Déjà Fu Documentation

Release HEAD

Michael Walker

Nov 16, 2019

User Documentation

[Déjà Fu is] A martial art in which the user's limbs move in time as well as space, [. . .] It is best described as "the feeling that you have been kicked in the head this way before"

Terry Pratchett, Thief of Time

CHAPTER 1

Getting Started

> [Déjà Fu is] A martial art in which the user's limbs move in time as well as space, [...] It is best described as "the feeling that you have been kicked in the head this way before"
>
> **Terry Pratchett, Thief of Time**

Déjà Fu is a unit-testing library for concurrent Haskell programs. Tests are deterministic and expressive, making it easy and convenient to test your threaded code. Available on GitHub, Hackage, and Stackage.

Features:

- An abstraction over the concurrency functionality in `IO`
- Deterministic testing of nondeterministic code
- Both complete (slower) and incomplete (faster) modes
- A unit-testing-like approach to writing test cases
- A property-testing-like approach to comparing stateful operations
- Testing of potentially nonterminating programs
- Integration with HUnit and tasty

There are a few different packages under the Déjà Fu umbrella:

Package	Version	Summary
concurrency	1.8.1.0	Typeclasses, functions, and data types for concurrency and STM
dejafu	2.1.0.1	Systematic testing for Haskell concurrency
hunit-dejafu	2.0.0.1	Déjà Fu support for the HUnit test framework
tasty-dejafu	2.0.0.1	Déjà Fu support for the tasty test framework

1.1 Installation

Install from Hackage globally:

```
$ cabal install dejafu
```

Or add it to your cabal file:

```
build-depends: ...
             , dejafu
```

Or to your package.yaml:

```
dependencies:
  ...
  - dejafu
```

1.2 Quick start guide

Déjà Fu supports unit testing, and comes with a helper function called `autocheck` to look for some common issues. Let's see it in action:

```
import Control.Concurrent.Classy

myFunction :: MonadConc m => m String
myFunction = do
  var <- newEmptyMVar
  fork (putMVar var "hello")
  fork (putMVar var "world")
  readMVar var
```

That `MonadConc` is a typeclass abstraction over concurrency, but we'll get onto that shortly. First, the result of testing:

```
> autocheck myFunction
[pass] Successful
[fail] Deterministic
    "hello" S0----S1--S0--

    "world" S0----S2--S0--
False
```

There are no concurrency errors, which is good; but the program is (as you probably spotted) nondeterministic!

Along with each result, Déjà Fu gives us a representative execution trace in an abbreviated form. `Sn` means that thread `n` started executing, and `Pn` means that thread `n` pre-empted the previously running thread.

1.3 Why Déjà Fu?

Testing concurrent programs is difficult, because in general they are nondeterministic. This leads to people using workarounds like running their testsuite many thousands of times; or running their testsuite while putting their machine under heavy load.

These approaches are inadequate for a few reasons:

- **How many runs is enough?** When you are just hopping to spot a bug by coincidence, how do you know to stop?

- **How do you know if you've fixed a bug you saw previously?** Because the scheduler is a black box, you don't know if the previously buggy schedule has been re-run.
- **You won't actually get that much scheduling variety!** Operating systems and language runtimes like to run threads for long periods of time, which reduces the variety you get (and so drives up the number of runs you need).

Déjà Fu addresses these points by offering *complete* testing. You can run a test case and be guaranteed to find all results with some bounds. These bounds can be configured, or even disabled! The underlying approach used is smarter than merely trying all possible executions, and will in general explore the state-space quickly.

If your test case is just too big for complete testing, there is also a random scheduling mode, which is necessarily *incomplete*. However, Déjà Fu will tend to produce much more scheduling variety than just running your test case in `IO` a bunch of times, and so bugs will tend to crop up sooner. Furthermore, as you get execution traces out, you can be certain that a bug has been fixed by following the trace by eye.

If you'd like to get involved with Déjà Fu, check out the "good first issue" label on the issue tracker.

1.4 Questions, feedback, discussion

- For general help talk to me in IRC (barrucadu in #haskell) or shoot me an email (mike@barrucadu.co.uk)
- For bugs, issues, or requests, please file an issue.

1.5 Bibliography

Déjà Fu has been produced as part of my Ph.D work, and wouldn't be possible without prior research. Here are the core papers:

- Bounded partial-order reduction, K. Coons, M. Musuvathi, and K. McKinley (2013)
- Dynamic Partial Order Reduction for Relaxed Memory Models, N. Zhang, M. Kusano, and C. Wang (2015)
- Concurrency Testing Using Schedule Bounding: an Empirical Study, P. Thompson, A. Donaldson, and A. Betts (2014)
- On the Verification of Programs on Relaxed Memory Models, A. Linden (2014)

CHAPTER 2

Typeclasses

We don't use the regular `Control.Concurrent` and `Control.Exception` modules, we use typeclass-generalised ones instead from the concurrency and exceptions packages.

2.1 Porting guide

If you want to test some existing code, you'll need to port it to the appropriate typeclass. The typeclass is necessary, because we can't peek inside `IO` and `STM` values, so we need to able to plug in an alternative implementation when testing.

Fortunately, this tends to be a fairly mechanical and type-driven process:

1. Import `Control.Concurrent.Classy.*` instead of `Control.Concurrent.*`
2. Import `Control.Monad.Catch` instead of `Control.Exception`
3. Change your monad type:
    - `IO a` becomes `MonadConc m => m a`
    - `STM a` becomes `MonadSTM stm => stm a`
4. Parameterise your state types by the monad:
    - `TVar` becomes `TVar stm`
    - `MVar` becomes `MVar m`
    - `IORef` becomes `IORef m`
5. Some functions are renamed:
    - `forkIO*` becomes `fork*`
    - `atomicModifyIORefCAS*` becomes `modifyIORefCAS*`
6. Fix the type errors

If you're lucky enough to be starting a new concurrent Haskell project, you can just program against the `MonadConc` interface.

2.2 What if I really need I/O?

You can use `MonadIO` and `liftIO` with `MonadConc`, for instance if you need to talk to a database (or just use some existing library which needs real I/O).

To test `IO`-using code, there are some rules you need to follow:

1. Given the same set of scheduling decisions, your `IO` code must be deterministic[1]
2. As dejafu can't inspect `IO` values, they should be kept small; otherwise dejafu may miss buggy interleavings
3. You absolutely cannot block on the action of another thread inside `IO`, or the test execution will just deadlock.

2.3 Deriving your own instances

There are `MonadConc` and `MonadSTM` instances for many common monad transformers. In the simple case, where you want an instance for a newtype wrapper around a type that has an instance, you may be able to derive it. For example:

```
{-# LANGUAGE GeneralizedNewtypeDeriving #-}
{-# LANGUAGE StandaloneDeriving #-}
{-# LANGUAGE UndecidableInstances #-}

data Env = Env

newtype MyMonad m a = MyMonad { runMyMonad :: ReaderT Env m a }
  deriving (Functor, Applicative, Monad)

deriving instance MonadThrow m => MonadThrow (MyMonad m)
deriving instance MonadCatch m => MonadCatch (MyMonad m)
deriving instance MonadMask  m => MonadMask  (MyMonad m)

deriving instance MonadConc m => MonadConc (MyMonad m)
```

`MonadSTM` needs a slightly different set of classes:

```
{-# LANGUAGE GeneralizedNewtypeDeriving #-}
{-# LANGUAGE StandaloneDeriving #-}
{-# LANGUAGE UndecidableInstances #-}

data Env = Env

newtype MyMonad m a = MyMonad { runMyMonad :: ReaderT Env m a }
  deriving (Functor, Applicative, Monad, Alternative, MonadPlus)

deriving instance MonadThrow m => MonadThrow (MyMonad m)
deriving instance MonadCatch m => MonadCatch (MyMonad m)

deriving instance MonadSTM m => MonadSTM (MyMonad m)
```

[1] This is only essential if you're using the systematic testing (the default). Nondeterministic `IO` won't break the random testing, it'll just make things more confusing.

Don't be put off by the use of `UndecidableInstances`, it's safe here.

CHAPTER 3

Unit Testing

Writing tests with Déjà Fu is a little different to traditional unit testing, as your test case may have multiple results. A "test" is a combination of your code, and a predicate which says something about the set of allowed results.

Most tests will look something like this:

```
dejafu "Assert the thing holds" myPredicate myAction
```

The `dejafu` function comes from `Test.DejaFu`. Another useful function is `dejafuWithSettings`; see *Execution settings*.

3.1 Actions

An action is just something with the type `MonadConc m => m a`, or `(MonadConc m, MonadIO m) => m a` for some `a` that your chosen predicate can deal with.

For example, some users on Reddit found a couple of apparent bugs in the auto-update package a while ago (thread here). As the package is simple and self-contained, I translated it to the `MonadConc` abstraction and wrote a couple of tests to replicate the bugs. Here they are:

```
deadlocks :: MonadConc m => m ()
deadlocks = do
  auto <- mkAutoUpdate defaultUpdateSettings
  auto

nondeterministic :: forall m. MonadConc m => m Int
nondeterministic = do
  var <- newIORef 0
  let settings = (defaultUpdateSettings :: UpdateSettings m ())
        { updateAction = atomicModifyIORef var (\x -> (x+1, x)) }
  auto <- mkAutoUpdate settings
  auto
  auto
```

These actions action could be tested with `autocheck`, and the issues would be revealed. The use of `ScopedTypeVariables` in the second is an unfortunate example of what can happen when everything becomes more polymorphic. But other than that, note how there is no special mention of Déjà Fu in the actions: it's just normal concurrent Haskell, simply written against a different interface.

The modified package is included in the test suite, if you want to see the full code.[1]

If the RTS supports bound threads (the `-threaded` flag was passed to GHC when linking), then the main thread of an action given to Déjà Fu will be bound, and further bound threads can be forked with the `forkOS` functions. If not, then attempting to fork a bound thread will raise an error.

3.2 Conditions

When a concurrent program of type `MonadConc m => m a` is executed, it may produce a value of type `a`, or it may experience a **condition** such as deadlock.

A condition does not necessarily cause your test to fail. It's important to be aware of what exactly your test is testing, to avoid drawing the wrong conclusions from a passing (or failing) test.

3.3 Setup and Teardown

Because dejafu drives the execution of the program under test, there are some tricks available to you which are not possible using normal concurrent Haskell.

If your test does some set-up work which is required for your test to work, but which is not the actual thing you are testing, you can define that as a **setup action**:

```
withSetup
  :: Program Basic n x
  -- ^ Setup action
  -> (x -> Program Basic n a)
  -- ^ Main program
  -> Program (WithSetup x) n a
```

dejafu will save the state at the end of the setup action, and efficiently restore that state in subsequent runs of the same test with a different schedule. This can be much more efficient than dejafu running the setup action normally every single time.

If you want to examine some state you created in your setup action even if your actual test case deadlocks or something, you can define a **teardown action**:

```
withSetupAndTeardown
  :: Program Basic n x
  -- ^ Setup action
  -> (x -> Either Condition y -> Program Basic n a)
  -- ^ Teardown action
  -> (x -> Program Basic n y)
  -- ^ Main program
  -> Program (WithSetupAndTeardown x y) n a
```

The teardown action is always executed.

[1] The predicates in dejafu-tests are a little confusing, as they're the opposite of what you would normally write! These predicates are checking that the bug is found, not that the code is correct.

Finally, if you want to ensure that some invariant holds over some shared state, you can define invariants in the setup action, which are checked atomically during the main action:

```
-- slightly contrived example
let setup = do
      var <- newEmptyMVar
      registerInvariant $ do
        value <- inspectMVar var
        when (x == Just 1) (throwM Overflow)
      pure var
in withSetup setup $ \var -> do
     fork $ putMVar var 0
     fork $ putMVar var 1
     tryReadMVar var
```

If the main action violates the invariant, it is terminated with an `InvariantFailure` condition, and any teardown action is run.

3.4 Predicates

There are a few predicates built in, and some helpers to define your own.

`abortsNever`	checks that the computation never aborts
`abortsAlways`	checks that the computation always aborts
`abortsSometimes`	checks that the computation aborts at least once

An **abort** is where the scheduler chooses to terminate execution early. If you see it, it probably means that a test didn't terminate before it hit the execution length limit. Aborts are hidden unless you use explicitly enable them, see *Execution settings*.

`deadlocksNever`	checks that the computation never deadlocks
`deadlocksAlways`	checks that the computation always deadlocks
`deadlocksSometimes`	checks that the computation deadlocks at least once

Deadlocking is where every thread becomes blocked. This can be, for example, if every thread is trying to read from an `MVar` that has been emptied.

`exceptionsNever`	checks that the main thread is never killed by an exception
`exceptionsAlways`	checks that the main thread is always killed by an exception
`exceptionsSometimes`	checks that the main thread is killed by an exception at least once

An uncaught **exception** in the main thread kills the process. These can be synchronous (thrown in the main thread) or asynchronous (thrown to it from a different thread).

`alwaysSame`	checks that the computation is deterministic and always produces a value
`alwaysSameOn f`	is like `alwaysSame`, but transforms the results with `f` first
`alwaysSameBy f`	is like `alwaysSame`, but uses `f` instead of `(==)` to compare
`notAlwaysSame`	checks that the computation is nondeterministic
`notAlwaysSameOn f`	is like `notAlwaysSame`, but transforms the results with `f` first
`notAlwaysSameBy f`	is like `notAlwaysSame`, but uses `f` instead of `(==)` to compare

Checking for **determinism** will also find nondeterministic failures: deadlocking (for instance) is still a result of a test!

`alwaysTrue p`	checks that `p` is true for every result
`somewhereTrue p`	checks that `p` is true for at least one result

These can be used to check custom predicates. For example, you might want all your results to be less than five.

`gives xs`	checks that the set of results is exactly `xs` (which may include conditions)
`gives' xs`	checks that the set of results is exactly `xs` (which may not include conditions)

These let you say exactly what you want the results to be. Your test will fail if it has any extra results, or misses a result.

You can check multiple predicates against the same collection of results using the `dejafus` and `dejafusWithSettings` functions. These avoid recomputing the results, and so may be faster than multiple `dejafu` / `dejafuWithSettings` calls; see *Performance tuning*.

3.5 Using HUnit and Tasty

By itself, Déjà Fu has no framework in place for named test groups and parallel execution or anything like that. It does one thing and does it well, which is running test cases for concurrent programs. HUnit and tasty integration is provided to get more of the features you'd expect from a testing framework.

The integration is provided by the hunit-dejafu and tasty-dejafu packages.

There's a simple naming convention used: the `Test.DejaFu` function `dejafuFoo` is wrapped in the appropriate way and exposed as `testDejafuFoo` from `Test.HUnit.DejaFu` and `Test.Tasty.DejaFu`.

Our example from the start becomes:

```
testDejafu "Assert the thing holds" myPredicate myAction
```

The `autocheck` function is exposed as `testAuto`.

