSTests.Parallel.PSObjectTests.TextXmlElementMember | Passed | |
| PSTests.Parallel.PSTypeExtensionsTests.TestIsNumeric | Passed | |
| PSTests.Parallel.PSVersionInfoTests.TestVersions | Passed | |
| PSTests.Parallel.SecuritySupportTests.TestScanContent | Passed | |
| PSTests.Parallel.UtilsTests.TestBoundedStack | Passed | |
| PSTests.Parallel.UtilsTests.TestConvertToJsonBasic | Passed | |
| PSTests.Parallel.UtilsTests.TestConvertToJsonCancellation | Passed | |
| PSTests.Parallel.UtilsTests.TestConvertToJsonWithEnum | Passed | |
| PSTests.Parallel.UtilsTests.TestConvertToJsonWithoutCompress | Passed | |
| PSTests.Parallel.UtilsTests.TestHistoryStack | Passed | |
| PSTests.Parallel.UtilsTests.TestIsWinPEHost | Passed | |
| PSTests.Parallel.WildcardPatternTests.TestEscape_Empty | Passed | |
| PSTests.Parallel.WildcardPatternTests.TestEscape_Null | Passed | |
| PSTests.Parallel.WildcardPatternTests.TestEscape_String | Passed | |
| PSTests.Parallel.WildcardPatternTests.TestEscape_String_NotEscape | Passed | |
| PSTests.Parallel.WildcardPatternTests.TestUnescape_Empty | Passed | |
| PSTests.Parallel.WildcardPatternTests.TestUnescape_Null | Passed | |
| PSTests.Parallel.WildcardPatternTests.TestUnescape_String | Passed | |
| PSTests.Sequential.SubsystemTests.GetSubsystemInfo | Passed | |
| PSTests.Sequential.SubsystemTests.RegisterSubsystem | Passed | |

| | | |
|---|---|---|
| PSTests.Sequential.SubsystemTests.UnregisterSubsystem | Passed | |
| PSTests.Parallel.PSCommandLineParserTests.TestParameter_CustomPipeName_With_Too_Long_Name | Skipped | |
| PSTests.Parallel.PSCommandLineParserTests.TestParameter_MTA | Skipped | |
| PSTests.Parallel.PSCommandLineParserTests.TestParameter_MTA_Not_IsWindowsDesktop | Skipped | |
| PSTests.Parallel.PSCommandLineParserTests.TestParameter_STA | Skipped | |
| PSTests.Parallel.PSCommandLineParserTests.TestParameter_STA_Not_IsWindowsDesktop | Skipped | |
| PSTests.Parallel.PSCommandLineParserTests.TestParameter_WindowsStyle_On_Unix | Skipped | |
| PSTests.Sequential.RunspaceTests.TestAppDomainProcessExitEvenHandlerNotLeaking | Skipped | |
| PSTests.Sequential.PowerShellPolicyTests.Utils_GetPolicySetting_BothConfigFilesEmpty | Passed | Coyote rewrite bug issue, class has instance as singleton accessed by static field, Coyote throws. If creating instances for each test, coyote works, and test passes. |
| PSTests.Sequential.PowerShellPolicyTests.Utils_GetPolicySetting_BothConfigFilesNotEmpty | Passed | Coyote rewrite bug issue, class has instance as singleton accessed by static field, Coyote throws. If creating instances for each test, coyote works, and test passes. |
| PSTests.Sequential.PowerShellPolicyTests.Utils_GetPolicySetting_EmptySystemConfig | Passed | Coyote rewrite bug |

| | | |
|---|---|---|
| | | issue, class has instance as singleton accessed by static field, Coyote throws. If creating instances for each test, coyote works, and test passes. |
| PSTests.Sequential.PowerShellPolicyTests.Utils_GetPolicySetting_EmptyUserConfig | Passed | Coyote rewrite bug issue, class has instance as singleton accessed by static field, Coyote throws. If creating instances for each test, coyote works, and test passes. |
| PSTests.Sequential.RunspaceTests.TestRunspaceWithPipeline | Stuck | Uses Thread internally, not in scope by coyote |
| PSTests.Sequential.RunspaceTests.TestRunspaceWithPowerShell | Stuck | Uses Thread internally, not in scope by coyote |
| PSTests.Sequential.RunspaceTests.TestRunspaceWithPowerShellAndInitialSessionState | Stuck | Uses Thread internally, not in scope by coyote |

## References


[1] i. frequency. [Online]. Available: https://software.intel.com/content/www/us/en/develop/blogs/why-has-cpu-frequency-ceased-to-grow.html.

[2] a. acm, "Extending Amdahl's law in the multicore era," [Online]. Available: https://dl.acm.org/doi/10.1145/1639562.1639571.

[3] [Online]. Available: http://nob.cs.ucdavis.edu/classes/ecs150-1999-02/dl-cond.html.

[4] "Coyote home," [Online]. Available: https://microsoft.github.io/coyote/#.

[5] [Online]. Available: https://web.archive.org/web/20070317015632/http://www.cse.ohio-state.edu/~gurari/course/cis680/cis680Ch19.html#QQ1-51-128.

[6] "A Randomized Scheduler with probabilistic guarantees for finding bugs," [Online]. Available: https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/asplos277-pct.pdf.

[7] "Learning-Based Controlled Concurrency Testing," [Online]. Available: https://www.microsoft.com/en-us/research/uploads/prod/2019/12/QL-OOPSLA-2020.pdf.

[8] Microsoft, "Task Class," [Online]. Available: https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.task?view=net-6.0.

[9] "xUnit," [Online]. Available: https://xunit.net/.

[10] "xunit fact theories," [Online]. Available: https://xunit.net/docs/getting-started/netcore/visual-studio#write-first-theory.

[11] j. evain. [Online]. Available: https://github.com/jbevain/cecil.

[12] "Roslyn," [Online]. Available: https://github.com/dotnet/roslyn.

[13] Microsoft. [Online]. Available: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/using-statement.

[14] Microsoft. [Online]. Available: https://docs.microsoft.com/en-us/dotnet/api/system.idisposable?view=net-5.0.

[15] Microsoft. [Online]. Available: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/try-finally.