Morpho VM: An Indirect Threaded Stackless Virtual Machine

Hallgrímur H. Gunnarsson

Tölvunarfræðideild
Háskóli Íslands
2010

Morpho VM: An Indirect Threaded Stackless Virtual Machine

Hallgrímur H. Gunnarsson

12 ECTS eininga ritgerð sem er hluti af
Baccalaureus Scientiarum gráðu í Tölvunarfræði

Leiðbeinandi
Snorri Agnarsson

Iðnaðarverkfræði-, vélaverkfræði- og tölvunarfræðideild
Verkfræði- og náttúruvísindasvið
Háskóli Íslands
Reykjavík, apríl 2010

Morpho VM: An Indirect Threaded Stackless Virtual Machine
Morpho VM
12 ECTS eininga ritgerð sem er hluti af *Baccalaureus Scientiarum* gráðu í Tölvunarfræði

Iðnaðarverkfræði-, vélaverkfræði- og tölvunarfræðideild
Verkfræði- og náttúruvísindasvið
Háskóli Íslands
Hjarðarhaga 6
107 Reykjavík

Sími: 525 4000

Skráningarupplýsingar:
Hallgrímur H. Gunnarsson, 2010, *Morpho VM: An Indirect Threaded Stackless Virtual Machine*,
BS ritgerð, Tölvunarfræðideild, Háskóli Íslands, 25 bls.

Prentun: Háskólaprent
Reykjavík, apríl 2010

Contents

Abstract

The Morpho Virtual Machine provides a simple high-level execution environment for dynamic languages. It is written in Java and runs on top of the Java Virtual Machine using an efficient dispatch technique based on indirect threading. Its primary design goal is to provide first-class architectural support for lightweight concurrency and high-level features such as first-class closures, block structure, nested lexical scopes, lightweight threads and fibers.

This report describes the design and implementation of the Morpho Virtual Machine.

Útdráttur

Morpho sýndarvélin býður upp á einfalt keyrslu umhverfi fyrir kvik forritunarmál. Hún er forrituð í Java forritunarmálinu og keyrir ofan á Java sýndarvélinni. Morpho sýndarvélin framkvæmir sýndarskipanir með skilvirkri aðferð sem byggist á óbeinni þræðingu (e. indirect threading). Helsta markmið Morpho sýndarvélarinnar er að veita beinan stuðning við hagnýta og ódýra samskeiða forritun og æðri eiginleika í forritunarmálum svo sem fyrsta-flokks lokanir, bálkmótun, földun, létta þræði og trefjar.

Í þessari skýrslu er farið yfir helstu þætti í hönnun og útfærslu Morpho sýndarvélarinnar.

List of Figures

1 Introduction

The Java Virtual Machine (JVM)[7] is one of the most famous and widely used process virtual machines at the time of writing. It offers a portable high-level execution environment that has been heavily optimized. It was originally designed for the statically-typed Java language, but it has become an attractive target for compilers of languages other than Java, e.g. Clojure (a functional Lisp dialect), Groovy (a scripting language), JRuby (an implementation of Ruby), Jython (an implementation of Python), Rhino (an implementation of JavaScript) and Scala (an object-oriented and functional language).

Any language with functionality that can be expressed in terms of Java bytecode (a valid class file) can be compiled to run on the JVM. Unfortunately, the expressive power of Java bytecode is for the most part limited to the set of language features present in the Java programming language. Features of more dynamic languages and functional languages, such as dynamic dispatch, first-class closures, tail-call optimizations, continuations, coroutines and fibers, are difficult or even impossible to express in Java bytecode. As a result, implementors must either abandon certain features or, when possible, use special tricks to work around the limitations of the JVM. For example, Clojure does not support continuations or tail-call optimizations, JRuby does not support continuations, and none of the languages listed above support coroutines or fibers.

The Morpho programming language is designed to be a massively scalable scripting language. It is a dynamic language that supports functional programming as well as imperative programming and object-oriented programming. It uses run-time typing rather than compile-time typing. It supports first-class closures, tail-recursion and tail-call removal, and it supports fibers as a building block for lightweight concurrency.

Because of the limitations of the JVM, the Morpho programming language cannot be compiled to Java bytecode. Instead it is compiled to run on the Morpho Virtual Machine, which in turns runs on top of the Java Virtual Machine. In effect, the Morpho programming language is interpreted, but the Morpho Virtual Machine uses an efficient dispatch technique (described in section 2.2) which has proven to be effective.

The Morpho Virtual Machine is small. The source code is approximately 8000 lines in 76 files. It is available for download at http://morpho.cs.hi.is/.

2 The Morpho Virtual Machine

2.1 Registers

The Morpho Virtual Machine supports the concurrent execution of multiple independent interpreters. Each interpreter has the following virtual registers:

1. Accumulator.

2. Activation record.

3. Program (an array of instructions).

4. Program counter.

The accumulator holds a single value that can be any Java object. The activation record register holds a reference to the current activation record. The program register holds a reference to the current instruction array. The Morpho VM supports multiple instruction arrays. The program counter is an index into the current instruction array. It holds the integer address of the current instruction.