CHAPTER 4

Refinement Testing

Déjà Fu also supports a form of property-testing where you can check things about the side-effects of stateful operations. For example, we can assert that `readMVar` is equivalent to sequencing `takeMVar` and `putMVar` like so:

```
prop_mvar_read_take_put =
  sig readMVar `equivalentTo` sig (\v -> takeMVar v >>= putMVar v)
```

Given the signature function, `sig`, defined in the next section. If we check this, our property fails!

```
> check prop_mvar_read_take_put
*** Failure: (seed Just 0)
    left:  [(Nothing,Just 0)]
    right: [(Nothing,Just 0),(Just Deadlock,Just 0)]
False
```

This is because `readMVar` is atomic, whereas sequencing `takeMVar` with `putMVar` is not, and so another thread can interfere with the `MVar` in the middle. The `check` and `equivalentTo` functions come from `Test.DejaFu.Refinement` (also re-exported from `Test.DejaFu`).

4.1 Signatures

A signature tells the property-tester something about the state your operation acts upon, it has a few components:

```
data Sig s o x = Sig
  { initialise ::       x -> ConcIO s
  , observe    :: s -> x -> ConcIO o
  , interfere  :: s -> x -> ConcIO ()
  , expression :: s      -> ConcIO ()
  }
```

- `s` is the **state type**, it's the thing which your operations mutate. For `readMVar`, the state is some `MVar a`.

- `o` is the **observation type**, it's some pure (and comparable) proxy for a snapshot of your mutable state. For `MVar a`, the observation is probably a `Maybe a`.
- `x` is the **seed type**, it's some pure value used to construct the initial mutable state. For `MVar a`, the seed is probably a `Maybe a`.
- `ConcIO` is just one of the instances of `MonadConc` that Déjà Fu defines for testing purposes. Just write code polymorphic in the monad as usual, and all will work.

The `initialise`, `observe`, and `expression` functions should be self-explanatory, but the `interfere` one may not be. It's the job of the `interfere` function to change the state in some way; it's run concurrently with the expression, to simulate the nondeterministic action of other threads.

Here's a concrete example for our `MVar` example:

```
sig :: (MVar ConcIO Int -> ConcIO a) -> Sig (MVar ConcIO Int) (Maybe Int) (Maybe Int)
sig e = Sig
{ initialise = maybe newEmptyMVar newMVar
, observe    = \v _ -> tryTakeMVar v
, interfere  = \v s -> tryTakeMVar v >> maybe (pure ()) (\x -> void $ tryPutMVar v (x␣
↪* 1000)) s
, expression = void . e
}
```

The `observe` function should be deterministic, but as it is run after the normal execution ends, it may have side-effects on the state. The `interfere` function can do just about anything[1], but a poor one may result in the property-checker being unable to distinguish between atomic and nonatomic expressions.

4.2 Properties

A property is a pair of signatures linked by one of three provided functions. These functions are:

Function	Operator	Checks that...
`equivalentTo`	`===`	... the left and right have exactly the same behaviours
`refines`	`=>=`	... every behaviour of the left is also a behaviour of the right
`strictlyRefines`	`->-`	... `left =>= right` holds but `left === right` does not

The signatures can have different state types, as long as the seed and observation types are the same. This lets you compare different implementations of the same idea: for example, comparing a concurrent stack implemented using `MVar` with one implemented using `IORef`.

Properties can have parameters, given in the obvious way:

```
check $ \a b c -> sig1 ... `op` sig2 ...
```

Under the hood, seed and parameter values are generated using the leancheck package, an enumerative property-based testing library. This means that any types you use will need to have a `Listable` instance.

You can also think about the three functions in terms of sets of results, where a result is a `(Maybe Failure, o)` value. A `Failure` is something like deadlocking, or being killed by an exception; `o` is the observation type. An observation is always made, even if execution of the expression fails.

[1] There are probably some concrete rules for a good function, but I haven't figured them out yet.

Function	Result-set operation
`refines`	For all seed and parameter assignments, subset-or-equal
`strictlyRefines`	For at least one seed and parameter assignment, proper subset; for all others, subset-or-equal
`equivalentTo`	For all seed and parameter assignments, equality

Finally, there is an `expectFailure` function, which inverts the expected result of a property.

The Déjà Fu testsuite has a collection of refinement properties, which may help you get a feel for this sort of testing.

4.3 Using HUnit and Tasty

As for unit testing, HUnit and tasty integration is provided for refinement testing in the hunit-dejafu and tasty-dejafu packages.

The `testProperty` function is used to check properties. Our example from the start becomes:

```
testProperty "Read is equivalent to Take then Put" prop_mvar_read_take_put
```

CHAPTER 5

Advanced Usage

Déjà Fu tries to have a sensible set of defaults, but there are some times when the defaults are not suitable. There are a lot of knobs provided to tweak how things work.

5.1 Execution settings

The `autocheckWithSettings`, `dejafuWithSettings`, and `dejafusWithSettings` let you provide a `Settings` value, which controls some of Déjà Fu's behaviour:

```
dejafuWithSettings mySettings "Assert the thing holds" myPredicate myAction
```

The available settings are:

- **"Way"**, how to explore the behaviours of the program under test.
- **Length bound**, a cut-off point to terminate an execution even if it's not done yet.
- **Memory model**, which affects how non-synchronised operations, such as `readIORef` and `writeIORef` behave.
- **Discarding**, which allows throwing away uninteresting results, rather than keeping them around in memory.
- **Early exit**, which allows exiting as soon as a result matching a predicate is found.
- **Representative traces**, keeping only one execution trace for each distinct result.
- **Trace simplification**, rewriting execution traces into a simpler form (particularly effective with the random testing).
- **Safe IO**, pruning needless schedules when your IO is only used to manage thread-local state.

See the `Test.DejaFu.Settings` module for more information.

5.2 Performance tuning

- Are you happy to trade space for time?

  Consider computing the results once and running multiple predicates over the output: this is what `dejafus` / `testDejafus` / etc does.

- Can you sacrifice completeness?

  Consider using the random testing functionality. See the `*WithSettings` functions.

- Would strictness help?

  Consider using the strict functions in `Test.DejaFu.SCT` (the ones ending with a `'`).

- Do you just want the set of results, and don't care about traces?

  Consider using `Test.DejaFu.SCT.resultsSet`.

- Do you know something about the sort of results you care about?

  Consider discarding results you *don't* care about. See the `*WithSettings` functions in `Test.DejaFu`, `Test.DejaFu.SCT`, and `Test.{HUnit,Tasty}.DejaFu`.

For example, let's say you want to know if your test case deadlocks, but you don't care about the execution trace, and you are going to sacrifice completeness because your possible state-space is huge. You could do it like this:

```
dejafuWithSettings
  ( set ldiscard
    -- "efa" == "either failure a", discard everything but deadlocks
    (Just $ \efa -> Just (if either isDeadlock (const False) efa then DiscardTrace␣
↪else DiscardResultAndTrace))
  . set lway
    -- try 10000 executions with random scheduling
    (randomly (mkStdGen 42) 10000)
  $ defaultSettings
  )
  -- the name of the test
  "Never Deadlocks"
  -- the predicate to check
  deadlocksNever
  -- your test case
  testCase
```

CHAPTER 6

1.x to 2.x

dejafu-2.0.0.0 is a super-major release which breaks compatibility with dejafu-1.x.

Highlights reel:

- Test cases are written in terms of a new `Program` type.
- The `Failure` type has been replaced with a `Condition` type (actually in 1.12).
- Random testing takes an optional length bound.
- Atomically-checked invariants over shared mutable state.

See the changelogs for the full details.

6.1 The `Program` type

The `ConcT` type is now an alias for `Program Basic`.

A `Program Basic` has all the instances `ConcT` did, defined using the `~` instance trick, so this shouldn't be a breaking change:

```
instance (pty ~ Basic)             => MonadTrans (Program pty)
instance (pty ~ Basic)             => MonadCatch (Program pty n)
instance (pty ~ Basic)             => MonadThrow (Program pty n)
instance (pty ~ Basic)             => MonadMask  (Program pty n)
instance (pty ~ Basic, Monad   n) => MonadConc  (Program pty n)
instance (pty ~ Basic, MonadIO n) => MonadIO    (Program pty n)
```

The `dontCheck` function has been removed in favour of `withSetup`:

```
do x <- dontCheck setup
   action x

-- becomes
```

(continues on next page)

(continued from previous page)

```
withSetup setup action
```

The `subconcurrency` function has been removed in favour of `withSetupAndTeardown`:

```
do x <- setup
   y <- subconcurrency (action x)
   teardown x y

-- becomes

withSetupAndTeardown setup teardown action
```

The `dontCheck` and `subconcurrency` functions used to throw runtime errors if nested. This is not possible with `withSetup` and `withSetupAndTeardown` due to their types:

```
withSetup
  :: Program Basic n x
  -- ^ Setup action
  -> (x -> Program Basic n a)
  -- ^ Main program
  -> Program (WithSetup x) n a

withSetupAndTeardown
  :: Program Basic n x
  -- ^ Setup action
  -> (x -> Either Condition y -> Program Basic n a)
  -- ^ Teardown action
  -> (x -> Program Basic n y)
  -- ^ Main program
  -> Program (WithSetupAndTeardown x y) n a
```

Previously, multiple calls to `subconcurrency` could be sequenced in the same test case. This is not possible using `withSetupAndTeardown`. If you rely on this behaviour, please file an issue.

6.2 The `Condition` type

This is a change in dejafu-1.12.0.0, but the alias `Failure = Condition` is removed in dejafu-2.0.0.0.

- The `STMDeadlock` and `Deadlock` constructors have been merged.
- Internal errors have been split into the `Error` type and are raised as exceptions, instead of being returned as conditions.

The name "failure" has been a recurring source of confusion, because an individual execution can "fail" without the predicate as a whole failing. My hope is that the more neutral "condition" will prevent this confusion.

6.3 Deprecated functions

All the deprecated special-purpose functions have been removed. Use more general `*WithSettings` functions instead.

6.4 Need help?

- For general help talk to me in IRC (barrucadu in #haskell) or shoot me an email (mike@barrucadu.co.uk)
- For bugs, issues, or requests, please file an issue.

CHAPTER 7

0.x to 1.x

dejafu-1.0.0.0 is a super-major release which breaks compatibility with dejafu-0.x quite significantly, but brings with it support for bound threads, and significantly improves memory usage in the general case.

Highlights reel:

- Most predicates now only need to keep around the failures, rather than all results.
- Support for bound threads (with concurrency-1.3.0.0).
- The `ST` / `IO` interface duplication is gone, everything is now monadic.
- Function parameter order is closer to other testing libraries.
- Much improved API documentation.

See the changelogs for the full details.

7.1 `ST` and `IO` functions

There is only one set of functions now. Testing bound threads requires being able to fork actual threads, so testing with `ST` is no longer possible. The `ConcST` type is gone, there is only `ConcIO`.

For dejafu change:

- `autocheckIO` to `autocheck`
- `dejafuIO` to `dejafu`
- `dejafusIO` to `dejafus`
- `autocheckWayIO` to `autocheckWay`
- `dejafuWayIO` to `dejafuWay`
- `dejafusWayIO` to `dejafusWay`
- `dejafuDiscardIO` to `dejafuDiscard`

- `runTestM` to `runTest`
- `runTestWayM` to `runTestWay`

If you relied on being able to get a pure result from the `ConcST` functions, you can no longer do this.

For hunit-dejafu and tasty-dejafu change:

- `testAutoIO` to `testAuto`
- `testDejafuIO` to `testDejafu`
- `testDejafusIO` to `testDejafus`
- `testAutoWayIO` to `testAutoWay`
- `testDejafuWayIO` to `testDejafuWay`
- `testDejafusWayIO` to `testDejafusWay`
- `testDejafuDiscardIO` to `testDejafuDiscard`

7.2 Function parameter order

Like HUnit, the monadic action to test is now the last parameter of the testing functions. This makes it convenient to write tests without needing to define the action elsewhere.

For dejafu change:

- `dejafu ma (s, p)` to `dejafu s p ma`
- `dejafus ma ps` to `dejafus ps ma`
- `dejafuWay way mem ma (s, p)` to `dejafuWay way mem s p ma`
- `dejafusWay way mem ma ps` to `dejafuWay way mem ps ma`
- `dejafuDiscard d way mem ma (s, p)` to `dejafuDiscard d way mem s p ma`

For hunit-dejafu and tasty-dejafu change:

- `testDejafu ma s p` to `testDejafu s p ma`
- `testDejafus ma ps` to `testDejafus ps ma`
- `testDejafuWay way mem ma s p` to `testDejafuWay way mem s p ma`
- `testDejafusWay way mem ma ps` to `testDejafusWay way mem ps ma`
- `testDejafuDiscard d way mem ma s p` to `testDejafuDiscard d way mem s p ma`

7.3 Predicates

The `Predicate a` type is now an alias for `ProPredicate a a`, defined like so:

```
data ProPredicate a b = ProPredicate
  { pdiscard :: Either Failure a -> Maybe Discard
  -- ^ Selectively discard results before computing the result.
  , peval :: [(Either Failure a, Trace)] -> Result b
  -- ^ Compute the result with the un-discarded results.
  }
```

If you use the predicate helper functions to construct a predicate, you do not need to change anything (and should get a nice reduction in your resident memory usage). If you supply a function directly, you can recover the old behaviour like so:

```
old :: ([(Either Failure a, Trace)] -> Result a) -> ProPredicate a a
old p = ProPredicate
  { pdiscard = const Nothing
  , peval = p
  }
```

The `alwaysTrue2` helper function is gone. If you use it, use `alwaysSameOn` or `alwaysSameBy` instead.

7.4 Need help?

- For general help talk to me in IRC (barrucadu in #haskell) or shoot me an email (mike@barrucadu.co.uk)
- For bugs, issues, or requests, please file an issue.

CHAPTER 8

Contributing

Thanks for caring about Déjà Fu!

8.1 Ways to contribute

Déjà Fu is a project under active development, there's always something to do. Here's a list of ideas to get you started:

- Submit bug reports.
- Submit feature requests.
- Got a particularly slow test case which you think should be faster? Open an issue for that too.
- Blog about how and why you use Déjà Fu.
- Check if any bugs which have been open for a while are still bugs.

If you want to contribute code, you could:

- Tackle one of the issues tagged "good first issue".
- Tackle a bigger issue, perhaps one of the roadmap issues!
- Run code coverage and try to fix a gap in the tests.
- Profile the test suite and try to improve a slow function.

Roadmap issues are priority issues (in my opinion), so help with those is especially appreciated.

If you have a support question, you can talk to me on IRC (#haskell on freenode) or send an email (mike@barrucadu.co.uk) rather than opening an issue. But maybe your question is a bug report about poor documentation in disguise!

8.2 Making the change

1. Talk to me!

I don't bite, and chances are I can quickly tell you where you should start. It's better to ask what seems like a stupid question than to waste a lot of time on the wrong approach.

2. Make the change.

   Figure out what needs to be changed, how to change it, and do it. If you're fixing a bug, make sure to add a minimal reproduction to Cases.Regressions in dejafu-tests.

3. Document the change.

   All top-level definitions should have a Haddock comment explaining what it does. If you've added or changed a top-level function, consider commenting its arguments too.

