```
/**
 * -----------------------------------------------------------------------------
 * "THE BEER-WARE LICENSE" (Revision 42):
 * <doktor@dyregod.dk> <mads@danquah.dk> <tkrogh@ruc.dk> <tnjr@ruc.dk> wrote
 * this file. As long as you retain this notice you can do whatever you want
 * with this stuff. If we meet some day, and you think this stuff is worth it,
 * you can buy us a beer in return.
 * -----------------------------------------------------------------------------
 */
package com.ibm.JikesRVM.opt;

import com.ibm.JikesRVM.*;
import com.ibm.JikesRVM.classloader.*;

import java.util.*;

import com.ibm.JikesRVM.opt.ir.*;
/**
 * Does the actual escape analysis, based on paper from 2000 by David Gay and Bj
arne Steensgaard
 *
 * Last change by: $Author: dyregod $
 *
 * $Header: /var/cvs/Jikes\040RVM/com/ibm/JikesRVM/opt/OPT_EscapeAnalysis.java,v
 1.8 2003/05/26 03:48:33 dyregod Exp $
 *
 * @version $Revision: 1.8 $
 */
public class OPT_EscapeAnalysis extends OPT_CompilerPhase implements OPT_Operato
rs
{
    public final boolean shouldPerform(OPT_Options options)
    {
        if (options.getOptLevel() >= 2 && (options.ESCAPE_IPA || options.ESCAPE_
NONIPA))
            return true;
        else
            return false;
    }
    private static OPT_OptimizationPlanCompositeElement eplan = getPlan();

    public void perform(OPT_IR ir)
    {
        OPT_EscapeSummary summary = OPT_EscapeSummaryDatabase.getEscapeSummary(i
r.getMethod());
        if (summary == null)
            summary = OPT_EscapeSummaryDatabase.createEscapeSummary(ir.getMethod
());
        summary.working = true;

        for (OPT_BasicBlockEnumeration oe = ir.getBasicBlocks(); oe.hasMoreEleme
nts();)
        {
            OPT_BasicBlock bb = oe.next();
            int ints = 0;
            for (OPT_InstructionEnumeration ie = bb.forwardInstrEnumerator(); ie
.hasMoreElements();)
            {
                OPT_Instruction i = ie.next();
                char opcode = i.getOpcode();
                switch (opcode)
                {
                    case IR_PROLOGUE_opcode :
```



```
                        prologue(i, summary);
                        break;
                    case CALL_opcode :
                //      call(i, summary, ir.options);
                        break;
                    case REF_MOVE_opcode :
                        move(i, summary);
                        break;
                    case PUTFIELD_opcode :
                        putfield(i, summary);
                        break;
                    case PUTSTATIC_opcode :
                        putstatic(i, summary);
                        break;
                    case ATHROW_opcode :
                        throwed(i, summary);
                        break;
                    case RETURN_opcode :
                        returns(i, summary);
                        break;
                        // only new and new unresolved, as we don't yet allocate
 arrays on stack
                    case NEW_UNRESOLVED_opcode :
                        news(i, summary);
                        break;
                    case NEW_opcode :
                        news(i, summary);
                        break;
                        // phi join
                    case PHI_opcode :
                        phi(i, summary);
                        break;
                    default :
                        break;
                }
            }
        }

        //
        if (ir.options.PRINT_ESCAPE_INFO != true)
            return;
        int s = 0;
        OPT_MethodRegisterProperties[] mregs =
            (OPT_MethodRegisterProperties[]) summary.store.values().toArray(new
OPT_MethodRegisterProperties[] {
        });
        System.out.println("***** BEGIN HIR for " + ir.getMethod() + " *****");
        ir.printInstructions();
        System.out.println("***** END HIR for " + ir.getMethod() + " *****");

        System.out.println("var:");
        OPT_Register[] regs = (OPT_Register[]) summary.store.keySet().toArray(ne
w OPT_Register[] {
        });
        for (int k = 0; k < mregs.length; k++)
        {
            if (!mregs[k].props[OPT_MethodRegisterProperties.ESCAPED]
                && !mregs[k].props[OPT_MethodRegisterProperties.LOOP]
                && !mregs[k].props[OPT_MethodRegisterProperties.RETURNED]
                && mregs[k].props[OPT_MethodRegisterProperties.FRESH])
                s++;
            System.out.print(
                regs[k]
```

```
                    + ": returned = "
                    + mregs[k].props[2]
                    + ", fresh = "
                    + mregs[k].props[0]
                    + ", escaped = "
                    + mregs[k].props[1]
                    + ", loop = "
                    + mregs[k].props[3]
                    + "\n");
        }
        System.out.println("parameters:");
        for (int i = 0; i < summary.parmprop.length; i++)
        {
            System.out.print(
                summary.parameters
                    + ": returned = "
                    + summary.parmprop[i].props[2]