2.2 Execution and dispatch

Programs intended to run on the Morpho Virtual Machine must be compiled into a compact binary executable format. Modules are compiled into `.mmod` files and executables into `.mexe` files. Both of these contain within them a sequence of Morpho VM instructions in serialized form. The Morpho Virtual Machine executes an `.mexe` file by loading it and then interpreting Morpho VM instructions until the program terminates.

When the loader reads a file, it decodes the instruction sequence and constructs an internal representation of it that is suitable for execution. The internal representation uses a form of indirect threading, where each VM instruction is a Java object and the instruction sequence is an array of VM instruction objects.

In general, a VM interpreter executes a virtual program by dispatching its virtual instructions in sequence. A dispatch consists of fetching, decoding and executing an instruction. A major concern in the design of VM interpreters is the cost of a dispatch, as it can make up most of the run-time of an interpreter[6].

Work on the dispatch problem has shown that the efficiency of dispatch techniques depends largely on the hardware platform, because dispatch is highly dependent on micro-architectural features, such as branch prediction. Up to 30-40% of interpreter execution time is due to stalls caused by branch misprediction[3]. Direct threaded interpreters are known to have very poor branch prediction properties and switched-interpreters are even worse[6, 3].

A CGO 2005[3] paper explores and identifies the cause of branch mispredictions in direct threading. Due to the poor correlation between the virtual and hardware control flow, what they call the context problem, direct threading's indirect branches are poorly predicted by the hardware. They propose an alternative dispatch technique called context threading that eliminates 95% of branch mispredictions by aligning the virtual and hardware control flow. The alignment is based on using linear virtual instructions that are dispatched with native calls and returns. The idea is inspired by subroutine threading in Forth.

Our approach to dispatch uses a form of indirect threading that follows the same idea. The Morpho VM uses linear virtual instructions that are dispatched with method calls and returns. Each virtual instruction is a Java object and its operands are stored within the object. A

virtual program is an array of such objects. The loader effectively "precompiles" the instructions by creating ready-to-run Java instruction objects that already contain their operands. This eliminates the need to decode instructions in the interpreter loop. The essence of the interpreter loop is shown below.

```
Operation[] code;

// ...

for (;;)
{
    code[pc++].execute(this);
}
```

2.3 Instruction set

The Morpho Virtual Machine currently offers 66 different instructions. New instructions can easily be created by subclassing the base Operation class. An exhaustive description of the entire set of instructions can be found in the Morpho VM manual[2].

2.4 Scheduler

The Morpho Virtual Machine supports the concurrent execution of multiple *processes*. When the VM starts, it starts a pool of Java worker threads. These are usually native OS threads that can take advantage of multi-processor hardware.

The virtual machine maintains a queue of *ready processes*. A special thread (the main thread) runs the VM scheduler. When a process becomes ready the scheduler takes it and schedules it for execution by a thread in the pool.

If multi-processing is not required then the main thread takes the role of the worker threads and executes the processes by itself.

2.5 Processes

A process consumes a single Java thread during execution. When a process is executed it runs until one of the following conditions is met and then it releases its thread back to the pool:

1. The process explicitly yields control.
2. The process is out of work (no ready fibers).

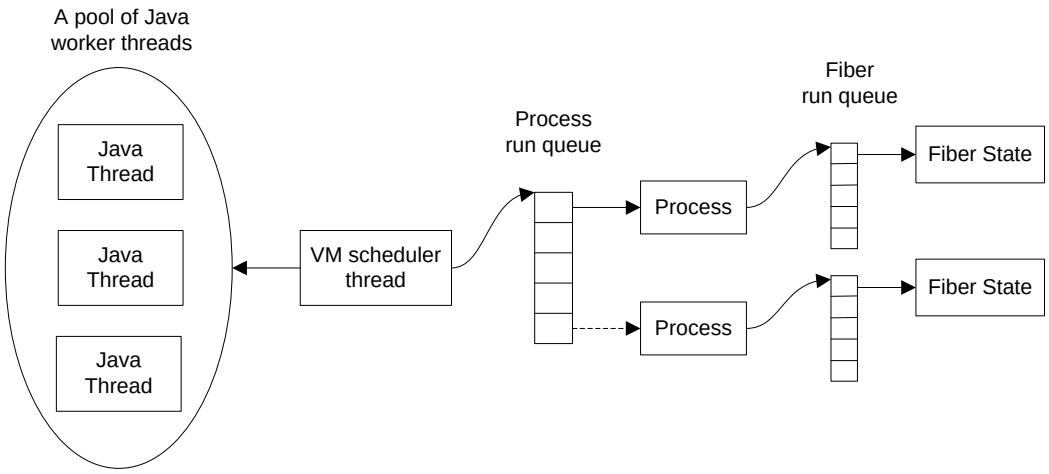

Figure 1: The scheduler

3. The process has executed a certain number of instructions and yields for fairness reasons.

If a process yields control then it is rescheduled for execution at a later time.