   If you've added a top-level definition, or changed one in a backwards-incompatible way, add an `@since unreleased` Haddock comment to it. This is to help prevent me from missing things when I update the changelog ahead of a release.

4. Submit a PR.

   Travis will run some checks, which might prompt further action. You should expect a response from me in a day or two.

Don't worry about your PR being perfect the first time. We'll work through any issues together, to ensure that Déjà Fu gets the best code it can.

8.3 Coding style

There isn't really a prescribed style. It's not quite the wild west though; keep these three rules in mind:

1. Be consistent.
2. Run stylish-haskell to format import lists.
3. Use hlint and weeder and fix lints unless you have a good reason not to.

Travis runs stylish-haskell, hlint, and weeder on all PRs.

8.4 Coverage

hpc can generate a coverage report from the execution of dejafu-tests:

```
$ stack build --coverage
$ stack exec dejafu-tests
$ stack hpc report --all dejafu-tests.tix
```

This will print some stats and generate an HTML coverage report:

```
Generating combined report
 52% expressions used (4052/7693)
 48% boolean coverage (63/129)
      43% guards (46/106), 31 always True, 9 always False, 20 unevaluated
      68% 'if' conditions (11/16), 2 always True, 3 unevaluated
      85% qualifiers (6/7), 1 unevaluated
 61% alternatives used (392/635)
 80% local declarations used (210/261)
 26% top-level declarations used (280/1063)
The combined report is available at /home/barrucadu/projects/dejafu/.stack-work/
↪install/x86_64-linux/nightly-2016-06-20/8.0.1/hpc/combined/custom/hpc_index.html
```

The highlighted code in the HTML report emphasises branch coverage:

- Red means a branch was evaluated as always false.
- Green means a branch was evaluated as always true.
- Yellow means an expression was never evaluated.

See also the stack coverage documentation.

8.5 Performance

GHC can generate performance statistics from the execution of dejafu-tests:

```
$ stack build --profile
$ stack exec  -- dejafu-tests +RTS -p
$ less dejafu-tests.prof
```

This prints a detailed breakdown of where memory and time are being spent:

```
    Mon Mar 20 19:26 2017 Time and Allocation Profiling Report  (Final)

       dejafu-tests +RTS -p -RTS

    total time  =      105.94 secs   (105938 ticks @ 1000 us, 1 processor)
    total alloc = 46,641,766,952 bytes  (excludes profiling overheads)

COST CENTRE                            MODULE                      %time %alloc

findBacktrackSteps.doBacktrack.idxs'  Test.DejaFu.SCT.Internal    21.9   12.0
==                                     Test.DejaFu.Common          12.4    0.0
yieldCount.go                          Test.DejaFu.SCT             12.1    0.0
dependent'                             Test.DejaFu.SCT              5.1    0.0
runThreads.go                          Test.DejaFu.Conc.Internal    2.7    4.1
[...]
```

Be careful, however! Compiling with profiling can significantly affect the behaviour of a program! Use profiling to get an idea of where the hot spots are, but make sure to confirm with a non-profiled build that things are actually getting faster.

If you compile with `-rtsopts` you can get some basic stats from a non-profiled build:

```
$ stack exec -- dejafu-tests +RTS -s

[...]
86,659,658,504 bytes allocated in the heap
13,057,037,448 bytes copied during GC
    13,346,952 bytes maximum residency (4743 sample(s))
       127,824 bytes maximum slop
            37 MB total memory in use (0 MB lost due to fragmentation)

                                     Tot time (elapsed)  Avg pause  Max pause
Gen  0     78860 colls,     0 par   32.659s  32.970s     0.0004s    0.0669s
Gen  1      4743 colls,     0 par    3.043s   3.052s     0.0006s    0.0086s

TASKS: 174069 (174065 bound, 4 peak workers (4 total), using -N1)
```

(continues on next page)

(continued from previous page)

```
SPARKS: 0 (0 converted, 0 overflowed, 0 dud, 0 GC'd, 0 fizzled)

INIT    time    0.001s  (  0.001s elapsed)
MUT     time   98.685s  (101.611s elapsed)
GC      time   35.702s  ( 36.022s elapsed)
EXIT    time    0.001s  (  0.007s elapsed)
Total   time  134.388s  (137.640s elapsed)

Alloc rate    878,145,635 bytes per MUT second

Productivity  73.4% of total user, 73.8% of total elapsed
```

8.6 Heap profiling

GHC can tell you where the memory is going:

```
$ stack build --profile
$ stack exec  -- dejafu-tests +RTS -hc
$ hp2ps -c dejafu-tests.hp
```

This will produce a graph of memory usage over time, as a postscript file, broken down by cost-centre which produced the data. There are a few different views:

- `-hm` breaks down the graph by module
- `-hd` breaks down the graph by closure description
- `-hy` breaks down the graph by type

I typically find `-hd` and `-hy` most useful. If you're feeling particularly brave, you can try `-hr`, which is intended to help track down memory leaks caused by unevaluated thunks.

CHAPTER 9

Supported GHC Versions

Déjà Fu supports the latest four GHC releases, after GHC 8.0. For testing purposes, we use Stackage LTSes as a proxy for GHC versions. The currently supported versions are:

GHC	Stackage	base
8.8		4.13.0.0
8.6	LTS 13.0	4.12.0.0
8.4	LTS 12.0	4.11.0.0
8.2	LTS 10.0	4.10.1.0
8.0	LTS 9.0	4.9.1.0

In practice, we may *compile with* older versions of GHC, but keeping them working is not a priority.

9.1 Adding new GHC releases

When a new version of GHC is released, we need to make some changes for everything to go smoothly. In general, these changes should only cause a **patch level version bump**.

1. Bump the upper bound of base and set up any needed conditional compilation
2. Add the GHC and base versions to the table.
3. Remove any unsupported versions from the table.
4. Make a patch release.

A new GHC release won't get a Stackage LTS for little while. When it does:

1. Add the LTS to the Travis script.
2. Update the resolver in the stack.yaml.
3. Put the LTS in the table.

9.2 Dropping old GHC releases

When we want to drop an unsupported version of GHC, we need to bump the version bound on base to preclude it. This is a backwards-incompatible change which causes a **major version bump**.

1. Remove the dropped GHC version from the Travis script.
2. Bump the lower bound of base.
3. Look through the other dependencies. Some may not work with our new lower bound on base, so we should bump those too.
4. Remove any now-irrelevant conditional compilation (mostly CPP, but there may also be some cabal file bits).
5. Make whatever change required the bump.
6. Make a major release.

GHC versions shouldn't be dropped just because we can, but here are some good reasons to do it:

- We want to bump the lower bounds of a dependency to a version which doesn't support that GHC.
- We want to add a new dependency which doesn't support that GHC.
- The conditional compilation needed to keep that GHC working is getting confusing.

CHAPTER 10

Release Process

1. Figure out what the next version number is. See the PVP page if unsure.
2. Update version numbers in the relevant cabal files:
   - Update the `version` field
   - Update the `tag` in the `source-repository` block
3. Fill in all `@since unreleased` Haddock comments with the relevant version number.
4. Update version numbers in the tables in the README and the Getting Started page.
5. Ensure the relevant CHANGELOG files have all the entries they should.
6. Add the release information to the relevant CHANGELOG files:
   - Change the `unreleased` title to the version number
   - Add the current date

     If it's early in the year (like January or February), make sure you don't put down the wrong year.
   - Add the git tag name
   - Add the Hackage URL
   - Add the contributors list
7. Commit.
8. Push to GitHub and wait for Travis to confirm everything is OK. If it's not OK, fix what is broken before continuing.
9. Merge the PR.
10. Tag the merge commit. Tags are in the form `<package>-<version>`, and the message is the changelog entry.
11. Push tags to GitHub.

When the merge commit successfully builds on `master` the updated packages will be pushed to Hackage by Travis.

See pull request #284 and the Travis build of its merge commit if any of this is unclear.

10.1 Pro tips

- If a release would have a combination of breaking and non-breaking changes, if possible make two releases: the non-breaking ones first, and then a major release with the breaking ones.

  This makes it possible for users who don't want the breaking changes to still benefit from the non-breaking improvements.

- Before uploading to Hackage, check you have no changes to the files (for example, temporarily changing the GHC options, or adding `trace` calls, for debugging reasons).

  `stack upload` will upload the files on the disk, not the files in version control, so your unwanted changes will be published!

CHAPTER 11

Release Notes

This project is versioned according to the PVP, the *de facto* standard Haskell versioning scheme.

CHAPTER 12

1.8.1.0 (2019-11-16)

- Git: tag concurrency-1.8.1.0
- Hackage: concurrency-1.8.1.0

12.1 Added

- (issue #303) `Control.Monad.Conc.Class.newTVarConc`, with a default implementation of `atomically . newTVar`.

12.1.1 1.8.0.0 (2019-10-04)

- Git: tag concurrency-1.8.0.0
- Hackage: concurrency-1.8.0.0

12.2 Added

- `MonadFail` instances for `Control.Monad.Conc.Class.IsConc` and `Control.Monad.STM.IsSTM`.

12.3 Changed

- Added `MonadFail` constraints to `Control.Concurrent.Classy.QSem.newQSem` and `Control.Concurrent.Classy.QSemN.newQSemN`.

12.4 Miscellaneous

- Fixed a compilation error with GHC 8.8

12.4.1 1.7.0.0 (2019-03-24)

- Git: tag concurrency-1.7.0.0
- Hackage: concurrency-1.7.0.0

12.5 Added

- The `Control.Monad.Conc.Class.supportsBoundThreads` function, like `rtsSupportsBoundThreads` but returns a monadic result.

12.6 Deprecated

- `Control.Monad.Conc.Class.rtsSupportsBoundThreads`, in favour of `supportsBoundThreads`.

12.6.1 1.6.2.0 (2018-11-28)

- Git: tag concurrency-1.6.2.0
- Hackage: concurrency-1.6.2.0

Contributors: gip (pull request #289).

12.7 Added

- (pull request #289) The `Control.Concurrent.Classy.BoundedChan` module.
- (pull request #289) The `Control.Concurrent.Classy.Lock` module.
- (pull request #289) The `Control.Concurrent.Classy.RWLock` module.

12.7.1 1.6.1.0 (2018-09-23)

- Git: tag concurrency-1.6.1.0
- Hackage: concurrency-1.6.1.0

12.8 Added

- (issue #286) Copy across additions from the stm package:
  - `Control.Concurrent.Classy.STM.TQueue.flushTQueue`

  - `Control.Concurrent.Classy.STM.TBQueue.flushTBQueue`
  - `Control.Concurrent.Classy.STM.TBQueue.lengthTBQueue`
  - `Control.Concurrent.Classy.STM.TVar.stateTVar`
- (issue #287) The `Control.Concurrent.Classy.STM.TSem` module.

12.9 Changed

- (issue #286) Copy across changes from the stm package:
  - Make definition of `readTQueue` consistent with `readTBQueue`

12.10 Miscellaneous

- The upper bound on stm is <2.6.

12.10.1 1.6.0.0 - IORefs (2018-07-01)

- Git: tag concurrency-1.6.0.0
- Hackage: concurrency-1.6.0.0

12.11 Added

- `Control.Concurrent.Classy.CRef`, deprecated `*CRef` functions and a `CRef` alias.

12.12 Changed

- (issue #274) `CRef` is now `IORef`: all functions, modules, and types have been renamed.

12.12.1 1.5.0.0 - No More 7.10 (2018-03-28)

- Git: tag concurrency-1.5.0.0
- Hackage: concurrency-1.5.0.0

12.13 Added

- (issue #132) `forkOSWithUnmask` in `MonadConc`

12.14 Changed

- (issue #132) `Control.Monad.Conc.Class.fork`, `forkOn`, `forkOS`, and `forkOSN` are top-level definitions.

12.15 Miscellaneous

- GHC 7.10 support is dropped. Dependency lower bounds are:
  - base: 4.9
  - array: 0.5.1
  - transformers: 0.5

12.15.1 1.4.0.2 (2018-03-11)

- Git: tag concurrency-1.4.0.2
- Hackage: concurrency-1.4.0.2

12.16 Miscellaneous

- (pull request #245) The upper bound on exceptions is <0.11.

12.16.1 1.4.0.1 (2018-02-26)

- Git: tag concurrency-1.4.0.1
- Hackage: concurrency-1.4.0.1

12.17 Miscellaneous

- The upper bound on exceptions is <0.10.

12.17.1 1.4.0.0 (2018-01-19)

- Git: tag concurrency-1.4.0.0
- Hackage: concurrency-1.4.0.0

12.18 Changed

- `Control.Monad.Conc.Class.peekTicket'` has a more concrete type, to make deriving newtype instances of `MonadConc` possible:
  - Old: `MonadConc m => proxy m -> Ticket m a -> a`
  - New: `MonadConc m => Proxy m -> Ticket m a -> a`

12.18.1 1.3.0.0 - The Bound Thread Release (2017-12-23)

- Git: tag concurrency-1.3.0.0
- Hackage: concurrency-1.3.0.0

Note: bound threads are only supported if you compile with GHC and link with -threaded.

12.19 Added

- (pull request #145) Bound thread variants of the `withAsync` functions:
  - `Control.Concurrent.Classy.Async.asyncBound`
  - `Control.Concurrent.Classy.Async.asyncBoundN`
  - `Control.Concurrent.Classy.Async.withAsyncBound`
  - `Control.Concurrent.Classy.Async.withAsyncBoundN`
- (pull request #145) Bound thread functions in `MonadConc`:
  - `Control.Monad.Conc.Class.forkOS`
  - `Control.Monad.Conc.Class.forkOSN`
  - `Control.Monad.Conc.Class.isCurrentThreadBound`
- (pull request #145) Helper functions for bound threads:
  - `Control.Monad.Conc.Class.runInBoundThread`
  - `Control.Monad.Conc.Class.runInUnboundThread`

12.20 Changed

- (pull request #145) `Control.Monad.Conc.Class.rtsSupportsBoundThreads` is a re-export from `Control.Concurrent`.

12.20.1 1.2.3.0 (2017-11-30)

- Git: tag concurrency-1.2.3.0
- Hackage: concurrency-1.2.3.0

12.21 Added

- (issue #148) Named thread variants of the `withAsync` functions:
  - `Control.Concurrent.Classy.Async.withAsyncN`
  - `Control.Concurrent.Classy.Async.withAsyncOnN`
  - `Control.Concurrent.Classy.Async.withAsyncWithUnmaskN`
  - `Control.Concurrent.Classy.Async.withAsyncOnWithUnmaskN`

12.21.1 1.2.2.0 (2017-11-05)

- Git: tag concurrency-1.2.2.0
- Hackage: concurrency-1.2.2.0

12.22 Added

- (issue #144) `IsConc` and `IsSTM` wrapper types:
  - `Control.Monad.Conc.Class.IsConc` (constructor unexported)
  - `Control.Monad.Conc.Class.toIsConc`
  - `Control.Monad.Conc.Class.fromIsConc`
  - `Control.Monad.STM.Class.IsSTM` (constructor unexported)
  - `Control.Monad.STM.Class.toIsSTM`
  - `Control.Monad.STM.Class.fromIsSTM`

12.23 Changed

- `Control.Monad.Conc.Class.modifyCRefCAS_` for transformer instances delegates to the underlying monad, rather than using the default definition in terms of `modifyCRefCAS`.

