jminusminus

Class NControlFlowGraph

- java.lang.Object
  - jminusminus.NControlFlowGraph

.

```
class NControlFlowGraph
extends Object
```

Representation of a control flow graph (cfg) for a method.

- **Field Summary**

Fields

Modifier and Type	Field and Description
ArrayList<NBasicBlock>	**basicBlocks** List of blocks forming the cfg for the method.
static int	**blockId** block identifier.
ArrayList<String>	**data** SPIM code for string literals added to the data segment.
String	**desc** Descriptor of the method this cfg corresponds to.
static int	**hirId** HIR instruction identifier.
TreeMap<Integer,NHIRInstruction>	**hirMap**

	Maps HIR instruction ids in this cfg to HIR instructions.
ArrayList<NInterval>	**intervals** Intervals allocated by the register allocation algorithm.
String	**labelPrefix** Used to construct jump labels in spim output.
static int	**lirId** HIR instruction identifier.
static int	**loopIndex** Loop identifier.
int	**maxIntervals** The total number of intervals.
String	**name** Name of the method this cfg corresponds to.
int	**offset** Stack offset counter..
ArrayList<NPhysicalRegister>	**pRegisters** Physical registers allocated for this cfg by the HIR to LIR conversion algorithm.
static int	**regId** Virtual register identifier.
ArrayList<NRegister>	**registers** Registers allocated for this cfg by the HIR to LIR conversion algorithm.

- **Constructor Summary**

Constructors

Constructor and Description
NControlFlowGraph(CLConstantPool cp, CLMethodInfo m) Construct an NControlFlowGraph object for a method given the constant pool for the class containing the method and the object containing information about the method.

- **Method Summary**

Methods

Modifier and Type	Method and Description
void	**allocatePhysicalRegisters**() Replace references to virtual registers in LIR instructions with references to physical registers.
NBasicBlock	**blockAt**(int id) The basic block at a particular instruction id.
void	**computeDominators**(NBasicBlock block, NBasicBlock pred)

	Compute the dominator of each block in this cfg recursively given the starting block and its predecessor.
void	**detectLoops**(NBasicBlock block, NBasicBlock pred) Implements loop detection algorithm to figure out if the specified block is a loop head or a loop tail.
void	**eliminateRedundantPhiFunctions**() Eliminate redundant phi functions of the form x = (y, x, x, ..., x) with y.
void	**hirToLir**() Convert the hir instructions in this cfg to lir instructions.
void	**optimize**() Carry out optimizations on the high-level instructions.
void	**orderBlocks**() Compute optimal ordering of the basic blocks in this cfg.
void	**removeUnreachableBlocks**() Remove blocks that cannot be reached from the begin block (B0).
void	**renumberLirInstructions**() Assign new ids to the LIR instructions in this cfg.
void	**resolvePhiFunctions**() Resolve the phi functions in this cfg, i.e., for each x = phi(x1, x2, ..., xn) generate an (LIR) move xi, x instruction at the end of the predecessor i of this block defining the phi function; if the instruction there is a branch, add the instruction prior to the branch.
void	**tuplesToHir**() Convert tuples in each block to their high-level (HIR) representations.
void	**writeHirToStdOut**(PrettyPrinter p) Write the hir instructions in this cfg to STDOUT.
void	**writeIntervalsToStdOut**(PrettyPrinter p) Write the intervals in this cfg to STDOUT.
void	**writeLirToStdOut**(PrettyPrinter p) Write the lir instructions in this cfg to STDOUT.
void	**writeTuplesToStdOut**(PrettyPrinter p) Write the tuples in this cfg to STDOUT.

- **Methods inherited from class java.lang.Object**

  clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

- **Field Detail**

  - **blockId**

  ```
  public staticint blockId
  ```

  block identifier.

- **hirId**

```
public staticint hirId
```

HIR instruction identifier.

- **lirId**

```
public staticint lirId
```

HIR instruction identifier.

- **regId**

```
public staticint regId
```

Virtual register identifier.

- **offset**

```
publicint offset
```

Stack offset counter..

- **loopIndex**

```
public staticint loopIndex
```

Loop identifier.

- **name**

```
publicString name
```


Name of the method this cfg corresponds to.

- **desc**

```
publicString desc
```


Descriptor of the method this cfg corresponds to.

- **basicBlocks**

```
publicArrayList<NBasicBlock> basicBlocks
```

List of blocks forming the cfg for the method.