The Morpho Virtual Machine supports the execution of multiple independent interpreters through the use of processes. Each process is a VM interpreter with its own set of registers and interpreter state. In addition, each process maintains a queue of *ready fibers*. In some sense, a process in Morpho is merely an abstraction for a set of fibers that can be collectively scheduled to run within a thread.

2.6 Fibers

Fibers are the fundamental unit of execution in the Morpho Virtual Machine. Each fiber belongs to a particular process and each fiber encapsulates control state (a thread of control) that allows its execution to be suspended and resumed. A VM interpreter that is running within a process always runs in the context of a particular fiber. Fibers that are ready for execution are placed on a run-queue within the process that owns them. When the process runs, it picks a fiber from its queue and runs it.

The fibers in a process are cooperatively multitasked. Each fiber runs until it explicitly yields, suspends, or performs a blocking operation.

By only yielding control at well defined points, the interleaving of fibers becomes predictable. This is in contrast to preemptive scheduling where the interleaving is arbitrary and not in any way subject to control by the program. This allows the programmer to maintain invariants across interleavings without the use of locks and other low-level synchronization primitives.

2.7 Stackless

The Morpho Virtual Machine interpreter is *stackless* in the sense that it decouples the stack of the interpreter (the Java stack) from the virtual stack of the program under interpretation.

Interpreters with coupled (or otherwise mingled) stacks are restricted in several ways. A good example is the CPython interpreter. For each function call in Python, there is a Python stack frame allocated on the heap, but there is also a corresponding frame on the C stack. The C stack cannot be manipulated in any portable way[1], and as a consequence of the coupling, the limitations of the C stack are imposed on the Python stack as well.

Without the ability to manipulate the stack, it becomes difficult to implement many high-level features that require stack manipulation, such as tail recursion and multi-tasking with fibers (or co-routines).

2.8 Activation records

The virtual stack of a fiber is represented as a chain of activation records that are allocated on the heap. The activation records are Java objects and as such, they are garbage collected by Java when they are no longer needed.

Each activation record contains the following values

1. Return address: the address (index) of the next instruction following a return from the called function to the calling function. The execution picks up where it left off when control was passed away by loading the stored return address as the program counter.
2. Control link: the continuation activation record to use when returning from the current function. The chain of control links traces the dynamic execution of a program.
3. Environment: the environment contains the parameters and local variables for the activation record. In addition, the environment contains a reference (access link) to the lexically enclosing environment. The chain of access links traces the static structure (enclosing lexical scopes) of a function.
4. Temporary variables

Figure 2 shows the relationship between a fiber and its associated activation records and environments.

The structure of the activation records makes it easy to implement features such as first-class closures, block structure and nested lexical scopes.

[1] Exceptions: getcontext/setcontext, setjmp/longjmp

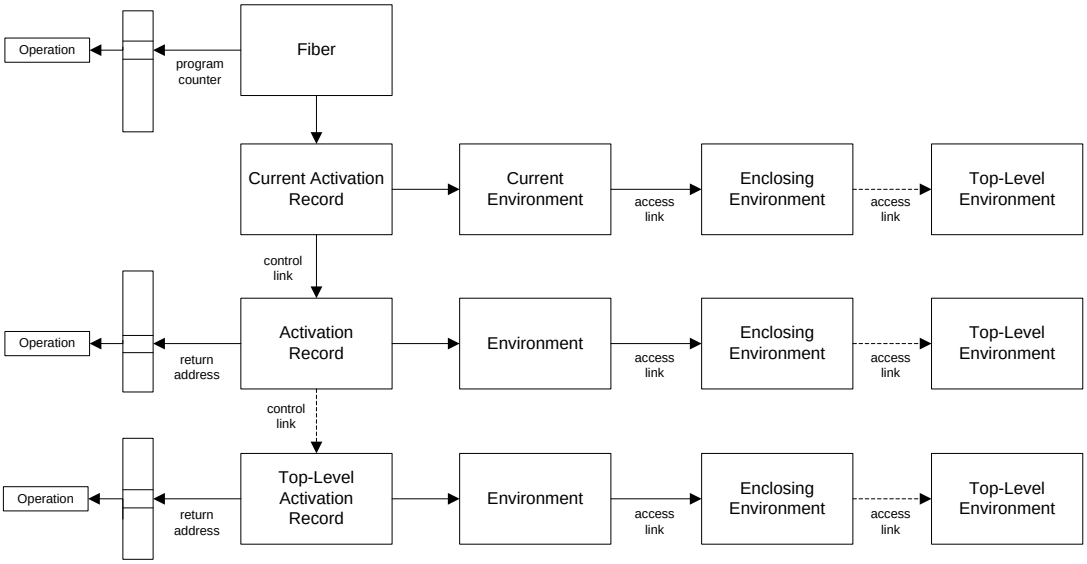

Figure 2: Activation records

3 Conclusion

This report presents the current state of the Morpho Virtual Machine. Its primary advantage over the Java Virtual Machine is the direct support it offers for high-level features such as first-class closures, lightweight threads and fibers. The expressive power of the Morpho Virtual Machine and its ability to offer direct support is derived from its stackless design and the heap-allocation of activation records. It is also what limits it to being interpreted instead of being compiled to Java bytecode.

