

Canal

Karel Klíč

September 26, 2012

Contents

1	Overview	9
1.1	Use cases	9
1.1.1	Analysis of program behaviour	9
1.1.2	Comparison with a specification	9
1.1.3	Conformance to environment constraints	9
2	Installation	11
2.1	Installation from source code on Fedora and Red Hat Enterprise Linux	11
I	Concepts	13
3	Preliminaries	15
4	LLVM	17
5	Abstract Interpretation	19
6	Memory Model	21
7	Array Abstract Domains	23
8	Structure Abstract Domain	25
9	Integer Abstract Domains	27
9.1	Integer Interval Domain $\mathcal{D}_i^\#$	27
9.2	Integer Bitfield Domain $\mathcal{D}_b^\#$	27
10	Abstract Domains for Floating-Point Numbers	29
11	Pointer Abstract Domains	31
12	Abstract Domain Combination	33

13 Wishlist	35
13.1 Reduced Product of Abstract Domains	35
13.2 Widening and Narrowing	35
13.3 String Abstract Domains	35
13.4 Trace Partitioning	35
13.5 Boolean Partitioning	35
13.6 Fixpoint Recalculation	35
13.7 Multi Threading	35
13.8 Symbolic Abstract Domains	36
13.9 Weakly-Relational Abstract Domains	36
13.10 Compositional Analysis	36
13.11 Parallelization	36
13.12 Custom Domains	36
 II Implementation	 37
14 Overview	39
15 Library Class Index	41
15.1 Class Hierarchy	41
15.2 Class List	42
16 Library Class Documentation	43
16.1 Canal::AccuracyDomain Class Reference	43
16.1.1 Detailed Description	44
16.1.2 Member Function Documentation	44
16.2 Canal::InterpreterBlock::BasicBlock Class Reference	45
16.3 Canal::Integer::Bitfield Class Reference	46
16.3.1 Detailed Description	48
16.3.2 Member Function Documentation	48
16.3.3 Member Data Documentation	50
16.4 Canal::Constructors Class Reference	51
16.5 Canal::Integer::Container Class Reference	52
16.5.1 Constructor & Destructor Documentation	54
16.5.2 Member Function Documentation	54
16.6 Canal::Domain Class Reference	57
16.6.1 Detailed Description	59

16.6.2 Member Function Documentation	59
16.7 Canal::Integer::Enumeration Class Reference	61
16.7.1 Member Function Documentation	63
16.8 Canal::Environment Class Reference	65
16.9 Canal::Array::ExactSize Class Reference	66
16.9.1 Detailed Description	68
16.9.2 Member Function Documentation	68
16.10 Canal::InterpreterBlock::Function Class Reference	69
16.10.1 Member Function Documentation	69
16.11 Canal::FunctionModel Class Reference	70
16.12 Canal::FunctionModelManager Class Reference	71
16.13 Canal::Pointer::InclusionBased Class Reference	72
16.13.1 Detailed Description	73
16.13.2 Constructor & Destructor Documentation	73
16.13.3 Member Function Documentation	74
16.13.4 Member Data Documentation	75
16.14 Canal::Array::Interface Class Reference	76
16.14.1 Member Function Documentation	76
16.15 Canal::InterpreterBlock::Interpreter Class Reference	78
16.15.1 Constructor & Destructor Documentation	78
16.16 Canal::Integer::Interval Class Reference	79
16.16.1 Detailed Description	81
16.16.2 Constructor & Destructor Documentation	81
16.16.3 Member Function Documentation	82
16.16.4 Member Data Documentation	83
16.17 Canal::Float::Interval Class Reference	84
16.17.1 Member Function Documentation	85
16.18 Canal::InterpreterBlock::Iterator Class Reference	86
16.18.1 Detailed Description	86
16.18.2 Member Function Documentation	87
16.18.3 Member Data Documentation	87
16.19 Canal::InterpreterBlock::IteratorCallback Class Reference	88
16.20 Canal::InterpreterBlock::Module Class Reference	89
16.21 Canal::Operations Class Reference	90
16.21.1 Detailed Description	93
16.21.2 Member Function Documentation	93

16.22	Canal::OperationsCallback Class Reference	95
16.22.1	Member Function Documentation	95
16.23	Canal::InterpreterBlock::OperationsCallback Class Reference	96
16.23.1	Member Function Documentation	96
16.24	Canal::APIntUtils::SCompare Struct Reference	97
16.25	Canal::Array::SingleItem Class Reference	98
16.25.1	Detailed Description	100
16.25.2	Member Function Documentation	100
16.25.3	Member Data Documentation	100
16.26	Canal::SlotTracker Class Reference	102
16.26.1	Detailed Description	103
16.26.2	Member Function Documentation	104
16.27	Canal::State Class Reference	105
16.27.1	Detailed Description	106
16.27.2	Member Function Documentation	106
16.27.3	Member Data Documentation	107
16.28	Canal::Structure Class Reference	108
16.28.1	Member Function Documentation	109
16.29	Canal::Pointer::Target Class Reference	110
16.29.1	Detailed Description	111
16.29.2	Constructor & Destructor Documentation	111
16.29.3	Member Function Documentation	111
16.29.4	Member Data Documentation	111
16.30	Canal::APIntUtils::UCompare Struct Reference	113
16.31	Canal::VariablePrecisionDomain Class Reference	114
16.31.1	Detailed Description	114
16.31.2	Member Function Documentation	114
17	Tool Class Index	115
17.1	Class Hierarchy	115
17.2	Class List	115
18	Tool Class Documentation	117
18.1	Arguments Class Reference	117
18.2	Command Class Reference	118
18.3	CommandBreak Class Reference	120
18.4	CommandCd Class Reference	122

18.5 CommandContinue Class Reference	124
18.6 CommandFile Class Reference	126
18.7 CommandFinish Class Reference	128
18.8 CommandHelp Class Reference	130
18.9 CommandInfo Class Reference	132
18.10CommandPrint Class Reference	134
18.11CommandPwd Class Reference	136
18.12CommandQuit Class Reference	138
18.13CommandRun Class Reference	140
18.14Commands Class Reference	142
18.15CommandShell Class Reference	144
18.16CommandShow Class Reference	146
18.17CommandStart Class Reference	148
18.18CommandStep Class Reference	150
18.19IteratorCallback Class Reference	152
18.20State Class Reference	153
19 Known Bugs	155
20 Wishlist	157
20.1 Callbacks Interface	157
20.2 Models	157
20.3 Support of Multiple Platforms	157
20.4 Automatic Tests	157
20.5 Graphical User Interface	157
Bibliography	159
Index	161

Chapter 1

Overview

For a sufficiently complex software system, its maintainability and extensibility is limited by our ability to understand and correctly approximate the behaviour of the system, trace the impact of system parts to each other, control the impact of modifications, ensure correctness of the critical parts, and fixing bugs before they cause serious consequences in production.

The maintainability and extensibility is affected by the programming language of the implementation. Efficient low-level languages such as C and C++ increase the complexity of the system by being closely aligned with hardware. Systems must handle memory management, operate on machine-dependent integers and floating point numbers, and cooperate with an environment with complex invariants and interdependencies.

Canal is a framework combining existing static analysis techniques in order to improve the maintainability, understanding, traceability and correctness of imperative programs in a coherent manner. The purpose of the framework is to make existing techniques accessible and evaluable, to support the implementation of new techniques, and to encourage experiments. Currently, techniques are often presented without proper experiments on real-world complex systems, or just with a proprietary implementation that cannot be investigated. As a consequence, actual applicability of many techniques for industrial use is unknown.

1.1 Use cases

1.1.1 Analysis of program behaviour

You can hook on the fixpoint of function calls to inspect the calculated abstract values. You can get abstract values of function call parameters.

1.1.2 Comparison with a specification

A set of pre- and post-conditions for functions, and variable-based or module-based automata. This can be defined for certain function or library, and library/function users are watched to conform to the specification.

1.1.3 Conformance to environment constraints

Double free, memory leaks, buffer overflow and underflow, division by zero, invalid access to memory, locking and concurrency errors, uncaught exceptions.

Chapter 2

Installation

This chapter provides instructions for installing Canal on supported platforms. Canal can be built and installed on most GNU/Linux operating systems.

2.1 Installation from source code on Fedora and Red Hat Enterprise Linux

Canal can be built, installed, and developed on a computer with the Red Hat Enterprise Linux 6 or Fedora 17 operating systems.

Prerequisites

Specific software packages are required by the build process and should be installed prior to building Canal:

- The `llvm-devel` and `clang` packages. On Red Hat Enterprise Linux, these packages can be obtained from Extra Packages for Enterprise Linux (EPEL) software repository.
- The `elfutils-devel` and `readline-devel` packages. These are needed for the command-line user interface tool.
- The `doxygen`, `graphviz`, and `texlive-latex` packages. These are needed to build the documentation. If not present, the documentation is not built.

Building and installing

Autotools (also known as GNU build system) are used to build Canal on these platforms. The first step is to configure the source code, telling Canal where various dependencies are located and where various files will be installed. To do so, go to the directory with the Canal source code tree and run the `configure` script:

```
./configure
```

It prints messages telling which features it checks and which dependencies it found.

Once the `configure` script has been run, you can compile Canal by running `make`:

```
make
```

After compiling Canal, you can verify your compiled program can operate well by running the test suite and seeing that all tests pass:

```
make check
```

Canal can be now installed. If it is going to be installed to system directories, special priviledges might be needed. To install Canal, run

```
make install
```

Part I

Concepts

Chapter 3

Preliminaries

As a preliminary step we shall define terms from the order theory. Detailed explanation can be found in [7] and [8].

A binary relation \sqsubseteq is *reflexive* on a set \mathcal{D} if every element is related to itself: $a \sqsubseteq a$ for all $a \in \mathcal{D}$. A binary relation \sqsubseteq is *antisymmetric* on a set \mathcal{D} if the following implication holds: $a \sqsubseteq b$ and $b \sqsubseteq a$ implies $a = b$. A binary relation \sqsubseteq is *transitive* on a set \mathcal{D} if whenever an element a is related to an element b , and b is in turn related to an element c , then a is also related to c : $a \sqsubseteq b$ and $b \sqsubseteq c$ implies $a \sqsubseteq c$.

A *partial order* \sqsubseteq is a binary relation on a set \mathcal{D} which is reflexive, antisymmetric and transitive. A *partial ordered set* or *poset* for short is an ordered pair $(\mathcal{D}, \sqsubseteq)$ of a set \mathcal{D} together with a partial ordering \sqsubseteq .

An element a in a poset $(\mathcal{D}, \sqsubseteq)$ is called *maximal* if it is not less than any other element in \mathcal{D} : $\nexists b \in \mathcal{D}, a \sqsubset b$. If there is an unique maximal element, we call it the *greatest element* and denote it by \top . Similarly, an element a in a poset $(\mathcal{D}, \sqsubseteq)$ is called *minimal* if it is not greater than any other element in \mathcal{D} : $\nexists b \in \mathcal{D}, b \sqsubset a$. If there is an unique minimal element, we call it the *least element* and denote it by \perp .

Let $(\mathcal{D}, \sqsubseteq)$ be a poset and $A \subseteq \mathcal{D}$. An element $u \in \mathcal{D}$ is an *upper bound* of A if $a \sqsubseteq u$ for all elements $a \in A$. The *least upper bound* or *lub* for short is an element x that is an upper bound on a subset A and is less than all other upper bounds on A ; such an element is denoted by $\sqcup A$. Similarly, an element $l \in \mathcal{D}$ is a *lower bound* of A if $l \sqsubseteq a$ for all elements $a \in A$. The *greatest lower bound* or *glb* for short is an element x that is a lower bound on a subset A and is greater than all other lower bounds on A ; such an element is denoted by $\sqcap A$.

A *lattice* $(\mathcal{D}, \sqsubseteq, \sqcup, \sqcap)$ is a partially ordered set in which any two elements $a, b \in \mathcal{D}$ have both a least upper bound, denoted by $a \sqcup b$, and a greatest lower bound, denoted by $a \sqcap b$. A *complete lattice* $(\mathcal{D}, \sqsubseteq, \sqcup, \sqcap, \perp, \top)$ is a partially ordered set in which every subset $A \subseteq \mathcal{D}$ has a least upper bound and a greatest lower bound. A complete lattice therefore has the greatest element \top defined as $\sqcup \mathcal{D}$, and the lowest element \perp defined as $\sqcap \mathcal{D}$.

A function $F \in \mathcal{D}_1 \rightarrow \mathcal{D}_2$ between two posets $(\mathcal{D}_1, \sqsubseteq_1)$ and $(\mathcal{D}_2, \sqsubseteq_2)$ is *monotonic* if $X \sqsubseteq_1 Y \implies F(X) \sqsubseteq_2 F(Y)$. A function $F \in \mathcal{D}_1 \rightarrow \mathcal{D}_2$ is *strict* if $F(\perp_1) = \perp_2$. A function $F \in \mathcal{D}_1 \rightarrow \mathcal{D}_2$ is *continuous* if it preserves the existing limits of increasing chains $(X_i)_{i \in I}$: $F(\sqcup_1 \{X_i \mid i \in I\}) = \sqcup_2 \{F(X_i) \mid i \in I\}$ whenever $\sqcup_1 \{X_i \mid i \in I\}$ exists.

A *fixpoint* of a function $F : \mathcal{D} \rightarrow \mathcal{D}$ on a poset $(\mathcal{D}, \sqsubseteq)$ is an element $x \in \mathcal{D}$ such that $F(x) = x$. A *prefixpoint* is an element $x \in \mathcal{D}$ such that $x \sqsubseteq F(x)$. A *postfixpoint* is an element $x \in \mathcal{D}$ such that $F(x) \sqsubseteq x$. A set of all fixpoints is denoted by $\text{fp}(F)$. A set of all prefixpoints is denoted by $\text{prefp}(F)$. A set of all postfixpoints is denoted by $\text{postfp}(F)$. The *least fixpoint* or *lfp* of a function F on a poset $(\mathcal{D}, \sqsubseteq)$ satisfies $\text{lfp} \in \text{fp}(F)$ and $\forall p \in \text{fp}(F) : \text{lfp} \sqsubseteq p$.

A *Galois connection* is a pair of two functions $\alpha : \mathcal{D}_1 \rightarrow \mathcal{D}_2$ and $\gamma : \mathcal{D}_2 \rightarrow \mathcal{D}_1$ on two preordered sets $(\mathcal{D}_1, \sqsubseteq_1)$ and $(\mathcal{D}_2, \sqsubseteq_2)$ iff $\forall d_1 \in \mathcal{D}_1, \forall d_2 \in \mathcal{D}_2 : \alpha(d_1) \sqsubseteq_2 d_2 \equiv d_1 \sqsubseteq_1 \gamma(d_2)$. It is denoted by

$$(\mathcal{D}_1, \sqsubseteq_1) \xrightleftharpoons[\alpha]{\gamma} (\mathcal{D}_2, \sqsubseteq_2).$$

Chapter 4

LLVM

Canal is built on the top of the LLVM [10] (Low-level Virtual Machine) compiler technology framework. Canal performs its static analysis over the LLVM intermediate representation, which is independent of source language and hide the complexity of target architecture. Canal is tested with C and C++ front-ends on 32-bit and 64-bit operating systems with little-endian memory layout, but it is expected that other source languages and platforms are supportable at low cost.

LLVM is suitable for efficient static analysis due to its design. Due to its type safety and Static Single Assignment (SSA) nature, most operations can be easily and precisely handled in static analysis. However, it is low enough level to support not only type conversion (creating a value of one data type from a value of another data type), but also type casting (changing the interpretation of the bit pattern representing a value from one type to another), pointer arithmetics, and manual memory management.

A subset of LLVM intermediate representation has been formalized in [17]. Figure 4.1 presents an updated abstract syntax that captures all attributes handled by Canal.

A module *mod* represents a translation unit of the input program. Most importantly, a module specifies list of *prod* that can be function declarations, function definitions, and global variables. It might also specify a target specific data layout string *layout* that specifies how data is to be laid out in memory, module-level inline assembler blocks *asm*, named types *namedt* that make the program shorter and easier to read, named metadata *namedm* that provide a collection of metadata, and aliases *alias* that act as a second name for the aliasee.

Types *typ* include arbitrary bit-width integers $isz \mid sz \in \mathbb{N}^*$, such as **i1**, **i8**, **i32**, **i64**. They also include floating point types *fp*. The **void** type does not represent any value and has no size. Pointers *typ** are used to specify memory locations. Arrays $[sz \times typ]$ have statically known size *sz*. Structures $\{\overline{typ_j}^j\}$ are defined as a list of types. Functions $typ \overline{typ_j}^j$ consist of a return type and a list of parameter types. Types can also be named by identifiers *id*, which is useful for the definition of recursive types. The **label** type represents code labels. The **metadata** type represents embedded metadata.

Modules	mod	$::=$	$\overline{layout} \overline{asm} \overline{namedt} \overline{namedm} \overline{alias} \overline{prod}$
Layouts	$layout$	$::=$	$\mathbf{bigendian} \mid \mathbf{littleendian} \mid \mathbf{ptr} \, sz \, align_{abi} \, align_{pref}$ $\mid \mathbf{int} \, sz \, align_{abi} \, align_{pref} \mid \mathbf{float} \, sz \, align_{abi} \, align_{pref}$ $\mid \mathbf{aggr} \, sz \, align_{abi} \, align_{pref} \mid \mathbf{vec} \, sz \, align_{abi} \, align_{pref}$ $\mid \mathbf{stack} \, sz \, align_{abi} \, align_{pref}$
Products	$prod$	$::=$	$id = \mathbf{global} \, typ \, const \, align \mid \mathbf{define} \, typ \, id(\overline{arg})\{\overline{b}\} \mid \mathbf{declare} \, typ \, id(\overline{arg})$
Floats	fp	$::=$	$\mathbf{half} \mid \mathbf{float} \mid \mathbf{double} \mid \mathbf{x86_fp80} \mid \mathbf{fp128} \mid \mathbf{ppc_fp128}$
Vec types	$vtyp$	$::=$	$fp \mid \mathbf{isz} \mid fp* \mid \mathbf{isz*}$
Types	typ	$::=$	$\mathbf{isz} \mid fp \mid \mathbf{void} \mid typ* \mid [sz \times typ] \mid [sz \times vtyp] \mid \{\overline{typ_j^j}\} \mid typ \, \overline{typ_j^j}$ $\mid id \mid \mathbf{label} \mid \mathbf{metadata}$
Values	val	$::=$	$id \mid \mathbf{cst}$
Binops	bop	$::=$	$\mathbf{add} \mid \mathbf{sub} \mid \mathbf{mul} \mid \mathbf{udiv} \mid \mathbf{sdiv} \mid \mathbf{urem} \mid \mathbf{srem} \mid \mathbf{shl} \mid \mathbf{lshr} \mid \mathbf{ashr}$ $\mid \mathbf{and} \mid \mathbf{or} \mid \mathbf{xor}$
Float ops	$fbop$	$::=$	$\mathbf{fadd} \mid \mathbf{fsub} \mid \mathbf{fmul} \mid \mathbf{fdiv} \mid \mathbf{frem}$
Extension	eop	$::=$	$\mathbf{zext} \mid \mathbf{sext} \mid \mathbf{fpext}$
Cast ops	cop	$::=$	$\mathbf{fptoui} \mid \mathbf{ptrtoint} \mid \mathbf{inttoptr} \mid \mathbf{bitcast}$
Trunc ops	$trop$	$::=$	$\mathbf{trunc}_{int} \mid \mathbf{trunc}_{fp}$
Constants	cst	$::=$	$\mathbf{isz} \, Int \mid fp \, Float \mid typ * id \mid (typ*) \, \mathbf{null} \mid typ \, \mathbf{zeroinitializer}$ $\mid typ[\overline{cst_j^j}] \mid \{\overline{cst_j^j}\} \mid typ \, \mathbf{undef} \mid bop \, cst_1 \, cst_2 \mid fbop \, cst_1 \, cst_2$ $\mid trop \, cst \, \mathbf{to} \, typ \mid eop \, cst \, \mathbf{to} \, typ \mid cop \, cst \, \mathbf{to} \, typ$ $\mid \mathbf{getelementptr} \, cst \, \overline{cst_j^j} \mid \mathbf{select} \, cst_0 \, cst_1 \, cst_2$ $\mid \mathbf{icmp} \, cond \, cst_1 \, cst_2 \mid \mathbf{fcmp} \, fcond \, cst_1 \, cst_2$
Blocks	b	$::=$	$l \, \overline{\phi} \, \overline{c} \, tmn$
ϕ nodes	ϕ	$::=$	$id = \mathbf{phi} \, typ \, [\overline{val_j, l_j}]^j$
Tmns	tmn	$::=$	$\mathbf{br} \, val \, l_1 \, l_2 \mid \mathbf{br} \, l \mid \mathbf{ret} \, typ \, val \mid \mathbf{ret} \, \mathbf{void} \mid \mathbf{unreachable}$
Commands	c	$::=$	$id = bop \, (\mathbf{int} \, sz) \, val_1 \, val_2 \mid id = fbop \, fp \, val_1 \, val_2$ $\mid id = \mathbf{store} \, typ \, val_1 \, val_2 \, align \mid id = \mathbf{malloc} \, typ \, val \, align \mid \mathbf{free} \, (typ*) \, val$ $\mid id = \mathbf{alloca} \, typ \, val \, align \mid id = trop \, typ_1 \, val \, \mathbf{to} \, typ_2$ $\mid id = eop \, typ_1 \, val \, \mathbf{to} \, typ_2 \mid id = cop \, typ_1 \, val \, \mathbf{to} \, typ_2$ $\mid id = \mathbf{select} \, val_0 \, typ \, val_1 \, val_2 \mid option \, id = \mathbf{call} \, typ_0 \, val_0 \, \overline{param}$ $\mid id = \mathbf{icmp} \, cond \, typ \, val_1 \, val_2 \mid id = \mathbf{fcmp} \, fcond \, fp \, val_1 \, val_2$ $\mid id = \mathbf{getelementptr} \, (typ*) \, val \, \overline{val_j^j} \mid id = \mathbf{load} \, (typ*) \, val \, align$ $\mid id = \mathbf{extractelement} \, [sz \times vtyp] \, val_1 \, val_2$ $\mid id = \mathbf{insertelement} \, [sz \times vtyp] \, val_1 \, val_2 \, val_3$

Figure 4.1: Abstract syntax for a subset of LLVM.