- **hirMap**

```
publicTreeMap<Integer,NHIRInstruction> hirMap
```

Maps HIR instruction ids in this cfg to HIR instructions.

- **registers**

```
publicArrayList<NRegister> registers
```

Registers allocated for this cfg by the HIR to LIR conversion algorithm.

- **maxIntervals**

```
publicint maxIntervals
```

The total number of intervals. This is used to name split children and grows as more intervals are created by spills.

- **pRegisters**

```
publicArrayList<NPhysicalRegister> pRegisters
```

Physical registers allocated for this cfg by the HIR to LIR conversion algorithm.

- **intervals**

```
publicArrayList<NInterval> intervals
```

Intervals allocated by the register allocation algorithm.

- **labelPrefix**

```
publicString labelPrefix
```

Used to construct jump labels in spim output.

- **data**

```
publicArrayList<String> data
```

SPIM code for string literals added to the data segment.

Constructor Detail

- **NControlFlowGraph**

```
publicNControlFlowGraph(CLConstantPoolcp,
                        CLMethodInfom)
```

Constructs an NControlFlowGraph object for a method given the constant pool for the class containing the method and the object containing information about the method.

Parameters:

`cp` - constant pool for the class containing the method.
`m` - contains information about the method.

Method Detail

- **detectLoops**

```
publicvoiddetectLoops(NBasicBlockblock,
                      NBasicBlockpred)
```

Implements loop detection algorithm to figure out if the specified block is a loop head or a loop tail. Also calculates the number of backward branches to the block.

Parameters:

`block` - a block.
`pred` - block's predecessor or null.

- **removeUnreachableBlocks**

```
publicvoidremoveUnreachableBlocks()
```

Remove blocks that cannot be reached from the begin block (B0). Also removes these blocks from the predecessor lists.

- **computeDominators**

```
publicvoidcomputeDominators(NBasicBlockblock,
                            NBasicBlockpred)
```

Compute the dominator of each block in this cfg recursively given the starting block and its predecessor.

Parameters:

`block` - starting block.
`pred` - block's predecessor.

- **tuplesToHir**

```
publicvoidtuplesToHir()
```

Convert tuples in each block to their high-level (HIR) representations.

- **optimize**

```
publicvoidoptimize()
```

Carry out optimizations on the high-level instructions.

- **eliminateRedundantPhiFunctions**

```
publicvoideliminateRedundantPhiFunctions()
```

Eliminate redundant phi functions of the form x = (y, x, x, ..., x) with y.

- **hirToLir**

```
publicvoidhirToLir()
```

Convert the hir instructions in this cfg to lir instructions.

- **resolvePhiFunctions**

```
publicvoidresolvePhiFunctions()
```


Resolve the phi functions in this cfg, i.e., for each x = phi(x1, x2, ..., xn) generate an (LIR) move xi, x instruction at the end of the predecessor i of the block defining the phi function. If the instruction there is a branch, add the instruction prior to the branch.

- **orderBlocks**

```
publicvoidorderBlocks()
```

Compute optimal ordering of the basic blocks in this cfg.

- **blockAt**

```
publicNBasicBlockblockAt(intid)
```

The basic block at a particular instruction id.

Parameters:

`id` - the (LIR) instruction id.

Returns:

the basic block.

- **renumberLirInstructions**

```
publicvoidrenumberLirInstructions()
```

Assign new ids to the LIR instructions in this cfg.

- **allocatePhysicalRegisters**

```
publicvoidallocatePhysicalRegisters()
```

Replace references to virtual registers in LIR instructions with references to physical registers.

- **writeTuplesToStdOut**

```
publicvoidwriteTuplesToStdOut(PrettyPrinterp)
```

Write the tuples in this cfg to STDOUT.

Parameters:

p - for pretty printing with indentation.

- **writeHirToStdOut**

```
publicvoidwriteHirToStdOut(PrettyPrinterp)
```

Write the hir instructions in this cfg to STDOUT.

Parameters:

p - for pretty printing with indentation.

- **writeLirToStdOut**

```
publicvoidwriteLirToStdOut(PrettyPrinterp)
```

Write the lir instructions in this cfg to STDOUT.

Parameters:

p - for pretty printing with indentation.

- **writeIntervalsToStdOut**

```
publicvoidwriteIntervalsToStdOut(PrettyPrinterp)
```


Write the intervals in this cfg to STDOUT.

Parameters:

p - for pretty printing with indentation.