What Does Monad Mean?

Tony Morris

Abstract

This presentation is intended for the November 2009 meeting of the Brisbane Functional Programming Group http://www.meetup.com/Brisbane-Functional-Programming-Group-BFG/.

The *monad* concept will be presented to an audience with the assumption of knowledge of the generics concept of either the Java or C# programming languages or some other form of parametric polymorphism. The concept will be presented in a way with the objective of supplying enough understanding to apply the practical implications and will not address the underlying mathematics or category theory.

The monad concept has attracted much mysticism, misunderstanding and mythology, which will also be addressed.

Questions and comments should be forwarded to Tony Morris [mailto:code@tmorris.net] Appendix B, *Tony Morris - PGP Key*.

Table of Contents

Introduction

Moggi first described monads for use in structuring programs [MoggiMonads1988]. Philip Wadler and Simon Peyton-Jones have worked further with the concept on introducing them to purely functional programming languages.

It is often said that monads are specific to functional programming. This is not true. Indeed, the term “functional programming” is itself quite a misnomer. It's simply practical programming and the thesis is composition and abstraction (see [WhyFP]). Monads are used in every single programming language. The only distinction is whether or not the user is aware of the fact.

This presentation will set aside the myths and present you with a concrete understanding of a rather simple concept that is easy to understand, but with far reaching implications. If the objectives of this presentation are met, you will be equipped to take advantage of this concept in your every-day programming and explore those implications further.

Mythology

Debunking

Debunking popular myths is often fraught with controversy, fallacy and futility, so before we do that, let's have fun with a myth that is not so popular nor controversial.

> Athena punished Medusa by turning her hair tresses into snakes. Perseus then used Medusa's head as a weapon in battles.

Once upon a time, this story was popularly believed despite its obvious absurdity. In our modern times, other stories are widely believed, again, despite absurdity. A few of those follow.

Monads are a hack for working with I/O in the real world

Other forms

- Monads are for impractical programming languages that try to be practical.
- Monads are for controlling side-effects.
- Without monads in $language we cannot do I/O.

Demystify

Suppose a function that reverses a `List` and a class called `Banana` that your local fruit man has on his server.

```
<A> List<A> reverse(List<A> list)
class Banana {}
```

It is correct to say that we can reverse a list of bananas, but it is **incorrect** to say that the `reverse` function is in some way related to bananas. Reversing a list of bananas is merely a single instance of an infinite possibility. For precisely this reason, it is also incorrect to say that monads are in some way related to I/O.

This metaphor might appear vague or inaccurate, but it is actually very (very) precise as you will see.

Monads are only for functional or esoteric programming languages

Other forms

- I don't use monads in my Java (or .NET or Ruby or Groovy or Python or ...) enterprise.
- Monads are for others using other languages, not me and my language.

Demystify

- FACT: All programmers, knowingly or not, use monads at least 100 times per day. **I'll put a house on it.**
- Like many myths, it is highly likely that the claimant has either no or a very poor understanding of the monad concept and is spreading the myth simply by repetition.
- Monads are at least as common as semi-colons in language syntaxes such as Java.

Monads make my program impure

This myth is often related to or borne of those already mentioned. Interestingly, what the claimant is actually referring to (which is not monads, but controlling effects with types) makes programs pure, that would otherwise be impure.

Terminology

A quick note on terminology

Laws, type constructors, kinds and higher-order polymorphism.

Monad laws

What do we mean by the *Monad Laws*?

- You may have heard of the monad laws. There are three of them. They even have names![1]
- Lots of interfaces have laws that are not enforced by the compiler.
- You are all familiar with many of them.

[1] Associativity, Left Identity, Right Identity

For example

Example 1. An abbreviated List interface using Java

```
interface List<T> {
  T get(int index);
  int length();
  List<T> reverse();
}
```

- It can be said that any instance (list) must satisfy list.get(list.length() - 1) == list.reverse().get(0)
- There are other laws that are implicit in this abbreviated interface.
- All monad instances must satisfy three laws, but we won't go into them. Instead, just be aware of what we mean by the monad laws.