The performance of the Morpho Virtual Machine has not been extensively tested and measured but early results are promising. The dispatch technique seems to offer a reasonably good performance in comparison to other interpreted languages. Scalability tests on modest hardware have shown that the Morpho Virtual Machine can support millions of concurrent fibers and the number of concurrent processes can be orders of magnitude times the number of native Java threads.

Future work includes improved support for native I/O that integrates with the concurrency model. Currently, a fiber may block all other fibers in the same process if it performs a blocking I/O operation. One approach to solving the problem is to wrap blocking I/O calls and use asynchronous I/O and event-dispatch to drive the scheduling of fibers. Given an efficient I/O library and the lightweight concurrency constructs of the Morpho Virtual Machine, it would be easy to build massively scalable network servers that are based on structured programming and sequential control flow, as opposed to event-based servers that use state machines and inverted control flow.

References

[1] A. Adya, J. Howell, M. Theimer, W. J. Bolosky, and J. R. Douceur. Cooperative task management without manual stack management. In *ATEC '02: Proceedings of the General Track of the annual conference on USENIX Annual Technical Conference*, pages 289–302, Berkeley, CA, USA, 2002. USENIX Association.

[2] S. Agnarsson. The morpho virtual machine manual, 2009. Technical report, University of Iceland.

[3] M. Berndl, B. Vitale, M. Zaleski, and A. D. Brown. Context threading: A flexible and efficient dispatch technique for virtual machine interpreters. In *In CGO '05: Proceedings of the international symposium on Code generation and optimization*, pages 15–26. IEEE Computer Society, 2005.

[4] H.-J. Boehm. Threads cannot be implemented as a library. *SIGPLAN Not.*, 40(6):261–268, 2005.

[5] F. Dabek, N. Zeldovich, F. Kaashoek, D. Mazières, and R. Morris. Event-driven programming for robust software. In *EW 10: Proceedings of the 10th workshop on ACM SIGOPS European workshop*, pages 186–189, New York, NY, USA, 2002. ACM.

[6] M. A. Ertl and D. Gregg. The structure and performance of efficient interpreters. *Journal of Instruction-Level Parallelism*, 5:2003, 2003.

[7] T. Lindholm and F. Yellin. *Java(TM) Virtual Machine Specification, The (2nd Edition)*. Prentice Hall PTR, 2 edition, April 1999.

[8] J. Ousterhout. Why threads are a bad idea (for most purposes), January 1996.

[9] H. Sutter. The free lunch is over: A fundamental turn toward concurrency in software. *Dr. Dobb's Journal*, 30(3), 2005.

[10] H. Sutter and J. Larus. Software and the concurrency revolution. *Queue*, 3(7):54–62, 2005.

[11] R. von Behren, J. Condit, and E. Brewer. Why events are a bad idea (for high-concurrency servers). In *HOTOS'03: Proceedings of the 9th conference on Hot Topics in Operating Systems*, pages 4–4, Berkeley, CA, USA, 2003. USENIX Association.

A Appendix

A.1 Machine.java

```
/*
Copyright 2009, 2010 Snorri Agnarsson, Hallgrímur H. Gunnarsson

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

   http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
*/

package is.hi.cs.morpho;

import java.util.LinkedList;
import java.util.HashSet;

public class Machine implements Runnable
{
    public static String[] args;
    private final LinkedList<Process> ready; // Current queue of ready processes
    private final HashSet<Process> waiting;
    private volatile boolean dead = false;

    public Machine()
    {
        ready = new LinkedList<Process>();
        waiting = new HashSet<Process>();
    }

    public Machine(Operation[] code, int startPC)
    {
        this();
        Process p = new Process(this);
        FiberState f = new FiberState(p);
        f.ar = new ActivationRecord(null,new Environment(),code);
        f.ar.control = f.ar;
        f.ar.rp = -1;
        f.pc = startPC;
        p.add(f);
        //add(p);  previous line does this
        run1();
    }

    public boolean isDead()
    {
        return dead;
    }

    public void kill()
    {
        dead = true;
    }

    public void run()
    {
        run(4);
    }

    public void run( int n )
```

```
{
    if( n<2 )
    {
        run1();
        return;
    }
    try
    {
        java.util.concurrent.ExecutorService executor =
            java.util.concurrent.Executors.newFixedThreadPool(n);

        for(;;)
        {
            Process p;
            p = getNextReady();
            if( p==null ) break;
            executor.execute(p);
        }
        executor.shutdown();
    }
    catch( InterruptedException e )
    {
        throw new Error(e);
    }
}

public void run1()
{
    try
    {
        for(;;)
        {
            Process p;
            p = getNextReady();
            if( p==null ) break;
            p.run();
        }
    }
    catch( InterruptedException e )
    {
        throw new Error(e);
    }
}

private Process getNextReady() throws InterruptedException
{
    Process res;
    try
    {
        synchronized(this)
        {
            for(;;)
            {
                if( dead ) return null;
                if( ready.isEmpty() )
                {
                    wait();
                    continue;
                }
                res = ready.remove();
                waiting.remove(res);
                break;
            }
        }
        return res;
    }
    catch( InterruptedException e )
    {
        throw new Error(e);
    }
}
```

```
    public synchronized void add( Process p )
    {
        if( p.getMachine() != this )
            throw new Error("Attempt to add a process to a machine that does not own the process");
        if( waiting.contains(p) ) return;
        waiting.add(p);
        ready.offer(p);
        notifyAll();
    }
}
```

