**Context:**

*From http://padhye.org/vasco/apidocs/vasco/Context.html:*

**Context<M,N,A>:** A value-based context for a context-sensitive inter-procedural data flow analysis.

**Type Parameters:**

M - the type of a method

N - the type of a node in the CFG

A - the type of a data flow value

A value-based context is identified as a pair of a method and the data flow value at the entry of the method, for forward flows, or the data flow value at the exit of the method, for backward flows. Thus, if two distinct calls are made to a method and each call-site has the same data flow value then it is considered that the target of that call is the same context. This concept allows termination in the presence of recursion as the number of contexts is limited by the size of the lattice (which must be finite).

Each value context has its own work-list of CFG nodes to analyse, and the results of analysis are stored in a map from nodes to the data flow values before/after the node.

**Soot functions and their llvm conversions:**

|  |  |
| --- | --- |
| int | [**compareTo**](http://padhye.org/vasco/apidocs/vasco/Context.html#compareTo(vasco.Context))([Context](http://padhye.org/vasco/apidocs/vasco/Context.html)<[M](http://padhye.org/vasco/apidocs/vasco/Context.html),[N](http://padhye.org/vasco/apidocs/vasco/Context.html),[A](http://padhye.org/vasco/apidocs/vasco/Context.html)> other) |
|  |  |

bool Context<M,N,A>::operator==(const Context<M, N, A>& c) const {

return id == c.getId();

}

Compares two contexts by their globally unique IDs. This functionality is useful in the framework's internal methods where ordered processing of newer contexts first helps speed up certain operations.

|  |  |
| --- | --- |
| [DirectedGraph](http://www.sable.mcgill.ca/soot/doc/soot/toolkits/graph/DirectedGraph.html?is-external=true)<[N](http://padhye.org/vasco/apidocs/vasco/Context.html)> | [**getControlFlowGraph**](http://padhye.org/vasco/apidocs/vasco/Context.html#getControlFlowGraph())() |

std::vector<N> Context<M,N,A>::getControlFlowGraph(void) {

std::vector<N> wl;

for(N BB: depth\_first(&method->getEntryBlock())) {

wl.push\_back(BB);

}

std::reverse(wl.begin(), wl.end());

return wl;

}

Returns a reference to the control flow graph of this context's method.

**Returns:**

a reference to the control flow graph of this context's method

|  |  |
| --- | --- |
| static int | [**getCount**](http://padhye.org/vasco/apidocs/vasco/Context.html#getCount())() |

(Maybe)

Context<M,N,A>::Context(M method, bool reverse): method(method), analysed(false), is\_null(false), reverse(reverse) {

id = ++count;

}

Returns the total number of contexts created so far.

|  |  |
| --- | --- |
| [A](http://padhye.org/vasco/apidocs/vasco/Context.html) | [**getEntryValue**](http://padhye.org/vasco/apidocs/vasco/Context.html#getEntryValue())() |

A Context<M,N,A>::getEntryValue(void) {

return entry\_value;

}

Returns a reference to the data flow value at the method entry.

|  |  |
| --- | --- |
| [A](http://padhye.org/vasco/apidocs/vasco/Context.html) | [**getExitValue**](http://padhye.org/vasco/apidocs/vasco/Context.html#getExitValue())() |

A Context<M,N,A>::getExitValue(void) {

return exit\_value;

}

Returns a reference to the data flow value at the method exit.

|  |  |
| --- | --- |
| int | [**getId**](http://padhye.org/vasco/apidocs/vasco/Context.html#getId())() |

int Context<M,N,A>::getId(void) const {

return id;

}

Returns the globally unique identifier of this context.

|  |  |
| --- | --- |
| [M](http://padhye.org/vasco/apidocs/vasco/Context.html) | [**getMethod**](http://padhye.org/vasco/apidocs/vasco/Context.html#getMethod())() |

M Context<M,N,A>::getMethod(void) {

return method;

}

Returns a reference to this context's method.

|  |  |
| --- | --- |
| [A](http://padhye.org/vasco/apidocs/vasco/Context.html) | [**getValueAfter**](http://padhye.org/vasco/apidocs/vasco/Context.html#getValueAfter(N))([N](http://padhye.org/vasco/apidocs/vasco/Context.html) node) |

A Context<M,N,A>::getValueAfter(N \_node) {

return out\_values[\_node];

}

Gets the data flow value at the exit of the given node.

**Parameters:**

node - a node in the control flow graph

**Returns:**

the data flow value at the exit of the given node

|  |  |
| --- | --- |
| [A](http://padhye.org/vasco/apidocs/vasco/Context.html) | [**getValueBefore**](http://padhye.org/vasco/apidocs/vasco/Context.html#getValueBefore(N))([N](http://padhye.org/vasco/apidocs/vasco/Context.html) node) |

A Context<M,N,A>::getValueBefore(N \_node) {

return in\_values[\_node];

}

Gets the data flow value at the entry of the given node.