                    + ", fresh = "
                    + summary.parmprop[i].props[0]
                    + ", escaped = "
                    + summary.parmprop[i].props[1]
                    + ", loop = "
                    + summary.parmprop[i].props[3]
                    + "\n");

        }

        System.out.println("Method is fresh? " + summary.fresh);
        System.out.println("Total new: " + summary.news);
        System.out.println("Total stackable: " + s);
    }
    private void prologue(OPT_Instruction i, OPT_EscapeSummary summary)
    {
        int k = Prologue.getNumberOfFormals(i);
        if (k == 0)
            k = 1;
        summary.setParameters(k);
        summary.parameters = new OPT_Operand[k - 1];
        for (int j = 0; j < summary.parameters.length; j++)
        {
            summary.parameters[j] = Prologue.getFormal(i, j + 1);
        }
    }
    private void putstatic(OPT_Instruction i, OPT_EscapeSummary summary)
    {
        OPT_Operand oper = PutStatic.getValue(i);
        if (oper.isRegister())
        {
            summary.setEscaped(oper.asRegister().register, true);
        }
    }
    private void throwed(OPT_Instruction i, OPT_EscapeSummary summary)
    {
        OPT_Operand oper = Athrow.getValue(i);
        if (oper.isRegister())
        {
            summary.setEscaped(oper.asRegister().register, true);
        }
    }
    private void returns(OPT_Instruction i, OPT_EscapeSummary summary)
    {
        OPT_Operand operand = Return.getVal(i);
```



```
        if (operand != null && operand.isRegister())
        {
            OPT_Register register = operand.asRegister().register;
            summary.setReturned(register, true);
            if (summary.getEscaped(register))
                summary.fresh = false;
            else
                summary.fresh = summary.getFresh(register);
        }
    }
    private void putfield(OPT_Instruction i, OPT_EscapeSummary summary)
    {
        OPT_Operand operand = PutField.getValue(i);
        if (operand.isRegister())
        {
            OPT_Register source = operand.asRegister().register;
            summary.setEscaped(source, true);
        }
    }
    private void phi(OPT_Instruction i, OPT_EscapeSummary summary)
    {
        OPT_Register des = Phi.getResult(i).asRegister().register;
        OPT_Operand source = Phi.getValue(i, 1);
        if (source != null && source.isRegister())
        {
            summary.setDependence(des, source.asRegister().register, OPT_MethodR
egisterProperties.ESCAPED);
            summary.setDependence(des, source.asRegister().register, OPT_MethodR
egisterProperties.RETURNED);
            summary.setLoop(source.asRegister().register, true);
        }
        source = Phi.getValue(i, 0);
        if (source != null && source.isRegister())
        {
            summary.setDependence(des, source.asRegister().register, OPT_MethodR
egisterProperties.ESCAPED);
            summary.setDependence(des, source.asRegister().register, OPT_MethodR
egisterProperties.RETURNED);
            summary.setLoop(source.asRegister().register, true);
        }
        summary.setFresh(des, false);
    }
    private void news(OPT_Instruction i, OPT_EscapeSummary summary)
    {
        OPT_Operand oper = New.getResult(i);
        if (oper.isRegister())
        {
            summary.setFresh(oper.asRegister().register, true);
        }
        summary.news++;
    }
    private void move(OPT_Instruction i, OPT_EscapeSummary summary)
    {
        OPT_Register dest = Move.getResult(i).asRegister().register;
        if (!Move.getVal(i).isRegister())
        {
            summary.setFresh(dest, false);
        }
        else
        {
            OPT_Register source = i.getOperand(1).asRegister().register;
            summary.setDependence(dest, source, OPT_MethodRegisterProperties.ESC
APED);
```

```
            summary.setDependence(dest, source, OPT_MethodRegisterProperties.LOO
P);
            summary.setDependence(dest, source, OPT_MethodRegisterProperties.RET
URNED);
        }
    }
    private void call(OPT_Instruction i, OPT_EscapeSummary summary, OPT_Options
options)
    {
        OPT_Register retval = null;
        OPT_Register source = null;
        OPT_Operand oper = Call.getResult(i);
        if (oper != null && oper.isRegister())
            retval = oper.asRegister().register;

        OPT_MethodOperand method = Call.getMethod(i);
        OPT_Operand[] parameters = new OPT_Operand[Call.getNumberOfParams(i)];
        for (int j = 0; j < parameters.length; j++)
        {
            parameters[j] = Call.getParam(i, j);
        }
        if (method == null)
        {
            for (int j = 0; j < parameters.length; j++)
            {
                if (parameters[j].isRegister())
                    summary.setEscaped(parameters[j].asRegister().register, true
);
            }
            return;
        }
        // we automagically detect if we are trying to evaluate recursively. It
will just say that its not fresh as fresh is init to false
        OPT_EscapeSummary es = OPT_EscapeSummaryDatabase.getEscapeSummary(method
.getTarget());