Type constructors and kinds

- In Java 1.5, List is not a type, because it requires a type variable [2].
- List<Long> is a type, but List is a *type constructor*.
- List has a *kind* which is denoted * -> * and reads “takes one type to reveal a type”.
- HashMap is a type constructor which is kinded (*,*) -> * since it takes two type variables to reveal a type.
- List<Long> has a *kind* which is denoted * (i.e. it is a type).

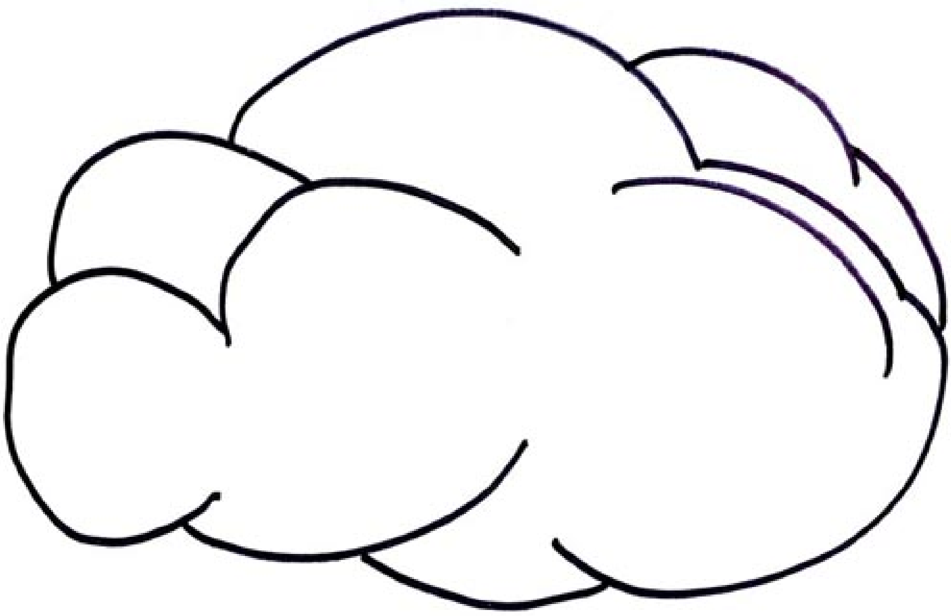
Higher-order Polymorphism

- Languages such as Java and C# have *first-order polymorphism* because they allow us to abstract on types. e.g. List<A> can have a reverse function that works on any element type (the A).
- More practical programming languages and type systems[3] allow us to abstract on *type constructors* as well.
- This feature is called higher-order (or higher-kinded) polymorphism.

[2] ignoring reverse compatibility where List is equivalent to List<?>
[3] Haskell, Scala, Clean (not F#)

So what is a Monad then?

It's a fluffy cloud



A monad is a fluffy cloud

Produced by a sonic boom

A monad is a cloud produced by the condensation of the sonic boom of a supersonic aircraft

Monad metaphors are stupid

Yeah OK, so monad metaphors are stupid and divertive.

Metaphors are training wheels that obscure learning opportunities and stall progress (object-oriented programming anyone?).

Let's get to the nuts of it.

First let's pretend that Java/C# are practical

Example 2. Pseudo-Java with an invented notation for higher-order polymorphism [4]

```
interface Transformer<X, Y> {
  Y transform(X x);
}

interface Monad<M> { // M :: * -> *
  <A> M<A> pure(A a);
  <A, B> M<B> bind(Transformer<A, M<B>> t, M<A> a);
}
```

Finished staring?

A monad is any instance of that interface[5]. That is **all it is**.

Example 3. The List monad

```
new Monad<List>() { // List :: * -> * (kind checks)
  public <A> List<A> pure(A a) {
    return List.single(a);
  }

  public <A, B> List<B> bind(Transformer<A, List<B>> t, List<A> a) {
    List<B> r = List.empty();
    for(e : a) r.addAll(t.transform(e));
    return r;
  }
};
```

Monads and more monads

In fact, there are lots of possible instances of the `Monad` interface. Even `Transformer` with one type variable applied is a monad.