12.23.1 1.2.1.2 (2017-10-14)

- Git: tag concurrency-1.2.1.2
- Hackage: concurrency-1.2.1.2

12.24 Fixed

- (issue #134) `Control.Monad.Conc.Class.forkWithUnmask` and `forkOnWithUnmask` for the `IO` instance does not infinitely loop (bug introduced in tag concurrency-1.2.1.1).

12.24.1 1.2.1.1 (2017-10-11)

- Git: tag concurrency-1.2.1.1
- Hackage: concurrency-1.2.1.1

12.25 Changed

- Named threads for `IO` are implemented with `GHC.Conc.labelThread`.

12.25.1 1.2.1.0 (2017-10-02)

- Git: tag concurrency-1.2.1.0
- Hackage: concurrency-1.2.1.0

12.26 Added

- (pull request #125) Named thread variants of the `async` functions:
  - `Control.Concurrent.Classy.Async.asyncN`
  - `Control.Concurrent.Classy.Async.asyncOnN`
  - `Control.Concurrent.Classy.Async.asyncWithUnmaskN`
  - `Control.Concurrent.Classy.Async.asyncOnWithUnmaskN`

12.26.1 1.2.0.0 (2017-09-16)

- Git: tag concurrency-1.2.0.0
- Hackage: concurrency-1.2.0.0

12.27 Changed

- `MonadPlus` is a superclass of `MonadSTM`.
- `Control.Monad.STM.Class.orElse` is a top-level alias for `mplus`.
- `Control.Monad.STM.Class.retry` is a top-level alias for `mzero`.

12.27.1 1.1.2.1 (2017-06-07)

- Git: tag concurrency-1.1.2.1
- Hackage: concurrency-1.1.2.1

12.28 Changed

- `Control.Concurrent.Classy.MVar.isEmptyMVar` does not briefly empties the `MVar`, and does not block.

12.28.1 1.1.2.0 (2017-04-05)

- Git: tag concurrency-1.1.2.0
- Hackage: concurrency-1.1.2.0

12.29 Added

- Missing functions copied from async:
  - `Control.Concurrent.Classy.Async.uninterruptibleCancel`
  - `Control.Concurrent.Classy.Async.replicateConcurrently`
  - `Control.Concurrent.Classy.Async.concurrently_`
  - `Control.Concurrent.Classy.Async.mapConcurrently_`
  - `Control.Concurrent.Classy.Async.forConcurrently_`
  - `Control.Concurrent.Classy.Async.replicateConcurrently_`
- `Control.Concurrent.Classy.Async.Concurrently` has a `Semigroup` instance when built with base >= 4.9.
- `Control.Concurrent.Classy.Async.Concurrently` has a `Monoid` instance.
- `Control.Monad.Conc.Class` re-exports `Control.Monad.Catch.mask_` and `uninterruptibleMask_`.

12.30 Changed

- (pull request #77) To match changes in async, `Control.Concurrent.Classy.Async.cancel` and `withAsync` block until the `Async` is killed.

12.31 Miscellaneous

- Every definition, class, and instance now has a Haddock `@since` annotation.

12.31.1 1.1.1.0 - The Async Release (2017-03-04)

- Git: tag concurrency-1.1.1.0
- Hackage: concurrency-1.1.1.0

12.32 Added

- The `Control.Concurrent.Classy.Async` module.

12.32.1 1.1.0.0 (2017-02-21)

- Git: tag concurrency-1.1.0.0
- Hackage: concurrency-1.1.0.0

12.33 Added

- `Control.Monad.Conc.Class.tryReadMVar`

12.34 Removed

- `Control.Monad.Conc.Class._concMessage`

12.34.1 1.0.0.0 - The Initial Release (2016-09-10)

- Git: tag concurrency-1.0.0.0
- Hackage: concurrency-1.0.0.0

12.35 Added

- Everything.

CHAPTER 13

Release Notes

This project is versioned according to the PVP, the *de facto* standard Haskell versioning scheme.

13.1 2.1.0.1 (2019-10-04)

- Git: tag dejafu-2.1.0.1
- Hackage: dejafu-2.1.0.1

13.1.1 Miscellaneous

- Fixed a compilation error with GHC 8.8
- The upper version bound on concurrency is <1.9.

13.2 2.1.0.0 (2019-03-24)

- Git: tag dejafu-2.1.0.0
- Hackage: dejafu-2.1.0.0

13.2.1 Added

- The `Test.DejaFu.Types.MonadDejaFu` typeclass, containing the primitives needed to run a concurrent program. There are instances for:
  - `IO`, which is probably the `MonadConc` instance people used previously, so there is no breaking change there.
  - `CatchT (ST t)`, meaning that concurrent programs can be run without `IO` once more.

- Thread action constructors for `MonadConc` `supportsBoundThreads` function:
  - `Test.DejaFu.Types.ThreadAction`, `SupportsBoundThreads`
  - `Test.DejaFu.Types.Lookahead`, `WillSupportsBoundThreads`

13.2.2 Changed

- Many functions which had a `MonadConc` constraint now have a `MonadDejaFu` constraint:
  - **In `Test.DejaFu`**
    - `autocheck`
    - `autocheckWay`
    - `autocheckWithSettings`
    - `dejafu`
    - `dejafuWay`
    - `dejafuWithSettings`
    - `dejafus`
    - `dejafusWay`
    - `dejafusWithSettings`
    - `runTest`
    - `runTestWay`
    - `runTestWithSettings`
  - **In `Test.DejaFu.Conc`**
    - `runConcurrent`
    - `recordSnapshot`
    - `runSnapshot`
  - **In `Test.DejaFu.SCT`**
    - `runSCT`
    - `resultsSet`
    - `runSCT'`
    - `resultsSet'`
    - `runSCTWithSettings`
    - `resultsSetWithSettings`
    - `runSCTWithSettings'`
    - `resultsSetWithSettings'`

13.2.3 Miscellaneous

- The version bound on concurrency is >=1.7 and <1.8.

13.3 2.0.0.1 (2019-03-14)

- Git: tag dejafu-2.0.0.1
- Hackage: dejafu-2.0.0.1

13.3.1 Fixed

- (issue #267) Throwing an asynchronous exception to the current thread interrupts the current thread even if it is masked.

13.4 2.0.0.0 (2019-02-12)

- Git: tag dejafu-2.0.0.0
- Hackage: dejafu-2.0.0.0

13.4.1 Added

- The `Program` types and their constructors (re-exported from `Test.DejaFu`):
  - `Test.DejaFu.Conc.Program`
  - `Test.DejaFu.Conc.Basic`
  - `Test.DejaFu.Conc.WithSetup`
  - `Test.DejaFu.Conc.WithSetupAndTeardown`
  - `Test.DejaFu.Conc.withSetup`
  - `Test.DejaFu.Conc.withTeardown`
  - `Test.DejaFu.Conc.withSetupAndTeardown`
- The `Invariant` type and associated functions (re-exported from `Test.DejaFu`):
  - `Test.DejaFu.Conc.Invariant`
  - `Test.DejaFu.Conc.registerInvariant`
  - `Test.DejaFu.Conc.inspectIORef`
  - `Test.DejaFu.Conc.inspectMVar`
  - `Test.DejaFu.Conc.inspectTVar`
- **New snapshotting functions:**
  - `Test.DejaFu.Conc.Snapshot`
  - `Test.DejaFu.Conc.recordSnapshot`
  - `Test.DejaFu.Conc.runSnapshot`
- `Test.DejaFu.Settings.llengthBound`, which now applies to all ways of testing.
- `Test.DejaFu.Types.isInvariantFailure` (re-exported from `Test.DejaFu`).
- `Test.DejaFu.runTestWithSettings` function.

- **A simplified form of the concurrency state:**
  - `Test.DejaFu.Types.ConcurrencyState`
  - `Test.DejaFu.Types.isBuffered`
  - `Test.DejaFu.Types.numBuffered`
  - `Test.DejaFu.Types.isFull`
  - `Test.DejaFu.Types.canInterrupt`
  - `Test.DejaFu.Types.canInterruptL`
  - `Test.DejaFu.Types.isMaskedInterruptible`
  - `Test.DejaFu.Types.isMaskedUninterruptible`

13.4.2 Changed

- `Test.DejaFu.Schedule.Scheduler` has a `ConcurrencyState` parameter.
- `Test.DejaFu.alwaysSameBy` and `Test.DejaFu.notAlwaysSameBy` return a representative trace for each unique condition.
- **Functions which took a `ConcT` now take a `Program pty`:**
  - `Test.DejaFu.autocheck`
  - `Test.DejaFu.autocheckWay`
  - `Test.DejaFu.autocheckWithSettings`
  - `Test.DejaFu.dejafu`
  - `Test.DejaFu.dejafuWay`
  - `Test.DejaFu.dejafuWithSettings`
  - `Test.DejaFu.dejafus`
  - `Test.DejaFu.dejafusWay`
  - `Test.DejaFu.dejafusWithSettings`
  - `Test.DejaFu.runTest`
  - `Test.DejaFu.runTestWay`
  - `Test.DejaFu.runTestWithSettings`
  - `Test.DejaFu.Conc.runConcurrent`
  - `Test.DejaFu.SCT.runSCT`
  - `Test.DejaFu.SCT.resultsSet`
  - `Test.DejaFu.SCT.runSCT'`
  - `Test.DejaFu.SCT.resultsSet'`
  - `Test.DejaFu.SCT.runSCTWithSettings`
  - `Test.DejaFu.SCT.resultsSetWithSettings`
  - `Test.DejaFu.SCT.runSCTWithSettings'`
  - `Test.DejaFu.SCT.resultsSetWithSettings'`

- `Test.DejaFu.Conc.ConcT` is an alias for `Program Basic`.
- **`Test.DejaFu.Types.Bounds`:**
  - Removed `boundLength` field.
- **`Test.DejaFu.Types.Condition`:**
  - Added `InvariantFailure` constructor
  - Removed `STMDeadlock` constructor
- **`Test.DejaFu.Types.Error`:**
  - Removed `NestedSubconcurrency`, `MultithreadedSubconcurrency`, and `LateDontCheck` constructors.
- **`Test.DejaFu.Types.Lookahead`:**
  - Added `WillRegisterInvariant` constructor
  - Removed `WillSubconcurrency`, `WillStopSubconcurrency`, and `WillDontCheck` constructors
- **`Test.DejaFu.Types.ThreadAction`:**
  - Added `RegisterInvariant` constructor
  - Removed `Subconcurrency`, `StopSubconcurrency`, and `DontCheck` constructors

13.4.3 Removed

- **The deprecated functions:**
  - `Test.DejaFu.dejafuDiscard`
  - `Test.DejaFu.SCT.runSCTDiscard`
  - `Test.DejaFu.SCT.runSCTDiscard'`
  - `Test.DejaFu.SCT.resultsSetDiscard`
  - `Test.DejaFu.SCT.resultsSetDiscard'`
  - `Test.DejaFu.SCT.sctBound`
  - `Test.DejaFu.SCT.sctBoundDiscard`
  - `Test.DejaFu.SCT.sctUniformRandom`
  - `Test.DejaFu.SCT.sctUniformRandomDiscard`
  - `Test.DejaFu.SCT.sctWeightedRandom`
  - `Test.DejaFu.SCT.sctWeightedRandomDiscard`
- The deprecated type `Test.DejaFu.Types.Failure`
- **Old snapshotting functions:**
  - `Test.DejaFu.Conc.DCSnapshot`
  - `Test.DejaFu.Conc.runForDCSnapshot`
  - `Test.DejaFu.Conc.runWithDCSnapshot`
  - `Test.DejaFu.Conc.canDCSnapshot`
  - `Test.DejaFu.Conc.threadsFromDCSnapshot`

- `Test.DejaFu.Conc.dontCheck`
- `Test.DejaFu.Conc.subconcurrency`
- `Test.DejaFu.Settings.defaultLengthBound`
- `Test.DejaFu.Types.isIncorrectUsage`

13.5 1.12.0.0 (2019-01-20)

- Git: tag dejafu-1.12.0.0
- Hackage: dejafu-1.12.0.0

13.5.1 Added

- `Test.DejaFu.Types.Error` for internal errors and misuses, with predicates:
  - `Test.DejaFu.Types.isSchedulerError`
  - `Test.DejaFu.Types.isIncorrectUsage`
- Deprecated `Test.DejaFu.Types.Failure` type synonym for `Condition`.
- The `Test.DejaFu.Settings.lshowAborts` option, to make SCT functions show `Abort` conditions.
- `Test.DejaFu.Utils.showCondition`

13.5.2 Changed

- Renamed `Test.DejaFu.Types.Failure` to `Test.DejaFu.Types.Condition`.
- The SCT functions drop `Left Abort` results by default, restore the old behaviour with `Test.DejaFu.Settings.lshowAborts`.

13.5.3 Removed

- `Test.DejaFu.Types.isInternalError`
- `Test.DejaFu.Types.isIllegalDontCheck`
- `Test.DejaFu.Types.isIllegalSubconcurrency`
- `Test.DejaFu.Utils.showFail`

13.6 1.11.0.5 (2019-01-17)

- Git: tag dejafu-1.11.0.5
- Hackage: dejafu-1.11.0.5

13.6.1 Miscellaneous

- The upper bound on leancheck is <0.10.

13.7 1.11.0.4 (2018-12-02)

- Git: tag dejafu-1.11.0.4
- Hackage: dejafu-1.11.0.4

Contributors: pepeiborra (pull request #290).

13.7.1 Miscellaneous

- (pull request #290) The upper bound on containers is <0.7.
- (pull request #290) The upper bound on leancheck is <0.9.

13.8 1.11.0.3 (2018-07-15)

- Git: tag dejafu-1.11.0.3
- Hackage: dejafu-1.11.0.3

13.8.1 Fixed

- (issue #275) In trace simplification, only remove a commit if there are no other buffered writes for that same *IORef*.

13.9 1.11.0.2 (2018-07-08)

- Git: tag dejafu-1.11.0.2
- Hackage: dejafu-1.11.0.2

13.9.1 Miscellaneous

- The upper bound on profunctors is <6.

13.10 1.11.0.1 (2018-07-02)

- Git: tag dejafu-1.11.0.1
- Hackage: dejafu-1.11.0.1

13.10.1 Miscellaneous

- The upper bound on contravariant is <1.6.

13.11 1.11.0.0 - IORefs (2018-07-01)

- Git: tag dejafu-1.11.0.0
- Hackage: dejafu-1.11.0.0

13.11.1 Changed

- (issue #274) `CRef` is now `IORef`: all functions, data constructors, and types have been renamed.
- The lower bound on concurrency is 1.6.

13.12 1.10.1.0 (2018-06-17)

- Git: tag dejafu-1.10.1.0
- Hackage: dejafu-1.10.1.0

13.12.1 Added

- (issue #224) The `Test.DejaFu.Settings.lsafeIO` option, for when all lifted IO is thread-safe (such as exclusively managing thread-local state).