        if (es != null)
        {
            if (es.fresh && retval != null)
                summary.setFresh(retval, true);

            checkParameters(summary, es, parameters, retval);
        }
        else if (options.ESCAPE_IPA)
        {


            //VM_Class c = method.getTarget().getDeclaringClass();
            //System.err.println("\n" + c);
            // hack to enable full ipa when using testharness
            //loadClass();
            if (method.getTarget() == null || method.getTarget().isAbstract())
                return;
            if (!(method.getTarget() instanceof VM_NormalMethod))
                return;
            es = OPT_EscapeSummaryDatabase.createEscapeSummary(method.getTarget(
));

            OPT_CompilationPlan plan = new OPT_CompilationPlan((VM_NormalMethod)
 method.getTarget(), eplan, null, options);
            plan.analyzeOnly = true;
            try
            {
                OPT_Compiler.compile(plan);
```



```
            }
            catch (OPT_MagicNotImplementedException e)
            {
                //ignore
            }
            checkParameters(summary, es, parameters, retval);
            if (es.fresh && retval != null)
                summary.setFresh(retval, true);


        }
    }
    private void checkParameters(OPT_EscapeSummary summary, OPT_EscapeSummary es
, OPT_Operand[] parameters, OPT_Register retval)
    {
        if (es.parmprop != null)
        {
            for (int j = 0; j < es.parmprop.length; j++)
            {
                if (!parameters[j].isRegister())
                    continue;
                if (es.parmprop[j].props[OPT_MethodRegisterProperties.RETURNED]
== true && retval != null)
                {
                    summary.setDependence(retval, parameters[j].asRegister().reg
ister, OPT_MethodRegisterProperties.ESCAPED);
                    summary.setDependence(retval, parameters[j].asRegister().reg
ister, OPT_MethodRegisterProperties.RETURNED);
                    summary.setDependence(retval, parameters[j].asRegister().reg
ister, OPT_MethodRegisterProperties.LOOP);
                }
                if (es.parmprop[j].props[OPT_MethodRegisterProperties.ESCAPED] =
= true)
                {
                    // safe as the parameter can no longer change its value
                    summary.setEscaped(parameters[j].asRegister().register, true
);
                }
            }
        }
        else
        {
            for (int j = 0; j < parameters.length; j++)
            {
                if (parameters[j].isRegister())
                    summary.setEscaped(parameters[j].asRegister().register, true
);
            }
        }

    }
    public String getName()
    {
        return "Escape Analysis";
    }
    private static OPT_OptimizationPlanCompositeElement getPlan()
    {

        return OPT_OptimizationPlanCompositeElement
            .compose(
                "Escape Analysis",
                new Object[] { new OPT_ConvertBCtoHIR(), new OPT_BuildLST(), new
 OPT_YieldPoints(), new OPT_EstimateBlockFrequencies(),
            // Simple flow-insensitive optimizations
```

```
            new OPT_Simple(true, true),

            // Perform peephole branch optimizations to clean-up before SSA stuf
f
            new OPT_BranchOptimizations(1, true, true),
            // CFG spliting
            new OPT_StaticSplitting(),
            // restructure loops
            new OPT_CFGTransformations(),
            // Simple flow-insensitive optimizations
            new OPT_Simple(true, true), new OPT_BuildLST(), new OPT_EstimateBloc
kFrequencies(),
            // Local copy propagation
            new OPT_LocalCopyProp(),
            // Local constant propagation
            new OPT_LocalConstantProp(),
            // Insert PI Nodes
            new OPT_PiNodes(true),
            // branch optimization
            new OPT_BranchOptimizations(0, true, true),
            // Compute dominators
            new OPT_DominatorsPhase(true),
            // compute dominance frontier
            new OPT_DominanceFrontier(),
            // load elimination
            new OPT_LoadElimination(1),
            // load elimination
            new OPT_LoadElimination(2),
            // load elimination
            new OPT_LoadElimination(3),
            // load elimination
            new OPT_LoadElimination(4),
            // load elimination
            new OPT_LoadElimination(5),
            // eliminate redundant conditional branches
            new OPT_RedundantBranchElimination(),
            // path sensitive constant propagation
            new OPT_SSATuneUp(),
            // escape analysis
            new OPT_EscapeAnalysis()});
    }

}
```



```
/**
 * ------------------------------------------------------------------------
 * "THE BEER-WARE LICENSE" (Revision 42):
 * <doktor@dyregod.dk> <mads@danquah.dk> <tkrogh@ruc.dk> <tnjr@ruc.dk> wrote
 * this file. As long as you retain this notice you can do whatever you want
 * with this stuff. If we meet some day, and you think this stuff is worth it,
 * you can buy us a beer in return.