A.2 Process.java

```
/*
Copyright 2009, 2010 Snorri Agnarsson, Hallgrímur H. Gunnarsson

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

   http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
*/

package is.hi.cs.morpho;

import java.util.LinkedList;
import java.util.Deque;
import java.util.concurrent.LinkedBlockingDeque;

public class Process implements Runnable
{
    private int pc;
    private ActivationRecord ar; // Current activation record
    private Operation[] code; // Current code array, same as ar.code
    private Object acc; // Current accumulator
    private volatile FiberState current; // Current fiber
    private final LinkedList<FiberState> ready; // Current queue of ready fibers
    private boolean dead = false; // True iff this process has been killed and should stop
    private final Machine m;

    public ActivationRecord theLastTransferSourceActivationRecord;
    // Only guaranteed valid from transfer/throw
    // until next transfer/throw or next end of
    // catch block

    public Process( Machine m )
    {
        ready = new LinkedList<FiberState>();
        //ready = new LinkedBlockingDeque<FiberState>();
        //ready = new ConcurrentLinkedQueue<FiberState>();
        this.pc = 0;
        this.ar = null;
        this.m = m;
    }

    public final Machine getMachine()
    {
        return m;
    }

    public synchronized final boolean isDead()
    {
```

```
    return dead;
}

public synchronized final void kill()
{
    dead = true;
}

public Operation[] getCode()
{
    return code;
}

public synchronized void loadFiber(FiberState p)
{
    this.ar = p.ar;
    this.pc = p.pc;
    this.acc = p.acc;
    this.code = p.ar.code;
}

public void add(FiberState f)
{
    synchronized(ready)
    {
        ready.offer(f);
        ready.notifyAll();
    }
    m.add(this);
}

public synchronized FiberState getCurrent()
{
    return current;
}

public synchronized void setCurrent( FiberState s )
{
    current = s;
}

public int getNumReady()
{
    synchronized(ready) {return ready.size();}
}

public synchronized void saveFiber(FiberState p)
{
    p.ar = this.ar;
    p.pc = this.pc;
    p.acc = this.acc;
}

public void run()
{
    try
    {
        startNicely();
    }
    catch( InterruptedException e )
    {
        throw new Error(e);
    }
}

public final Object start() throws InterruptedException
{
    for(;;)
    {
        synchronized(this)
        {
            if( dead ) return acc;
```

```
            synchronized(ready) {current = ready.poll();}
            while( current==null )
            {
                wait();
                synchronized(ready) {current = ready.poll();}
            }
            loadFiber(current);
        }
        try
        {
            for (;;)
            {
                code[pc++].execute(this);
            }
        }
        catch( KillMachine e )
        {
            kill();
            m.kill();
            return acc;
        }
        catch( Die e )
        {
            kill();
            return acc;
        }
        catch( Throwable e )
        {
            synchronized(this)
            {
                FiberState exception_continuation =
                    this.ar==null ? null :
                                    this.ar.exception_continuation;

                if( exception_continuation == null )
                {
                    System.err.println("No exception handler.");
                    System.err.println("PC="+pc);
                    kill();
                    throw new Error(e);
                }
                theLastTransferSourceActivationRecord = this.ar;
                this.acc = e;
                this.pc = exception_continuation.pc;
                this.ar = exception_continuation.ar;
                this.code = exception_continuation.ar.code;
                if( current!=null )
                {
                    saveFiber(current);
                    synchronized(ready) {ready.offer(current);ready.notifyAll();}
                }
            }
        }
    }
}

public final synchronized void startNicely() throws InterruptedException
{
    int n=0;
    for(;;)
    {
        if( dead ) return;
        synchronized(ready) {current = ready.poll();}
        if( current==null ) return;
        loadFiber(current);
        try
        {
            for (;;)
            {
                if( n++ > 100000 )
                {
                    saveFiber(current);
```

```
                    synchronized(ready) {ready.addFirst(current);}
                    m.add(this);
                    return;
                }
                code[pc++].execute(this);
                if( current==null )
                {
                    if( dead ) return;
                    m.add(this);
                    return;
                }
            }
        }
        catch( KillMachine e )
        {
            kill();
            m.kill();
            return;
        }
        catch( Die e )
        {
            kill();
            return;
        }
        catch( Throwable e )
        {
            FiberState exception_continuation =
                this.ar==null ?  null :
                                 this.ar.exception_continuation;

            if( exception_continuation == null )
            {
                System.err.println("No exception handler.");
                System.err.println("PC="+pc);
                kill();
                throw new Error(e);
            }
            theLastTransferSourceActivationRecord = this.ar;
            this.acc = e;
            this.pc = exception_continuation.pc;
            this.ar = exception_continuation.ar;
            this.code = exception_continuation.ar.code;
            if( current!=null )
            {
                saveFiber(current);
                synchronized(ready) {ready.offer(current);ready.notifyAll();}
            }
        }
    }
}

public synchronized final void setPC(int location)
{
    this.pc = location;
}

public synchronized final int getPC()
{
    return this.pc;
}

public synchronized final ActivationRecord getAR()
{
    return this.ar;
}

public synchronized final Object getAcc()
{
    return this.acc;
}

public synchronized final void setAcc(Object o)
```

```
    {
        this.acc = o;
    }

    public synchronized final void setAR(ActivationRecord ar)
    {
        this.ar = ar;
        if( ar!=null && ar.code != null )
            this.code = ar.code;
        else
            throw new Error("AR invalid");
    }

    public synchronized Object getControlTrace()
    {
        ActivationRecord tmp = theLastTransferSourceActivationRecord;
        int n=0;
        while( tmp != null )
        {
            n++;
            tmp = tmp.control;
        }
        Object[] trace = new Object[n];
        int i=0;
        tmp = ar;
        while( tmp != null )
        {
            Object[] rec = new Object[6];  // [fname,line,file,fnameorg,lineorg,fileorg]
            rec[0] = tmp.function_name;
            rec[1] = tmp.line==0 ? null : (""+tmp.line);
            rec[2] = tmp.filename;
            Operation op = tmp.rp-2 >= 0 ? code[tmp.rp-2] : null;
            if( op instanceof Operation_CallInfo )
            {
                Operation_CallInfo info = (Operation_CallInfo)op;
                rec[3] = info.function_name;
                rec[4] = info.line==0 ? null : info.line;
                rec[5] = info.file_name;
            }
            trace[i++] = rec;
            //System.out.println(tmp.function_name);
            tmp = tmp.control;
        }
        return trace;
    }

    private java.util.HashMap<Integer,Object> theVarTable = new java.util.HashMap<Integer,Object>();

    public synchronized void storeGlobal( int idx, Object val )
    {
        if( theVarTable==null )
            theVarTable = new java.util.HashMap<Integer,Object>();
        theVarTable.put(idx,val);
    }

    public synchronized Object fetchGlobal( int idx )
    {
        if( theVarTable==null ) return null;
        return theVarTable.get(idx);
    }
}
```