Chapter 5

Abstract Interpretation

Define: context sensitivity context sensitivity lattice (infinite height due to recursion) path sensitivity path sensitivity lattice (infinite height due to loops) flow sensitivity

Call graph Call stack Operational fixpoint calculation. Equation-based fixpoint calculation.

Abstract interpreter can be either operational or equation-based. Our interpreter is operational.

Chapter 6

Memory Model

Memory abstraction appeared in [13].

Our memory abstraction for abstract interpretation recognizes four kinds of memory:

Register-like stack memory This is function-level memory that is released automatically when function returns. We denote such a memory by LLVM-style names starting with the percent sign %. Memory either has a name (e.g. `%result`) or a number is generated to serve as a name (e.g. `%32` denotes thirty-second unnamed instruction call in a function).

Stack memory allocated by `alloca` This is also a function-level memory that is released automatically when function returns. The difference to register-like stack memory is that this memory is accessed by LLVM exclusively via pointers. We denote such a memory by names starting with `%^`. Every piece of memory has a name corresponding to the place where the memory has been allocated (`alloca` has been called). So if the memory has been allocated by an instruction call `%ptr = alloca i32, align 4`, it can be denoted by `%^ptr`.

Global variables Global variables are module-wise and are valid for the whole program run. We denote such a memory by LLVM-style names starting with `@`.

Heap memory Heap memory is also valid for the whole program run. We denote such a memory by names starting by `@^`. Every piece of memory has a name corresponding to the place where the memory has been allocated (`malloc` or similar function has been called). Name of the function is also included in the place name, so if a function `createString` contains an instruction call `%result = call i8* @malloc(i32 1)`, we can denote the memory allocated on this place by `@^createString:result`.

As it can be seen from the style of memory denotation, every piece of memory is associated with a place in the program. This means all operations affecting a memory block allocated at certain place forms a single abstract value. Context-sensitive abstract interpretation helps to increase the precision of this memory abstraction.

Chapter 7

Array Abstract Domains

Chapter 8

Structure Abstract Domain

Chapter 9

Integer Abstract Domains

9.1 Integer Interval Domain $\mathcal{D}_i^\#$

The interval domain was first presented in [1]. It was particularly well described in [9]. More precise machine integer interval domain appeared in [16].

$$D_i^\# \stackrel{\text{def}}{=} \{[l, h] \mid l, h \in \mathbb{Z} \cup \{\pm\infty\}\}$$

9.2 Integer Bitfield Domain $\mathcal{D}_b^\#$

Described in [16]. The domain associates two integers z and o to each variable. The integers represent bit masks for bits that can be set to 0 (zero) and to 1 (one).

Chapter 10

Abstract Domains for Floating-Point Numbers

Precise machine floating-point abstraction appeared in [16].

Chapter 11

Pointer Abstract Domains

Pointer can be casted to a number via the `ptrtoint` instruction. Usually, the resulting memory offset is used to achieve pointer arithmetics that are not available via `getelementptr` semantics.

Chapter 12

Abstract Domain Combination

Trees of abstract domains as done in ASTRÉE are described in [14].

Chapter 13

Wishlist

13.1 Reduced Product of Abstract Domains

Including cooperation.

13.2 Widening and Narrowing

Implement widening and narrowing operators for integers and other abstract domains as required.

13.3 String Abstract Domains

Implement abstract domains specific for C strings.

13.4 Trace Partitioning

Move context sensitivity to an abstract domain based on trace partitioning [11]. This change will allow us to introduce path-sensitivity just by extending this domain.

13.5 Boolean Partitioning

13.6 Fixpoint Recalculation

Allow to recompute fixpoint with a few variables changing their abstract domain layout.

13.7 Multi Threading

Multi-threading abstraction for Abstract Interpretation appeared in [15].

13.8 Symbolic Abstract Domains

Mine. Symbolic Methods to Enhance the Precision of Numerical Abstract Domains.

13.9 Weakly-Relational Abstract Domains

Implement weakly relational integer and floating-point abstract domains.

13.10 Compositional Analysis

Analyze functions, modules, or libraries separately, and merge the results afterwards. Theory can be found in [4] and [6].

13.11 Parallelization

Make abstract interpreter to use multiple threads for fixpoint calculation on symmetric multiprocessor systems. See [12].

13.12 Custom Domains

Active user/group (uid/gid) domain, priviledges domain. Opened files domain. Environment variables domain. File domain.

Part II

Implementation

Chapter 14

Overview

Canal can be used for a static analysis of real-world complex software systems written in efficient low-level languages C and C++. It uses the LLVM intermediate representation for the static analysis.

Canal is implemented in the C++ language as defined in the C++98 standard (ISO/IEC 14882:1998). It uses the C++ standard library and some additional libraries:

- LLVM core libraries. All versions from 2.8 up to 3.1 are supported.
- Clang compiler. Any version working with a supported version of LLVM should work.
- GNU readline. Any BSD-licensed reimplementations can be used as an alternative.
- elfutils. This library is used only on Linux-based operating systems.

Chapter 15

Library Class Index

15.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Canal::AccuracyDomain	43
Canal::Float::Interval	84
Canal::Integer::Bitfield	46
Canal::Integer::Container	52
Canal::Integer::Enumeration	61
Canal::Integer::Interval	79
Canal::Pointer::InclusionBased	72
Canal::InterpreterBlock::BasicBlock	45
Canal::Constructors	51
Canal::Domain	57
Canal::Array::ExactSize	66
Canal::Array::SingleItem	98
Canal::Float::Interval	84
Canal::Integer::Bitfield	46
Canal::Integer::Container	52
Canal::Integer::Enumeration	61
Canal::Integer::Interval	79
Canal::Pointer::InclusionBased	72
Canal::Structure	108
Canal::Environment	65
Canal::InterpreterBlock::Function	69
Canal::FunctionModel	70
Canal::FunctionModelManager	71
Canal::Array::Interface	76
Canal::Array::ExactSize	66
Canal::Array::SingleItem	98
Canal::Structure	108
Canal::InterpreterBlock::Interpreter	78
Canal::InterpreterBlock::Iterator	86
Canal::InterpreterBlock::IteratorCallback	88
Canal::InterpreterBlock::Module	89
Canal::Operations	90

Canal::OperationsCallback	95
Canal::InterpreterBlock::OperationsCallback	96
Canal::APIntUtils::SCompare	97
Canal::SlotTracker	102
Canal::State	105
Canal::Pointer::Target	110
Canal::APIntUtils::UCompare	113
Canal::VariablePrecisionDomain	114

15.2 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Canal::AccuracyDomain (Base class for abstract domains with the concept of value accuracy)	43
Canal::InterpreterBlock::BasicBlock	45
Canal::Integer::Bitfield	46
Canal::Constructors	51
Canal::Integer::Container	52
Canal::Domain (Base class for all abstract domains)	57
Canal::Integer::Enumeration	61
Canal::Environment	65
Canal::Array::ExactSize	66
Canal::InterpreterBlock::Function	69
Canal::FunctionModel	70
Canal::FunctionModelMANager	71
Canal::Pointer::InclusionBased (Inclusion-based flow-insensitive abstract pointer)	72
Canal::Array::Interface	76
Canal::InterpreterBlock::Interpreter	78
Canal::Integer::Interval (Abstracts integer values as a interval min - max)	79
Canal::Float::Interval	84
Canal::InterpreterBlock::Iterator	86
Canal::InterpreterBlock::IteratorCallback	88
Canal::InterpreterBlock::Module	89
Canal::Operations	90
Canal::OperationsCallback	95
Canal::InterpreterBlock::OperationsCallback	96
Canal::APIntUtils::SCompare	97
Canal::Array::SingleItem (This array type is very imprecise)	98
Canal::SlotTracker	102
Canal::State (Abstract memory state)	105
Canal::Structure	108
Canal::Pointer::Target (TODO: Pointers to functions)	110
Canal::APIntUtils::UCompare	113
Canal::VariablePrecisionDomain (Base class for abstract domains that can lower the precision and memory requirements on demand)	114

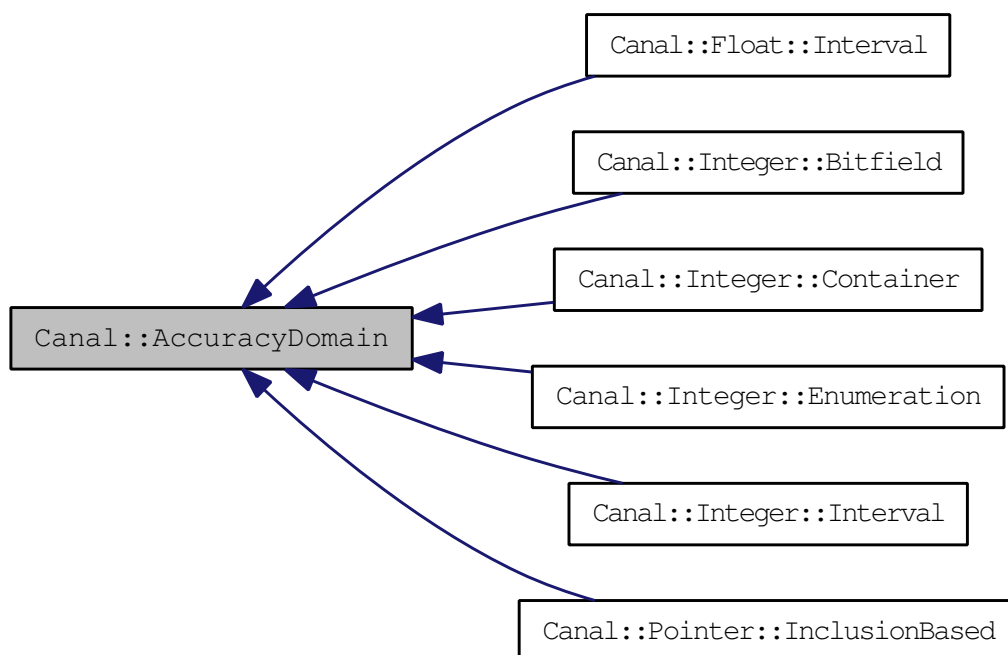
Chapter 16

Library Class Documentation

16.1 Canal::AccuracyDomain Class Reference

Base class for abstract domains with the concept of value accuracy.

`#include <Domain.h>`Inheritance diagram for Canal::AccuracyDomain:



Public Member Functions

- virtual float accuracy () const
- virtual bool isBottom () const
Is it the lowest value.
- virtual void setBottom ()

Set to the lowest value.

- virtual bool isTop () const

Is it the highest value.

- virtual void setTop ()

Set it to the top value of lattice.

16.1.1 Detailed Description

Base class for abstract domains with the concept of value accuracy.

16.1.2 Member Function Documentation

16.1.2.1 float Canal::AccuracyDomain::accuracy () const [virtual]

Get accuracy of the abstract value (0 - 1). In finite-height lattices, it is determined by the position of the value in the lattice.

Accuracy 0 means that the value represents all possible values (top). Accuracy 1 means that the value represents the most precise and exact value (bottom).

Reimplemented in Canal::Float::Interval, Canal::Integer::Bitfield, Canal::Integer::Container, Canal::Integer::Enumeration, Canal::Integer::Interval, and Canal::Pointer::InclusionBased.

The documentation for this class was generated from the following files:

- lib/Domain.h
- lib/Domain.cpp

16.2 Canal::InterpreterBlock::BasicBlock Class Reference

Collaboration diagram for Canal::InterpreterBlock::BasicBlock:



Public Member Functions

- **BasicBlock** (const llvm::BasicBlock &basicBlock, const Constructors &constructors)
- const llvm::BasicBlock & **getBasicBlock** () const
- llvm::BasicBlock::const_iterator **begin** () const
- llvm::BasicBlock::const_iterator **end** () const
- State & **getInputState** ()
- State & **getOutputState** ()

Protected Attributes

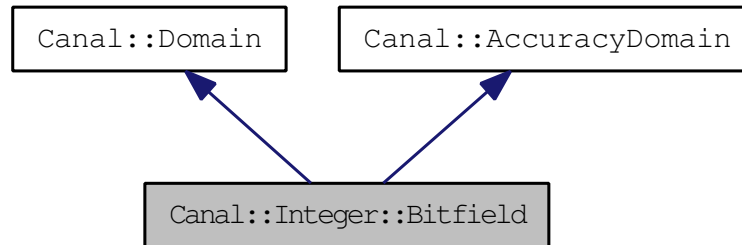
- `const llvm::BasicBlock & mBasicBlock`
- State `mInputState`
- State `mOutputState`

The documentation for this class was generated from the following files:

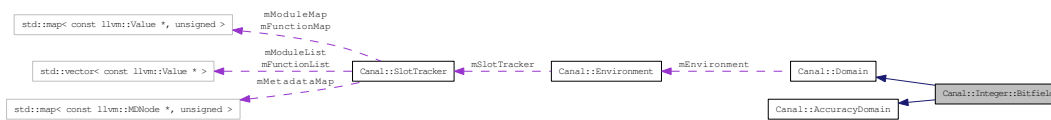
- lib/InterpreterBlockBasicBlock.h
- lib/InterpreterBlockBasicBlock.cpp

16.3 Canal::Integer::Bitfield Class Reference

#include <IntegerBitfield.h> Inheritance diagram for Canal::Integer::Bitfield:



Collaboration diagram for Canal::Integer::Bitfield:



Public Member Functions

- Bitfield (const Environment &environment, unsigned bitWidth)
Initializes to the lowest value.
- Bitfield (const Environment &environment, const llvm::APInt &number)
Initializes to the given value.
- unsigned getBitWidth () const
Return the number of bits of the represented number.
- int getBitValue (unsigned pos) const
- void setBitValue (unsigned pos, int value)
- bool signedMin (llvm::APInt &result) const
- bool signedMax (llvm::APInt &result) const
- bool unsignedMin (llvm::APInt &result) const
- bool unsignedMax (llvm::APInt &result) const
- bool isSingleValue () const
Does these bits represent single value?
- virtual Bitfield * clone () const
- virtual Bitfield * cloneCleaned () const
- virtual bool operator== (const Domain &value) const
Implementation of Domain::operator==().
- virtual void merge (const Domain &value)
Implementation of Domain::merge().

- virtual size_t memoryUsage () const
Implementation of Domain::memoryUsage().
- virtual std::string toString () const
Implementation of Domain::toString().
- virtual bool matchesString (const std::string &text, std::string &rational) const
Implementation of Domain::matchesString().
- virtual void add (const Domain &a, const Domain &b)
Implementation of Domain::add().
- virtual void sub (const Domain &a, const Domain &b)
Implementation of Domain::sub().
- virtual void mul (const Domain &a, const Domain &b)
Implementation of Domain::mul().
- virtual void udiv (const Domain &a, const Domain &b)
Implementation of Domain::udiv().
- virtual void sdiv (const Domain &a, const Domain &b)
Implementation of Domain::sdiv().
- virtual void urem (const Domain &a, const Domain &b)
Implementation of Domain::urem().
- virtual void srem (const Domain &a, const Domain &b)
Implementation of Domain::srem().
- virtual void shl (const Domain &a, const Domain &b)
Implementation of Domain::shl().
- virtual void lshr (const Domain &a, const Domain &b)
Implementation of Domain::lshr().
- virtual void ashr (const Domain &a, const Domain &b)
Implementation of Domain::ashr().
- virtual void and_ (const Domain &a, const Domain &b)
Implementation of Domain::and_().
- virtual void or_ (const Domain &a, const Domain &b)
Implementation of Domain::or_().
- virtual void xor_ (const Domain &a, const Domain &b)
Implementation of Domain::xor_().
- virtual void icmp (const Domain &a, const Domain &b, llvm::CmpInst::Predicate predicate)
Implementation of Domain::icmp().

- virtual float accuracy () const
Implementation of AccuracyDomain::accuracy().
- virtual bool isBottom () const
Implementation of AccuracyDomain::isBottom().
- virtual void setBottom ()
Implementation of AccuracyDomain::setBottom().
- virtual bool isTop () const
Implementation of AccuracyDomain::isTop().
- virtual void setTop ()
Implementation of AccuracyDomain::setTop().

Public Attributes

- llvm::APInt mZeroes
- llvm::APInt mOnes

16.3.1 Detailed Description

Abstracts integers as a bitfield.

For every bit, we have 4 possible states: mZeroes mOnes State ----- 0 0 Nothing was set to the bit (lowest lattice value - bottom) 1 0 The bit is set to 0 0 1 The bit is set to 1 1 1 The bit can be both 0 and 1 (highest lattice value - top)

16.3.2 Member Function Documentation

16.3.2.1 Bitfield * Canal::Integer::Bitfield::clone () const [virtual]

Implementation of Domain::clone(). Covariant return type.

Implements Canal::Domain.

16.3.2.2 Bitfield * Canal::Integer::Bitfield::cloneCleaned () const [virtual]

Implementation of Domain::cloneCleaned(). Covariant return type.

Implements Canal::Domain.

16.3.2.3 int Canal::Integer::Bitfield::getBitValue (unsigned pos) const

Returns 0 if the bit is known to be 0. Returns 1 if the bit is known to be 1. Returns -1 if the bit value is unknown. Returns 2 if the bit is either 1 or 0.

16.3.2.4 void Canal::Integer::Bitfield::setBitValue (unsigned pos, int value)

Sets the bit. If value is 0 or 1, the bit is set to represent exactly 0 or 1. If value is -1, the bit is set to represent unknown value. If value is 2, the bit is set to represent both 0 and 1.

16.3.2.5 bool Canal::Integer::Bitfield::signedMax (llvm::APInt & result) const

Highest signed number represented by this abstract domain.

Parameters:

result Filled by the maximum value if it is known. Otherwise, the value is undefined.

Returns:

True if the result is known and the parameter was set to correct value.

16.3.2.6 bool Canal::Integer::Bitfield::signedMin (llvm::APInt & result) const

Lowest signed number represented by this abstract domain.

Parameters:

result Filled by the minimum value if it is known. Otherwise, the value is undefined.

Returns:

True if the result is known and the parameter was set to correct value.

16.3.2.7 bool Canal::Integer::Bitfield::unsignedMax (llvm::APInt & result) const

Highest unsigned number represented by this abstract domain.

Parameters:

result Filled by the maximum value if it is known. Otherwise, the value is undefined.

Returns:

True if the result is known and the parameter was set to correct value.

16.3.2.8 bool Canal::Integer::Bitfield::unsignedMin (llvm::APInt & result) const

Lowest unsigned number represented by this abstract domain.

Parameters:

result Filled by the minimum value if it is known. Otherwise, the value is undefined.

Returns:

True if the result is known and the parameter was set to correct value.

16.3.3 Member Data Documentation

16.3.3.1 `llvm::APInt Canal::Integer::Bitfield::mOnes`

When a bit in `mOnes` is 1, the value is known to contain one at this position.

16.3.3.2 `llvm::APInt Canal::Integer::Bitfield::mZeroes`

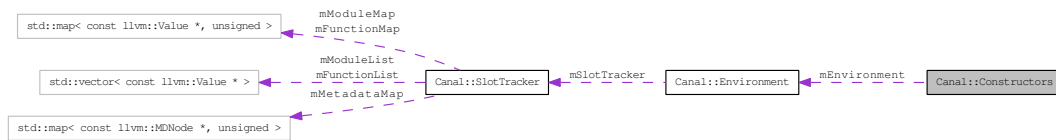
When a bit in `mZeroes` is 1, the value is known to contain zero at this position.