13.13 1.10.0.0 (2018-06-17)

- Git: tag dejafu-1.10.0.0
- Hackage: dejafu-1.10.0.0

13.13.1 Added

- The `Test.DejaFu.notAlwaysSameOn` and `notAlwaysSameBy` predicates, generalising `notAlwaysSame`.

13.13.2 Changed

- `Test.DejaFu.autocheck` and related functions use the `successful` predicate, rather than looking specifically for deadlocks and uncaught exceptions.
- (issue #259) The `Test.DejaFu.alwaysSame`, `alwaysSameOn`, `alwaysSameBy`, and `notAlwaysSame` predicates fail if the computation under test fails.

13.14 1.9.1.0 (2018-06-10)

- Git: tag dejafu-1.9.1.0
- Hackage: dejafu-1.9.1.0

13.14.1 Added

- A `Test.DejaFu.successful` predicate, to check that a computation never fails.

13.15 1.9.0.0 (2018-06-10)

- Git: tag dejafu-1.9.0.0
- Hackage: dejafu-1.9.0.0

13.15.1 Changed

- (issue #190) `Test.DejaFu.Types.Throw` and `ThrowTo` have a `Bool` parameter, which is `True` if the exception kills the thread.

13.16 1.8.0.0 (2018-06-03)

- Git: tag dejafu-1.8.0.0
- Hackage: dejafu-1.8.0.0

13.16.1 Changed

- (issue #258) Length bounding is disabled by default. This is not a breaking API change, but it is a breaking semantics change.

13.17 1.7.0.0 (2018-06-03)

- Git: tag dejafu-1.7.0.0
- Hackage: dejafu-1.7.0.0

13.17.1 Changed

- (issue #237) `Test.DejaFu.SCT.sctWeightedRandom` and `sctWeightedRandomDiscard` no longer take the number of executions to use the same weights for as a parameter.

13.17.2 Removed

- (issue #237) The deprecated function `Test.DejaFu.Settings.swarmy`.

13.18 1.6.0.0 (2018-05-11)

- Git: tag dejafu-1.6.0.0
- Hackage: dejafu-1.6.0.0

13.18.1 Removed

- The deprecated module `Test.DejaFu.Defaults`.

13.19 1.5.1.0 (2018-03-29)

- Git: tag dejafu-1.5.1.0
- Hackage: dejafu-1.5.1.0

13.19.1 Added

- (issue #210) `Test.DejaFu.Types.Weaken` and `Strengthen` newtype wrappers around discard functions, with `Semigroup`, `Monoid`, `Contravariant`, and `Divisible` instances corresponding to `weakenDiscard` and `strengthenDiscard`.

13.20 1.5.0.0 - No More 7.10 (2018-03-28)

- Git: tag dejafu-1.5.0.0
- Hackage: dejafu-1.5.0.0

13.20.1 Miscellaneous

- GHC 7.10 support is dropped. Dependency lower bounds are:
  - base: 4.9
  - concurrency: 1.5
  - transformers: 0.5
- The upper bound on concurrency is 1.6.

13.21 1.4.0.0 (2018-03-17)

- Git: tag dejafu-1.4.0.0
- Hackage: dejafu-1.4.0.0

13.21.1 Changed

- (issue #201) `Test.DejaFu.Conc.ConcT r n a` drops its `r` parameter, becoming `ConcT n a`.
- (issue #201) All functions drop the `MonadConc` constraint.

13.21.2 Removed

- (issue #201) The `MonadRef` and `MonadAtomicRef` instances for `Test.DejaFu.Conc.ConcT`.
- (issue #198) The `Test.DejaFu.Types.Killed` thread action, which was unused.

13.21.3 Fixed

- (issue #250) Add missing dependency for `throwTo` actions.

13.22 1.3.2.0 (2018-03-12)

- Git: tag dejafu-1.3.2.0
- Hackage: dejafu-1.3.2.0

13.22.1 Added

- (issue #183) SCT settings for trace simplification:
  - `Test.DejaFu.Settings.lequality`
  - `Test.DejaFu.Settings.lsimplify`
- (pull request #248) `Test.DejaFu.Utils.toTIdTrace` to extract thread IDs from a trace.
- (pull request #248) SCT setting to make some recoverable errors fatal: `Test.DejaFu.Settings.ldebugFatal`

13.22.2 Performance

- (pull request #248) Prune some unnecessary interleavings of `CRef` actions in systematic testing when using sequential consistency.

13.23 1.3.1.0 (2018-03-11)

- Git: tag dejafu-1.3.1.0
- Hackage: dejafu-1.3.1.0

13.23.1 Added

- (pull request #246) `Generic` instances for:
  - `Test.DejaFu.Types.ThreadId`
  - `Test.DejaFu.Types.CRefId`
  - `Test.DejaFu.Types.MVarId`
  - `Test.DejaFu.Types.TVarId`
  - `Test.DejaFu.Types.Id`

  - `Test.DejaFu.Types.ThreadAction`
  - `Test.DejaFu.Types.Lookahead`
  - `Test.DejaFu.Types.TAction`
  - `Test.DejaFu.Types.Decision`
  - `Test.DejaFu.Types.Failure`
  - `Test.DejaFu.Types.Bounds`
  - `Test.DejaFu.Types.PreemptionBound`
  - `Test.DejaFu.Types.FairBound`
  - `Test.DejaFu.Types.LengthBound`
  - `Test.DejaFu.Types.Discard`
  - `Test.DejaFu.Types.MemType`
  - `Test.DejaFu.Types.MonadFailException`
- (pull request #246) `NFData` instance for `Test.DejaFu.Types.MonadFailException`

13.23.2 Fixed

- (issue #199) Missing cases in the `NFData` instances for `Test.DejaFu.Types.ThreadAction` and `TAction`

13.24 1.3.0.3 (2018-03-11)

- Git: tag dejafu-1.3.0.3
- Hackage: dejafu-1.3.0.3

13.24.1 Miscellaneous

- (pull request #245) The upper bound on exceptions is <0.11.

13.25 1.3.0.2 (2018-03-11)

- Git: tag dejafu-1.3.0.2
- Hackage: dejafu-1.3.0.2

13.25.1 Fixed

- (pull request #244) Add missing dependency for `setNumCapabilities` actions.

13.26 1.3.0.1 (2018-03-08)

- Git: tag dejafu-1.3.0.1
- Hackage: dejafu-1.3.0.1

13.26.1 Fixed

- (pull request #242) A compilation error when building with exceptions-0.9.0.

13.27 1.3.0.0 (2018-03-06)

- Git: tag dejafu-1.3.0.0
- Hackage: dejafu-1.3.0.0

13.27.1 Deprecated

- (pull request #240) `Test.DejaFu.Settings.swarmy`

13.28 1.2.0.0 - The Settings Release (2018-03-06)

- Git: tag dejafu-1.2.0.0
- Hackage: dejafu-1.2.0.0

Contributors: qrilka (pull request #236).

13.28.1 Added

- (pull request #238) A record-based approach to SCT configuration:
  - `Test.DejaFu.Settings` (re-exported from `Test.Dejafu` and `Test.DejaFu.SCT`)
  - `Test.DejaFu.Settings.Settings`
  - `Test.DejaFu.Settings.defaultSettings`
  - `Test.DejaFu.Settings.fromWayAndMemType`
  - **Lenses:**
    * `Test.DejaFu.Settings.lway`
    * `Test.DejaFu.Settings.lmemtype`
    * `Test.DejaFu.Settings.ldiscard`
    * `Test.DejaFu.Settings.learlyExit`
    * `Test.DejaFu.Settings.ldebugShow`
    * `Test.DejaFu.Settings.ldebugPrint`
  - **Lens helpers:**

        * `Test.DejaFu.Settings.get`
        * `Test.DejaFu.Settings.set`
    - **Runners:**
        * `Test.DejaFu.SCT.runSCTWithSettings`
        * `Test.DejaFu.SCT.runSCTWithSettings'`
        * `Test.DejaFu.SCT.resultsSetWithSettings`
        * `Test.DejaFu.SCT.resultsSetWithSettings'`
- (pull request #238) Settings-based test functions:
    - `Test.DejaFu.autocheckWithSettings`
    - `Test.DejaFu.dejafuWithSettings`
    - `Test.DejaFu.dejafusWithSettings`
    - `Test.DejaFu.runTestWithSettings`

13.28.2 Deprecated

- (pull request #238) SCT function variants:
    - `Test.DejaFu.SCT.runSCTDiscard`
    - `Test.DejaFu.SCT.resultSetDiscard`
    - `Test.DejaFu.SCT.runSCTDiscard'`
    - `Test.DejaFu.SCT.resultSetDiscard'`
    - `Test.DejaFu.SCT.sctBound`
    - `Test.DejaFu.SCT.sctBoundDiscard`
    - `Test.DejaFu.SCT.sctUniformRandom`
    - `Test.DejaFu.SCT.sctUniformRandomDiscard`
    - `Test.DejaFu.SCT.sctWeightedRandom`
    - `Test.DejaFu.SCT.sctWeightedRandomDiscard`
- (pull request #238) The `Test.DejaFu.Defaults` module. Import `Test.DejaFu.Settings` instead.
- (pull request #238) `Test.DejaFu.dejafuDiscard`.

13.28.3 Removed

- (pull request #238) `Test.DejaFu.Defaults.defaultDiscarder`, as the discard function is optional.

13.29 1.1.0.2 (2018-03-01)

- Git: tag dejafu-1.1.0.2
- Hackage: dejafu-1.1.0.2

13.29.1 Miscellaneous

- (pull request #235) The documentation for `Test.DejaFu.Conc.dontCheck` and `subconcurrency` clarify that an illegal use does not necessarily cause a failing test.

13.30 1.1.0.1 (2018-02-26)

- Git: tag dejafu-1.1.0.1
- Hackage: dejafu-1.1.0.1

Contributors: qrilka (pull request #229).

13.30.1 Miscellaneous

- The upper bound on exceptions is <0.10.

13.31 1.1.0.0 (2018-02-22)

- Git: tag dejafu-1.1.0.0
- Hackage: dejafu-1.1.0.0

Contributors: qrilka (pull request #228).

13.31.1 Added

- (pull request #219) The testing-only `Test.DejaFu.Conc.dontCheck` function, and associated definitions:
  - `Test.DejaFu.Types.DontCheck`
  - `Test.DejaFu.Types.WillDontCheck`
  - `Test.DejaFu.Types.IllegalDontCheck`
  - `Test.DejaFu.Types.isIllegalDontCheck`
- (pull request #219) A snapshotting approach based on `Test.DejaFu.Conc.dontCheck`:
  - `Test.DejaFu.Conc.runForDCSnapshot`
  - `Test.DejaFu.Conc.runWithDCSnapshot`
  - `Test.DejaFu.Conc.canDCSnapshot`
  - `Test.DejaFu.Conc.threadsFromDCSnapshot`

13.31.2 Changed

- (pull request #219) SCT functions automatically use the snapshotting mechanism when possible.

13.32 1.0.0.2 (2018-02-18)

- Git: tag dejafu-1.0.0.2
- Hackage: dejafu-1.0.0.2

Contributors: qrilka (pull request #214).

13.32.1 Changed

- (issue #193) Deterministically assign commit thread IDs.

13.32.2 Fixed

- (issue #189) Remove an incorrect optimisation in systematic testing for `getNumCapabilities` and `setNumCapabilities`.
- (issue #204) Fix missed interleavings in systematic testing with some uses of STM.
- (issue #205) Fix `forkOS` being recorded in an execution trace as if it were a `fork`.

13.32.3 Miscellaneous

- (pull request #180) Doctest Haddock examples in `Test.DejaFu` and `Test.DejaFu.Refinement`.
- (pull request #185, pull request #215) Check some more internal invariants and throw on error.
- (pull request #214) Remove unnecessary use of `head`.

13.33 1.0.0.1 (2018-01-19)

- Git: tag dejafu-1.0.0.1
- Hackage: dejafu-1.0.0.1

13.33.1 Miscellaneous

- The upper bound on concurrency is <1.5.

13.34 1.0.0.0 - The API Friendliness Release (2017-12-23)

- Git: tag dejafu-1.0.0.0
- Hackage: dejafu-1.0.0.0

13.34.1 Added

- `Test.DejaFu.alwaysSameOn` and `alwaysSameBy` predicate helpers.
- `Test.DejaFu.SCT.strengthenDiscard` and `weakenDiscard` functions to combine discard functions.
- (issue #124) The `Test.DejaFu.ProPredicate` type, which contains both an old-style `Predicate` and a discard function. It is also a `Profunctor`, parameterised by the input and output types.
- (issue #124) `Test.DejaFu.alwaysNothing` and `somewhereNothing` predicate helpers, like `alwaysTrue` and `somewhereTrue`, to lift regular functions into a `ProPredicate`.
- (issue #137) The `Test.DejaFu.Types.Id` type.
- (pull request #145) Thread action and lookahead values for bound threads:
  - `Test.DejaFu.Types.ForkOS`
  - `Test.DejaFu.Types.IsCurrentThreadBound`
  - `Test.DejaFu.Types.WillForkOS`
  - `Test.DejaFu.Types.WillIsCurrentThreadBound`
- (issue #155) `Test.DejaFu.Types` and `Test.DejaFu.Utils` modules, each containing some of what was in `Test.DejaFu.Common`.

13.34.2 Changed

- All testing functions require `MonadConc`, `MonadRef`, and `MonadIO` constraints. Testing with `ST` is no longer possible.
- The `Test.DejaFu.alwaysSame` predicate helper gives the simplest trace leading to each distinct result.
- The `MonadIO Test.DejaFu.Conc.ConcIO` instance is now the more general `MonadIO n => MonadIO (ConcT r n)`.
- (issue #121) The chosen thread is no longer redundantly included in trace lookahead.
- (issue #123) All testing functions in `Test.DejaFu` take the action to run as the final parameter.
- (issue #124) All testing functions in `Test.DejaFu` have been generalised to take a `ProPredicate` instead of a `Predicate`.
- (issue #124) The `Test.DejaFu.Predicate` type is an alias for `ProPredicate a a`.
- (issue #124) The `Test.DejaFu.Result` type no longer includes a number of cases checked.
- (issue #137) The `Test.DejaFu.Types.ThreadId`, `CRefId`, `MVarId`, and `TVarId` types are now wrappers for an `Id`.
- (pull request #145) If built with the threaded runtime, the main thread in a test is executed as a bound thread.
- (issue #155) The `Test.DejaFu.SCT.Discard` type is defined in `Test.DejaFu.Types`, and re-exported from `Test.DejaFu.SCT`.
- (issue #155) The `Test.DejaFu.Schedule.tidOf` and `decisionOf` functions are defined in `Test.DejaFu.Utils`, but not re-exported from `Test.DejaFu.Schedule`.