A.3 FiberState.java

```
/*
Copyright 2009, 2010 Snorri Agnarsson, Hallgrímur H. Gunnarsson

Licensed under the Apache License, Version 2.0 (the "License");
```

```
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

   http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
*/

package is.hi.cs.morpho;

class FiberState
{
    public int pc;
    public ActivationRecord ar;
    public Object acc;
    public final Process proc;

    private java.util.HashMap<Integer,Object> theVarTable = null;

    public FiberState( Process p )
    {
        proc = p;
    }

    public synchronized void storeGlobal( int idx, Object val )
    {
        if( theVarTable==null )
            theVarTable = new java.util.HashMap<Integer,Object>();
        theVarTable.put(idx,val);
    }

    public synchronized Object fetchGlobal( int idx )
    {
        if( theVarTable==null ) return null;
        return theVarTable.get(idx);
    }

    public final Process getProcess()
    {
        return proc;
    }
}
```

A.4 ActivationRecord.java

```
/*
Copyright 2009, 2010 Snorri Agnarsson, Hallgrímur H. Gunnarsson

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

   http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
*/

package is.hi.cs.morpho;

public class ActivationRecord
```

```
{
    public ActivationRecord control;
    public int rp;
    public Environment env;
    public Object[] tmpvars = null;
    public FiberState exception_continuation;
    public Operation[] code;

    // Traceback info, set on entry to function
    public String function_name = null; // name of the function in original source file
    public String filename = null; // name of source file
    public int line = 0; // approximate line where function is defined

    public ActivationRecord( ActivationRecord outer, Environment env, Operation[] code )
    {
        this.env = env;
        if( outer!=null )
            this.exception_continuation = outer.exception_continuation;
        this.code = code;
    }

    public final void put( int n, Object o )
    {
        if (n >= 0) {
            env.put(n, o);
            return;
        }

        n = -1-n;

        if (tmpvars == null) {
            tmpvars = new Object[n<3?3:n+1];
        } else if (tmpvars.length <= n) {
            int k = 2*tmpvars.length;

            if (k <= n)
                k = n+1;

            Object[] newtmp = new Object[k];

            System.arraycopy(tmpvars, 0, newtmp, 0, tmpvars.length);
            tmpvars = newtmp;
        }

        tmpvars[n] = o;
    }

    public final Object get(int n)
    {
        if (n >= 0)
            return env.get(n);

        n = -1-n;

        if (tmpvars == null || tmpvars.length <= n)
            return null;

        return tmpvars[n];
    }
}
```