The documentation for this class was generated from the following files:

- `lib/IntegerBitfield.h`
- `lib/IntegerBitfield.cpp`

16.4 Canal::Constructors Class Reference

Collaboration diagram for Canal::Constructors:



Public Member Functions

- **Constructors** (const Environment &environment)
- Domain * **create** (const llvm::Type &type) const
- Domain * **create** (const llvm::Constant &value) const

Static Public Member Functions

- static const llvm::fltSemantics * **getFloatingPointSemantics** (const llvm::Type &type)

Protected Attributes

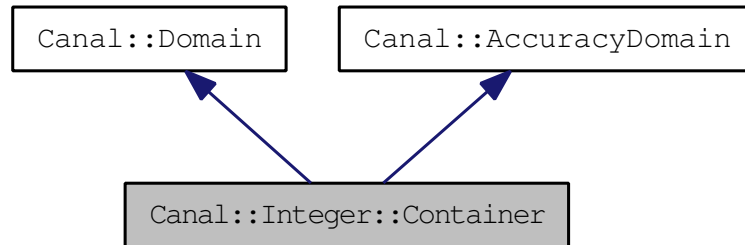
- const Environment & **mEnvironment**

The documentation for this class was generated from the following files:

- lib/Constructors.h
- lib/Constructors.cpp

16.5 Canal::Integer::Container Class Reference

Inheritance diagram for Canal::Integer::Container:



Collaboration diagram for Canal::Integer::Container:



Public Member Functions

- **Container** (const Environment &environment, unsigned bitWidth)
- Container (const Environment &environment, const llvm::APInt &number)
- Container (const Container &container)

Copy constructor. Creates independent copy of the container.

- virtual ~Container ()

Destructor: Deletes the contents of the container.

- unsigned **getBitWidth** () const
- Bitfield & **getBitfield** ()
- const Bitfield & **getBitfield** () const
- Enumeration & **getEnumeration** ()
- const Enumeration & **getEnumeration** () const
- Interval & **getInterval** ()
- const Interval & **getInterval** () const
- bool signedMin (Ilvm::APInt &result) const
- bool signedMax (Ilvm::APInt &result) const
- bool unsignedMin (Ilvm::APInt &result) const
- bool unsignedMax (Ilvm::APInt &result) const
- virtual Container * clone () const
- virtual Container * cloneCleaned () const
- virtual bool operator== (const Domain &value) const

Implementation of Domain::operator==().

- virtual void merge (const Domain &value)

Implementation of `Domain::merge()`.

- virtual size_t memoryUsage () const
Implementation of Domain::memoryUsage().
- virtual std::string toString () const
Implementation of Domain::toString().
- virtual bool matchesString (const std::string &text, std::string &rational) const
- virtual void add (const Domain &a, const Domain &b)
Implementation of Domain::add().
- virtual void sub (const Domain &a, const Domain &b)
Implementation of Domain::sub().
- virtual void mul (const Domain &a, const Domain &b)
Implementation of Domain::mul().
- virtual void udiv (const Domain &a, const Domain &b)
Implementation of Domain::udiv().
- virtual void sdiv (const Domain &a, const Domain &b)
Implementation of Domain::sdiv().
- virtual void urem (const Domain &a, const Domain &b)
Implementation of Domain::urem().
- virtual void srem (const Domain &a, const Domain &b)
Implementation of Domain::srem().
- virtual void shl (const Domain &a, const Domain &b)
Implementation of Domain::shl().
- virtual void lshr (const Domain &a, const Domain &b)
Implementation of Domain::lshr().
- virtual void ashr (const Domain &a, const Domain &b)
Implementation of Domain::ashr().
- virtual void and_ (const Domain &a, const Domain &b)
Implementation of Domain::and_().
- virtual void or_ (const Domain &a, const Domain &b)
Implementation of Domain::or_().
- virtual void xor_ (const Domain &a, const Domain &b)
Implementation of Domain::xor_().
- virtual void icmp (const Domain &a, const Domain &b, llvm::CmpInst::Predicate predicate)
Implementation of Domain::icmp().

- virtual void fcmp (const Domain &a, const Domain &b, llvm::CmpInst::Predicate predicate)
Implementation of Domain::fcmp().
- virtual float accuracy () const
Implementation of AccuracyDomain::accuracy().
- virtual bool isBottom () const
Implementation of AccuracyDomain::isBottom().
- virtual void setBottom ()
Implementation of AccuracyDomain::setBottom().
- virtual bool isTop () const
Implementation of AccuracyDomain::isTop().
- virtual void setTop ()
Implementation of AccuracyDomain::setTop().
- bool isSingleValue () const
Find out whether all representations contain only single value.

Public Attributes

- std::vector< Domain * > **mValues**

16.5.1 Constructor & Destructor Documentation

16.5.1.1 Canal::Integer::Container::Container (const Environment & *environment*, const llvm::APInt & *number*)

Creates a new container with an initial value. Signedness, number of bits is taken from the provided number.

16.5.2 Member Function Documentation

16.5.2.1 Container * Canal::Integer::Container::clone () const [virtual]

Implementation of Domain::clone(). Covariant return type.

Implements Canal::Domain.

16.5.2.2 Container * Canal::Integer::Container::cloneCleaned () const [virtual]

Implementation of Domain::cloneCleaned(). Covariant return type.

Implements Canal::Domain.

16.5.2.3 **bool Canal::Integer::Container::matchesString (const std::string & *text*, std::string & *rationale*) const [virtual]**

Implementation of Domain::matchesString(). Examples: integer enumeration -8 integer enumeration -10 2 4 6 8 interval -10 8

Implements Canal::Domain.

16.5.2.4 **bool Canal::Integer::Container::signedMax (llvm::APInt & *result*) const**

Highest signed number represented by this container. Uses the abstract domain (enum, interval, bits) with highest precision.

Parameters:

result Filled by the maximum value if it is known. Otherwise, the value is undefined.

Returns:

True if the result is known and the parameter was set to correct value.

16.5.2.5 **bool Canal::Integer::Container::signedMin (llvm::APInt & *result*) const**

Lowest signed number represented by this container. Uses the abstract domain (enum, interval, bits) with highest precision.

Parameters:

result Filled by the minimum value if it is known. Otherwise, the value is undefined.

Returns:

True if the result is known and the parameter was set to correct value.

16.5.2.6 **bool Canal::Integer::Container::unsignedMax (llvm::APInt & *result*) const**

Highest unsigned number represented by this container. Uses the abstract domain (enum, interval, bits) with highest precision.

Parameters:

result Filled by the maximum value if it is known. Otherwise, the value is undefined.

Returns:

True if the result is known and the parameter was set to correct value.

16.5.2.7 **bool Canal::Integer::Container::unsignedMin (llvm::APInt & *result*) const**

Lowest unsigned number represented by this container. Uses the abstract domain (enum, interval, bits) with highest precision.

Parameters:

result Filled by the minimum value if it is known. Otherwise, the value is undefined.

Returns:

True if the result is known and the parameter was set to correct value.

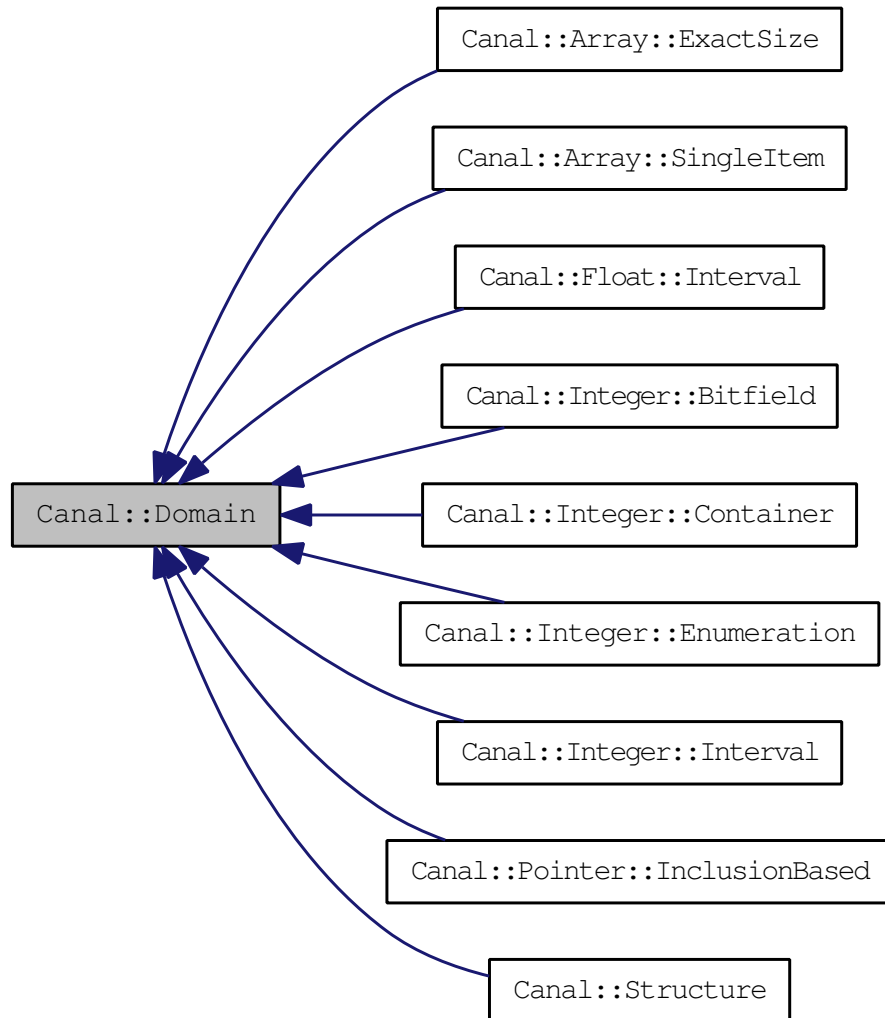
The documentation for this class was generated from the following files:

- lib/IntegerContainer.h
- lib/IntegerContainer.cpp

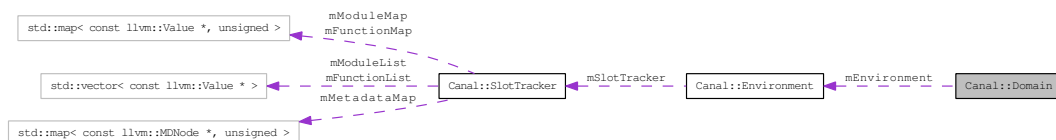
16.6 Canal::Domain Class Reference

Base class for all abstract domains.

#include <Domain.h> Inheritance diagram for Canal::Domain:



Collaboration diagram for Canal::Domain:



Public Types

- typedef void(Domain::* **CastOperation**)(const Domain &)

- typedef void(Domain::* **BinaryOperation**)(const Domain &, const Domain &)
- typedef void(Domain::* **CmpOperation**)(const Domain &, const Domain &, llvm::CmpInst::Predicate predicate)

Public Member Functions

- Domain (const Environment &environment)
Standard constructor.
- virtual ~Domain ()
Virtual destructor.
- virtual Domain * clone () const =0
Create a copy of this value.
- virtual Domain * cloneCleaned () const =0
- virtual bool operator== (const Domain &value) const =0
- virtual bool operator!= (const Domain &value) const
Inequality is implemented by calling the equality operator.
- virtual void merge (const Domain &value)
Merge another value into this one.
- virtual size_t memoryUsage () const =0
Get memory usage (used byte count) of this abstract value.
- virtual std::string toString () const =0
Create a string representation of the abstract value.
- virtual bool matchesString (const std::string &text, std::string &rationale) const =0
- virtual void add (const Domain &a, const Domain &b)
Implementation of instructions operating on values.
- virtual void **fadd** (const Domain &a, const Domain &b)
- virtual void **sub** (const Domain &a, const Domain &b)
- virtual void **fsub** (const Domain &a, const Domain &b)
- virtual void **mul** (const Domain &a, const Domain &b)
- virtual void **fmul** (const Domain &a, const Domain &b)
- virtual void udiv (const Domain &a, const Domain &b)
Unsigned division.
- virtual void sdiv (const Domain &a, const Domain &b)
Signed division.
- virtual void fdiv (const Domain &a, const Domain &b)
Floating point division.
- virtual void **urem** (const Domain &a, const Domain &b)
- virtual void **srem** (const Domain &a, const Domain &b)
- virtual void **frem** (const Domain &a, const Domain &b)

- virtual void **shl** (const Domain &a, const Domain &b)
- virtual void **lshr** (const Domain &a, const Domain &b)
- virtual void **ashr** (const Domain &a, const Domain &b)
- virtual void **and_** (const Domain &a, const Domain &b)
- virtual void **or_** (const Domain &a, const Domain &b)
- virtual void **xor_** (const Domain &a, const Domain &b)
- virtual void **icmp** (const Domain &a, const Domain &b, llvm::CmpInst::Predicate predicate)
- virtual void **fcmp** (const Domain &a, const Domain &b, llvm::CmpInst::Predicate predicate)
- virtual void **trunc** (const Domain &value)
- virtual void **zext** (const Domain &value)
- virtual void **sext** (const Domain &value)
- virtual void **fptrunc** (const Domain &value)
- virtual void **fpext** (const Domain &value)
- virtual void **fpoui** (const Domain &value)
- virtual void **fpotsi** (const Domain &value)
- virtual void **uitofp** (const Domain &value)
- virtual void **sitofp** (const Domain &value)

Public Attributes

- const Environment & **mEnvironment**

16.6.1 Detailed Description

Base class for all abstract domains.

16.6.2 Member Function Documentation

16.6.2.1 void Canal::Domain::add (const Domain & *a*, const Domain & *b*) [virtual]

Implementation of instructions operating on values. Load the abstract value state from a string representation.

Parameters:

text The textual representation. It must not contain any text that does not belong to this abstract value state.

Returns:

True if the text has been successfully parsed and the state has been set from the text. False otherwise.

16.6.2.2 virtual Domain* Canal::Domain::cloneCleaned () const [pure virtual]

This is used to obtain instance of the value type and to get an empty value at the same time.

Implemented in Canal::Array::ExactSize, Canal::Array::SingleItem, Canal::Float::Interval, Canal::Integer::Bitfield, Canal::Integer::Container, Canal::Integer::Enumeration, Canal::Integer::Interval, Canal::Pointer::InclusionBased, and Canal::Structure.

16.6.2.3 virtual bool Canal::Domain::matchesString (const std::string & *text*, std::string & *rationale*) const [pure virtual]

Checks if the abstract value internal state matches the text description. Full coverage of the state is not expected, the text can contain just partial information.

Implemented in Canal::Array::ExactSize, Canal::Array::SingleItem, Canal::Float::Interval, Canal::Integer::Bitfield, Canal::Integer::Container, Canal::Integer::Enumeration, Canal::Integer::Interval, Canal::Pointer::InclusionBased, and Canal::Structure.

16.6.2.4 virtual bool Canal::Domain::operator== (const Domain & *value*) const [pure virtual]

Implementing this is mandatory. Values are compared while computing the fixed point.

16.6.2.5 virtual std::string Canal::Domain::toString () const [pure virtual]

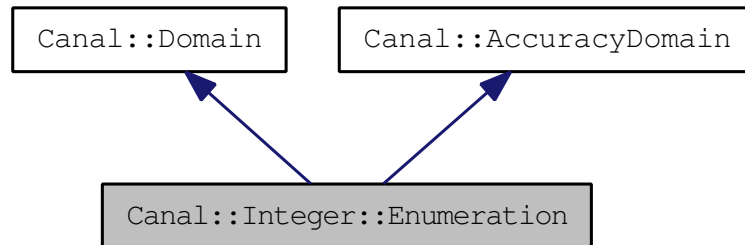
Create a string representation of the abstract value. An idea for different memory interpretation. virtual Domain *castTo(const llvm::Type *itemType, int offset) const = 0;

Implemented in Canal::Array::ExactSize, Canal::Array::SingleItem, Canal::Float::Interval, Canal::Integer::Bitfield, Canal::Integer::Container, Canal::Integer::Enumeration, Canal::Integer::Interval, Canal::Pointer::InclusionBased, and Canal::Structure.

The documentation for this class was generated from the following files:

- lib/Domain.h
- lib/Domain.cpp

Inheritance diagram for Canal::Integer::Enumeration:



```

graph LR
    subgraph STL_Map_Value_Unsigned [std::map<const llvm::Value *, unsigned>]
        direction TB
        mModuleMap[mModuleMap]
        mFunctionMap[mFunctionMap]
    end

    subgraph STL_Vector_Value [std::vector<const llvm::Value *>]
        direction TB
        mModuleList[mModuleList]
        mFunctionList[mFunctionList]
        mMetadataMap[mMetadataMap]
    end

    subgraph STL_Map_Module_Unsigned [std::map<const llvm::Module *, unsigned>]
        direction TB
        mModuleMap
        mFunctionMap
    end

    subgraph Canali_Environment [Canali::Environment]
        direction TB
        mEnvironment[mEnvironment]
    end

    subgraph Canali_Domain [Canali::Domain]
        direction TB
        mValues[mValues]
    end

    subgraph Canali_Integer_Enumeration [Canali::Integer::Enumeration]
        direction TB
        mValues
    end

    subgraph STL_Set_APInt [std::set<llvm::APInt, UCompare>]
        direction TB
        mValues
    end

    STL_Map_Value_Unsigned --> mModuleMap
    STL_Map_Value_Unsigned --> mFunctionMap
    STL_Vector_Value --> mModuleList
    STL_Vector_Value --> mFunctionList
    STL_Map_Module_Unsigned --> mModuleMap
    STL_Map_Module_Unsigned --> mFunctionMap
    Canali_Environment --> mEnvironment
    Canali_Domain --> mValues
    Canali_Integer_Enumeration --> mValues
    STL_Set_APInt --> mValues
    mModuleMap -.-> mModuleList
    mFunctionMap -.-> mFunctionList
    mModuleList -.-> mMetadataMap
    mFunctionList -.-> mMetadataMap
    mEnvironment -.-> Canali_Domain
    mValues -.-> Canali_Integer_Enumeration

```

- Enumeration (const Environment &environment, unsigned bitWidth)
Initializes to the lowest value.
- Enumeration (const Environment &environment, const llvm::APInt &number)
Initializes to the given value.
- unsigned **getBitWidth** () const
- bool signedMin (llvm::APInt &result) const
- bool signedMax (llvm::APInt &result) const
- bool unsignedMin (llvm::APInt &result) const
- bool unsignedMax (llvm::APInt &result) const
- bool isSingleValue () const
Does this enumeration represent single value?
- virtual Enumeration * clone () const
- virtual Enumeration * cloneCleaned () const
- virtual bool operator== (const Domain &value) const
Implementation of Domain::operator==(.).
- virtual void merge (const Domain &value)
Implementation of Domain::merge().
- virtual size_t memoryUsage () const
Implementation of Domain::memoryUsage().

- virtual std::string toString () const
Implementation of Domain::toString().
- virtual bool matchesString (const std::string &text, std::string &rationale) const
- virtual void add (const Domain &a, const Domain &b)
Implementation of Domain::add().
- virtual void sub (const Domain &a, const Domain &b)
Implementation of Domain::sub().
- virtual void mul (const Domain &a, const Domain &b)
Implementation of Domain::mul().
- virtual void udiv (const Domain &a, const Domain &b)
Implementation of Domain::udiv().
- virtual void sdiv (const Domain &a, const Domain &b)
Implementation of Domain::sdiv().
- virtual void urem (const Domain &a, const Domain &b)
Implementation of Domain::urem().
- virtual void srem (const Domain &a, const Domain &b)
Implementation of Domain::srem().
- virtual void shl (const Domain &a, const Domain &b)
Implementation of Domain::shl().
- virtual void lshr (const Domain &a, const Domain &b)
Implementation of Domain::lshr().
- virtual void ashr (const Domain &a, const Domain &b)
Implementation of Domain::ashr().
- virtual void and_ (const Domain &a, const Domain &b)
Implementation of Domain::and_().
- virtual void or_ (const Domain &a, const Domain &b)
Implementation of Domain::or_().
- virtual void xor_ (const Domain &a, const Domain &b)
Implementation of Domain::xor_().
- virtual void icmp (const Domain &a, const Domain &b, llvm::CmpInst::Predicate predicate)
Implementation of Domain::icmp().
- virtual void fcmp (const Domain &a, const Domain &b, llvm::CmpInst::Predicate predicate)
Implementation of Domain::fcmp().
- virtual float accuracy () const

Implementation of AccuracyDomain::accuracy().

- virtual bool isBottom () const

Implementation of AccuracyDomain::isBottom().

- virtual void setBottom ()

Implementation of AccuracyDomain::setBottom().

- virtual bool isTop () const

Implementation of AccuracyDomain::isTop().

- virtual void setTop ()

Implementation of AccuracyDomain::setTop().

Public Attributes

- APIntUtils::USet **mValues**
- bool **mTop**
- unsigned **mBitWidth**

Static Public Attributes

- static const unsigned int **mMaxSize** = 40

Protected Member Functions

- void **applyOperation** (const Domain &a, const Domain &b, APIntUtils::Operation operation1, APIntUtils::OperationWithOverflow operation2)

16.7.1 Member Function Documentation

16.7.1.1 Enumeration * Canal::Integer::Enumeration::clone () const [virtual]

Implementation of Domain::clone(). Covariant return type.

Implements Canal::Domain.

16.7.1.2 Enumeration * Canal::Integer::Enumeration::cloneCleaned () const [virtual]

Implementation of Domain::cloneCleaned(). Covariant return type.

Implements Canal::Domain.

16.7.1.3 bool Canal::Integer::Enumeration::matchesString (const std::string &text, std::string &rationale) const [virtual]

Implementation of Domain::matchesString(). Examples of allowed input: enumeration 3 -4 5 enumeration 0xff 0xfa enumeration 0x1000000F 0xABABABAB enumeration top enumeration empty enumeration empty

Implements Canal::Domain.