[4] or C# with a slight syntax change
[5] and also satisfies the three monad laws

Example 4. The Transformer<X, _> monad

```
new Monad<Transformer<X, _>>() {
  public <A> Transformer<X, A> pure(A a) {
    return new Transformer<X, A>() {
      public A transform(X x) { return a; }
    };
  }

  public <A, B> Transformer<X, B> bind(
        Transformer<A, Transformer<X, B>> t, Transformer<X, A> a) {
    return new Transformer<X, B>() {
      public B transform(X x) { return
        t.transform(a.transform(x)).transform(x); }
    };
  }
};
```

More examples

Example 5. A value is a monad [6]

```
interface Value<T> {
  T value();
}

new Monad<Value>() {
  public <A> Value<A> pure(A a) {
    return new Value<A>() { public A value() { return a; } };
  }

  public <A, B> Value<B> bind(Transformer<A, Value<B>> t, Value<A> a) {
    return t.transform(a.value());
  }
};
```

Other examples include

- Nullable values
- Chaining exceptions
- Disjoint union
- Continuations
- Functions
- State
- Parsers
- Side-effects (!!!)

[6] more correctly called The Identity Monad

So what, why do I care?

Repetition is for solving non-problems using clumsy languages

We can write functions that abstract on the type constructor and needn't be repeated. Similar to a `sort` function that accepts any value that has declared itself the ability to compare, these functions can act on any type constructor that has declared itself a monad.

- ‹A› M‹List‹A›› sequence(List‹M‹A›› list)
- ‹A› M‹List‹A›› replicate(int n, M‹A› a)
- ‹A› M‹A› join(M‹M‹A›› a)
- *etc. etc.*

And it's not just monads

We can do this because we have *higher-order polymorphism*. We can also generalise applicative functors, comonads, arrows, covariant and contravariant functors, binary covariant functors, structure traversal and many other concepts that are well documented in the literature.

Monads get far more attention than they deserve!

C#, LINQ Query Comprehensions and Monads

C# 3.0

- Microsoft introduced LINQ query comprehensions into C# version 3.0.
- Many people are mistaken in believing LINQ is about enumerable data structures or SQL queries.
- These are just monad instances! LINQ query comprehensions are syntactic sugar for working with *any* monad.
- What we have called `bind`, C#/LINQ calls `SelectMany` and what we have called `Transformer`, C# calls `Func`. C# also introduces syntactic sugar for instantiating `Func`.

Syntax sugar

`x => method(x)` is syntax sugar for a `Func` instance that takes a variable (x) and runs `method` on it.

Example 6. LINQ Query Comprehension [7]

```
from i in a
from j in b
from k in c
select foo(i, j, k);
```

[7] `from`, `in`, and `select` are keywords.

Example 7. Without the syntax sugar

```
a.SelectMany(i =>
b.SelectMany(j =>
c.Select(k =>
foo(i, j, k))));
```

LINQ

In the previous examples, the query comprehension can work on *any monad*. Types declare their monad simply by implementing the `SelectMany` (and `Select`) method.

Notice that a general LINQ query comprehension cannot have a type since it would require higher-order polymorphism. Consequently, we cannot write functions to run on any monad.

C# is catching up to practical languages (unlike Java), but still has an awful long way to go!

Scala for-comprehensions and Monads

Scala

- Scala has had for-comprehensions from the beginning, but not always higher-kinded polymorphism[8].
- What C# calls `SelectMany` and we have called `bind`, Scala calls `flatMap`.
- Like LINQ, for-comprehensions provide syntactic sugar for calls to `flatMap` (and others) and also like LINQ are not type-safe[9].

For-comprehensions

Example 8. Scala for-comprehension [10]

```
for(i <- a;
    j <- b;
    k <- c)
yield foo(i, j, k)
```

Example 9. Without syntax sugar

```
a flatMap (i =>
b flatMap (j =>
c map (k =>
foo(i, j, k))))
```

Transforming side-effects using monads

Suppose a program in a typical language

[8] Scala By Example (DRAFT October 28, 2009) still incorrectly states that Scala's type system is too weak to express the generalisation.
[9] This is because of historical reasons, unlike C#/LINQ, which is due to type system limitations.
[10] `for` and `yield` are keywords.

```
T t = e1();
e2(t);
U u = e3(t);
V v = e4(t, u);
return e5(u, v);
```

We are going to change some of the syntax rules.