13.34.3 Removed

- The `IO` specific testing functions:
  - `Test.DejaFu.autocheckIO`
  - `Test.DejaFu.dejafuIO`
  - `Test.DejaFu.dejafusIO`
  - `Test.DejaFu.autocheckWayIO`
  - `Test.DejaFu.dejafuWayIO`
  - `Test.DejaFu.dejafusWayIO`
  - `Test.DejaFu.dejafuDiscardIO`
  - `Test.DejaFu.runTestM`
  - `Test.DejaFu.runTestWayM`
- The `Test.DejaFu.Conc.ConcST` type alias.
- The `MonadBaseControl IO Test.DejaFu.Conc.ConcIO` typeclass instance.
- The `Test.DejaFu.alwaysTrue2` function, which had confusing behaviour.
- The `Test.DejaFu.Common.TTrace` type synonym for `[TAction]`.
- The `Test.DejaFu.Common.preEmpCount` function.
- Re-exports of `Decision` and `NonEmpty` from `Test.DejaFu.Schedule`.
- (issue #155) The `Test.DejaFu.Common` and `Test.DejaFu.STM` modules.

13.34.4 Fixed

- In refinement property testing, a blocking interference function is not reported as a deadlocking execution.

13.34.5 Performance

- (issue #124) Passing tests should use substantially less memory.
- (issue #168) Prune some unnecessary interleavings of `MVar` actions in systematic testing.

13.34.6 Miscellaneous

- The lower bound on concurrency is >=1.3.

13.35 0.9.1.2 (2017-12-12)

- Git: tag dejafu-0.9.1.2
- Hackage: dejafu-0.9.1.2

13.35.1 Miscellaneous

- The upper bound on leancheck is <0.8.

13.36 0.9.1.1 (2017-12-08)

- Git: tag dejafu-0.9.1.1
- Hackage: dejafu-0.9.1.1

13.36.1 Fixed

- (issue #160) Fix an off-by-one issue with nested masks during systematic testing.

13.37 0.9.1.0 (2017-11-26)

- Git: tag dejafu-0.9.1.0
- Hackage: dejafu-0.9.1.0

13.37.1 Added

- `MonadFail` instance for `Test.DejaFu.Conc.ConcT`.
- `MonadFail` instance for `Test.DejaFu.STM.STMLike`.

13.37.2 Changed

- Pretty-printed traces display a pre-emption following a yield with a little "p".

13.37.3 Fixed

- Some incorrect Haddock `@since` comments.

13.38 0.9.0.3 (2017-11-06)

- Git: tag dejafu-0.9.0.3
- Hackage: dejafu-0.9.0.3

13.38.1 Fixed

- (issue #138) Fix missed interleavings in systematic testing with some relaxed memory programs.

13.39 0.9.0.2 (2017-11-02)

- Git: tag dejafu-0.9.0.2
- Hackage: dejafu-0.9.0.2

13.39.1 Changed

- A fair bound of 0 prevents yielding or delaying.

13.39.2 Performance

- Prune some unnecessary interleavings of STM transactions in systematic testing.

13.40 0.9.0.1 (2017-10-28)

- Git: tag dejafu-0.9.0.1
- Hackage: dejafu-0.9.0.1

13.40.1 Fixed

- (issue #139) Fix double pop of exception handler stack.

13.41 0.9.0.0 (2017-10-11)

- Git: tag dejafu-0.9.0.0
- Hackage: dejafu-0.9.0.0

13.41.1 Added

- Failure predicates (also exported from `Test.DejaFu`):
  - `Test.DejaFu.Common.isAbort`
  - `Test.DejaFu.Common.isDeadlock`
  - `Test.DejaFu.Common.isIllegalSubconcurrency`
  - `Test.DejaFu.Common.isInternalError`
  - `Test.DejaFu.Common.isUncaughtException`
- Thread action and lookahead values for `threadDelay`:
  - `Test.DejaFu.Common.ThreadDelay`
  - `Test.DejaFu.Common.WillThreadDelay`

13.41.2 Changed

- The `UncaughtException` constructor for `Test.DejaFu.Common.Failure` now includes the exception value.
- Uses of `threadDelay` are no longer reported in the trace as a use of `yield`.

13.41.3 Removed

- The `Bounded`, `Enum`, and `Read` instances for `Test.DejaFu.Common.Failure`.

13.42 0.8.0.0 (2017-09-26)

- Git: tag dejafu-0.8.0.0
- Hackage: dejafu-0.8.0.0

13.42.1 Changed

- (issue #80) STM traces now include the ID of a newly-created `TVar`.
- (issue #106) Schedulers are not given the execution trace so far.
- (issue #120) Traces only include a single action of lookahead.
- (issue #122) The `Test.DejaFu.Scheduler.Scheduler` type is now a newtype, rather than a type synonym.

13.43 0.7.3.0 (2017-09-26)

- Git: tag dejafu-0.7.3.0
- Hackage: dejafu-0.7.3.0

13.43.1 Added

- The `Test.DejaFu.Common.threadNames` function.

13.43.2 Fixed

- (issue #101) Named threads which are only started by a pre-emption are shown in the pretty-printed trace key.
- (issue #118) Escaping a mask by raising an exception correctly restores the masking state (#118).

13.44 0.7.2.0 (2017-09-16)

- Git: tag dejafu-0.7.2.0
- Hackage: dejafu-0.7.2.0

13.44.1 Added

- `Alternative` and `MonadPlus` instances for `Test.DejaFu.STM.STM`.

13.44.2 Fixed

- The `Eq` and `Ord` instances for `Test.DejaFu.Common.ThreadId`, `CRefId`, `MVarId`, and `TVarId` are consistent.

13.44.3 Miscellaneous

- The upper bound on concurrency is <1.2.

13.45 0.7.1.3 (2017-09-08)

- Git: tag dejafu-0.7.1.3
- Hackage: dejafu-0.7.1.3

13.45.1 Fixed

- (issue #111) Aborted STM transactions are correctly rolled back.

13.45.2 Performance

- (issue #105) Use a more efficient approach for an internal component of the systematic testing.

13.46 0.7.1.2 (2017-08-21)

- Git: tag dejafu-0.7.1.2
- Hackage: dejafu-0.7.1.2

13.46.1 Fixed

- (issue #110) Errors thrown with `Control.Monad.fail` are correctly treated as asynchronous exceptions.

13.47 0.7.1.1 (2017-08-16)

- Git: tag dejafu-0.7.1.1
- Hackage: dejafu-0.7.1.1

13.47.1 Performance

- (issue #64) Greatly reduce memory usage in systematic testing when discarding traces by using an alternative data structure.
  - Old: `O(max trace length * number of executions)`
  - New: `O(max trace length * number of traces kept)`

13.48 0.7.1.0 - The Discard Release (2017-08-10)

- Git: tag dejafu-0.7.1.0
- Hackage: dejafu-0.7.1.0

13.48.1 Added

- (issue #90) A way to selectively discard results or traces:
  - Type: `Test.DejaFu.SCT.Discard`
  - Functions: `Test.DejaFu.SCT.runSCTDiscard`, `resultsSetDiscard`, `sctBoundDiscard`, `sctUniformRandomDiscard`, and `sctWeightedRandomDiscard`.
- (issue #90) Discarding variants of the testing functions:
  - `Test.DejaFu.dejafuDiscard`
  - `Test.DejaFu.dejafuDiscardIO`
- (issue #90) `Test.DejaFu.Defaults.defaultDiscarder`.

13.48.2 Performance

- (issue #90) The `Test.DejaFu.SCT.resultsSet` and `resultsSet'` functions discard traces as they are produced, rather than all at the end.

13.49 0.7.0.2 (2017-06-12)

- Git: tag dejafu-0.7.0.2
- Hackage: dejafu-0.7.0.2

13.49.1 Changed

- Remove unnecessary typeclass constraints from `Test.DejaFu.Refinement.check`, `check'`, `checkFor`, and `counterExamples`.

13.49.2 Miscellaneous

- Remove an unnecessary dependency on monad-loops.

13.50 0.7.0.1 (2017-06-09)

- Git: tag dejafu-0.7.0.1
- Hackage: dejafu-0.7.0.1

13.50.1 Performance

- The `Test.DejaFu.Refinement.check`, `check'`, and `checkFor` functions no longer need to compute all counterexamples before showing only one.
- The above and `counterExamples` are now faster even if there is only a single counterexample in some cases.

13.51 0.7.0.0 - The Refinement Release (2017-06-07)

- Git: tag dejafu-0.7.0.0
- Hackage: dejafu-0.7.0.0

13.51.1 Added

- The `Test.DejaFu.Refinement` module, re-exported from `Test.DejaFu`.
- The `Test.DejaFu.SCT.sctUniformRandom` function for SCT via random scheduling.
- Smart constructors for `Test.DejaFu.SCT.Way` (also re-exported from `Test.DejaFu`):
  - `Test.DejaFu.SCT.systematically`, like the old `Systematically`.
  - `Test.DejaFu.SCT.randomly`, like the old `Randomly`.
  - `Test.DejaFu.SCT.uniformly`, a new uniform (as opposed to weighted) random scheduler.
  - `Test.DejaFu.SCT.swarmy`, like the old `Randomly` but which can use the same weights for multiple executions.

13.51.2 Changed

- The `default*` values are defined in `Test.DejaFu.Defaults` and re-exported from `Test.DejaFu`.
- The `Test.DejaFu.SCT.sctRandom` function is now called `sctWeightedRandom` and can re-use the same weights for multiple executions.

13.51.3 Removed

- The `Test.DejaFu.SCT.Way` type is now abstract, so its constructors are no longer exported:
  - `Test.DejaFu.SCT.Systematically`
  - `Test.DejaFu.SCT.Randomly`
- The `Test.DejaFu.SCT.sctPreBound`, `sctFairBound`, and `sctLengthBound` functions.

13.51.4 Fixed

- (issue #81) `Test.DejaFu.Conc.subconcurrency` no longer re-uses IDs.

13.52 0.6.0.0 (2017-04-08)

- Git: tag dejafu-0.6.0.0
- Hackage: dejafu-0.6.0.0

13.52.1 Changed

- The `Test.DejaFu.Conc.Conc n r a` type is `ConcT r n a`, and has a `MonadTrans` instance.
- The `Test.DejaFu.SCT.Way` type is a GADT, and does not expose the type parameter of the random generator.

13.52.2 Removed

- The `NFData` instance for `Test.DejaFu.SCT.Way`.

13.52.3 Miscellaneous

- `Test.DejaFu.Common` forms part of the public API.
- Every definition, class, and instance now has a Haddock `@since` annotation.

13.53 0.5.1.3 (2017-04-05)

- Git: tag dejafu-0.5.1.3
- Hackage: dejafu-0.5.1.3

13.53.1 Miscellaneous

- The version bounds on concurrency are 1.1.*.

13.54 0.5.1.2 (2017-03-04)

- Git: tag dejafu-0.5.1.2
- Hackage: dejafu-0.5.1.2

Note: this version was misnumbered! It should have caused a minor version bump!

13.54.1 Added

- `MonadRef` and `MonadAtomicRef` instances for `Test.DejaFu.Conc.Conc` using `CRef`.

13.54.2 Fixed

- A long-standing bug where if the main thread is killed with a `throwTo`, the throwing neither appears in the trace nor correctly terminates the execution.

13.54.3 Miscellaneous

- The upper bound on concurrency is <1.1.1.

13.55 0.5.1.1 (2017-02-25)

- Git: tag dejafu-0.5.1.1
- Hackage: dejafu-0.5.1.1

13.55.1 Fixed

- Fix using incorrect correct scheduler state after a *subconcurrency* action.
- Fix infinite loop in SCT of subconcurrency.

13.56 0.5.1.0 (2017-02-25)

- Git: tag dejafu-0.5.1.0
- Hackage: dejafu-0.5.1.0

13.56.1 Added

- `NFData` instances for:
  - `Test.DejaFu.Result`
  - `Test.DejaFu.Common.ThreadId`
  - `Test.DejaFu.Common.CRefId`
  - `Test.DejaFu.Common.MVarId`
  - `Test.DejaFu.Common.TVarId`
  - `Test.DejaFu.Common.IdSource`
  - `Test.DejaFu.Common.ThreadAction`
  - `Test.DejaFu.Common.Lookahead`
  - `Test.DejaFu.Common.ActionType`
  - `Test.DejaFu.Common.TAction`
  - `Test.DejaFu.Common.Decision`
  - `Test.DejaFu.Common.Failure`
  - `Test.DejaFu.Common.MemType`

  - `Test.DejaFu.SCT.Bounds`
  - `Test.DejaFu.SCT.PreemptionBound`
  - `Test.DejaFu.SCT.FairBound`
  - `Test.DejaFu.SCT.LengthBound`
  - `Test.DejaFu.SCT.Way`
  - `Test.DejaFu.STM.Result`
- `Eq`, `Ord`, and `Show` instances for `Test.DejaFu.Common.IdSource`.
- Strict variants of `Test.DejaFu.SCT.runSCT` and `resultsSet`: `runSCT'` and `resultsSet'`.

13.57 0.5.0.2 (2017-02-22)

- Git: tag dejafu-0.5.0.2
- Hackage: dejafu-0.5.0.2

Note: this version was misnumbered! It should have caused a major version bump!

13.57.1 Added

- `StopSubconcurrency` constructor for `Test.DejaFu.Common.ThreadAction`.

13.57.2 Changed

- A `Test.DejaFu.Common.StopConcurrency` action appears in the execution trace immediately after the end of a `Test.DejaFu.Conc.subconcurrency` action.

13.57.3 Fixed

- A `Test.DejaFu.Conc.subconcurrency` action inherits the number of capabilities from the outer computation.

13.57.4 Miscellaneous

- `Test.DejaFu.SCT` compiles with `MonoLocalBinds` enabled (implied by `GADTs` and `TypeFamilies`), which may be relevant to hackers.

13.58 0.5.0.1 (2017-02-21)

- Git: tag dejafu-0.5.0.1
- Hackage: ps!**

13.58.1 Fixed

- `readMVar` is considered a "release action" for the purposes of fair-bounding.

13.59 0.5.0.0 - The Way Release (2017-02-21)

- Git: tag dejafu-0.5.0.0
- Hackage: dejafu-0.5.0.0

13.59.1 Added

- `Eq` instances for `Test.DejaFu.Common.ThreadAction` and `Lookahead`.
- Thread action and lookahead values for `tryReadMVar`:
  - `Test.DejaFu.Common.TryReadMVar`
  - `Test.DejaFu.Common.WillTryReadMVar`
- The testing-only `Test.DejaFu.Conc.subconcurrency` function.
- SCT through weighted random scheduling: `Test.DejaFu.SCT.sctRandom`.
- The `Test.DejaFu.SCT.Way` type, used by the new functions `runSCT` and `resultsSet`.

13.59.2 Changed

- All the functions which took a `Test.DejaFu.SCT.Bounds` now take a `Way` instead.

13.59.3 Fixed

- Some previously-missed `CRef` action dependencies are no longer missed.

13.59.4 Miscellaneous

- The version bounds on concurrency are 1.1.0.*.
- A bunch of things were called "Var" or "Ref", these are now consistently "MVar" and "CRef".
- Significant performance improvements in both time and space.
- The dpor package has been merged back into this, as it turned out not to be very generally useful.