 * ------------------------------------------------------------------------
 */
package com.ibm.JikesRVM.opt;

import java.util.HashMap;
import java.util.Iterator;

import com.ibm.JikesRVM.opt.ir.*;
/**
 * Contains info on all registers in a method
 *
 * Last change by: $Author: dyregod $
 *
 * $Header: /var/cvs/Jikes\040RVM/com/ibm/JikesRVM/opt/OPT_EscapeSummary.java,v
1.7 2003/05/26 03:48:33 dyregod Exp $
 *
 * @version $Revision: 1.7 $
 */
public class OPT_EscapeSummary
{
    public HashMap store = new HashMap();
    public boolean fresh = false;
    boolean working = false;
    public int news = 0;
    public OPT_Operand[] parameters;
    public OPT_MethodRegisterProperties[] parmprop;

    public void setParameters(int i)
    {
        parmprop = new OPT_MethodRegisterProperties[i-1];
        for (int j = 0; j < parmprop.length; j++)
        {
            parmprop[j] = new OPT_MethodRegisterProperties();
        }
    }
    private int isParameter(OPT_Register register)
    {
        for (int i = 0; i < parameters.length; i++)
        {
            if (parameters[i].isRegister())
            {
                if (parameters[i].asRegister().register.equals(register))
                {
                    return i;
                }
            }
        }
        return -1;
    }
    public void setProperty(OPT_Register register, int prop, boolean value)
    {
        int i = isParameter(register);
        if (i != -1)
        {
            parmprop[i].props[prop] = value;
        }
```

```
        else
        {

            OPT_MethodRegisterProperties mreg;
            Object o = store.get(register);
            if (o == null)
            {
                mreg = new OPT_MethodRegisterProperties();
                store.put(register, mreg);
            }
            else
            {
                mreg = (OPT_MethodRegisterProperties) o;
            }
            mreg.props[prop] = value;
        }
        resolveDependencies(register);
    }
    public void setReturned(OPT_Register register, boolean returned)
    {
        setProperty(register, OPT_MethodRegisterProperties.RETURNED, returned);
    }
    public void setFresh(OPT_Register register, boolean fresh)
    {
        setProperty(register, OPT_MethodRegisterProperties.FRESH, fresh);
    }
    public void setEscaped(OPT_Register register, boolean escaped)
    {
        setProperty(register, OPT_MethodRegisterProperties.ESCAPED, escaped);
    }
    public void setLoop(OPT_Register register, boolean loop)
    {
        setProperty(register, OPT_MethodRegisterProperties.LOOP, loop);
    }
    public void setDependence(OPT_Register trig, OPT_Register tar, int prop)
    {
        setDependence(trig, tar, prop, this);
    }
    public void setDependence(OPT_Register trigger, OPT_Register target, int pro
p, OPT_EscapeSummary summary)
    {
        OPT_MethodRegisterProperties mreg;
        Object o = store.get(trigger);
        if (o == null)
        {
            mreg = new OPT_MethodRegisterProperties();
            //mreg.number = trigger.number;
            store.put(trigger, mreg);
        }
        o = store.get(target);
        if (o == null)
        {
            mreg = new OPT_MethodRegisterProperties();
            store.put(target, mreg);
        }
        Dependency dep = new Dependency(target, prop);
        ((OPT_MethodRegisterProperties) store.get(trigger)).deps.add(dep);
    }
    public void resolveDependencies(OPT_Register register)
    {
        OPT_MethodRegisterProperties smreg;
        int i = isParameter(register);
        if (i != -1)
```



```
            smreg = parmprop[i];
        else
            smreg
             = (OPT_MethodRegisterProperties) store.get(register);
        Iterator iter = smreg.deps.iterator();
        while (iter.hasNext())
        {
            Dependency element = (Dependency) iter.next();
            OPT_MethodRegisterProperties tmreg = (OPT_MethodRegisterProperties)
store.get(element.register);
            // hvis den når true så kan dependency slettes på smreg.