Change the syntax

- Remove type-annotations.
- Swap the left/right of an assignment.
- `void` methods will return a value that is ignored (let's call it _).

```
e1() = t;
e2(t) = _;
e3(t) = u;
e4(t, u) = v;
return e5(u, v);
```

More changes

- Replace semi-colons with =>.
- Replace `return` with `pure`.
- Replace = with `flatMap`.

Example 10. Zing!

```
e1() flatMap t =>
e2(t) flatMap _ =>
e3(t) flatMap u =>
e4(t, u) flatMap v =>
pure e5(u, v);
```

More fun

- Applicative functors [http://www.soi.city.ac.uk/~ross/papers/Applicative.pdf]
- Arrows [http://www.cs.chalmers.se/~rjmh/afp-arrows.pdf]
- Structure traversal (iteration) [http://www.comlab.ox.ac.uk/jeremy.gibbons/publications/iterator.pdf]
- Questions?

Special Thanks

- Dr. Andrew Wines, Orthopaedic Surgeon, Sydney.
- Dr. Lachlan Steffen, General Practitioner, Brisbane.
- Dr. Jennie Noakes, Radiologist, Sydney.

- The anonymous practitioners of medical science and research for your enormous sacrifices toward the betterment of humanity. Your contributions are appreciated.

Bibliography

[MoggiMonads1988] Eugenio Moggi . 1988 . *Computational lambda-calculus and monads* .

[WhyFP] John Hughes . 1984 . *Why Functional Programming Matters* .

A. Code Examples

Scala

Monad.scala

```
// Monad interface
trait Monad[M[_]] {
  def pure[A](a: A): M[A]
  def bind[A, B](a: M[A], f: A => M[B]): M[B]
}

object Monad {
  // Monad instances
  val ValueMonad = new Monad[Function0] {
    def pure[A](a: A) = () => a
    def bind[A, B](a: Function0[A], f: A => Function0[B]) =
      f(a.apply)
  }

  val ListMonad = new Monad[List] {
    def pure[A](a: A) = List(a)
    def bind[A, B](a: List[A], f: A => List[B]) =
      a flatMap f
  }

  val OptionMonad = new Monad[Option] {
    def pure[A](a: A) = Some(a)
    def bind[A, B](a: Option[A], f: A => Option[B]) =
      a flatMap f
  }

  trait PA[T[_, _], A] {
    type Apply[B] = T[A, B]
  }

  def Function1Monad[X] = new Monad[PA[Function1, X]#Apply] {
    def pure[A](a: A) = _ => a
    def bind[A, B](a: X => A, f: A => X => B) =
      x => f(a(x))(x)
  }

  // Monad functions
  def sequence[M[_], A](as: List[M[A]], m: Monad[M]) =
    as.foldRight[M[List[A]]](m pure Nil)((h, t) =>
      m.bind(h, (a: A) => m.bind(t, (as: List[A]) => m pure (a :: as))))
```

```
  def join[M[_], A](a: M[M[A]], m: Monad[M]) = m.bind(a, (z: M[A]) => z)

  def replicate[M[_], A](n: Int, a: M[A], m: Monad[M]) =
    sequence(List.fill(n)(a), m)

  // etc. etc. (Monad functions)
}

// Monad demo
object Main {
  import Monad._

  def main(args: Array[String]) {
    println(sequence(List(List(1, 2, 3), List(4, 5, 6)), ListMonad))
    println(sequence(List(Some(7), Some(8)), OptionMonad))
    println(join(List(List(1, 2, 3), List(4, 5, 6)), ListMonad))
    println(join(Some(Some(7)), OptionMonad))
    println(replicate(3, "abc".toList, ListMonad) map (_.mkString))
    println(replicate(4, Some(8), OptionMonad))
  }
}

// scala-2.8.0.r19410-b20091106023416

/*
List(List(1, 4), List(1, 5), List(1, 6), List(2, 4), List(2, 5),
  List(2, 6), List(3, 4), List(3, 5), List(3, 6))
Some(List(7, 8))
List(1, 2, 3, 4, 5, 6)
Some(7)
List(aaa, aab, aac, aba, abb, abc, aca, acb, acc, baa,
  bab, bac, bba, bbb, bbc, bca, bcb, bcc, caa, cab, cac,
  cba, cbb, cbc, cca, ccb, ccc)
Some(List(8, 8, 8, 8))
*/
```