13.60 0.4.0.0 - The Packaging Release (2016-09-10)

- Git: tag dejafu-0.4.0.0
- Hackage: dejafu-0.4.0.0

13.60.1 Added

- The `Test.DejaFu.runTestM` and `runTestM'` functions.
- The `Test.DejaFu.Conc.runConcurrent` function.
- The `Test.DejaFu.STM.runTransaction` function.
- The `Test.DejaFu.Common` module.

13.60.2 Changed

- The `Control.*` modules have all been split out into a separate concurrency package.
- The `Test.DejaFu.Deterministic` module has been renamed to `Test.DejaFu.Conc`.
- Many definitions from other modules have been moved to the `Test.DejaFu.Common` module.
- The `Test.DejaFu.autocheck'` function takes the schedule bounds as a parameter.
- The `Test.DejaFu.Conc.Conc` type no longer has the STM type as a parameter.
- The `ST` specific functions in `Test.DejaFu.SCT` are polymorphic in the monad.
- The termination of the main thread in execution traces appears as a single `Stop`, rather than the previous `Lift, Stop`.
- Execution traces printed by the helpful functions in `Test.DejaFu` include a key of thread names.

13.60.3 Removed

- The `Test.DejaFu.runTestIO` and `runTestIO'` functions: use `runTestM` and `runTestM'` instead.
- The `Test.DejaFu.Conc.runConcST` and `runConcIO` functions: use `runConcurrent` instead.
- The `Test.DejaFu.STM.runTransactionST` and `runTransactionIO` functions: use `runTransaction` instead.
- The `IO` specific functions in `Test.DejaFu.SCT`.

13.61 0.3.2.1 (2016-07-21)

- Git: tag dejafu-0.3.2.1
- Hackage: dejafu-0.3.2.1

13.61.1 Fixed

- (issue #55) Fix incorrect detection of deadlocks with some nested STM transactions.

13.62 0.3.2.0 (2016-06-06)

- Git: tag dejafu-0.3.2.0
- Hackage: dejafu-0.3.2.0

13.62.1 Fixed

- (issue #40) Fix missing executions with daemon threads with uninteresting first actions. This is significantly faster with dpor-0.2.0.0.

13.62.2 Performance

- When using dpor-0.2.0.0, greatly improve dependency inference of exceptions during systematic testing.
- Improve dependency inference of STM transactions during systematic testing.

13.63 0.3.1.1 (2016-05-26)

- Git: tag dejafu-0.3.1.1
- Hackage: dejafu-0.3.1.1

13.63.1 Miscellaneous

- Now supports GHC 8.

13.64 0.3.1.0 (2016-05-02)

- Git: tag dejafu-0.3.1.0
- Hackage: dejafu-0.3.1.0

13.64.1 Fixed

- Fix inaccurate counting of pre-emptions in an execution trace when relaxed memory commit actions are present.

13.65 0.3.0.0 (2016-04-03)

- Git: tag dejafu-0.3.0.0
- Hackage: dejafu-0.3.0.0

The minimum supported version of GHC is now 7.10.

I didn't write proper release notes, and this is so far back I don't really care to dig through the logs.

13.66 0.2.0.0 (2015-12-01)

- Git: tag 0.2.0.0
- Hackage: dejafu-0.2.0.0

I didn't write proper release notes, and this is so far back I don't really care to dig through the logs.

13.67 0.1.0.0 - The Initial Release (2015-08-27)

- Git: tag 0.1.0.0
- Hackage: dejafu-0.1.0.0

13.67.1 Added

- Everything.

CHAPTER 14

Release Notes

This project is versioned according to the PVP, the *de facto* standard Haskell versioning scheme.

14.1 2.0.0.1 (2019-03-24)

- Git: tag hunit-dejafu-2.0.0.1
- Hackage: hunit-dejafu-2.0.0.1

14.1.1 Miscellaneous

- The upper bound on dejafu is <2.2

14.2 2.0.0.0 (2019-02-12)

- Git: tag hunit-dejafu-2.0.0.0
- Hackage: hunit-dejafu-2.0.0.0

14.2.1 Added

- **Re-exports for the `Program` types and their constructors:**
  - `Test.HUnit.DejaFu.Program`
  - `Test.HUnit.DejaFu.Basic`
  - `Test.HUnit.DejaFu.ConcT`
  - `Test.HUnit.DejaFu.ConcIO`
  - `Test.HUnit.DejaFu.WithSetup`

  - `Test.HUnit.DejaFu.WithSetupAndTeardown`
  - `Test.HUnit.DejaFu.withSetup`
  - `Test.HUnit.DejaFu.withTeardown`
  - `Test.HUnit.DejaFu.withSetupAndTeardown`
- **Re-exports for the `Invariant` type and its functions:**
  - `Test.HUnit.DejaFu.Invariant`
  - `Test.HUnit.DejaFu.registerInvariant`
  - `Test.HUnit.DejaFu.inspectIORef`
  - `Test.HUnit.DejaFu.inspectMVar`
  - `Test.HUnit.DejaFu.inspectTVar`

14.2.2 Changes

- **Functions which took a `ConcIO` now take a `Program pty IO`:**
  - `Test.HUnit.DejaFu.testAuto`
  - `Test.HUnit.DejaFu.testAutoWay`
  - `Test.HUnit.DejaFu.testAutoWithSettings`
  - `Test.HUnit.DejaFu.testDejafu`
  - `Test.HUnit.DejaFu.testDejafuWay`
  - `Test.HUnit.DejaFu.testDejafuWithSettings`
  - `Test.HUnit.DejaFu.testDejafus`
  - `Test.HUnit.DejaFu.testDejafusWay`
  - `Test.HUnit.DejaFu.testDejafusWithSettings`

14.2.3 Removed

- **The deprecated functions:**
  - `Test.HUnit.DejaFu.testDejafuDiscard`
  - `Test.HUnit.DejaFu.testDejafusDiscard`

14.2.4 Miscellaneous

- The lower bound on dejafu is >=2.0.

14.3 1.2.1.0 (2019-01-20)

- Git: tag hunit-dejafu-1.2.1.0
- Hackage: hunit-dejafu-1.2.1.0

14.3.1 Added

- Re-export of the `Condition` type from dejafu. If using dejafu < 1.12, this is an alias for `Failure`.

14.3.2 Miscellaneous

- The upper bound on dejafu is <1.13

14.4 1.2.0.6 (2018-07-01)

- Git: tag hunit-dejafu-1.2.0.6
- Hackage: hunit-dejafu-1.2.0.6

14.4.1 Miscellaneous

- The upper bound on dejafu is <1.12.

14.5 1.2.0.5 (2018-06-17)

- Git: tag hunit-dejafu-1.2.0.5
- Hackage: hunit-dejafu-1.2.0.5

14.5.1 Miscellaneous

- The upper bound on dejafu is <1.11.

14.6 1.2.0.4 (2018-06-10)

- Git: tag hunit-dejafu-1.2.0.4
- Hackage: hunit-dejafu-1.2.0.4

14.6.1 Miscellaneous

- The upper bound on dejafu is <1.10.

14.7 1.2.0.3 (2018-06-03)

- Git: tag hunit-dejafu-1.2.0.3
- Hackage: hunit-dejafu-1.2.0.3

14.7.1 Miscellaneous

- The upper bound on dejafu is <1.9.

14.8 1.2.0.2 (2018-06-03)

- Git: tag hunit-dejafu-1.2.0.2
- Hackage: hunit-dejafu-1.2.0.2

14.8.1 Miscellaneous

- The upper bound on dejafu is <1.8.

14.9 1.2.0.1 (2018-05-11)

- Git: tag hunit-dejafu-1.2.0.1
- Hackage: hunit-dejafu-1.2.0.1

14.9.1 Miscellaneous

- The upper bound on dejafu is <1.7.

14.10 1.2.0.0 - No More 7.10 (2018-03-28)

- Git: tag hunit-dejafu-1.2.0.0
- Hackage: hunit-dejafu-1.2.0.0

14.10.1 Miscellaneous

- GHC 7.10 support is dropped. Dependency lower bounds are:
  - base: 4.9
  - dejafu: 1.5
  - HUnit: 1.3.1
- The upper bound on dejafu is 1.6.

14.11 1.1.0.3 (2018-03-17)

- Git: tag hunit-dejafu-1.1.0.3
- Hackage: hunit-dejafu-1.1.0.3

14.11.1 Miscellaneous

- (pull request #251) The upper bound on dejafu is <1.5.

14.12 1.1.0.2 (2018-03-11)

- Git: tag hunit-dejafu-1.1.0.2
- Hackage: hunit-dejafu-1.1.0.2

14.12.1 Miscellaneous

- (pull request #245) The upper bound on exceptions is <0.11.

14.13 1.1.0.1 (2018-03-06)

- Git: tag hunit-dejafu-1.1.0.1
- Hackage: hunit-dejafu-1.1.0.1

14.13.1 Miscellaneous

- The upper bound on dejafu is <1.4.

14.14 1.1.0.0 - The Settings Release (2018-03-06)

- Git: tag hunit-dejafu-1.1.0.0
- Hackage: hunit-dejafu-1.1.0.0

14.14.1 Added

- (pull request #238) Settings-based test functions:
  - `Test.HUnit.DejaFu.testAutoWithSettings`
  - `Test.HUnit.DejaFu.testDejafuWithSettings`
  - `Test.HUnit.DejaFu.testDejafusWithSettings`
- (pull request #238) Re-export of `Test.DejaFu.Settings`.

14.14.2 Deprecated

- (pull request #238) `Test.HUnit.DejaFu.testDejafuDiscard` and `testDejafusDiscard`.

14.14.3 Removed

- (pull request #238) The re-export of `Test.DejaFu.Defaults.defaultDiscarder`.

14.14.4 Miscellaneous

- The version bounds on dejafu are >=1.2 && <1.3.

14.15 1.0.1.2 (2018-02-26)

- Git: tag hunit-dejafu-1.0.1.2
- Hackage: hunit-dejafu-1.0.1.2

14.15.1 Miscellaneous

- The upper bound on exceptions is <0.10.

14.16 1.0.1.1 (2018-02-22)

- Git: tag hunit-dejafu-1.0.1.1
- Hackage: hunit-dejafu-1.0.1.1

14.16.1 Miscellaneous

- The upper bound on dejafu is <1.2.

14.17 1.0.1.0 (2018-02-13)

- Git: tag hunit-dejafu-1.0.1.0
- Hackage: hunit-dejafu-1.0.1.0

14.17.1 Added

- (pull request #200) `Test.HUnit.DejaFu.testDejafusDiscard` function.

14.18 1.0.0.0 - The API Friendliness Release (2017-12-23)

- Git: tag hunit-dejafu-1.0.0.0
- Hackage: hunit-dejafu-1.0.0.0

14.18.1 Added

- (issue #124) Re-exports of `Test.DejaFu.Predicate` and `ProPredicate`.

14.18.2 Changed

- All testing functions require `MonadConc`, `MonadRef`, and `MonadIO` constraints. Testing with `ST` is no longer possible.
- (issue #123) All testing functions take the action to run as the final parameter.
- (issue #124) All testing functions have been generalised to take a `Test.DejaFu.ProPredicate` instead of a `Predicate`.

14.18.3 Removed

- The `Test.DejaFu.Conc.ConcST` specific functions.
- The orphan `Testable` and `Assertable` instances for `Test.DejaFu.Conc.ConcST t ()`.

14.18.4 Miscellaneous

- The version bounds on dejafu are >=1.0 && <1.1.

14.19 0.7.1.1 (2017-11-30)

- Git: tag hunit-dejafu-0.7.1.1
- Hackage: hunit-dejafu-0.7.1.1

14.19.1 Fixed

- A missing Haddock `@since` comments.

14.20 0.7.1.0 (2017-11-30)

- Git: tag hunit-dejafu-0.7.1.0
- Hackage: hunit-dejafu-0.7.1.0

14.20.1 Added

- `Test.HUnit.DejaFu.testPropertyFor` function.

14.21 0.7.0.2 (2017-10-11)

- Git: tag hunit-dejafu-0.7.0.2
- Hackage: hunit-dejafu-0.7.0.2

14.21.1 Miscellaneous

- The upper bound on dejafu is <0.10.

14.22 0.7.0.1 (2017-09-26)

- Git: tag hunit-dejafu-0.7.0.1
- Hackage: hunit-dejafu-0.7.0.1

14.22.1 Miscellaneous

- The upper bound on dejafu is <0.9.

14.23 0.7.0.0 - The Discard Release (2017-08-10)

- Git: tag hunit-dejafu-0.7.0.0
- Hackage: hunit-dejafu-0.7.0.0

14.23.1 Added

- Re-export for `Test.DejaFu.SCT.Discard` and `Test.DejaFu.Defaults.defaultDiscarder`.
- `Test.HUnit.DejaFu.testDejafuDiscard` and `testDejafuDiscardIO` functions.

14.23.2 Miscellaneous

- The lower bound on dejafu is >=0.7.1.

14.24 0.6.0.0 - The Refinement Release (2017-06-07)

- Git: tag hunit-dejafu-0.6.0.0
- Hackage: hunit-dejafu-0.6.0.0

14.24.1 Added

- `Test.HUnit.DejaFu.testProperty` function
- Re-exports for `Test.DejaFu.SCT.systematically`, `randomly`, `uniformly`, and `swarmy`.
- Re-exports for `Test.DejaFu.Defaults.defaultWay`, `defaultMemType`, and `defaultBounds`.

14.24.2 Removed

- Re-exports of the `Test.DejaFu.SCT.Way` constructors: `Systematically` and `Randomly`.

14.24.3 Miscellaneous

- The version bounds on dejafu are >=0.7 && <0.8.

14.25 0.5.0.0 - The Way Release (2017-04-08)

- Git: tag hunit-dejafu-0.5.0.0
- Hackage: hunit-dejafu-0.5.0.0

14.25.1 Changed

- Due to changes in dejafu, the `Way` type no longer takes a parameter; it is now a GADT.

14.25.2 Miscellaneous

- Every definition, class, and instance now has a Haddock `@since` annotation.
- The version bounds on dejafu are >=0.6 && <0.7.
- Remove an unnecessary dependency on random.

14.26 0.4.0.1 (2017-03-20)

- Git: tag hunit-dejafu-0.4.0.1
- Hackage: hunit-dejafu-0.4.0.1

14.26.1 Miscellaneous

- The upper bound on HUnit is <1.7.

14.27 0.4.0.0 (2017-02-21)

- Git: tag hunit-dejafu-0.4.0.0
- Hackage: hunit-dejafu-0.4.0.0

14.27.1 Added

- Re-export of `Test.DejaFu.SCT.Way`.

14.27.2 Changed

- All the functions which took a `Test.DejaFu.SCT.Bounds` now take a `Way`.

14.27.3 Miscellaneous

- The version bounds on dejafu are >=0.5 && <0.6.
- Dependency on random with bounds >=1.0 && <1.2.