16.7.1.4 **bool Canal::Integer::Enumeration::signedMax (llvm::APInt & *result*) const**

Highest signed number represented by this abstract domain.

Parameters:

result Filled by the maximum value if it is known. Otherwise, the value is undefined.

Returns:

True if the result is known and the parameter was set to correct value.

16.7.1.5 **bool Canal::Integer::Enumeration::signedMin (llvm::APInt & *result*) const**

Lowest signed number represented by this abstract domain.

Parameters:

result Filled by the minimum value if it is known. Otherwise, the value is undefined.

Returns:

True if the result is known and the parameter was set to correct value.

16.7.1.6 **bool Canal::Integer::Enumeration::unsignedMax (llvm::APInt & *result*) const**

Highest unsigned number represented by this abstract domain.

Parameters:

result Filled by the maximum value if it is known. Otherwise, the value is undefined.

Returns:

True if the result is known and the parameter was set to correct value.

16.7.1.7 **bool Canal::Integer::Enumeration::unsignedMin (llvm::APInt & *result*) const**

Lowest unsigned number represented by this abstract domain.

Parameters:

result Filled by the minimum value if it is known. Otherwise, the value is undefined.

Returns:

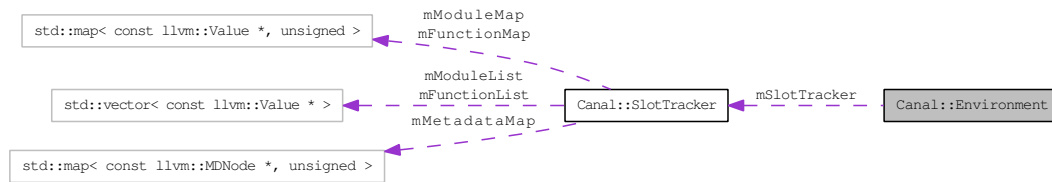
True if the result is known and the parameter was set to correct value.

The documentation for this class was generated from the following files:

- lib/IntegerEnumeration.h
- lib/IntegerEnumeration.cpp

16.8 Canal::Environment Class Reference

Collaboration diagram for Canal::Environment:



Public Member Functions

- **Environment** (const llvm::Module *module)
- llvm::LLVMContext & **getContext** () const
- const llvm::Module & **getModule** () const
- const llvm::TargetData & **getTargetData** () const
- SlotTracker & **getSlotTracker** () const

Protected Attributes

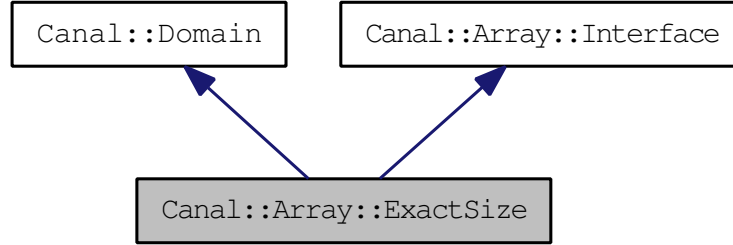
- const llvm::Module * **mModule**
- llvm::TargetData **mTargetData**
- SlotTracker **mSlotTracker**

The documentation for this class was generated from the following files:

- lib/Environment.h
- lib/Environment.cpp

16.9 Canal::Array::ExactSize Class Reference

#include <ArrayExactSize.h> Inheritance diagram for Canal::Array::ExactSize:



Collaboration diagram for Canal::Array::ExactSize:



Public Member Functions

- **ExactSize** (const Environment &environment)
- **ExactSize** (const ExactSize &exactSize)
- **size_t size** () const
- virtual ExactSize * clone () const
- virtual ExactSize * cloneCleaned () const
- virtual bool operator== (const Domain &value) const

Implementation of Domain::operator==().

- virtual void merge (const Domain &value)

Implementation of Domain::merge().

- virtual size_t memoryUsage () const

Implementation of Domain::memoryUsage().

- virtual std::string toString () const

Implementation of Domain::toString().

- virtual bool matchesString (const std::string &text, std::string &rationale) const

Implementation of Domain::matchesString().

- virtual void add (const Domain &a, const Domain &b)

Implementation of Domain::add().

- virtual void fadd (const Domain &a, const Domain &b)

Implementation of Domain::fadd().

- virtual void sub (const Domain &a, const Domain &b)
Implementation of Domain::sub().
- virtual void fsub (const Domain &a, const Domain &b)
Implementation of Domain::fsub().
- virtual void mul (const Domain &a, const Domain &b)
Implementation of Domain::mul().
- virtual void fmul (const Domain &a, const Domain &b)
Implementation of Domain::fmul().
- virtual void udiv (const Domain &a, const Domain &b)
Implementation of Domain::udiv().
- virtual void sdiv (const Domain &a, const Domain &b)
Implementation of Domain::sdiv().
- virtual void fdiv (const Domain &a, const Domain &b)
Implementation of Domain::fdiv().
- virtual void urem (const Domain &a, const Domain &b)
Implementation of Domain::urem().
- virtual void srem (const Domain &a, const Domain &b)
Implementation of Domain::srem().
- virtual void frem (const Domain &a, const Domain &b)
Implementation of Domain::frem().
- virtual void shl (const Domain &a, const Domain &b)
Implementation of Domain::shl().
- virtual void lshr (const Domain &a, const Domain &b)
Implementation of Domain::lshr().
- virtual void ashr (const Domain &a, const Domain &b)
Implementation of Domain::ashr().
- virtual void and_ (const Domain &a, const Domain &b)
Implementation of Domain::and_().
- virtual void or_ (const Domain &a, const Domain &b)
Implementation of Domain::or_().
- virtual void xor_ (const Domain &a, const Domain &b)
Implementation of Domain::xor_().
- virtual void icmp (const Domain &a, const Domain &b, llvm::CmpInst::Predicate predicate)
Implementation of Domain::icmp().

- virtual void fcmp (const Domain &a, const Domain &b, llvm::CmpInst::Predicate predicate)
Implementation of Domain::fcmp().
- virtual std::vector< Domain * > getItem (const Domain &offset) const
Implementation of Array::Interface::getItem().
- virtual Domain * getItem (uint64_t offset) const
Implementation of Array::Interface::getItem().
- virtual void setItem (const Domain &offset, const Domain &value)
Implementation of Array::Interface::setItem().
- virtual void setItem (uint64_t offset, const Domain &value)
Implementation of Array::Interface::setItem().

Public Attributes

- std::vector< Domain * > **mValues**

16.9.1 Detailed Description

Array with exact size and limited length. It keeps all array members separately, not losing precision at all.

16.9.2 Member Function Documentation

16.9.2.1 ExactSize * Canal::Array::ExactSize::clone () const [virtual]

Implementation of Domain::clone(). Covariant return type.

Implements Canal::Domain.

16.9.2.2 ExactSize * Canal::Array::ExactSize::cloneCleaned () const [virtual]

Implementation of Domain::cloneCleaned(). Covariant return type.

Implements Canal::Domain.

The documentation for this class was generated from the following files:

- lib/ArrayExactSize.h
- lib/ArrayExactSize.cpp

16.10 Canal::InterpreterBlock::Function Class Reference

Collaboration diagram for Canal::InterpreterBlock::Function:



Public Member Functions

- **Function** (const llvm::Function &function, const Constructors &constructors)
- const llvm::Function & **getFunction** () const
- const llvm::BasicBlock & **getEntryBlock** () const
- BasicBlock & **getBasicBlock** (const llvm::BasicBlock &llvmBasicBlock)
- std::vector< BasicBlock * >::const_iterator **begin** () const
- std::vector< BasicBlock * >::const_iterator **end** () const
- State & **getInputState** ()
- const State & **getInputState** () const
- const State & **getOutputState** () const
- llvm::StringRef **getName** () const
- void updateBasicBlockInputState (BasicBlock &basicBlock)
- void updateOutputState ()

Update function output state from basic block output states.

Protected Attributes

- const llvm::Function & **mFunction**
- std::vector< BasicBlock * > **mBasicBlocks**
- State **mInputState**
- State **mOutputState**

16.10.1 Member Function Documentation

16.10.1.1 void Canal::InterpreterBlock::Function::updateBasicBlockInputState (BasicBlock & basicBlock)

Update basic block input state from its predecessors and function input state.

Parameters:

basicBlock Must be a member of this function. Its input state is updated.

The documentation for this class was generated from the following files:

- lib/InterpreterBlockFunction.h
- lib/InterpreterBlockFunction.cpp

16.11 Canal::FunctionModel Class Reference

Public Member Functions

- bool **canHandle** (llvm::Function *function, bool implementationAvailable)
- void **handle** (llvm::Function *function, State &state)

The documentation for this class was generated from the following file:

- lib/FunctionModel.h

16.12 Canal::FunctionModelManager Class Reference

Public Member Functions

- bool **canHandle** (llvm::Function *function, bool implementationAvailable)
- void **handle** (llvm::Function *function, State &state)

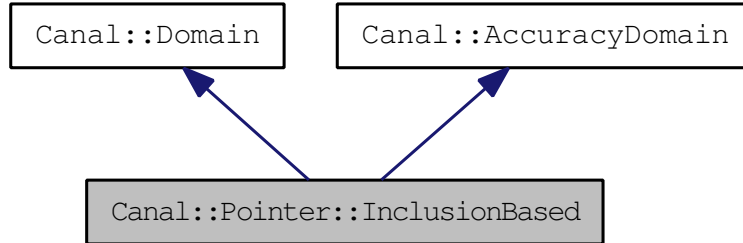
The documentation for this class was generated from the following file:

- lib/FunctionModelManager.h

16.13 Canal::Pointer::InclusionBased Class Reference

Inclusion-based flow-insensitive abstract pointer.

#include <Pointer.h> Inheritance diagram for Canal::Pointer::InclusionBased:



Collaboration diagram for Canal::Pointer::InclusionBased:



Public Member Functions

- InclusionBased (const Environment &environment, const llvm::Type &type)
Standard constructor.
- InclusionBased (const InclusionBased &second)
Copy constructor.
- InclusionBased (const InclusionBased &second, const llvm::Type &newType)
- virtual ~InclusionBased ()
Standard destructor.
- void addTarget (Target::Type type, const llvm::Value *instruction, const llvm::Value *target, const std::vector< Domain * > &offsets, Domain *numericOffset)
- Domain * dereferenceAndMerge (const State &state) const
- InclusionBased * bitcast (const llvm::Type &type) const
Creates a copy of this object with a different pointer type.
- InclusionBased * getElementPtr (const std::vector< Domain * > &offsets, const llvm::Type &type) const
- void store (const Domain &value, State &state)
- virtual InclusionBased * clone () const
- virtual InclusionBased * cloneCleaned () const
- virtual bool operator== (const Domain &value) const
Implementation of Domain::operator==().
- bool isSingleTarget () const
Does this pointer point to single target?

- virtual void merge (const Domain &value)
Implementation of Domain::merge().
- virtual size_t memoryUsage () const
Implementation of Domain::memoryUsage().
- virtual std::string toString () const
Implementation of Domain::toString().
- virtual bool matchesString (const std::string &text, std::string &rationale) const
Implementation of Domain::matchesString().
- virtual float accuracy () const
Implementation of AccuracyDomain::accuracy().
- virtual bool isBottom () const
Implementation of AccuracyDomain::isBottom().
- virtual void setBottom ()
Implementation of AccuracyDomain::setBottom().
- virtual bool isTop () const
Implementation of AccuracyDomain::isTop().
- virtual void setTop ()
Implementation of AccuracyDomain::setTop().

Public Attributes

- PlaceTargetMap mTargets
- const llvm::Type & mType
The type object is owned by the LLVM framework.
- bool mTop
If true, this pointer can point anywhere.

16.13.1 Detailed Description

Inclusion-based flow-insensitive abstract pointer.

16.13.2 Constructor & Destructor Documentation

16.13.2.1 Canal::Pointer::InclusionBased::InclusionBased (const InclusionBased & *second*, const llvm::Type & *newType*)

Copy constructor which changes the pointer type. Useful for bitcast and getelementptr operations.

16.13.3 Member Function Documentation

16.13.3.1 `void Canal::Pointer::InclusionBased::addTarget (Target::Type type, const llvm::Value * instruction, const llvm::Value * target, const std::vector< Domain * > & offsets, Domain * numericOffset)`

Add a new target to the pointer.

Parameters:

type Type of the referenced memory.

instruction Place where the pointer target is added.

target Represents the target memory block. If type is Constant, it must be NULL. Otherwise, it must be a valid pointer to an instruction. This is a key to State::mFunctionBlocks, State::mFunctionVariables, State::mGlobalBlocks, or State::mGlobalVariables, depending on the type.

offsets Offsets in the getelementptr style. The provided vector might be empty. The newly created pointer target becomes the owner of the objects in the vector.

numericOffset Numerical offset that is used in addition to the getelementptr style offset and after they have been applied. It might be NULL, which indicates the offset 0. The newly created pointer target becomes the owner of the numerical offset when it's provided. This parameter is mandatory for pointers of Constant type, because it contains the constant.

16.13.3.2 `InclusionBased * Canal::Pointer::InclusionBased::clone () const [virtual]`

Implementation of Domain::clone(). Covariant return type -- it really overrides Domain::clone().

Implements Canal::Domain.

16.13.3.3 `InclusionBased * Canal::Pointer::InclusionBased::cloneCleaned () const [virtual]`

Implementation of Domain::cloneCleaned(). Covariant return type.

Implements Canal::Domain.

16.13.3.4 `Domain * Canal::Pointer::InclusionBased::dereferenceAndMerge (const State & state) const`

Dereference all targets and merge the results into single abstract value. The returned value is owned by the caller.

Returns:

It might return NULL.

16.13.3.5 `InclusionBased * Canal::Pointer::InclusionBased::getElementPtr (const std::vector< Domain * > & offsets, const llvm::Type & type) const`

Creates a copy of this object pointing to subtargets.

Parameters:

offsets Pointer takes ownership of the values inside the vector.

16.13.4 Member Data Documentation

16.13.4.1 PlaceTargetMap Canal::Pointer::InclusionBased::mTargets

llvm::Value represents a position in the program. It points to the instruction where the target was assigned/stored to the pointer.

16.13.4.2 const llvm::Type& Canal::Pointer::InclusionBased::mType

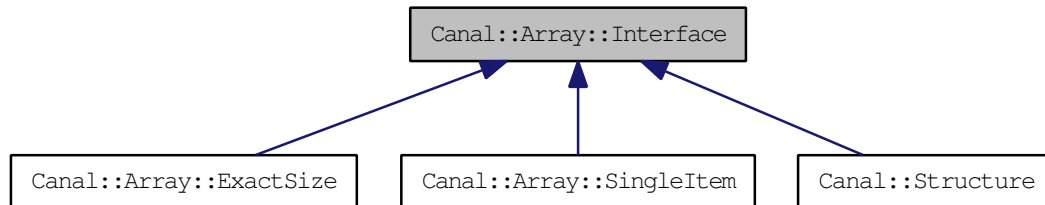
The type object is owned by the LLVM framework. Type of the object the pointer is pointing to. It might be incompatible with the type of the actual abstract value. Conversion is needed during store and load operations in such a case.

The documentation for this class was generated from the following files:

- lib/Pointer.h
- lib/Pointer.cpp

16.14 Canal::Array::Interface Class Reference

Inheritance diagram for Canal::Array::Interface:



Public Member Functions

- Domain * getValue (const Domain &offset) const
- Domain * getValue (uint64_t offset) const
- virtual std::vector< Domain * > getItem (const Domain &offset) const =0
- virtual Domain * getItem (uint64_t offset) const =0
- virtual void setItem (const Domain &offset, const Domain &value)=0
- virtual void setItem (uint64_t offset, const Domain &value)=0

16.14.1 Member Function Documentation

16.14.1.1 virtual Domain* Canal::Array::Interface::getItem (uint64_t offset) const [pure virtual]

Get the array item pointed by the provided offset. Returns internal array item that is owned by the array. Caller must not delete the item.

Note:

The uint64_t offset variant exists because of the extractvalue instruction, which provides exact numeric offsets.

For future array domains it might be necessary to extend this method to return a list of values.

Implemented in Canal::Array::ExactSize, Canal::Array::SingleItem, and Canal::Structure.

16.14.1.2 virtual std::vector<Domain*> Canal::Array::Interface::getItem (const Domain & offset) const [pure virtual]

Get the array items pointed by the provided offset. Returns internal array items that are owned by the array. Caller must not delete the items.

Implemented in Canal::Array::ExactSize, Canal::Array::SingleItem, and Canal::Structure.

16.14.1.3 Domain* Canal::Array::Interface::getValue (uint64_t offset) const

Gets the value representing the array item pointed by the provided offset. Caller is responsible for deleting the returned value.

Note:

The uint64_t offset variant exists because of the extractvalue instruction, which provides exact numeric offsets.

16.14.1.4 Domain * Canal::Array::Interface::getValue (const Domain & offset) const

Gets the value representing the array item or items pointed by the provided offset. Caller is responsible for deleting the returned value.

16.14.1.5 virtual void Canal::Array::Interface::setItem (uint64_t offset, const Domain & value) [pure virtual]**Parameters:**

value The method does not take the ownership of this memory. It copies the contents of the value instead.

Note:

The uint64_t offset variant exists because of the insertvalue instruction, which provides exact numeric offsets.

Implemented in Canal::Array::ExactSize, Canal::Array::SingleItem, and Canal::Structure.

16.14.1.6 virtual void Canal::Array::Interface::setItem (const Domain & offset, const Domain & value) [pure virtual]**Parameters:**

value The method does not take the ownership of this memory. It copies the contents of the value instead.

Implemented in Canal::Array::ExactSize, Canal::Array::SingleItem, and Canal::Structure.

The documentation for this class was generated from the following files:

- lib/ArrayInterface.h
- lib/ArrayInterface.cpp

16.15 Canal::InterpreterBlock::Interpreter Class Reference

Collaboration diagram for Canal::InterpreterBlock::Interpreter:



Public Member Functions

- Interpreter (const llvm::Module *module)
- const Environment & **getEnvironment** () const
- SlotTracker & **getSlotTracker** () const
- const Constructors & **getConstructors** () const
- const Module & **getModule** () const
- const Operations & **getOperations** () const
- Iterator & **getIterator** ()
- const Iterator & **getIterator** () const
- const State & **getCurrentState** () const
- const Function & **getCurrentFunction** () const
- const BasicBlock & **getCurrentBasicBlock** () const
- const llvm::Instruction & **getCurrentInstruction** () const

Protected Member Functions

- Domain * **onFunctionCall** (const llvm::Function &function, const std::vector< Domain * > &arguments)

Protected Attributes

- Environment **mEnvironment**
- Constructors **mConstructors**
- Module **mModule**
- OperationsCallback **mOperationsCallback**
- Operations **mOperations**
- Iterator **mIterator**

16.15.1 Constructor & Destructor Documentation

16.15.1.1 Canal::InterpreterBlock::Interpreter::Interpreter (const llvm::Module * module)

Parameters:

module Interpreter takes ownership of the module.

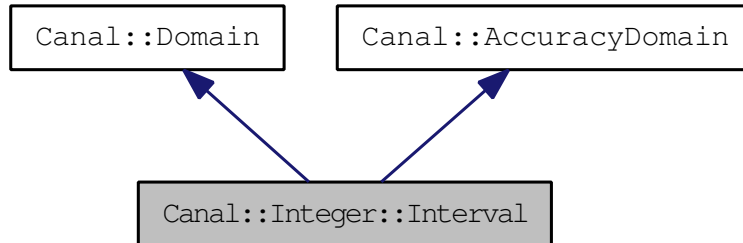
The documentation for this class was generated from the following files:

- lib/InterpreterBlock.h
- lib/InterpreterBlock.cpp

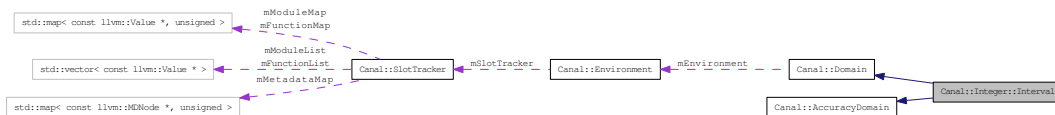
16.16 Canal::Integer::Interval Class Reference

Abstracts integer values as a interval min - max.

#include <IntegerInterval.h> Inheritance diagram for Canal::Integer::Interval:



Collaboration diagram for Canal::Integer::Interval:



Public Member Functions

- Interval (const Environment &environment, unsigned bitWidth)
Standard constructor.
- Interval (const Environment &environment, const llvm::APInt &constant)
Standard constructor.

- unsigned **getBitWidth** () const
- bool signedMin (llvm::APInt &result) const
- bool signedMax (llvm::APInt &result) const
- bool unsignedMin (llvm::APInt &result) const
- bool unsignedMax (llvm::APInt &result) const
- bool isSingleValue () const
- bool isSignedSingleValue () const

Returns true if the interval represents a signed single value.

- bool isUnsignedSingleValue () const

Returns true if the interval represents a unsigned single value.