C#

Monad.cs

```
using System;
using System.Collections.Generic;
using System.Linq;

static class MonadFunctions {
  public static IEnumerable<A> joinEnumerable<A>(IEnumerable<IEnumerable<A>> a)
    // Forced repetition.
    return a.SelectMany(z => z);
  }

  public static IQueryable<A> joinQueryable<A>(IQueryable<IQueryable<A>> a) {
    // Repeats for each type constructor.
    return a.SelectMany(z => z);
  }

  // join*
}
```

```
class M {
  public static void Main(string[] args) {
    var i = new[]{1, 2, 3};
    var j = new[]{4, 5, 6};
    Print(MonadFunctions.joinEnumerable(new[]{i as IEnumerable<int>, j}));
  }

  static void Print<A>(IEnumerable<A> a) {
    foreach(A i in a)
      Console.Write("{0} ", i);
    Console.WriteLine();
  }
}
```

Haskell

Monad.hs

```
-- Monad interface
class Monad' m where
  bind :: m a -> (a -> m b) -> m b
  pure :: a -> m a

newtype Value a = Value {
  val :: a
}

-- Monad instances
instance Monad' Value where
  bind (Value a) f = f a
  pure = Value

instance Monad' [] where -- aka List
  bind = (>>=)
  pure = return

instance Monad' Maybe where -- aka Option
  bind = (>>=)
  pure = return

instance Monad' ((->) t) where -- aka Function1
  bind a f = \x -> f (a x) x
  pure = const

-- Monad functions
sequence' :: (Monad' m) => [m a] -> m [a]
sequence' = foldr (\h t -> bind h (bind t . (pure .) . (:))) (pure [])

join' :: (Monad' m) => m (m a) -> m a
join' a = bind a id

replicate' :: (Monad m) => Int -> m a -> m [a]
replicate' n a = sequence (replicate n a)

-- etc. etc. Monad functions

main = do print (sequence' [[1, 2, 3], [4, 5, 6]])
```

```
        print (sequence' [Just 7, Just 8])
        print (join' [[1, 2, 3], [4, 5, 6]])
        print (join' (Just (Just 7)))
        print (replicate' 3 "abc")
        print (replicate' 4 (Just 8))

{-
[[1,4],[1,5],[1,6],[2,4],[2,5],[2,6],[3,4],[3,5],[3,6]]
Just [7,8]
[1,2,3,4,5,6]
Just 7
["aaa","aab","aac","aba","abb","abc","aca","acb","acc","baa",
  "bab","bac","bba","bbb","bbc","bca","bcb","bcc","caa","cab","cac",
  "cba","cbb","cbc","cca","ccb","ccc"]
Just [8,8,8,8]
-}
```