A.5 Environment.java

```
/*
Copyright 2009, 2010 Snorri Agnarsson, Hallgrímur H. Gunnarsson

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
```

```
You may obtain a copy of the License at

   http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
*/

package is.hi.cs.morpho;

import java.util.Vector;

public class Environment
{
    public Environment access;
    public Object[] localvars = null;

    public final void put(int n, Object o)
    {
        if (localvars == null) {
            localvars = new Object[n<3?3:n+1];
        } else if (localvars.length <= n) {
            int k = 2*localvars.length;

            if (k <= n)
                k = n+1;

            Object[] newtmp = new Object[k];

            System.arraycopy(localvars, 0, newtmp, 0, localvars.length);
            localvars = newtmp;
        }

        localvars[n] = o;
    }

    public final Object get(int n)
    {
        if (localvars == null || localvars.length <= n)
            return null;

        return localvars[n];
    }
}
```

A.6 Operation_Call.java

```
/*
Copyright 2009, 2010 Snorri Agnarsson, Hallgrímur H. Gunnarsson

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

   http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
*/

package is.hi.cs.morpho;
```

```
import java.io.InputStream;
import java.io.OutputStream;

public class Operation_Call extends Operation
{
    private int offset;
    private int parpos;

    public void setArg(int i, Comparable o)
    {
        switch (i) {
            case 0:
                this.offset = (Integer)o;
                break;
            case 1:
                this.parpos = (Integer)o;
                break;
        }
    }

    public int argCount()
    {
        return 2;
    }

    public Comparable getArg( int i )
    {
        switch(i)
        {
            case 0: return offset;
            case 1: return parpos;
            default:
                    throw new Error("Out of bounds");
        }
    }

    public void execute(Process m) throws Die, Yield, Exception
    {
        ActivationRecord current = m.getAR();
        Environment params = parpos==0 ? null : (Environment)current.get(parpos);
        if (offset < -1000000000) {
            int n = -offset-1000000000;
            try
            {
                m.setAcc(Builtins.execute(m, params, n));
                return;
            }
            catch( Yield e )
            {
                throw e;
            }
            catch( Exception e )
            {
                current.function_name = "Builtin operation "+Builtins.getOpName(n);
                current.line = 0;
                throw e;
            }
            catch( Error e )
            {
                current.function_name = "Builtin operation "+Builtins.getOpName(n);
                current.line = 0;
                throw e;
            }
        }


        if( params==null )
        {
            params = new Environment();
        }

        ActivationRecord other = new ActivationRecord(current,params,m.getCode());
```

```
        other.rp = m.getPC();
        other.control = current;

        m.setAR(other);
        m.setPC(m.getPC() + offset);

        if( parpos!=0 ) current.put(parpos, null);
    }
}
```

A.7 Operation_Become.java

```
/*
Copyright 2009, 2010 Snorri Agnarsson, Hallgrímur H. Gunnarsson

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

   http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
*/

package is.hi.cs.morpho;

import java.io.InputStream;
import java.io.OutputStream;

public class Operation_Become extends Operation
{
    private int offset;
    private int parpos;
    private int returnLevel = 0;

    public void setArg(int i, Comparable o)
    {
        switch (i) {
            case 0:
                this.offset = (Integer)o;
                break;
            case 1:
                this.parpos = (Integer)o;
                break;
            case 2:
                this.returnLevel = (Integer)o;
                break;
        }
    }

    public int argCount()
    {
        return 3;
    }

    public Comparable getArg( int i )
    {
        switch(i)
        {
            case 0:
                return offset;
            case 1:
                return parpos;
```

```
        case 2:
            return returnLevel;
        default:
            throw new Error("Out of bounds");
    }
}

public void execute(Process m) throws Die, Yield, Exception
{
    ActivationRecord current = m.getAR();
    Environment params;
    if( parpos==0 )
    {
        params = null;
    }
    else
    {
        params = (Environment)current.get(parpos);
        current.put(parpos, null);
    }
    if (offset < -1000000000) {
        int n=-offset-1000000000;
        int lev=returnLevel;
        while( lev>0 )
        {
            current=current.control;
            lev--;
        }
        m.setPC(current.rp);
        m.setAR(current.control);

        try
        {
            m.setAcc(Builtins.execute(m, params, n));
            return;
        }
        catch( Yield e )
        {
            throw e;
        }
        catch( Exception e )
        {
            m.setAR(current); // restore
            current.function_name = "Builtin operation "+Builtins.getOpName(n);
            current.line = 0;
            throw e;
        }
        catch( Error e )
        {
            m.setAR(current); // restore
            current.function_name = "Builtin operation "+Builtins.getOpName(n);
            current.line = 0;
            throw e;
        }
    }

    if( params==null )
    {
        params = new Environment();
    }

    ActivationRecord other = new ActivationRecord(current,params,m.getCode());
    int lev = returnLevel;
    while( lev > 0 )
    {
        current = current.control;
        lev--;
    }

    other.rp = current.rp;
    other.control = current.control;
```

```
        m.setAR(other);
        m.setPC(m.getPC() + offset);

    }
}
```