14.28 0.3.0.3 (2016-10-22)

- Git: tag hunit-dejafu-0.3.0.3
- Hackage: hunit-dejafu-0.3.0.3

14.28.1 Miscellaneous

- The upper bound on HUnit is <1.6.

14.29 0.3.0.2 (2016-09-10)

- Git: tag hunit-dejafu-0.3.0.2
- Hackage: hunit-dejafu-0.3.0.2

14.29.1 Miscellaneous

- The upper bound on dejafu is <0.5.

14.30 0.3.0.1 (2016-05-26)

- Git: tag hunit-dejafu-0.3.0.1
- Hackage: hunit-dejafu-0.3.0.1

14.30.1 Miscellaneous

- The lower bound on base is >=4.8.
- The upper bound on dejafu is <0.4.

14.31 0.3.0.0 (2016-04-28)

- Git: tag hunit-dejafu-0.3.0.0
- Hackage: hunit-dejafu-0.3.0.0

14.31.1 Added

- Orphan `Assertable` and `Testable` instances for `Test.DejaFu.Conc.ConcST t ()` and `ConcIO ()`.
- Re-export `Test.DejaFu.SCT.Bounds`.

14.31.2 Miscellaneous

- The version bounds on dejafu are >=0.2

14.32 0.2.1.0 (2016-04-03)

- Git: tag hunit-dejafu-0.2.1.0

Note: this was never pushed to Hackage, whoops!

14.32.1 Miscellaneous

- The version bounds on dejafu are 0.3.*.

14.33 0.2.0.0 - The Initial Release (2015-12-01)

- Git: tag 0.2.0.0
- Hackage: hunit-dejafu-0.2.0.0

14.33.1 Added

- Everything.

CHAPTER 15

Release Notes

This project is versioned according to the PVP, the *de facto* standard Haskell versioning scheme.

15.1 2.0.0.1 (2019-03-24)

- Git: tag tasty-dejafu-2.0.0.1
- Hackage: tasty-dejafu-2.0.0.1

15.1.1 Miscellaneous

- The upper bound on dejafu is <2.2

15.2 2.0.0.0 (2019-02-12)

- Git: tag tasty-dejafu-2.0.0.0
- Hackage: tasty-dejafu-2.0.0.0

15.2.1 Added

- **Re-exports for the `Program` types and their constructors:**
  - `Test.Tasty.DejaFu.Program`
  - `Test.Tasty.DejaFu.Basic`
  - `Test.Tasty.DejaFu.ConcT`
  - `Test.Tasty.DejaFu.ConcIO`
  - `Test.Tasty.DejaFu.WithSetup`

  - `Test.Tasty.DejaFu.WithSetupAndTeardown`
  - `Test.Tasty.DejaFu.withSetup`
  - `Test.Tasty.DejaFu.withTeardown`
  - `Test.Tasty.DejaFu.withSetupAndTeardown`
- **Re-exports for the `Invariant` type and its functions:**
  - `Test.Tasty.DejaFu.Invariant`
  - `Test.Tasty.DejaFu.registerInvariant`
  - `Test.Tasty.DejaFu.inspectIORef`
  - `Test.Tasty.DejaFu.inspectMVar`
  - `Test.Tasty.DejaFu.inspectTVar`

15.2.2 Changes

- **Functions which took a `ConcIO` now take a `Program pty IO`:**
  - `Test.Tasty.DejaFu.testAuto`
  - `Test.Tasty.DejaFu.testAutoWay`
  - `Test.Tasty.DejaFu.testAutoWithSettings`
  - `Test.Tasty.DejaFu.testDejafu`
  - `Test.Tasty.DejaFu.testDejafuWay`
  - `Test.Tasty.DejaFu.testDejafuWithSettings`
  - `Test.Tasty.DejaFu.testDejafus`
  - `Test.Tasty.DejaFu.testDejafusWay`
  - `Test.Tasty.DejaFu.testDejafusWithSettings`

15.2.3 Removed

- **The deprecated functions:**
  - `Test.Tasty.DejaFu.testDejafuDiscard`
  - `Test.Tasty.DejaFu.testDejafusDiscard`

15.2.4 Miscellaneous

- The lower bound on dejafu is >=2.0.

15.3 1.2.1.0 (2019-01-20)

- Git: tag tasty-dejafu-1.2.1.0
- Hackage: tasty-dejafu-1.2.1.0

15.3.1 Added

- Re-export of the `Condition` type from dejafu. If using dejafu < 1.12, this is an alias for `Failure`.

15.3.2 Miscellaneous

- The upper bound on dejafu is <1.13

15.4 1.2.0.8 (2018-12-02)

- Git: tag tasty-dejafu-1.2.0.8
- Hackage: tasty-dejafu-1.2.0.8

15.4.1 Miscellaneous

- The upper bound on tasty is <1.3.

15.5 1.2.0.7 (2018-07-01)

- Git: tag tasty-dejafu-1.2.0.7
- Hackage: tasty-dejafu-1.2.0.7

15.5.1 Miscellaneous

- The upper bound on dejafu is <1.12.

15.6 1.2.0.6 (2018-06-17)

- Git: tag tasty-dejafu-1.2.0.6
- Hackage: tasty-dejafu-1.2.0.6

15.6.1 Miscellaneous

- The upper bound on dejafu is <1.11.

15.7 1.2.0.5 (2018-06-10)

- Git: tag tasty-dejafu-1.2.0.5
- Hackage: tasty-dejafu-1.2.0.5

15.7.1 Miscellaneous

- The upper bound on dejafu is <1.10.

15.8 1.2.0.4 (2018-06-03)

- Git: tag tasty-dejafu-1.2.0.4
- Hackage: tasty-dejafu-1.2.0.4

15.8.1 Miscellaneous

- The upper bound on dejafu is <1.9.

15.9 1.2.0.3 (2018-06-03)

- Git: tag tasty-dejafu-1.2.0.3
- Hackage: tasty-dejafu-1.2.0.3

15.9.1 Miscellaneous

- The upper bound on dejafu is <1.8.

15.10 1.2.0.2 (2018-05-12)

- Git: tag tasty-dejafu-1.2.0.2
- Hackage: tasty-dejafu-1.2.0.2

15.10.1 Miscellaneous

- The upper bound on tasty is <1.2.

15.11 1.2.0.1 (2018-05-11)

- Git: tag tasty-dejafu-1.2.0.1
- Hackage: tasty-dejafu-1.2.0.1

15.11.1 Miscellaneous

- The upper bound on dejafu is <1.7.

15.12 1.2.0.0 - No More 7.10 (2018-03-28)

- Git: tag tasty-dejafu-1.2.0.0
- Hackage: tasty-dejafu-1.2.0.0

15.12.1 Miscellaneous

- GHC 7.10 support is dropped. Dependency lower bounds are:
  - base: 4.9
  - dejafu: 1.5
- The upper bound on dejafu is 1.6.

15.13 1.1.0.2 (2018-03-17)

- Git: tag tasty-dejafu-1.1.0.2
- Hackage: tasty-dejafu-1.1.0.2

15.13.1 Miscellaneous

- The upper bound on dejafu is <1.5.

15.14 1.1.0.1 (2018-03-06)

- Git: tag tasty-dejafu-1.1.0.1
- Hackage: tasty-dejafu-1.1.0.1

15.14.1 Miscellaneous

- The upper bound on dejafu is <1.4.

15.15 1.1.0.0 - The Settings Release (2018-03-06)

- Git: tag tasty-dejafu-1.1.0.0
- Hackage: tasty-dejafu-1.1.0.0

15.15.1 Added

- (pull request #238) Settings-based test functions:
  - `Test.Tasty.DejaFu.testAutoWithSettings`
  - `Test.Tasty.DejaFu.testDejafuWithSettings`

  - `Test.Tasty.DejaFu.testDejafusWithSettings`

- (pull request #238) Re-export of `Test.DejaFu.Settings`.

15.15.2 Deprecated

- (pull request #238) `Test.Tasty.DejaFu.testDejafuDiscard` and `testDejafusDiscard`.

15.15.3 Removed

- (pull request #238) The re-export of `Test.DejaFu.Defaults.defaultDiscarder`.

15.15.4 Miscellaneous

- The version bounds on dejafu are >=1.2 && <1.3.

15.16 1.0.1.1 (2018-02-22)

- Git: tag tasty-dejafu-1.0.1.1
- Hackage: tasty-dejafu-1.0.1.1

15.16.1 Miscellaneous

- The upper bound on dejafu is <1.2.

15.17 1.0.1.0 (2018-02-13)

- Git: tag tasty-dejafu-1.0.1.0
- Hackage: tasty-dejafu-1.0.1.0

15.17.1 Added

- (pull request #195) `Test.Tasty.DejaFu.testDejafusDiscard` function.

15.18 1.0.0.1 (2018-01-09)

- Git: tag tasty-dejafu-1.0.0.1
- Hackage: tasty-dejafu-1.0.0.1

15.18.1 Miscellaneous

- The upper bound on tasty is <1.1.

15.19 1.0.0.0 - The API Friendliness Release (2017-12-23)

- Git: tag tasty-dejafu-1.0.0.0
- Hackage: tasty-dejafu-1.0.0.0

15.19.1 Added

- (issue #124) Re-exports of `Test.DejaFu.Predicate` and `ProPredicate`.

15.19.2 Changed

- All testing functions require `MonadConc`, `MonadRef`, and `MonadIO` constraints. Testing with `ST` is no longer possible.
- (issue #123) All testing functions take the action to run as the final parameter.
- (issue #124) All testing functions have been generalised to take a `Test.DejaFu.ProPredicate` instead of a `Predicate`.

15.19.3 Removed

- The `Test.DejaFu.Conc.ConcST` specific functions.
- The orphan `IsTest` instance for `Test.DejaFu.Conc.ConcST t (Maybe String)`.

15.19.4 Miscellaneous

- The version bounds on dejafu are >=1.0 && <1.1.

15.20 0.7.1.1 (2017-11-30)

- Git: tag tasty-dejafu-0.7.1.1
- Hackage: tasty-dejafu-0.7.1.1

15.20.1 Fixed

- A missing Haddock `@since` comments.

15.21 0.7.1.0 (2017-11-30)

- Git: tag tasty-dejafu-0.7.1.0
- Hackage: tasty-dejafu-0.7.1.0

15.21.1 Added

- `Test.Tasty.DejaFu.testPropertyFor` function.

15.22 0.7.0.3 (2017-11-02)

- Git: tag tasty-dejafu-0.7.0.3
- Hackage: tasty-dejafu-0.7.0.3

15.22.1 Miscellaneous

- The upper bound on tasty is <0.13.

15.23 0.7.0.2 (2017-10-11)

- Git: tag tasty-dejafu-0.7.0.2
- Hackage: tasty-dejafu-0.7.0.2

15.23.1 Miscellaneous

- The upper bound on dejafu is <0.10.

15.24 0.7.0.1 (2017-09-26)

- Git: tag tasty-dejafu-0.7.0.1
- Hackage: tasty-dejafu-0.7.0.1

15.24.1 Miscellaneous

- The upper bound on dejafu is <0.9.

15.25 0.7.0.0 - The Discard Release (2017-08-10)

- Git: tag tasty-dejafu-0.7.0.0
- Hackage: tasty-dejafu-0.6.0.0

15.25.1 Added

- Re-export for `Test.DejaFu.SCT.Discard` and `Test.DejaFu.Defaults.defaultDiscarder`.
- `Test.Tasty.DejaFu.testDejafuDiscard` and `testDejafuDiscardIO` functions.

15.25.2 Miscellaneous

- The lower bound on dejafu is >=0.7.1.

15.26 0.6.0.0 - The Refinement Release (2017-04-08)

- Git: tag tasty-dejafu-0.6.0.0
- Hackage: tasty-dejafu-0.6.0.0

15.26.1 Added

- `Test.Tasty.DejaFu.testProperty` function
- Re-exports for `Test.DejaFu.SCT.systematically`, `randomly`, `uniformly`, and `swarmy`.
- Re-exports for `Test.DejaFu.Defaults.defaultWay`, `defaultMemType`, and `defaultBounds`.

15.26.2 Removed

- Re-exports of the `Test.DejaFu.SCT.Way` constructors: `Systematically` and `Randomly`.

15.26.3 Miscellaneous

- The version bounds on dejafu are >=0.7 && <0.8.

15.27 0.5.0.0 - The Way Release (2017-04-08)

- Git: tag tasty-dejafu-0.5.0.0
- Hackage: tasty-dejafu-0.5.0.0

15.27.1 Changed

- Due to changes in dejafu, the `Way` type no longer takes a parameter; it is now a GADT.

15.27.2 Miscellaneous

- Every definition, class, and instance now has a Haddock `@since` annotation.
- The version bounds on dejafu are >=0.6 && <0.7.

15.28 0.4.0.0 (2017-02-21)

- Git: tag tasty-dejafu-0.4.0.0
- Hackage: tasty-dejafu-0.4.0.0

15.28.1 Added

- Re-export of `Test.DejaFu.SCT.Way`.
- Orphan `IsOption` instance for `Test.DejaFu.SCT.Way`. Command-line parameters are:
  - "systematically": systematic testing with the default bounds
  - "randomly": 100 executions with a fixed random seed

15.28.2 Changed

- All the functions which took a `Test.DejaFu.SCT.Bounds` now take a `Way`.

15.28.3 Miscellaneous

- The version bounds on dejafu are >=0.5 && <0.6.
- Dependency on random with bounds >=1.0 && <1.2.

15.29 0.3.0.2 (2016-09-10)

- Git: tag tasty-dejafu-0.3.0.2
- Hackage: tasty-dejafu-0.3.0.2

15.29.1 Miscellaneous

- The upper bound on dejafu is <0.5.

15.30 0.3.0.1 (2016-05-26)

- Git: tag tasty-dejafu-0.3.0.1
- Hackage: tasty-dejafu-0.3.0.1

15.30.1 Miscellaneous

- The lower bound on base is >=4.8.
- The upper bound on dejafu is <0.4.

15.31 0.3.0.0 (2016-04-28)

- Git: tag tasty-dejafu-0.3.0.0
- Hackage: tasty-dejafu-0.3.0.0

15.31.1 Added

- Orphan `IsTest` instances for `Test.DejaFu.Conc.ConcST t (Maybe String)` and `ConcIO (Maybe String)`.
- Orphan `IsOption` instances for `Test.DejaFu.SCT.Bounds` and `MemType`. Command-line parameters are:
  - "sc": sequential consistency
  - "tso": total store order
  - "pso": partial store order
- Re-export `Test.DejaFu.SCT.Bounds`.

15.31.2 Miscellaneous

- The version bounds on dejafu are >=0.2

15.32 0.1.1.0 (2016-04-03)

- Git: tag tasty-dejafu-0.1.1.0

Note: this was misnumbered (it should have been 0.2.1.0) *and* was never pushed to Hackage, whoops!

15.32.1 Miscellaneous

- The version bounds on dejafu are 0.3.*.

15.33 0.2.0.0 - The Initial Release (2015-12-01)

- Git: tag 0.2.0.0
- Hackage: tasty-dejafu-0.2.0.0

15.33.1 Added

- Everything.