B. Tony Morris - PGP Key


```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1.4.6 (GNU/Linux)

mQGiBETNyC0RBAC3MYSZSbDZhBLKra2YUphB9OO6+qMFl/v2Lq8590ZfeE2WjIOu
c/KGKyOigXztMrA4+iekUjM4FA8E6AlBRQiAqZK8HF0ftX5hDpuSyEKkZe3jcxxI
BbhwX/SWHtDEVzwNmvO1wEnwhRE1oY/BCmy+bQ9wmAjlNav4UbOIcXcJUwCgz6FF
UwDvPxzybgd9FNS14BE37n8D/AzGmGjunBW/x+g/ndwu6WEpTEn1ZNdja6VrNXSG
xQ8XM+NBulroAIYX+YWdHsKnuGvSKCgVoc1ifVHdztA9sksID5GBGzhmVbIP2E16
w/LEzqqwruv9dX0Y1b7n8hcnvbl4DgEgLgeQ+VgJfLUkY2jFZ2m3dEBRH9USSgB/
V2pBBACP4Us2ZBYEXtMG29g6GqyeeJLEP34PLYVHWJZbet/wPQsHFhYahhzjwZGG
Srp0JUpegJOdaqX/Y0nio2whgCpqJcrbGuUBqNgQI3k4gvBwnPyx7jM0kfHfFORG
lq9SDbfLpV0EJx6fWXGStW33zsQQvenDcV2F5czzKQcy66BFwbQhVG9ueSBNb3Jy
aXMgPHRtb3JyaXNAdG1vcnJpcy5uZXQ+iGMEExECACMFAkTNyC0FCQlmAYAGCwkI
BwMCBBUCCAMEFgIDAQIeAQIXgAAKCRCaemCth7qvrf0pAKCFii2pI2W1BKVFuQcw
yoNxP0CAUACgyhuI9isCvtrOkeyjDmCVueRCdaC5Ag0ERM3IThAIAJ6A1z+d40ve
WvPIhpFiGfoS8UX4YdgYpo2mC/orY0xszBitogtaTHQHU5YDemGg81plNg9I3DbM
Er8uyONV4DqF6RbLj4w+iA6zn93+PTEZ73ydhxF6vDuojpVZPXVzXzpgyXHkEVLC
3hKL9oVlEsh+DWCvCiSAIy780JZ3FNVuMC3VH4qKxTw0CwPuuZvVfnMoIRfpODRR
fVEk2VDor+lr8kqJkBaHgN5o/AvOXC7QCYadwbEkpr0ecxIZ1VcASYytIIM3YNL7
ZcHWwU5PCNLOdMXPqOdthhDhsHkKJNEXXr0YsjX/bQqYOUqYKPDyqh/yrrRO9Ro6
7eTSbfIguycAAwcH/2waLIQR8qYKxPknNuSdsOOqF2jf2gglL/7uMsIzjfkFzgHo
+GNHw9tmlZqD3yzaZ/N7Yv08ujRHhmWPBYAWRICBM3qo0zMJ9kI5XWRobeRQpLtf
YxxIOenq8R9t6YU9ryHdqf+P+Fi38eN5ERTDhNLrJOnO5/TA+of97BWCmdtJMlWM
RaHtqXxwo02Yi65IqKx6L7oOvT7Gh4NV2eglz8ZafPEoP8+V8ER7rwBYPiLk4Mse
oVImjveq2dmLUip9OPwznoaeyC8zB0mJ13m/KNC+CffkBgoXpMPiKzbu5YTjVw++
MlEmfgL42yDK0hnokmW2i9y1RBh/T1VQQeQbUaeITAQYEQIADAUCRM3ITgUJCWYB
gAAKCRCaemCth7qvrcbtAJ9j3C6lKNRB3uKcrfze66jAVQh0qACaAysOK82TcQ/2
73ryR0xWMFnpGqg=
=bMTb
-----END PGP PUBLIC KEY BLOCK-----
```


C. Licence

a process analogous to cinematography; a work of drawing, painting, architecture, sculpture, engraving or lithography; a photographic work to which are assimilated works expressed by a process analogous to photography; a work of applied art; an illustration, map, plan, sketch or three-dimensional work relative to geography, topography, architecture or science; a performance; a broadcast; a phonogram; a compilation of data to the extent it is protected as a copyrightable work; or a work performed by a variety or circus performer to the extent it is not otherwise considered a literary or artistic work.

9. "You" means an individual or entity exercising rights under this License who has not previously violated the terms of this License with respect to the Work, or who has received express permission from the Licensor to exercise rights under this License despite a previous violation.

10. "Publicly Perform" means to perform public recitations of the Work and to communicate to the public those public recitations, by any means or process, including by wire or wireless means or public digital performances; to make available to the public Works in such a way that members of the public may access these Works from a place and at a place individually chosen by them; to perform the Work to the public by any means or process and the communication to the public of the performances of the Work, including by public digital performance; to broadcast and rebroadcast the Work by any means including signs, sounds or images.