A.8 CMorphoRunner.java

```
/*
Copyright 2009, 2010 Snorri Agnarsson, Hallgrímur H. Gunnarsson

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

   http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
*/

package is.hi.cs.morpho;

import java.io.InputStream;
import java.util.zip.InflaterInputStream;
import java.io.DataInput;
import java.io.FileInputStream;
import java.io.DataInputStream;
import java.io.DataOutput;
import java.io.FileOutputStream;
import java.io.DataOutputStream;

public class CMorphoRunner
{
    static public Module getBasis()
        throws Exception
    {
        InputStream in = null;
        DataInputStream din = null;
        try{
            in = new CMorphoRunner().getClass().getClassLoader().getResourceAsStream("basis.mmod");
            in = new InflaterInputStream(in);
            din = new DataInputStream(in);
        }
        catch(Exception e)
        {
            e.printStackTrace();
            throw e;
        }
        Module res = new Module();
        res.load(din);
        din.close();
        return res;
    }

    static public void compile( String[] args )
        throws Exception
    {
        Loader.registerOps();
        Operation[] code = null;
        int pc;
        DataInputStream in=null;
        try
        {
            in = new DataInputStream(
```

```
                new InflaterInputStream(
                    new CMorphoRunner().getClass()
                                       .getClassLoader()
                                       .getResourceAsStream("cmorpho.mexe")
                )
             );

        Loader loader = new Loader(in);
        pc = loader.loadInt();
        int n = loader.loadInt();
        code = new Operation[n];
        for( int i=0 ; i!=n ; i++ )
        {
            byte typecode = loader.loadByte();
            if( typecode == -128 )
            {
                code[i] = loader.loadOperation();
            }
            else
            {
                code[i] = code[loader.loadInt()];
            }
        }
    }
    finally
    {
        if(in==null)
        {
            System.err.println("cmorpho.mexe resource not found");
        }
        else
        {
            in.close();
        }
    }
    String[] newArgs = new String[args.length-1];
    System.arraycopy(args,1,newArgs,0,args.length-1);
    Machine.args = newArgs;
    Machine m = new Machine(code, pc);
}

static public void main( String[] args ) throws Exception
{
    if( args.length < 1 ||
        args[0].equals("--help") ||
        args[0].equals("-h") ||
        args[0].equals("-?") )
    {
        System.err.println("Usage:");
        System.err.println("  morpho <name> <arg>...");
        System.err.println("Or:");
        System.err.println("  java -jar morpho.jar <name> <arg>...");
        System.err.println("Or:");
        System.err.println("  morpho [--compile|-c] [<opt>|<filename>]...");
        System.err.println("Or:");
        System.err.println("  java -jar morpho.jar [--compile|-c] [<opt>|<filename>]...");
        System.err.println("   where <opt> = [--asm|--noasm|--debug|--nodebug|--encoding NAME]");
        System.err.println("Or:");
        System.err.println("  morpho [--help|-h|-?]");
        System.err.println("Or:");
        System.err.println("  java -jar morpho.jar [--help|-h|-?]");
        System.exit(1);
    }
    if( args[0].equals("--compile") || args[0].equals("-c") )
    {
        compile(args);
        return;
    }
    Loader.registerOps();
    Operation[] code = null;
    int pc;
    DataInputStream in=null;
```

```
        try
        {
            in = new DataInputStream(
                 new java.util.zip.InflaterInputStream(
                 new FileInputStream(args[0]+".mexe")));

            Loader loader = new Loader(in);
            pc = loader.loadInt();
            int n = loader.loadInt();
            code = new Operation[n];
            for( int i=0 ; i!=n ; i++ )
            {
                byte typecode = loader.loadByte();
                if( typecode == -128 )
                {
                    code[i] = loader.loadOperation();
                }
                else
                {
                    code[i] = code[loader.loadInt()];
                }
            }
        }
        finally
        {
            if(in==null)
            {
                System.err.println("File "+args[0]+".mexe"+" not found");
            }
            else
            {
                in.close();
            }
        }
        Machine.args = args;
        Machine m = new Machine(code, pc);
    }
}
```

A.9 Closure.java

```
/*
Copyright 2009, 2010 Snorri Agnarsson, Hallgrímur H. Gunnarsson

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

   http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
*/

package is.hi.cs.morpho;

public class Closure
{
    public int pc;
    public Environment access;
    public int nparams;
    public Operation[] code;

    public Closure(int pc, Environment access, int nparams, Operation[] code)
    {
```

```
        this.pc = pc;
        this.access = access;
        this.nparams = nparams;
        this.code = code;
    }

    public void call( Process m, Environment params )
    {
        ActivationRecord current = m.getAR();
        ActivationRecord other = new ActivationRecord(current,params,code);

        other.rp = m.getPC();
        other.control = current;
        params.access = this.access;

        m.setAR(other);
        m.setPC(this.pc);
    }
}
```