- virtual Interval * clone () const
- virtual Interval * cloneCleaned () const
- virtual bool operator== (const Domain &value) const

Implementation of Domain::operator==().

- virtual void merge (const Domain &value)

Implementation of Domain::merge().

- virtual size_t memoryUsage () const
Implementation of Domain::memoryUsage().
- virtual std::string toString () const
Implementation of Domain::toString().
- virtual bool matchesString (const std::string &text, std::string &rational) const
Implementation of Domain::matchesString().
- virtual void add (const Domain &a, const Domain &b)
Implementation of Domain::add().
- virtual void sub (const Domain &a, const Domain &b)
Implementation of Domain::sub().
- virtual void mul (const Domain &a, const Domain &b)
Implementation of Domain::mul().
- virtual void udiv (const Domain &a, const Domain &b)
Implementation of Domain::udiv().
- virtual void sdiv (const Domain &a, const Domain &b)
Implementation of Domain::sdiv().
- virtual void urem (const Domain &a, const Domain &b)
Implementation of Domain::urem().
- virtual void srem (const Domain &a, const Domain &b)
Implementation of Domain::srem().
- virtual void shl (const Domain &a, const Domain &b)
Implementation of Domain::shl().
- virtual void lshr (const Domain &a, const Domain &b)
Implementation of Domain::lshr().
- virtual void ashr (const Domain &a, const Domain &b)
Implementation of Domain::ashr().
- virtual void and_ (const Domain &a, const Domain &b)
Implementation of Domain::and_().
- virtual void or_ (const Domain &a, const Domain &b)
Implementation of Domain::or_().
- virtual void xor_ (const Domain &a, const Domain &b)
Implementation of Domain::xor_().
- virtual void icmp (const Domain &a, const Domain &b, llvm::CmpInst::Predicate predicate)

Implementation of Domain::icmp().

- virtual float accuracy () const

Implementation of AccuracyDomain::accuracy().

- virtual bool isBottom () const

Implementation of AccuracyDomain::isBottom().

- virtual void setBottom ()

Implementation of AccuracyDomain::setBottom().

- virtual bool isTop () const

Implementation of AccuracyDomain::isTop().

- virtual void setTop ()

Implementation of AccuracyDomain::setTop().

Public Attributes

- bool mEmpty

Indicates an empty interval.

- bool **mSignedTop**

- llvm::APInt mSignedFrom

The number is included in the interval.

- llvm::APInt mSignedTo

The number is included in the interval.

- bool **mUnsignedTop**

- llvm::APInt mUnsignedFrom

The number is included in the interval.

- llvm::APInt mUnsignedTo

The number is included in the interval.

16.16.1 Detailed Description

Abstracts integer values as a interval min - max.

16.16.2 Constructor & Destructor Documentation

16.16.2.1 Canal::Integer::Interval::Interval (const Environment & *environment*, unsigned *bitWidth*)

Standard constructor. Initializes an empty interval.

16.16.3 Member Function Documentation

16.16.3.1 `Interval * Canal::Integer::Interval::clone () const` [virtual]

Implementation of `Domain::clone()`. Covariant return type.

Implements `Canal::Domain`.

16.16.3.2 `Interval * Canal::Integer::Interval::cloneCleaned () const` [virtual]

Implementation of `Domain::cloneCleaned()`. Covariant return type.

Implements `Canal::Domain`.

16.16.3.3 `bool Canal::Integer::Interval::isSingleValue () const`

Returns true if the interval represents a single number. Signed and unsigned representations might differ, though.

16.16.3.4 `bool Canal::Integer::Interval::signedMax (llvm::APInt & result) const`

Highest signed number represented by this abstract domain.

Parameters:

result Filled by the maximum value if it is known. Otherwise, the value is undefined.

Returns:

True if the result is known and the parameter was set to correct value.

16.16.3.5 `bool Canal::Integer::Interval::signedMin (llvm::APInt & result) const`

Lowest signed number represented by this abstract domain.

Parameters:

result Filled by the minimum value if it is known. Otherwise, the value is undefined.

Returns:

True if the result is known and the parameter was set to correct value.

16.16.3.6 `bool Canal::Integer::Interval::unsignedMax (llvm::APInt & result) const`

Highest unsigned number represented by this abstract domain.

Parameters:

result Filled by the maximum value if it is known. Otherwise, the value is undefined.

Returns:

True if the result is known and the parameter was set to correct value.

16.16.3.7 bool Canal::Integer::Interval::unsignedMin (llvm::APInt & *result*) const

Lowest unsigned number represented by this abstract domain.

Parameters:

result Filled by the minimum value if it is known. Otherwise, the value is undefined.

Returns:

True if the result is known and the parameter was set to correct value.

16.16.4 Member Data Documentation

16.16.4.1 bool Canal::Integer::Interval::mEmpty

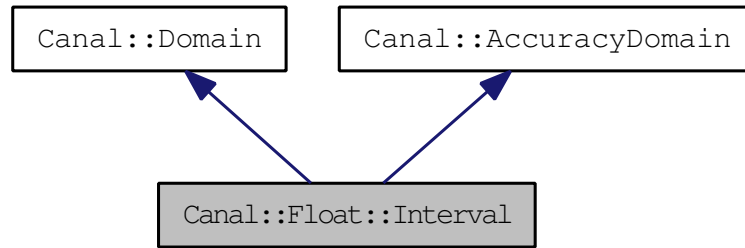
Indicates an empty interval. When it is set to true, other members' values are not considered as valid.

The documentation for this class was generated from the following files:

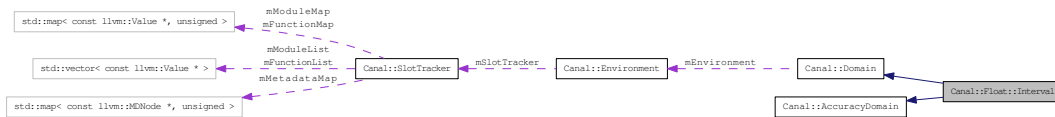
- lib/IntegerInterval.h
- lib/IntegerInterval.cpp

16.17 Canal::Float::Interval Class Reference

Inheritance diagram for Canal::Float::Interval:



Collaboration diagram for Canal::Float::Interval:



Public Member Functions

- **Interval** (const Environment &environment, const llvm::fltSemantics &semantics)
- int **compare** (const Interval &value, llvm::CmpInst::Predicate predicate) const
- bool **isNaN** () const
- const llvm::fltSemantics & **getSemantics** () const
- bool **isSingleValue** () const
- bool **intersects** (const Interval &value) const
- llvm::APFloat **getMax** () const
- llvm::APFloat **getMin** () const
- virtual Interval * **clone** () const

Create a copy of this value.

- virtual Interval * **cloneCleaned** () const
- virtual bool **operator==** (const Domain &value) const
- virtual void **merge** (const Domain &value)
- virtual size_t **memoryUsage** () const

Get memory usage (used byte count) of this abstract value.

- virtual std::string **toString** () const

Create a string representation of the abstract value.

- virtual bool **matchesString** (const std::string &text, std::string &rationale) const
- virtual float **accuracy** () const
- virtual bool **isBottom** () const

Is it the lowest value.

- virtual void **setBottom** ()

Set to the lowest value.

- virtual bool isTop () const

Is it the highest value.

- virtual void setTop ()

Set it to the top value of lattice.

Public Attributes

- bool mEmpty
- bool mTop
- llvm::APFloat mFrom
- llvm::APFloat mTo

16.17.1 Member Function Documentation

16.17.1.1 float Canal::Float::Interval::accuracy () const [virtual]

Get accuracy of the abstract value (0 - 1). In finite-height lattices, it is determined by the position of the value in the lattice.

Accuracy 0 means that the value represents all possible values (top). Accuracy 1 means that the value represents the most precise and exact value (bottom).

Reimplemented from Canal::AccuracyDomain.

16.17.1.2 Interval * Canal::Float::Interval::cloneCleaned () const [virtual]

This is used to obtain instance of the value type and to get an empty value at the same time.

Implements Canal::Domain.

16.17.1.3 bool Canal::Float::Interval::matchesString (const std::string & text, std::string & rationale) const [virtual]

Checks if the abstract value internal state matches the text description. Full coverage of the state is not expected, the text can contain just partial information.

Implements Canal::Domain.

16.17.1.4 std::string Canal::Float::Interval::toString () const [virtual]

Create a string representation of the abstract value. An idea for different memory interpretation. virtual Domain *castTo(const llvm::Type *itemType, int offset) const = 0;

Implements Canal::Domain.

The documentation for this class was generated from the following files:

- lib/FloatInterval.h
- lib/FloatInterval.cpp

16.18 Canal::InterpreterBlock::Iterator Class Reference

`#include <InterpreterBlockIterator.h>` Collaboration diagram for Canal::InterpreterBlock::Iterator:



Public Member Functions

- **Iterator** (Module &module, Operations &operations)
- void **initialize** ()
- void **interpretInstruction** ()
- void **setCallback** (IteratorCallback &callback)
- bool **isInitialized** () const
- const State & **getCurrentState** () const
- const Function & **getCurrentFunction** () const
- const BasicBlock & **getCurrentBasicBlock** () const
- const llvm::Instruction & **getCurrentInstruction** () const

Protected Member Functions

- void **nextInstruction** ()

Protected Attributes

- Module & **mModule**
- Operations & **mOperations**
- bool mChanged
- bool mInitialized
- std::vector< Function * >::const_iterator mFunction
- std::vector< BasicBlock * >::const_iterator mBasicBlock
- llvm::BasicBlock::const_iterator mInstruction

The instruction that will be interpreted in the next step.

- State mState
Current state.
- IteratorCallback * mCallback
Callback functions.

16.18.1 Detailed Description

Basic iterator that iterates over the whole program until a fixpoint is reached.

16.18.2 Member Function Documentation

16.18.2.1 void Canal::InterpreterBlock::Iterator::interpretInstruction ()

One step of the interpreter. Interprets the instruction and moves to the next one.

16.18.3 Member Data Documentation

16.18.3.1 std::vector<BasicBlock*>::const_iterator Canal::InterpreterBlock::Iterator::mBasicBlock [protected]

Basic block of the instruction that will be interpreted in the next step.

16.18.3.2 bool Canal::InterpreterBlock::Iterator::mChanged [protected]

Indication of changed abstract state during last loop through the program.

16.18.3.3 std::vector<Function*>::const_iterator Canal::InterpreterBlock::Iterator::mFunction [protected]

Function of the instruction that will be interpreted in the next step.

16.18.3.4 bool Canal::InterpreterBlock::Iterator::mInitialized [protected]

Indication that the iterator has been initialized and started iterating.

The documentation for this class was generated from the following files:

- lib/InterpreterBlockIterator.h
- lib/InterpreterBlockIterator.cpp

16.19 Canal::InterpreterBlock::IteratorCallback Class Reference

Public Member Functions

- virtual void **onFixpointReached** ()
- virtual void **onModuleEnter** ()
- virtual void **onModuleExit** ()
- virtual void **onFunctionEnter** (Function &function)
- virtual void **onFunctionExit** (Function &function)
- virtual void **onBasicBlockEnter** (BasicBlock &basicBlock)
- virtual void **onBasicBlockExit** (BasicBlock &basicBlock)
- virtual void **onInstructionEnter** (const llvm::Instruction &instruction)
- virtual void **onInstructionExit** (const llvm::Instruction &instruction)

The documentation for this class was generated from the following file:

- lib/InterpreterBlockIteratorCallback.h

16.20 Canal::InterpreterBlock::Module Class Reference

Collaboration diagram for Canal::InterpreterBlock::Module:



Public Member Functions

- **Module** (const llvm::Module &module, const Constructors &constructors)
- std::vector< Function * >::const_iterator **begin** () const
- std::vector< Function * >::const_iterator **end** () const
- Function * **getFunction** (const char *name) const
- Function * **getFunction** (const std::string &name) const
- Function * **getFunction** (const llvm::Function &function) const

Protected Attributes

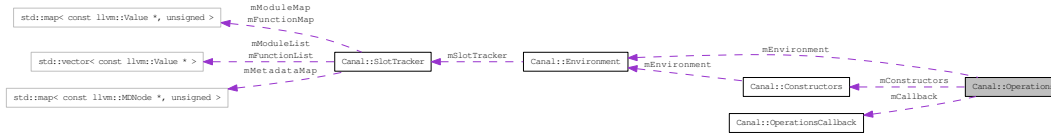
- const llvm::Module & **mModule**
- std::vector< Function * > **mFunctions**

The documentation for this class was generated from the following files:

- lib/InterpreterBlockModule.h
- lib/InterpreterBlockModule.cpp

16.21 Canal::Operations Class Reference

#include <Operations.h> Collaboration diagram for Canal::Operations:



Public Member Functions

- **Operations** (const Environment &environment, const Constructors &constructors, OperationsCallback &callback)
- void interpretInstruction (const llvm::Instruction &instruction, State &state)
Interprets current instruction.

Protected Member Functions

- Domain * variableOrConstant (const llvm::Value &place, State &state, llvm::OwningPtr< Domain > &constant) const
- template<typename T >
void **interpretCall** (const T &instruction, State &state)
- void **binaryOperation** (const llvm::BinaryOperator &instruction, State &state, Domain::BinaryOperation operation)
- template<typename T >
bool **getElementPtrOffsets** (std::vector< Domain * > &result, T iteratorStart, T iteratorEnd, const State &state)
- void **castOperation** (const llvm::CastInst &instruction, State &state, Domain::CastOperation operation)
- void **cmpOperation** (const llvm::CmpInst &instruction, State &state, Domain::CmpOperation operation)
- virtual void ret (const llvm::ReturnInst &instruction, State &state)
- virtual void br (const llvm::BranchInst &instruction, State &state)
- virtual void switch_ (const llvm::SwitchInst &instruction, State &state)
- virtual void indirectbr (const llvm::IndirectBrInst &instruction, State &state)
- virtual void invoke (const llvm::InvokeInst &instruction, State &state)
- virtual void unreachable (const llvm::UnreachableInst &instruction, State &state)
- virtual void add (const llvm::BinaryOperator &instruction, State &state)
Sum of two operands. It's a binary operator.
- virtual void fadd (const llvm::BinaryOperator &instruction, State &state)
- virtual void sub (const llvm::BinaryOperator &instruction, State &state)
Difference of two operands. It's a binary operator.
- virtual void fsub (const llvm::BinaryOperator &instruction, State &state)
- virtual void mul (const llvm::BinaryOperator &instruction, State &state)
Product of two operands. It's a binary operator.

- virtual void fmul (const llvm::BinaryOperator &instruction, State &state)
Product of two operands. It's a binary operator.
- virtual void udiv (const llvm::BinaryOperator &instruction, State &state)
- virtual void sdiv (const llvm::BinaryOperator &instruction, State &state)
- virtual void fdiv (const llvm::BinaryOperator &instruction, State &state)
- virtual void urem (const llvm::BinaryOperator &instruction, State &state)
Unsigned division remainder. It's a binary operator.
- virtual void srem (const llvm::BinaryOperator &instruction, State &state)
Signed division remainder. It's a binary operator.
- virtual void frem (const llvm::BinaryOperator &instruction, State &state)
Floating point remainder. It's a binary operator.
- virtual void shl (const llvm::BinaryOperator &instruction, State &state)
It's a bitwise binary operator.
- virtual void lshr (const llvm::BinaryOperator &instruction, State &state)
It's a bitwise binary operator.
- virtual void ashr (const llvm::BinaryOperator &instruction, State &state)
It's a bitwise binary operator.
- virtual void and_ (const llvm::BinaryOperator &instruction, State &state)
It's a bitwise binary operator.
- virtual void or_ (const llvm::BinaryOperator &instruction, State &state)
It's a bitwise binary operator.
- virtual void xor_ (const llvm::BinaryOperator &instruction, State &state)
It's a bitwise binary operator.
- virtual void extractelement (const llvm::ExtractElementInst &instruction, State &state)
It's a vector operation.
- virtual void insertelement (const llvm::InsertElementInst &instruction, State &state)
It's a vector operation.
- virtual void shufflevector (const llvm::ShuffleVectorInst &instruction, State &state)
It's a vector operation.
- virtual void extractvalue (const llvm::ExtractValueInst &instruction, State &state)
It's an aggregate operation.
- virtual void insertvalue (const llvm::InsertValueInst &instruction, State &state)
It's an aggregate operation.
- virtual void alloca_ (const llvm::AllocaInst &instruction, State &state)
It's a memory access operation.

- virtual void load (const llvm::LoadInst &instruction, State &state)
It's a memory access operation.
- virtual void store (const llvm::StoreInst &instruction, State &state)
It's a memory access operation.
- virtual void getelementptr (const llvm::GetElementPtrInst &instruction, State &state)
It's a memory addressing operation.
- virtual void trunc (const llvm::TruncInst &instruction, State &state)
It's a conversion operation.
- virtual void zext (const llvm::ZExtInst &instruction, State &state)
It's a conversion operation.
- virtual void sext (const llvm::SExtInst &instruction, State &state)
It's a conversion operation.
- virtual void fptrunc (const llvm::FPTruncInst &instruction, State &state)
It's a conversion operation.
- virtual void fpext (const llvm::FPExtInst &instruction, State &state)
It's a conversion operation.
- virtual void fptoui (const llvm::FPToUIInst &instruction, State &state)
It's a conversion operation.
- virtual void fptosi (const llvm::FPToSIInst &instruction, State &state)
It's a conversion operation.
- virtual void uitofp (const llvm::UIToFPInst &instruction, State &state)
It's a conversion operation.
- virtual void sitofp (const llvm::SIToFPInst &instruction, State &state)
It's a conversion operation.
- virtual void ptrtoint (const llvm::PtrToIntInst &instruction, State &state)
It's a conversion operation.
- virtual void inttoptr (const llvm::IntToPtrInst &instruction, State &state)
It's a conversion operation.
- virtual void bitcast (const llvm::BitCastInst &instruction, State &state)
It's a conversion operation.
- virtual void **icmp** (const llvm::ICmpInst &instruction, State &state)
- virtual void **fcmp** (const llvm::FCmpInst &instruction, State &state)
- virtual void **phi** (const llvm::PHINode &instruction, State &state)
- virtual void **select** (const llvm::SelectInst &instruction, State &state)
- virtual void **call** (const llvm::CallInst &instruction, State &state)
- virtual void **va_arg** (const llvm::VArgInst &instruction, State &state)

Protected Attributes

- const Environment & **mEnvironment**
- const Constructors & **mConstructors**
- OperationsCallback & **mCallback**

16.21.1 Detailed Description

Context-sensitive flow-insensitive operational abstract interpreter. Interprets instructions in abstract domain.

This is an abstract class, which is used as a base class for actual abstract interpretation implementations.

16.21.2 Member Function Documentation

16.21.2.1 void Canal::Operations::br (const llvm::BranchInst & *instruction*, State & *state*)
[protected, virtual]

Transfer to a different basic block in the current function. It's a terminator instruction.

16.21.2.2 void Canal::Operations::fadd (const llvm::BinaryOperator & *instruction*, State & *state*)
[protected, virtual]

Sum of two operands. It's a binary operator. The operands are floating point or vector of floating point values.

16.21.2.3 void Canal::Operations::fdiv (const llvm::BinaryOperator & *instruction*, State & *state*)
[protected, virtual]

Quotient of two operands. It's a binary operator. The operands are floating point or vector of floating point values.

16.21.2.4 void Canal::Operations::fsub (const llvm::BinaryOperator & *instruction*, State & *state*)
[protected, virtual]

Difference of two operands. It's a binary operator. The operands are floating point or vector of floating point values.

16.21.2.5 void Canal::Operations::indirectbr (const llvm::IndirectBrInst & *instruction*, State & *state*)
[protected, virtual]

An indirect branch to a label within the current function, whose address is specified by "address". It's a terminator instruction.

16.21.2.6 void Canal::Operations::invoke (const llvm::InvokeInst & *instruction*, State & *state*)
[protected, virtual]

Transfer to a specified function, with the possibility of control flow transfer to either the 'normal' label or the 'exception' label. It's a terminator instruction.

16.21.2.7 `void Canal::Operations::ret (const llvm::ReturnInst & instruction, State & state)`
[protected, virtual]

Return control flow (and optionally a value) from a function back to the caller. It's a terminator instruction.

16.21.2.8 `void Canal::Operations::sdiv (const llvm::BinaryOperator & instruction, State & state)`
[protected, virtual]

Quotient of two operands. It's a binary operator. The operands are integer or vector of integer values.

16.21.2.9 `void Canal::Operations::switch_ (const llvm::SwitchInst & instruction, State & state)`
[protected, virtual]

Transfer control flow to one of several different places. It is a generalization of the 'br' instruction, allowing a branch to occur to one of many possible destinations. It's a terminator instruction.

16.21.2.10 `void Canal::Operations::udiv (const llvm::BinaryOperator & instruction, State & state)` [protected, virtual]

Quotient of two operands. It's a binary operator. The operands are integer or vector of integer values.

16.21.2.11 `void Canal::Operations::unreachable (const llvm::UnreachableInst & instruction, State & state)` [protected, virtual]

No defined semantics. This instruction is used to inform the optimizer that a particular portion of the code is not reachable. It's a terminator instruction.