11. "Reproduce" means to make copies of the Work by any means including without limitation by sound or visual recordings and the right of fixation and reproducing fixations of the Work, including storage of a protected performance or phonogram in digital form or other electronic medium.

2. Fair Dealing Rights. Nothing in this License is intended to reduce, limit, or restrict any uses free from copyright or rights arising from limitations or exceptions that are provided for in connection with the copyright protection under copyright law or other applicable laws.

3. License Grant. Subject to the terms and conditions of this License, Licensor hereby grants You a worldwide, royalty-free, non-exclusive, perpetual (for the duration of the applicable copyright) license to exercise the rights in the Work as stated below:

   1. to Reproduce the Work, to incorporate the Work into one or more Collections, and to Reproduce the Work as incorporated in the Collections;

   2. to create and Reproduce Adaptations provided that any such Adaptation, including any translation in any medium, takes reasonable steps to clearly label, demarcate or otherwise identify that changes were made to the original Work. For example, a translation could be marked "The original work was translated from English to Spanish," or a modification could indicate "The original work has been modified.";

   3. to Distribute and Publicly Perform the Work including as incorporated in Collections; and,

   4. to Distribute and Publicly Perform Adaptations.

   5. For the avoidance of doubt:

      1. Non-waivable Compulsory License Schemes. In those jurisdictions in which the right to collect royalties through any statutory or compulsory licensing scheme cannot be waived, the Licensor reserves the exclusive right to collect such royalties for any exercise by You of the rights granted under this License;

      2. Waivable Compulsory License Schemes. In those jurisdictions in which the right to collect royalties through any statutory or compulsory licensing scheme can be waived, the Licensor waives the exclusive right to collect such royalties for any exercise by You of the rights granted under this License; and,

      3. Voluntary License Schemes. The Licensor waives the right to collect royalties, whether individually or, in the event that the Licensor is a member of a collecting society that adminis-

ters voluntary licensing schemes, via that society, from any exercise by You of the rights granted under this License.

The above rights may be exercised in all media and formats whether now known or hereafter devised. The above rights include the right to make such modifications as are technically necessary to exercise the rights in other media and formats. Subject to Section 8(f), all rights not expressly granted by Licensor are hereby reserved.

4. Restrictions. The license granted in Section 3 above is expressly made subject to and limited by the following restrictions:

    1. You may Distribute or Publicly Perform the Work only under the terms of this License. You must include a copy of, or the Uniform Resource Identifier (URI) for, this License with every copy of the Work You Distribute or Publicly Perform. You may not offer or impose any terms on the Work that restrict the terms of this License or the ability of the recipient of the Work to exercise the rights granted to that recipient under the terms of the License. You may not sublicense the Work. You must keep intact all notices that refer to this License and to the disclaimer of warranties with every copy of the Work You Distribute or Publicly Perform. When You Distribute or Publicly Perform the Work, You may not impose any effective technological measures on the Work that restrict the ability of a recipient of the Work from You to exercise the rights granted to that recipient under the terms of the License. This Section 4(a) applies to the Work as incorporated in a Collection, but this does not require the Collection apart from the Work itself to be made subject to the terms of this License. If You create a Collection, upon notice from any Licensor You must, to the extent practicable, remove from the Collection any credit as required by Section 4(c), as requested. If You create an Adaptation, upon notice from any Licensor You must, to the extent practicable, remove from the Adaptation any credit as required by Section 4(c), as requested.