            if (smreg.props[element.prop])
            {
                tmreg.props[element.prop] = true;
                //smreg.deps.remove(element);
            }
        }
    }
    public boolean getFresh(OPT_Register register)
    {
        Object o = store.get(register);
        if (o == null)
        {
            return false;
        }
        else
        {
            return ((OPT_MethodRegisterProperties) o).props[OPT_MethodRegisterPr
operties.FRESH];
        }
    }
    public boolean getEscaped(OPT_Register register)
    {
        Object o = store.get(register);
        if (o == null)
        {
            return false;
        }
        else
        {
            return ((OPT_MethodRegisterProperties) o).props[OPT_MethodRegisterPr
operties.ESCAPED];
        }
    }
    public boolean isWorking()
    {
        return working;
    }
    public void setWorking(boolean w)
    {
        working = w;
    }
}
class Dependency
{
    OPT_Register register;
    int prop;
    public Dependency(OPT_Register register, int prop)
    {
        this.register = register;
        this.prop = prop;
    }
}
```

OPT_EscapeSummaryDatabase.java


```
/**
 * -----------------------------------------------------------------------------
 * "THE BEER-WARE LICENSE" (Revision 42):
 * <doktor@dyregod.dk> <mads@danquah.dk> <tkrogh@ruc.dk> <tnjr@ruc.dk> wrote
 * this file. As long as you retain this notice you can do whatever you want
 * with this stuff. If we meet some day, and you think this stuff is worth it,
 * you can buy us a beer in return.
 * -----------------------------------------------------------------------------
 */
package com.ibm.JikesRVM.opt;

import java.util.HashMap;

import com.ibm.JikesRVM.classloader.VM_Method;
/**
 * Contains all method summaries
 *
 * Last change by: $Author: dyregod $
 *
 * $Header: /var/cvs/Jikes\040RVM/com/ibm/JikesRVM/opt/OPT_EscapeSummaryDatabase
.java,v 1.2 2003/05/25 12:27:52 dyregod Exp $
 *
 * @version $Revision: 1.2 $
 */
public class OPT_EscapeSummaryDatabase
{
    static HashMap database = new HashMap();
    public static OPT_EscapeSummary[] getAll()
    {
        return (OPT_EscapeSummary[]) database.values().toArray(new OPT_EscapeSum
mary[] {});
    }
    public static OPT_EscapeSummary getEscapeSummary(VM_Method method)
    {
        return (OPT_EscapeSummary) database.get(method);
    }
    public static OPT_EscapeSummary createEscapeSummary(VM_Method method)
    {
        OPT_EscapeSummary summary = new OPT_EscapeSummary();
        database.put(method, summary);
        return summary;
    }

}
```

OPT_MethodRegisterProperties.java


```
/**
 * -----------------------------------------------------------------------------
 * "THE BEER-WARE LICENSE" (Revision 42):
 * <doktor@dyregod.dk> <mads@danquah.dk> <tkrogh@ruc.dk> <tnjr@ruc.dk> wrote
 * this file. As long as you retain this notice you can do whatever you want
 * with this stuff. If we meet some day, and you think this stuff is worth it,
 * you can buy us a beer in return.
 * -----------------------------------------------------------------------------
 */
package com.ibm.JikesRVM.opt;

import java.util.LinkedList;
/**
 * Contains properties for each method
 *
 * Last change by: $Author: dyregod $
 *
 * $Header: /var/cvs/Jikes\040RVM/com/ibm/JikesRVM/opt/OPT_MethodRegisterPropert
ies.java,v 1.2 2003/05/25 12:27:52 dyregod Exp $
 *
 * @version $Revision: 1.2 $
 */
public class OPT_MethodRegisterProperties
{
    // ought to be ok as code i supposed on ssa form
    public static final int FRESH = 0;
    public static final int ESCAPED = 1;
    public static final int RETURNED = 2;
    public static final int LOOP = 3;
    public boolean[] props = new boolean[4];
    public LinkedList deps = new LinkedList();
}
```