16.21.2.12 `Domain * Canal::Operations::variableOrConstant (const llvm::Value & place, State & state, llvm::OwningPtr< Domain > & constant) const` [protected]

Given a place in source code, return the corresponding variable from the abstract interpreter state. If the place contains a constant, fill the provided constant variable with it.

Returns:

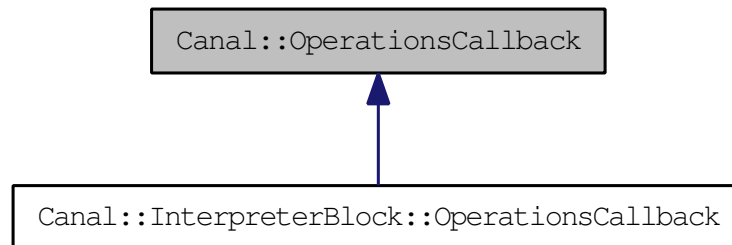
Returns a pointer to the variable if it is found in the state. Returns a pointer to the provided constant if the place contains a constant. Otherwise, it returns NULL.

The documentation for this class was generated from the following files:

- lib/Operations.h
- lib/Operations.cpp

16.22 Canal::OperationsCallback Class Reference

Inheritance diagram for Canal::OperationsCallback:



Public Member Functions

- virtual void onFunctionCall (const llvm::Function &function, const State &callState, State &resultState, const llvm::Value &resultPlace)=0

16.22.1 Member Function Documentation

16.22.1.1 virtual void Canal::OperationsCallback::onFunctionCall (const llvm::Function &function, const State &callState, State &resultState, const llvm::Value &resultPlace) [pure virtual]

Parameters:

callState Contains function arguments and referenced memory blocks.

resultState A state where the changes caused by the function can be merged.

resultPlace A place in the resultState where the returned value should be merged.

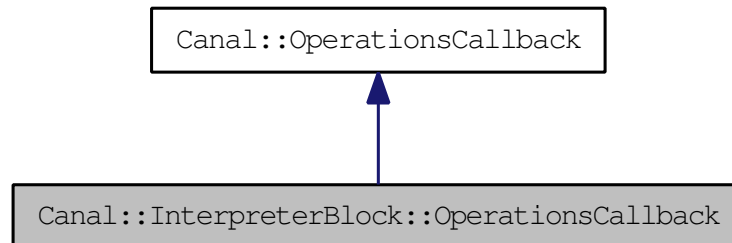
Implemented in Canal::InterpreterBlock::OperationsCallback.

The documentation for this class was generated from the following file:

- lib/OperationsCallback.h

16.23 Canal::InterpreterBlock::OperationsCallback Class Reference

Inheritance diagram for Canal::InterpreterBlock::OperationsCallback:



Collaboration diagram for Canal::InterpreterBlock::OperationsCallback:



Public Member Functions

- **OperationsCallback** (Module &module, Constructors &mConstructors)
- virtual void onFunctionCall (const llvm::Function &function, const State &callState, State &resultState, const llvm::Value &resultPlace)

Protected Attributes

- Module & **mModule**
- Constructors & **mConstructors**

16.23.1 Member Function Documentation

16.23.1.1 void Canal::InterpreterBlock::OperationsCallback::onFunctionCall (const llvm::Function &function, const State &callState, State &resultState, const llvm::Value &resultPlace) [virtual]

Parameters:

callState Contains function arguments and referenced memory blocks.

resultState A state where the changes caused by the function can be merged.

resultPlace A place in the resultState where the returned value should be merged.

Implements Canal::OperationsCallback.

The documentation for this class was generated from the following files:

- lib/InterpreterBlockOperationsCallback.h
- lib/InterpreterBlockOperationsCallback.cpp

16.24 Canal::APIntUtils::SCompare Struct Reference

Public Member Functions

- bool **operator()** (const llvm::APInt &a, const llvm::APInt &b) const

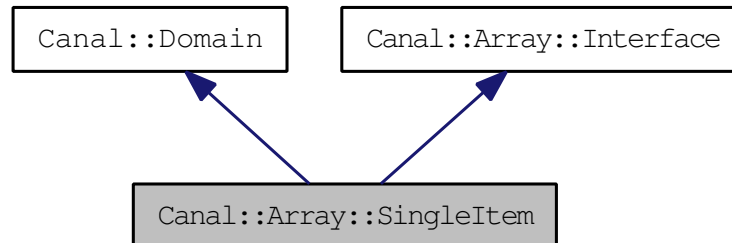
The documentation for this struct was generated from the following file:

- lib/APIntUtils.h

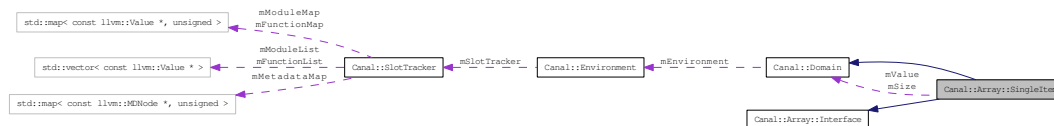
16.25 Canal::Array::SingleItem Class Reference

This array type is very imprecise.

```
#include <ArraySingleItem.h>Inheritance diagram for Canal::Array::SingleItem:
```



Collaboration diagram for Canal::Array::SingleItem:



Public Member Functions

- **SingleItem** (const Environment &environment)
- **SingleItem** (const SingleItem &singleItem)
- virtual SingleItem * clone () const
- virtual SingleItem * cloneCleaned () const
- virtual bool operator== (const Domain &value) const

Implementation of Domain::operator==().

- virtual void merge (const Domain &value)

Implementation of `Domain::merge()`.

- virtual size_t memoryUsage () const

Implementation of `Domain::memoryUsage()`.

- virtual std::string toString () const

Implementation of Domain::toString().

- virtual bool matchesString (const std::string &text, std::string &rationale) const

Implementation of `Domain::matchesString()`.

- virtual void add (const Domain &a, const Domain &b)

Implementation of Domain::add().

- virtual void fadd (const Domain &a, const Domain &b)

Implementation of `Domain::fadd()`.

- virtual void sub (const Domain &a, const Domain &b)
Implementation of Domain::sub().
- virtual void fsub (const Domain &a, const Domain &b)
Implementation of Domain::fsub().
- virtual void mul (const Domain &a, const Domain &b)
Implementation of Domain::mul().
- virtual void fmul (const Domain &a, const Domain &b)
Implementation of Domain::fmul().
- virtual void udiv (const Domain &a, const Domain &b)
Implementation of Domain::udiv().
- virtual void sdiv (const Domain &a, const Domain &b)
Implementation of Domain::sdiv().
- virtual void fdiv (const Domain &a, const Domain &b)
Implementation of Domain::fdiv().
- virtual void urem (const Domain &a, const Domain &b)
Implementation of Domain::urem().
- virtual void srem (const Domain &a, const Domain &b)
Implementation of Domain::srem().
- virtual void frem (const Domain &a, const Domain &b)
Implementation of Domain::frem().
- virtual void shl (const Domain &a, const Domain &b)
Implementation of Domain::shl().
- virtual void lshr (const Domain &a, const Domain &b)
Implementation of Domain::lshr().
- virtual void ashr (const Domain &a, const Domain &b)
Implementation of Domain::ashr().
- virtual void and_ (const Domain &a, const Domain &b)
Implementation of Domain::and_().
- virtual void or_ (const Domain &a, const Domain &b)
Implementation of Domain::or_().
- virtual void xor_ (const Domain &a, const Domain &b)
Implementation of Domain::xor_().
- virtual void icmp (const Domain &a, const Domain &b, llvm::CmpInst::Predicate predicate)

Implementation of Domain::icmp().

- virtual void fcmp (const Domain &a, const Domain &b, llvm::CmpInst::Predicate predicate)

Implementation of Domain::fcmp().

- virtual std::vector< Domain * > getItem (const Domain &offset) const

Implementation of Array::Interface::getItem().

- virtual Domain * getItem (uint64_t offset) const

Implementation of Array::Interface::getItem().

- virtual void setItem (const Domain &offset, const Domain &value)

Implementation of Array::Interface::setItem().

- virtual void setItem (uint64_t offset, const Domain &value)

Implementation of Array::Interface::setItem().

Public Attributes

- Domain * **mValue**
- Domain * mSize

16.25.1 Detailed Description

This array type is very imprecise. The most trivial array type. It treats all array members as a single value. This means all the operations on the array are merged and used to move the single value up in its lattice.

16.25.2 Member Function Documentation

16.25.2.1 SingleItem * Canal::Array::SingleItem::clone () const [virtual]

Implementation of Domain::clone(). Covariant return type.

Implements Canal::Domain.

16.25.2.2 SingleItem * Canal::Array::SingleItem::cloneCleaned () const [virtual]

Implementation of Domain::cloneCleaned(). Covariant return type.

Implements Canal::Domain.

16.25.3 Member Data Documentation

16.25.3.1 Domain* Canal::Array::SingleItem::mSize

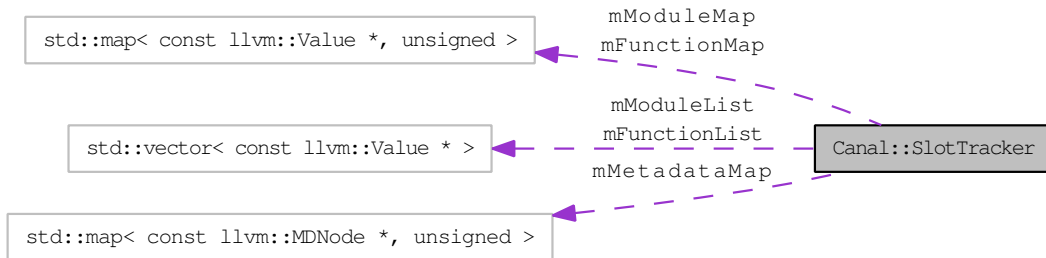
Number of elements in the array. It is either a Constant or Integer::Container.

The documentation for this class was generated from the following files:

- `lib/ArraySingleItem.h`
- `lib/ArraySingleItem.cpp`

16.26 Canal::SlotTracker Class Reference

#include <SlotTracker.h> Collaboration diagram for Canal::SlotTracker:



Public Types

- typedef std::map< const llvm::MDNode *, unsigned >::iterator mdn_iterator
MDNode map iterators.

Public Member Functions

- SlotTracker (const llvm::Module &module)
Construct from a module.
- void setActiveFunction (const llvm::Function &function)
- int getLocalSlot (const llvm::Value &value)
- const llvm::Value * **getLocalSlot** (int num)
- int getGlobalSlot (const llvm::Value &value)
Get the slot number of a global value.
- const llvm::Value * **getGlobalSlot** (int num)
- int getMetadataSlot (const llvm::MDNode &node)
Get the slot number of a MDNode.
- mdn_iterator **mdn_begin** ()
- mdn_iterator **mdn_end** ()
- unsigned **mdn_size** () const
- bool **mdn_empty** () const

Protected Types

- typedef std::map< const llvm::Value *, unsigned > ValueMap
A mapping of Values to slot numbers.
- typedef std::vector< const llvm::Value * > **ValueList**

Protected Member Functions

- void initialize ()
This function does the actual initialization.
- void createFunctionSlot (const llvm::Value &value)
Insert the specified Value into the slot table.*
- void createModuleSlot (const llvm::GlobalValue &value)
Insert the specified GlobalValue into the slot table.*
- void createMetadataSlot (const llvm::MDNode &node)
Insert the specified MDNode into the slot table.*
- void processModule ()
- void processFunction ()

Protected Attributes

- const llvm::Module & mModule
The module for which we are holding slot numbers.
- bool **mModuleProcessed**
- const llvm::Function * mFunction
The function for which we are holding slot numbers.
- bool **mFunctionProcessed**
- ValueMap mModuleMap
The slot map for the module level data.
- ValueList **mModuleList**
- unsigned **mModuleNext**
- ValueMap mFunctionMap
The slot map for the function level data.
- ValueList **mFunctionList**
- unsigned **mFunctionNext**
- std::map< const llvm::MDNode *, unsigned > mMetadataMap
The slot map for MDNodes.
- unsigned **mMetadataNext**

16.26.1 Detailed Description

This class provides computation of slot numbers. Initial version was taken from LLVM source code (lib/VMCore/AsmWriter.cpp).

16.26.2 Member Function Documentation

16.26.2.1 `int Canal::SlotTracker::getLocalSlot (const llvm::Value & value)`

Get the slot number for a value that is local to a function. Return the slot number of the specified value in its type plane. If something is not in the SlotTracker, return -1.

16.26.2.2 `void Canal::SlotTracker::processFunction ()` **[protected]**

Add all of the functions arguments, basic blocks, and instructions.

16.26.2.3 `void Canal::SlotTracker::processModule ()` **[protected]**

Add all of the module level global variables (and their initializers) and function declarations, but not the contents of those functions.

16.26.2.4 `void Canal::SlotTracker::setActiveFunction (const llvm::Function & function)`

If you'd like to deal with a function instead of just a module, use this method to get its data into the SlotTracker.

The documentation for this class was generated from the following files:

- lib/SlotTracker.h
- lib/SlotTracker.cpp

16.27 Canal::State Class Reference

Abstract memory state.

#include <State.h> Collaboration diagram for Canal::State:



Public Member Functions

- **State** (const State &state)
- State & **operator=** (const State &state)
- bool **operator==** (const State &state) const
- bool **operator!=** (const State &state) const
- void clear ()
Clears everything. Releases all memory.
- void clearFunctionLevel ()
Clears function variables, blocks and returned value.
- void merge (const State &state)
Merge everything.
- void mergeGlobal (const State &state)
Merge global variables and blocks.
- void mergeReturnedValue (const State &state)
Merge the returned value.
- void mergeFunctionBlocks (const State &state)
Merge function blocks only.
- void mergeForeignFunctionBlocks (const State &state, const llvm::Function ¤tFunction)
- void addGlobalVariable (const llvm::Value &place, Domain *value)
- void addFunctionVariable (const llvm::Value &place, Domain *value)
- void **addGlobalBlock** (const llvm::Value &place, Domain *value)
- void addFunctionBlock (const llvm::Value &place, Domain *value)
Adds a value created by alloca to the stack.
- const PlaceValueMap & **getGlobalVariables** () const
- const PlaceValueMap & **getGlobalBlocks** () const
- const PlaceValueMap & **getFunctionVariables** () const
- const PlaceValueMap & **getFunctionBlocks** () const
- Domain * findVariable (const llvm::Value &place) const
- Domain * findBlock (const llvm::Value &place) const

Public Attributes

- Domain * mReturnedValue
Value returned from function.

Protected Attributes

- PlaceValueMap mGlobalVariables
- PlaceValueMap mGlobalBlocks
- PlaceValueMap mFunctionVariables
- PlaceValueMap mFunctionBlocks

16.27.1 Detailed Description

Abstract memory state. Consists of function-level variables (also called registers) and stack memory, global variables and heap, and return value. All variables are in abstract domain.

16.27.2 Member Function Documentation

16.27.2.1 void Canal::State::addFunctionVariable (const llvm::Value & *place*, Domain * *value*)

Adds a register-type value to the stack.

Parameters:

place Represents a place in the program where the function variable is assigned. Usually it is an instance of llvm::Instruction for a result of the instruction. It might also be an instance of llvm::Argument, which represents a function call parameter.

See also:

To add a value created by alloca to the stack, use the method addFunctionBlock.

16.27.2.2 void Canal::State::addGlobalVariable (const llvm::Value & *place*, Domain * *value*)

Parameters:

place Represents a place in the program where the global variable is defined and assigned.

16.27.2.3 Domain * Canal::State::findBlock (const llvm::Value & *place*) const

Search both global and function blocks for a place. If the place is found, the block is returned. Otherwise NULL is returned.

16.27.2.4 Domain * Canal::State::findVariable (const llvm::Value & *place*) const

Search both global and function variables for a place. If the place is found, the variable is returned. Otherwise NULL is returned.

16.27.2.5 void Canal::State::mergeForeignFunctionBlocks (const State & *state*, const llvm::Function & *currentFunction*)

Merge function memory blocks external to a function. This is used after a function call, where the modifications of the global state need to be merged to the state of the caller, but its local state is not relevant.

16.27.3 Member Data Documentation

16.27.3.1 PlaceValueMap Canal::State::mFunctionBlocks [protected]

Nameless memory/values allocated on the stack. The values are referenced either by a pointer in mFunctionVariables or mGlobalVariables, or by another item in mFunctionBlocks or mGlobalBlocks.

The members of the list are owned by this class, so they are deleted in the state destructor.

16.27.3.2 PlaceValueMap Canal::State::mFunctionVariables [protected]

The value pointer does *not* point to mFunctionBlocks! To connect with a mFunctionBlocks item, create a Pointer object that contains a pointer to a StackBlocks item.

The key (llvm::Value*) is not owned by this class. It is not deleted. The value (Domain*) memory is owned by this class, so it is deleted in state destructor.

16.27.3.3 PlaceValueMap Canal::State::mGlobalBlocks [protected]

Nameless memory/values allocated on the heap. It's referenced either by a pointer somewhere on a stack, by a global variable, or by another Block or stack Block.

The keys are not owned by this class. They represent the place where the block has been allocated. The values are owned by this class, so they are deleted in the state destructor.

16.27.3.4 PlaceValueMap Canal::State::mGlobalVariables [protected]

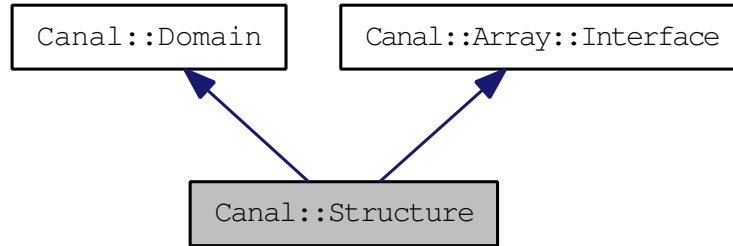
The key (llvm::Value*) is not owned by this class. It is not deleted. The value (Domain*) memory is owned by this class, so it is deleted in state destructor.

The documentation for this class was generated from the following files:

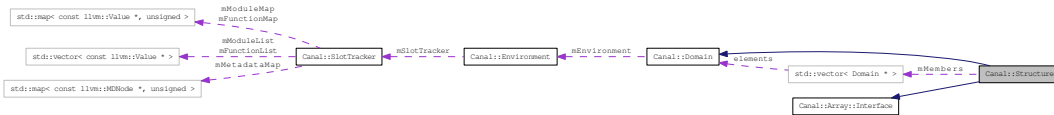
- lib/State.h
- lib/State.cpp

16.28 Canal::Structure Class Reference

Inheritance diagram for Canal::Structure:



Collaboration diagram for Canal::Structure:



Public Member Functions

- **Structure** (const Environment &environment)
- **Structure** (const Structure &structure)
- virtual Structure * clone () const
- virtual Structure * cloneCleaned () const
- virtual bool operator== (const Domain &value) const

Implementation of Domain::operator==().

- virtual void merge (const Domain &value)

Implementation of Domain::merge().

- virtual size_t memoryUsage () const

Implementation of Domain::memoryUsage().

- virtual std::string toString () const

Implementation of Domain::toString().

- virtual bool matchesString (const std::string &text, std::string &rationale) const

Implementation of Domain::matchesString().

- virtual std::vector< Domain * > getItem (const Domain &offset) const

Implementation of Array::Interface::getItem().

- virtual Domain * getItem (uint64_t offset) const

Implementation of Array::Interface::getItem().

- virtual void setItem (const Domain &offset, const Domain &value)

Implementation of Array::Interface::set().

- virtual void setItem (uint64_t offset, const Domain &value)

Implementation of Array::Interface::set().

Public Attributes

- std::vector< Domain * > **mMembers**

16.28.1 Member Function Documentation

16.28.1.1 Structure * Canal::Structure::clone () const [virtual]

Implementation of Domain::clone(). Covariant return type.

Implements Canal::Domain.

16.28.1.2 Structure * Canal::Structure::cloneCleaned () const [virtual]

Implementation of Domain::cloneCleaned(). Covariant return type.

Implements Canal::Domain.

The documentation for this class was generated from the following files:

- lib/Structure.h
- lib/Structure.cpp

16.29.1 Detailed Description

TODO: Pointers to functions. Pointer target -- where the pointer points to. Pointer can:

- be set to a fixed constant (memory area), such as 0xbaadfood
- point to a heap object (global block)
- point to a stack object (alloca, function block) Pointer can point to some offset in an array.

16.29.2 Constructor & Destructor Documentation

16.29.2.1 Canal::Pointer::Target::Target (const Environment & *environment*, Type *type*, const llvm::Value * *target*, const std::vector< Domain * > & *offsets*, Domain * *numericOffset*)

Standard constructor.

Parameters:

type Type of the referenced memory.

target Represents the target memory block. If type is Constant, it must be NULL. Otherwise, it must be a valid pointer to an instruction. This is a key to State::mFunctionBlocks, State::mFunctionVariables, State::mGlobalBlocks, or State::mGlobalVariables, depending on the type.

offsets Offsets in the getelementptr style. The provided vector might be empty. The newly created pointer target becomes the owner of the objects in the vector.

numericOffset Numerical offset that is used in addition to the getelementptr style offset and after they have been applied. It might be NULL, which indicates the offset 0. The target becomes the owner of the numerical offset when it's provided. This parameter is mandatory for pointers of Constant type, because it contains the constant.