    2. You may Distribute or Publicly Perform an Adaptation only under the terms of: (i) this License; (ii) a later version of this License with the same License Elements as this License; (iii) a Creative Commons jurisdiction license (either this or a later license version) that contains the same License Elements as this License (e.g., Attribution-ShareAlike 3.0 US)); (iv) a Creative Commons Compatible License. If you license the Adaptation under one of the licenses mentioned in (iv), you must comply with the terms of that license. If you license the Adaptation under the terms of any of the licenses mentioned in (i), (ii) or (iii) (the "Applicable License"), you must comply with the terms of the Applicable License generally and the following provisions: (I) You must include a copy of, or the URI for, the Applicable License with every copy of each Adaptation You Distribute or Publicly Perform; (II) You may not offer or impose any terms on the Adaptation that restrict the terms of the Applicable License or the ability of the recipient of the Adaptation to exercise the rights granted to that recipient under the terms of the Applicable License; (III) You must keep intact all notices that refer to the Applicable License and to the disclaimer of warranties with every copy of the Work as included in the Adaptation You Distribute or Publicly Perform; (IV) when You Distribute or Publicly Perform the Adaptation, You may not impose any effective technological measures on the Adaptation that restrict the ability of a recipient of the Adaptation from You to exercise the rights granted to that recipient under the terms of the Applicable License. This Section 4(b) applies to the Adaptation as incorporated in a Collection, but this does not require the Collection apart from the Adaptation itself to be made subject to the terms of the Applicable License.

    3. If You Distribute, or Publicly Perform the Work or any Adaptations or Collections, You must, unless a request has been made pursuant to Section 4(a), keep intact all copyright notices for the Work and provide, reasonable to the medium or means You are utilizing: (i) the name of the Original Author (or pseudonym, if applicable) if supplied, and/or if the Original Author and/or Licensor designate another party or parties (e.g., a sponsor institute, publishing entity, journal) for attribution ("Attribution Parties") in Licensor's copyright notice, terms of service or by other reasonable means, the name of such party or parties; (ii) the title of the Work if supplied; (iii) to the extent reasonably practicable, the URI, if any, that Licensor specifies to be associated with the Work, unless such URI does not refer to the copyright notice or licensing information for the Work; and (iv) , consistent with Ssection 3(b), in the case of an Adaptation, a credit identifying

the use of the Work in the Adaptation (e.g., "French translation of the Work by Original Author," or "Screenplay based on original Work by Original Author"). The credit required by this Section 4(c) may be implemented in any reasonable manner; provided, however, that in the case of a Adaptation or Collection, at a minimum such credit will appear, if a credit for all contributing authors of the Adaptation or Collection appears, then as part of these credits and in a manner at least as prominent as the credits for the other contributing authors. For the avoidance of doubt, You may only use the credit required by this Section for the purpose of attribution in the manner set out above and, by exercising Your rights under this License, You may not implicitly or explicitly assert or imply any connection with, sponsorship or endorsement by the Original Author, Licensor and/or Attribution Parties, as appropriate, of You or Your use of the Work, without the separate, express prior written permission of the Original Author, Licensor and/or Attribution Parties.

4. Except as otherwise agreed in writing by the Licensor or as may be otherwise permitted by applicable law, if You Reproduce, Distribute or Publicly Perform the Work either by itself or as part of any Adaptations or Collections, You must not distort, mutilate, modify or take other derogatory action in relation to the Work which would be prejudicial to the Original Author's honor or reputation. Licensor agrees that in those jurisdictions (e.g. Japan), in which any exercise of the right granted in Section 3(b) of this License (the right to make Adaptations) would be deemed to be a distortion, mutilation, modification or other derogatory action prejudicial to the Original Author's honor and reputation, the Licensor will waive or not assert, as appropriate, this Section, to the fullest extent permitted by the applicable national law, to enable You to reasonably exercise Your right under Section 3(b) of this License (right to make Adaptations) but not otherwise.

5. Representations, Warranties and Disclaimer

   UNLESS OTHERWISE MUTUALLY AGREED TO BY THE PARTIES IN WRITING, LICENSOR OFFERS THE WORK AS-IS AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE WORK, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTIBILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO SUCH EXCLUSION MAY NOT APPLY TO YOU.

6. Limitation on Liability. EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. Termination

   1. This License and the rights granted hereunder will terminate automatically upon any breach by You of the terms of this License. Individuals or entities who have received Adaptations or Collections from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 5, 6, 7, and 8 will survive any termination of this License.

   2. Subject to the above terms and conditions, the license granted here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

8. Miscellaneous

   1. Each time You Distribute or Publicly Perform the Work or a Collection, the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License.