**Parameters:**

node - a node in the control flow graph

**Returns:**

the data flow value at the entry of the given node

|  |  |
| --- | --- |
| [NavigableSet](http://download.oracle.com/javase/6/docs/api/java/util/NavigableSet.html?is-external=true" \o "class or interface in java.util)<[N](http://padhye.org/vasco/apidocs/vasco/Context.html)> | [**getWorkList**](http://padhye.org/vasco/apidocs/vasco/Context.html#getWorkList())() |

std::vector<N> Context<M,N,A>::getWorklist(void) {

return worklist;

}

Returns a reference to this context's work-list.

|  |  |
| --- | --- |
| boolean | [**isAnalysed**](http://padhye.org/vasco/apidocs/vasco/Context.html#isAnalysed())() |

bool Context<M,N,A>::isAnalysed(void) {

return analysed;

}

Returns whether or not this context has been analysed at least once.

**Returns:**

true if the context has been analysed at least once, or false otherwise

|  |  |
| --- | --- |
| void | [**setEntryValue**](http://padhye.org/vasco/apidocs/vasco/Context.html#setEntryValue(A))([A](http://padhye.org/vasco/apidocs/vasco/Context.html) entryValue) |

void Context<M,N,A>::setEntryValue(A \_entry\_value) {

entry\_value = \_entry\_value;

return;

}

Sets the entry flow of this context.

**Parameters:**

entryValue - the new data flow value at the method entry

|  |  |
| --- | --- |
| void | [**setExitValue**](http://padhye.org/vasco/apidocs/vasco/Context.html#setExitValue(A))([A](http://padhye.org/vasco/apidocs/vasco/Context.html) exitValue) |

void Context<M,N,A>::setExitValue(A \_exit\_value) {

exit\_value = \_exit\_value;

return;

}

Sets the exit flow of this context.

**Parameters:**

exitValue - the new data flow value at the method exit

|  |  |
| --- | --- |
| void | [**setValueAfter**](http://padhye.org/vasco/apidocs/vasco/Context.html#setValueAfter(N,%20A))([N](http://padhye.org/vasco/apidocs/vasco/Context.html) node, [A](http://padhye.org/vasco/apidocs/vasco/Context.html) value) |

void Context<M,N,A>::setValueAfter(N \_node, A \_value) {

out\_values[\_node] = \_value;

return;

}

Sets the data flow value at the exit of the given node.

**Parameters:**

node - a node in the control flow graph

value - the new data flow at the node exit

|  |  |
| --- | --- |
| void | [**setValueBefore**](http://padhye.org/vasco/apidocs/vasco/Context.html#setValueBefore(N,%20A))([N](http://padhye.org/vasco/apidocs/vasco/Context.html) node, [A](http://padhye.org/vasco/apidocs/vasco/Context.html) value) |

void Context<M,N,A>::setValueBefore(N \_node, A \_value) {

in\_values[\_node] = \_value;

return;

}

Sets the data flow value at the entry of the given node.

**Parameters:**

node - a node in the control flow graph

value - the new data flow at the node entry

**Extra functions to be used:**

std::vector<N> Context<M,N,A>::getPredsOf(N \_node) {

std::vector<N> wl;

for(N Pred: predecessors(\_node)) {

wl.push\_back(Pred);

}

return wl;

}

std::vector<N> Context<M,N,A>::getSuccsOf(N \_node) {

std::vector<N> wl;

for(N Succ: successors(\_node)) {

wl.push\_back(Succ);

}

return wl;

}

std::vector<N> Context<M,N,A>::getTails(void) {

std::vector<N> wl;

for(N BB: depth\_first(&method->getEntryBlock())) {

for(auto &I: \*BB) {

if(llvm::isa<llvm::ReturnInst>(I)) {

wl.push\_back(BB);

break;

}

}

}

return wl;

}

bool Context<M,N,A>::isCall(N \_node) {

for(auto &I: \*\_node) {

if(llvm::isa<llvm::CallInst>(I)) {

return true;

}

}

return false;

}

void Context<M,N,A>::markAnalysed(void) {

analysed = true;

return;

}

void Context<M,N,A>::unmarkAnalysed(void) {

analysed = false;

return;

}

void Context<M,N,A>::addToWorklist(N \_node) {

worklist.push\_back(\_node);

return;

}

bool Context<M,N,A>::isEmptyWorklist(void) {

return worklist.empty();

}

N Context<M,N,A>::getAndPopWorklist(void) {

N \_node = worklist.back();

worklist.pop\_back();

return \_node;

}

N Context<M,N, A>::getLastWorklist(void) {

if(worklist.empty()) return NULL;

return worklist.back();

}