16.29.3 Member Function Documentation

16.29.3.1 std::vector< Domain * > Canal::Pointer::Target::dereference (const State & *state*) const

Dereference the target in a certain state. Dereferencing might result in multiple values being returned due to the nature of mOffsets (offsets might include integer intervals). The returned pointers point to the memory owned by State and its abstract domains -- caller must not release the memory.

16.29.4 Member Data Documentation

16.29.4.1 const llvm::Value* Canal::Pointer::Target::mInstruction

Valid when the target represents an anonymous memory block. This is a key to either State::mGlobalBlocks or State::mFunctionBlocks. The referenced llvm::Value instance is owned by the LLVM framework and not by this class.

16.29.4.2 Domain* Canal::Pointer::Target::mNumericOffset

An additional numeric offset on the top of mOffsets. The value represents a number of bytes. This class owns the memory. It might be NULL instead of 0.

16.29.4.3 `std::vector<Domain*> Canal::Pointer::Target::mOffsets`

Array or struct offsets in the `GetElementPtr` style. This class owns the memory.

The documentation for this class was generated from the following files:

- `lib/PointerTarget.h`
- `lib/PointerTarget.cpp`

16.30 Canal::APIntUtils::UCompare Struct Reference

Public Member Functions

- **bool operator()** (const llvm::APInt &a, const llvm::APInt &b) const

The documentation for this struct was generated from the following file:

- lib/APIntUtils.h

16.31 Canal::VariablePrecisionDomain Class Reference

Base class for abstract domains that can lower the precision and memory requirements on demand.

```
#include <Domain.h>
```

Public Member Functions

- virtual bool limitMemoryUsage (size_t size)

16.31.1 Detailed Description

Base class for abstract domains that can lower the precision and memory requirements on demand.

16.31.2 Member Function Documentation

16.31.2.1 bool Canal::VariablePrecisionDomain::limitMemoryUsage (size_t *size*) [virtual]

Decrease memory usage of this value below the provided size in bytes. Returns true if the memory usage was limited, false when it was not possible.

The documentation for this class was generated from the following files:

- lib/Domain.h
- lib/Domain.cpp

Chapter 17

Tool Class Index

17.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Arguments	117
Command	118
CommandBreak	120
CommandCd	122
CommandContinue	124
CommandFile	126
CommandFinish	128
CommandHelp	130
CommandInfo	132
CommandPrint	134
CommandPwd	136
CommandQuit	138
CommandRun	140
CommandShell	144
CommandShow	146
CommandStart	148
CommandStep	150
Commands	142
IteratorCallback	152
State	153

17.2 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Arguments	117
Command	118
CommandBreak	120
CommandCd	122
CommandContinue	124
CommandFile	126
CommandFinish	128

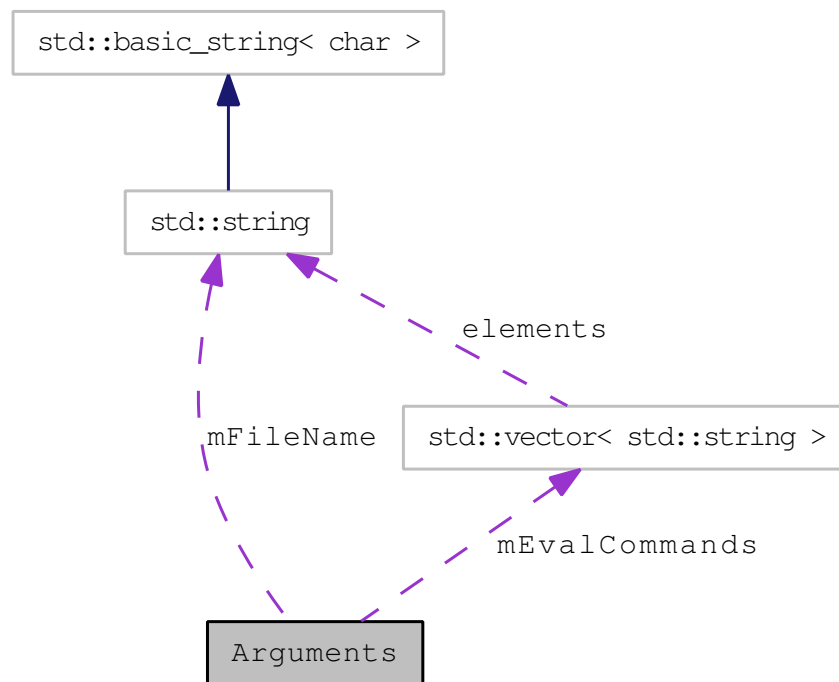
CommandHelp	130
CommandInfo	132
CommandPrint	134
CommandPwd	136
CommandQuit	138
CommandRun	140
Commands	142
CommandShell	144
CommandShow	146
CommandStart	148
CommandStep	150
IteratorCallback	152
State	153

Chapter 18

Tool Class Documentation

18.1 Arguments Class Reference

Collaboration diagram for Arguments:



Public Attributes

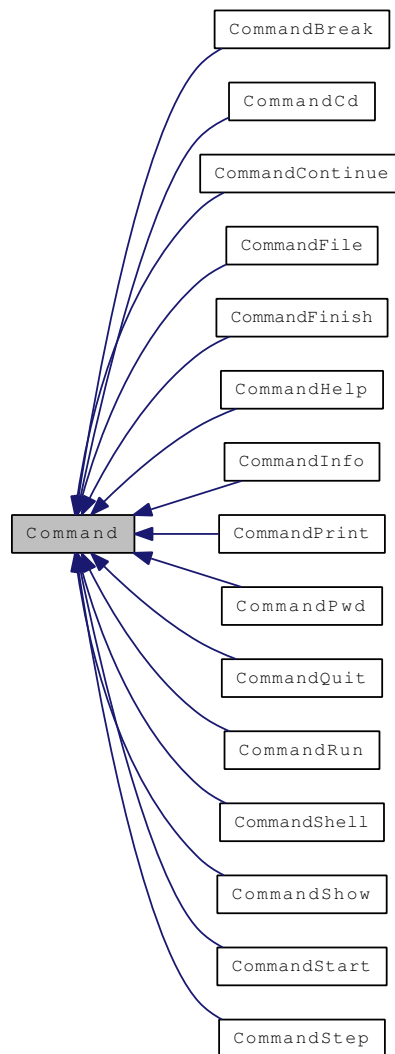
- `std::vector< std::string > mEvalCommands`
- `std::string mFileName`

The documentation for this class was generated from the following file:

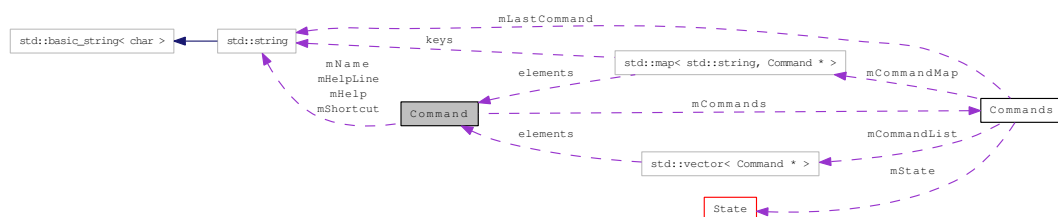
- `tool/Canal.cpp`

18.2 Command Class Reference

Inheritance diagram for Command:



Collaboration diagram for Command:



Public Member Functions

- **Command** (const std::string &name, const std::string &shortcut, const std::string &helpLine, const std::string &help, Commands &commands)
- virtual ~Command ()

Standard virtual destructor.

- const std::string & **getName** () const
- const std::string & **getShortcut** () const
- const std::string & **getHelpLine** () const
- const std::string & **getHelp** () const
- virtual std::vector< std::string > **getCompletionMatches** (const std::vector< std::string > &args, int pointArg, int pointArgOffset) const
- virtual void **run** (const std::vector< std::string > &args)=0

Protected Attributes

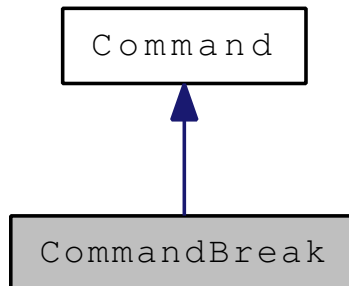
- std::string **mName**
- std::string **mShortcut**
- std::string **mHelpLine**
- std::string **mHelp**
- Commands & **mCommands**

The documentation for this class was generated from the following files:

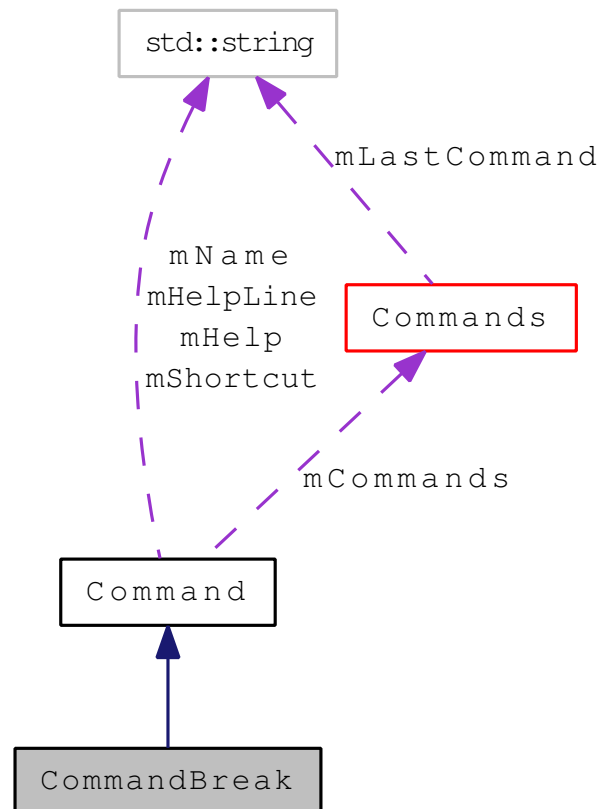
- tool/Command.h
- tool/Command.cpp

18.3 CommandBreak Class Reference

Inheritance diagram for CommandBreak:



Collaboration diagram for CommandBreak:



Public Member Functions

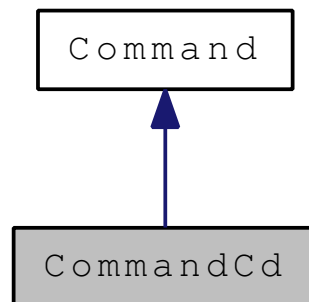
- **CommandBreak** (Commands &commands)
- virtual void **run** (const std::vector< std::string > &args)

The documentation for this class was generated from the following files:

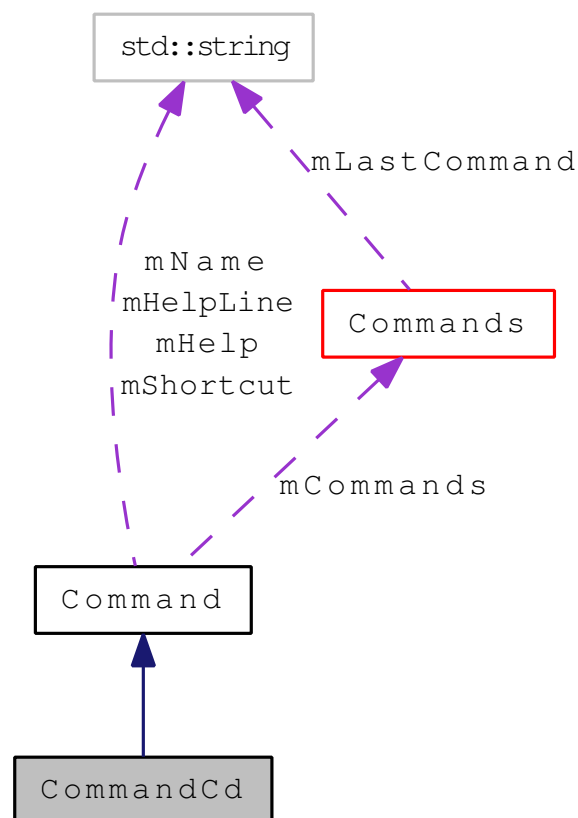
- `tool/CommandBreak.h`
- `tool/CommandBreak.cpp`

18.4 CommandCd Class Reference

Inheritance diagram for CommandCd:



Collaboration diagram for CommandCd:



Public Member Functions

- **CommandCd** (`Commands &commands`)
- virtual `std::vector< std::string > getCompletionMatches` (`const std::vector< std::string > &args`, `int pointArg`, `int pointArgOffset`) `const`

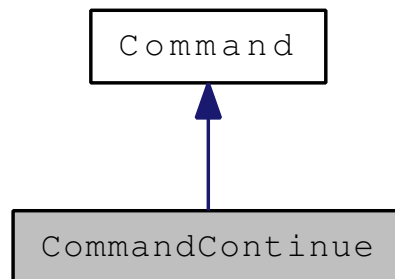
- virtual void **run** (const std::vector< std::string > &args)

The documentation for this class was generated from the following files:

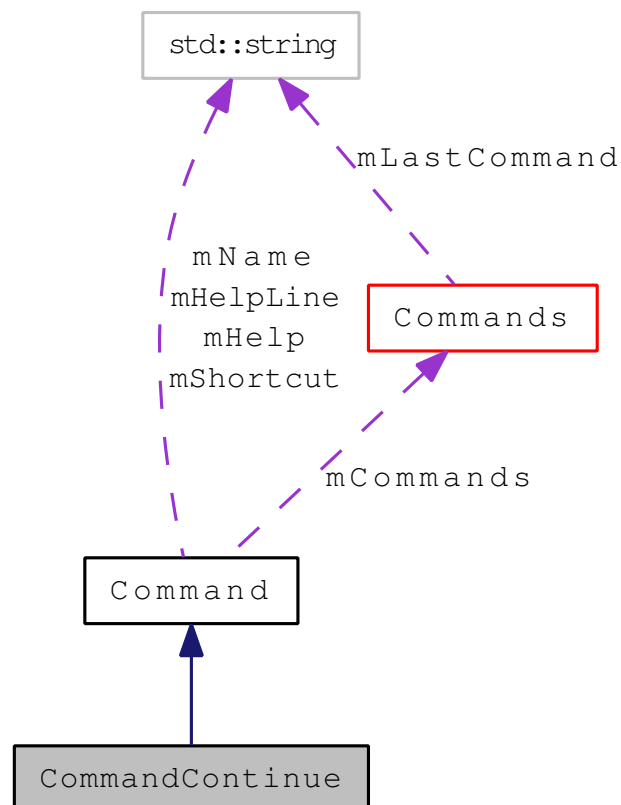
- tool/CommandCd.h
- tool/CommandCd.cpp

18.5 CommandContinue Class Reference

Inheritance diagram for CommandContinue:



Collaboration diagram for CommandContinue:



Public Member Functions

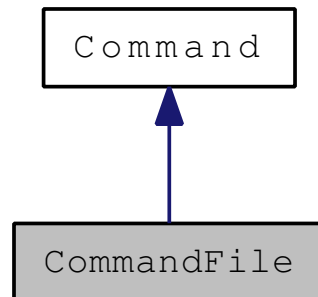
- **CommandContinue** (Commands &commands)
- virtual void **run** (const std::vector< std::string > &args)

The documentation for this class was generated from the following files:

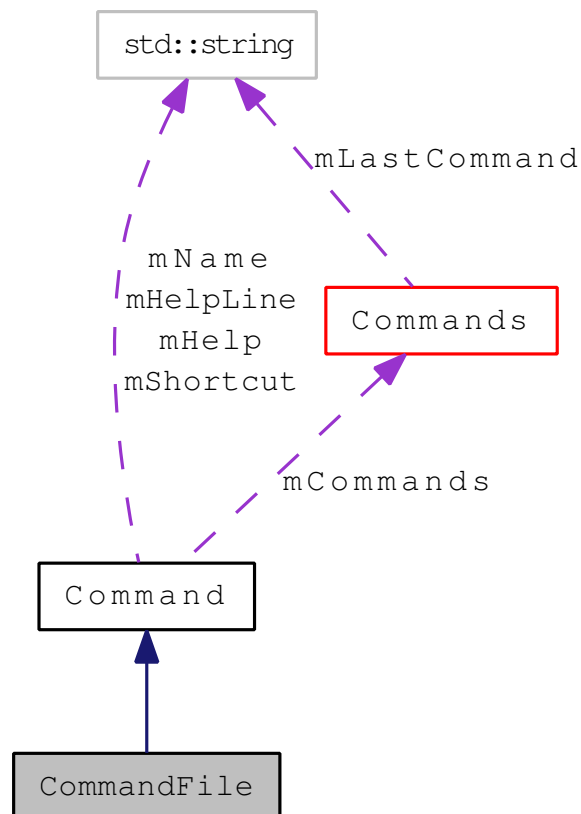
- `tool/CommandContinue.h`
- `tool/CommandContinue.cpp`

18.6 CommandFile Class Reference

Inheritance diagram for CommandFile:



Collaboration diagram for CommandFile:



Public Member Functions

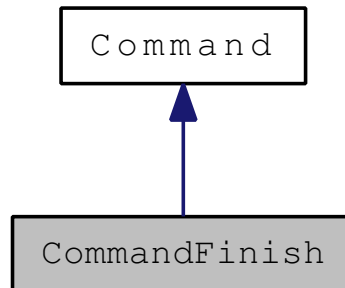
- **CommandFile** (`Commands &commands`)
- virtual `std::vector< std::string > getCompletionMatches` (`const std::vector< std::string > &args`, `int pointArg`, `int pointArgOffset`) `const`
- virtual void **run** (`const std::vector< std::string > &args`)

The documentation for this class was generated from the following files:

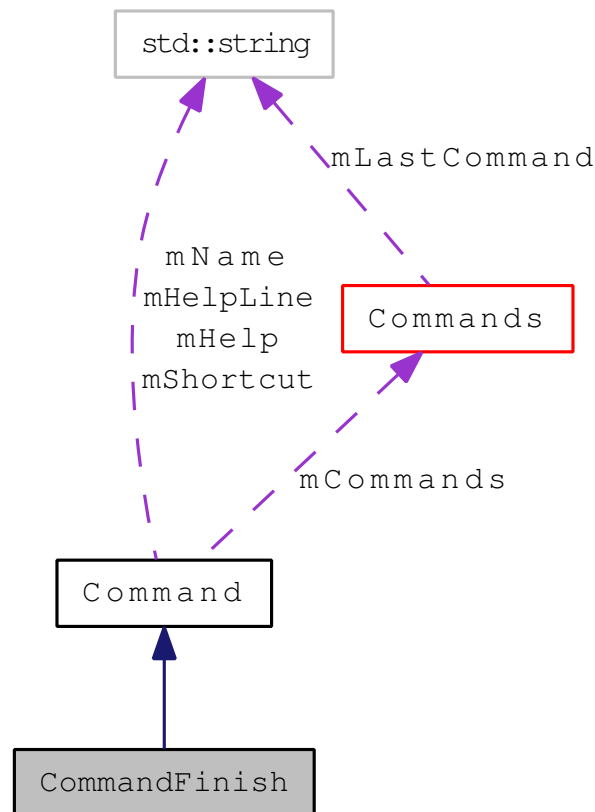
- `tool/CommandFile.h`
- `tool/CommandFile.cpp`

18.7 CommandFinish Class Reference

Inheritance diagram for CommandFinish:



Collaboration diagram for CommandFinish:



Public Member Functions

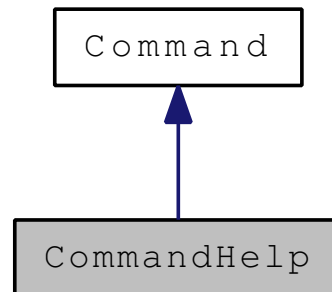
- **CommandFinish** (Commands &commands)
- virtual void **run** (const std::vector< std::string > &args)

The documentation for this class was generated from the following files:

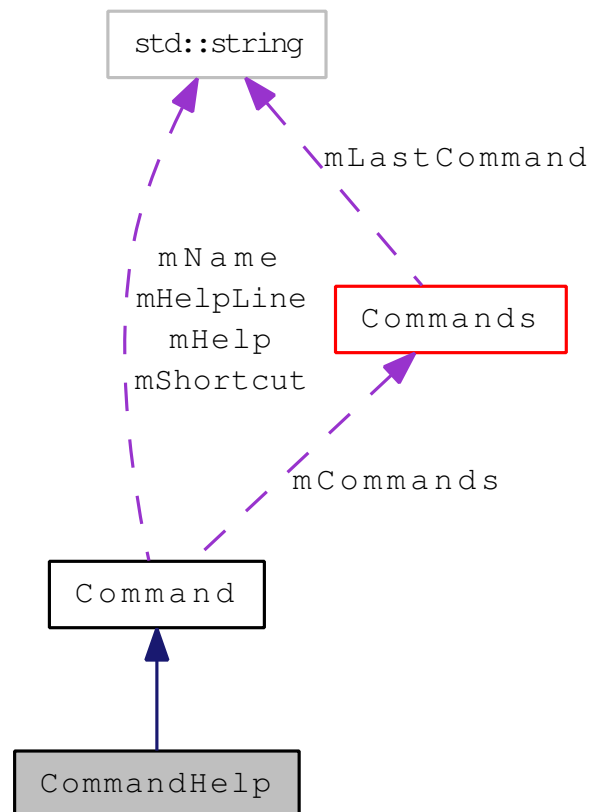
- `tool/CommandFinish.h`
- `tool/CommandFinish.cpp`

18.8 CommandHelp Class Reference

Inheritance diagram for CommandHelp:



Collaboration diagram for CommandHelp:



Public Member Functions

- **CommandHelp** (Commands &commands)
- virtual `std::vector< std::string >` **getCompletionMatches** (const `std::vector< std::string >` &args, int pointArg, int pointArgOffset) const
- virtual void **run** (const `std::vector< std::string >` &args)

Protected Member Functions

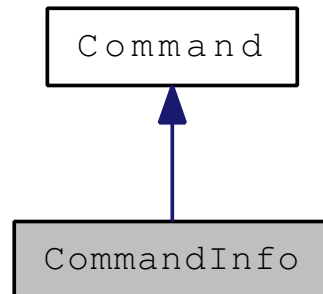
- void **allCommandsHelp** ()

The documentation for this class was generated from the following files:

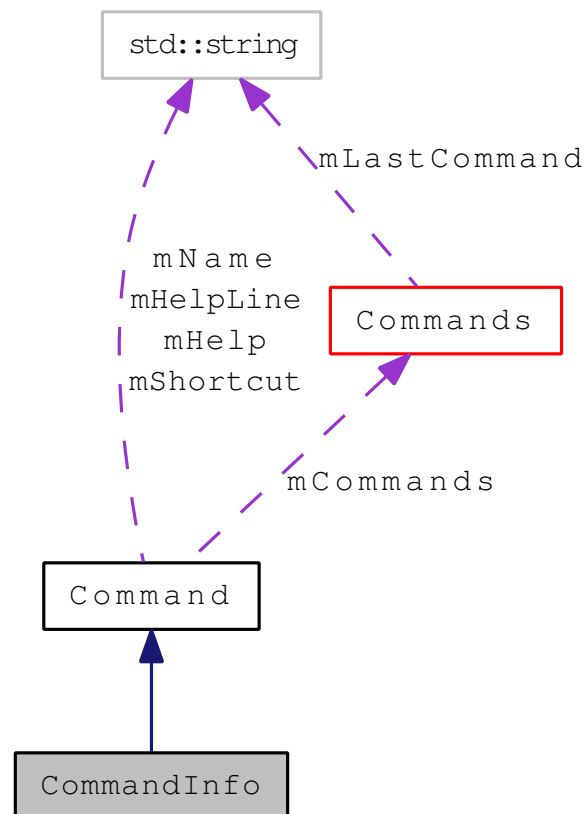
- tool/CommandHelp.h
- tool/CommandHelp.cpp

18.9 CommandInfo Class Reference

Inheritance diagram for CommandInfo:



Collaboration diagram for CommandInfo:



Public Member Functions

- **CommandInfo** (`Commands &commands`)
- virtual `std::vector< std::string >` **getCompletionMatches** (`const std::vector< std::string > &args`, `int pointArg`, `int pointArgOffset`) `const`
- virtual void **run** (`const std::vector< std::string > &args`)

Protected Member Functions

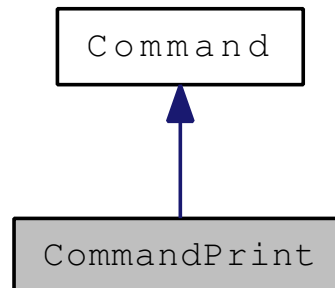
- void **infoModule** ()

The documentation for this class was generated from the following files:

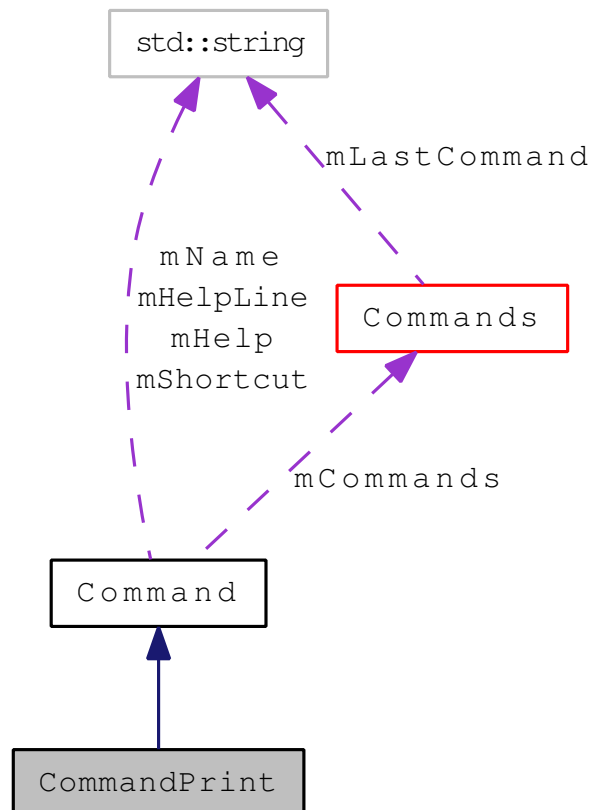
- tool/CommandInfo.h
- tool/CommandInfo.cpp

18.10 CommandPrint Class Reference

Inheritance diagram for CommandPrint:



Collaboration diagram for CommandPrint:



Public Member Functions

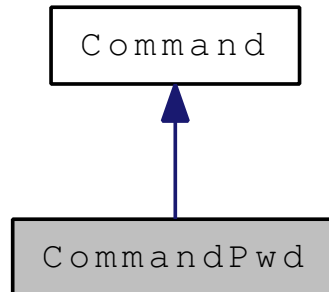
- **CommandPrint** (Commands &commands)
- virtual `std::vector< std::string >` **getCompletionMatches** (const `std::vector< std::string >` &args, int pointArg, int pointArgOffset) const
- virtual void **run** (const `std::vector< std::string >` &args)

The documentation for this class was generated from the following files:

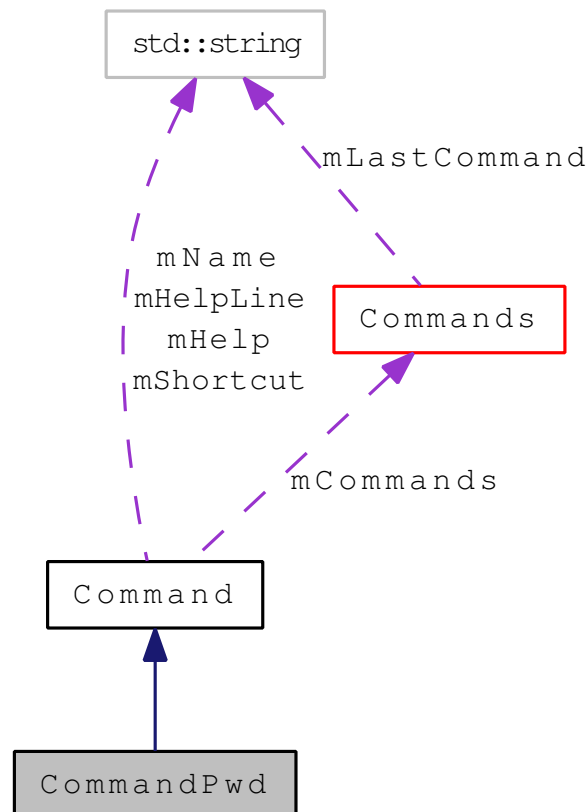
- `tool/CommandPrint.h`
- `tool/CommandPrint.cpp`

18.11 CommandPwd Class Reference

Inheritance diagram for CommandPwd:



Collaboration diagram for CommandPwd:



Public Member Functions

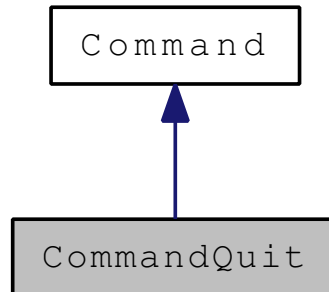
- **CommandPwd** (`Commands &commands`)
- virtual void **run** (`const std::vector< std::string > &args`)

The documentation for this class was generated from the following files:

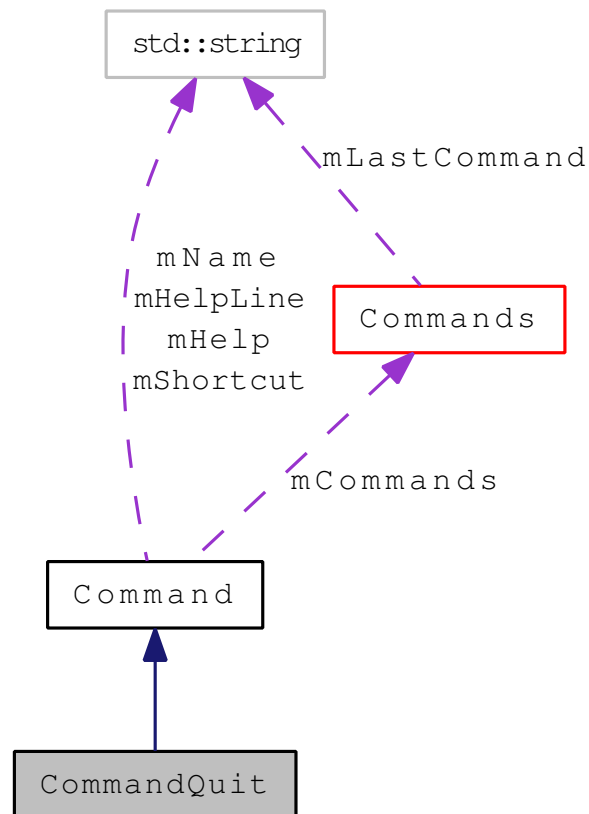
- `tool/CommandPwd.h`
- `tool/CommandPwd.cpp`

18.12 CommandQuit Class Reference

Inheritance diagram for CommandQuit:



Collaboration diagram for CommandQuit:



Public Member Functions

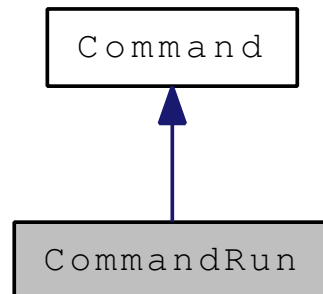
- **CommandQuit** (`Commands &commands`)
- virtual void **run** (`const std::vector< std::string > &args`)

The documentation for this class was generated from the following files:

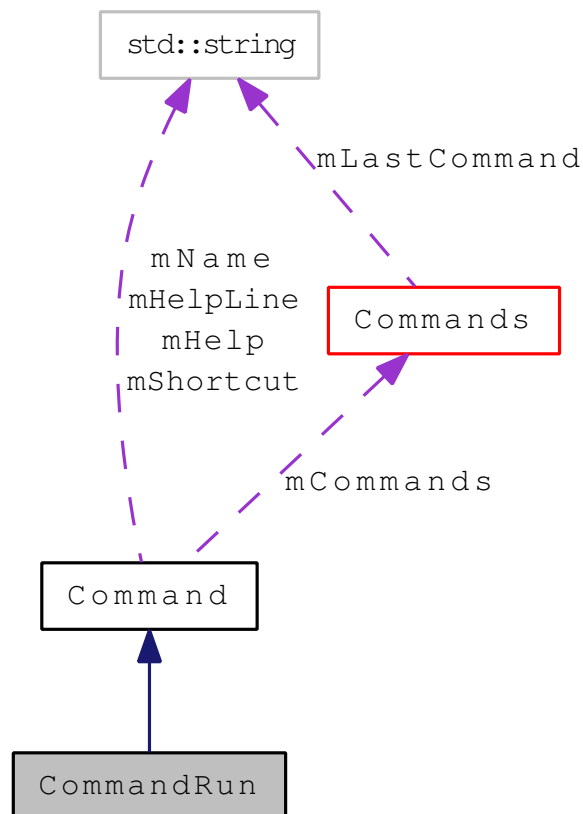
- `tool/CommandQuit.h`
- `tool/CommandQuit.cpp`

18.13 CommandRun Class Reference

Inheritance diagram for CommandRun:



Collaboration diagram for CommandRun:



Public Member Functions

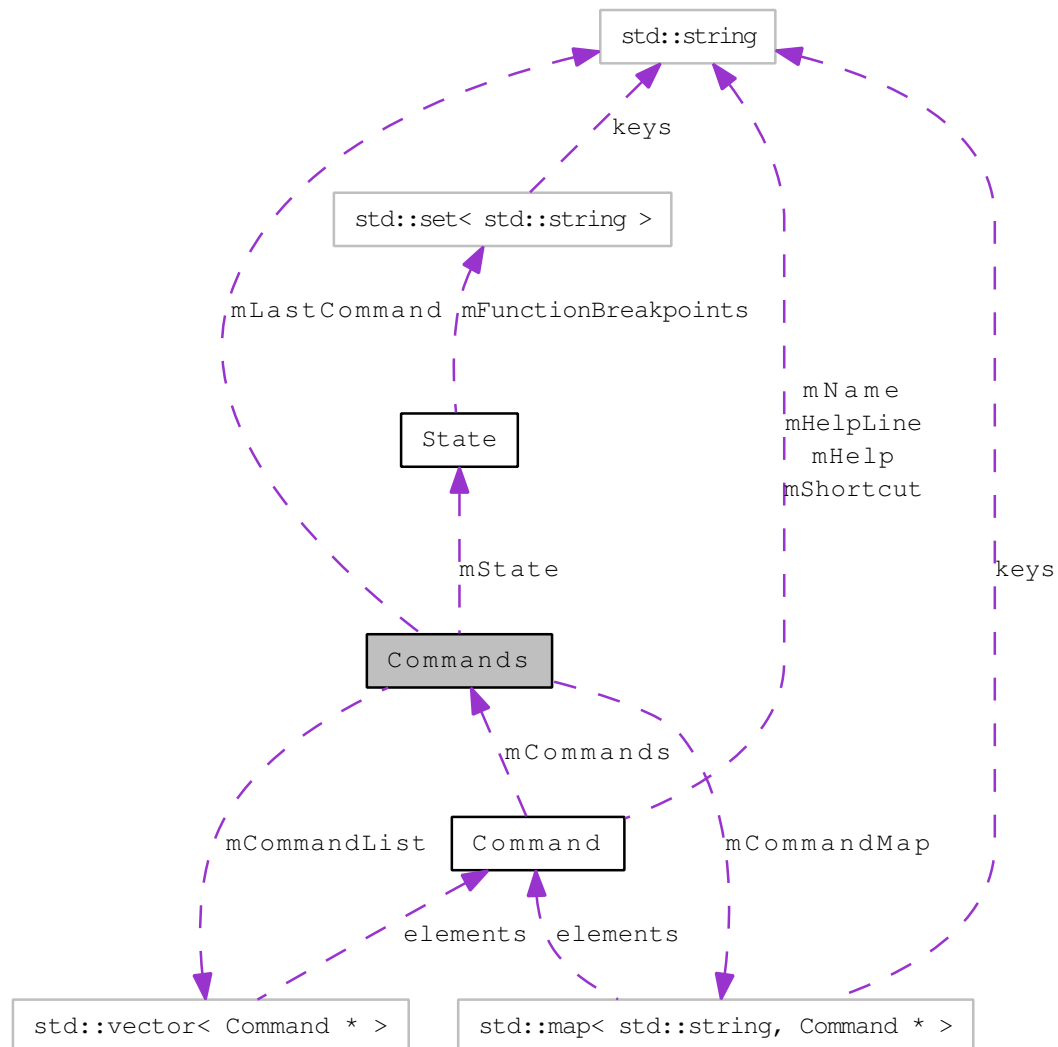
- **CommandRun** (`Commands &commands`)
- virtual void **run** (`const std::vector< std::string > &args`)

The documentation for this class was generated from the following files:

- `tool/CommandRun.h`
- `tool/CommandRun.cpp`

18.14 Commands Class Reference

Collaboration diagram for Commands:



Public Types

- typedef std::map< std::string, Command * > **CommandMap**

Public Member Functions

- std::vector< std::string > **getCompletionMatches** (const std::string &text, int point) const
- std::vector< std::string > **getCommandMatches** (const std::string &command) const
- void **executeLine** (const std::string &line)
- Command * **getCommand** (const std::string &name)
- const Command * **getCommand** (const std::string &name) const
- State * **getState** ()

- `const State * getState () const`
- `void createState (const llvm::Module *module)`

Public Attributes

- `std::vector< Command * > mCommandList`
- `CommandMap mCommandMap`

Protected Attributes

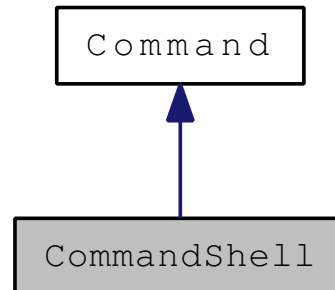
- `std::string mLastCommand`
- `State * mState`

The documentation for this class was generated from the following files:

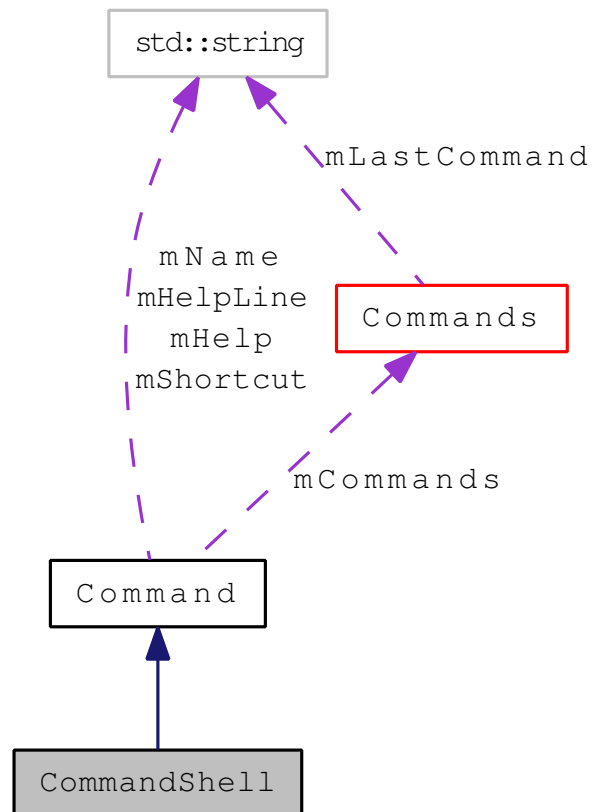
- `tool/Commands.h`
- `tool/Commands.cpp`

18.15 CommandShell Class Reference

Inheritance diagram for CommandShell:



Collaboration diagram for CommandShell:



Public Member Functions

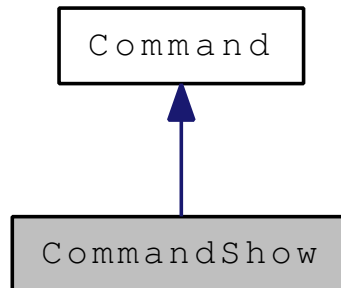
- **CommandShell** (`Commands &commands`)
- virtual void **run** (`const std::vector< std::string > &args`)

The documentation for this class was generated from the following files:

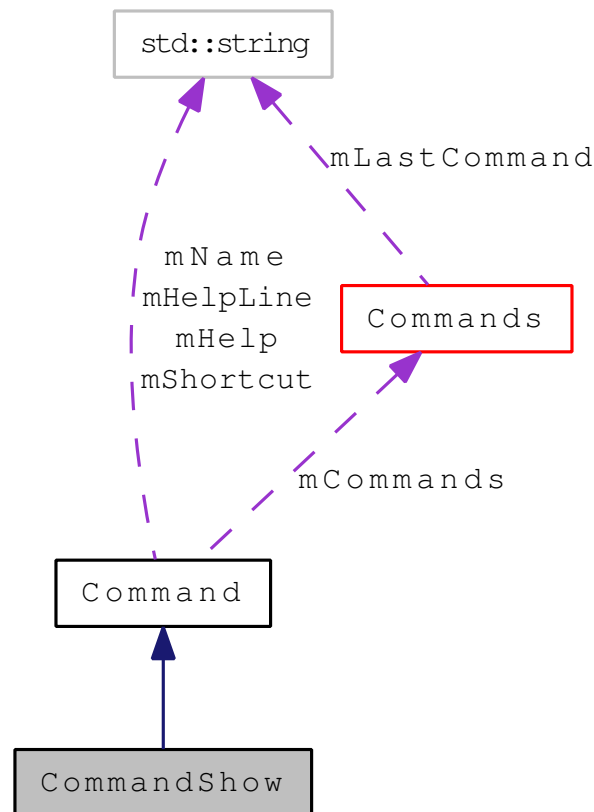
- `tool/CommandShell.h`
- `tool/CommandShell.cpp`

18.16 CommandShow Class Reference

Inheritance diagram for CommandShow:



Collaboration diagram for CommandShow:



Public Member Functions

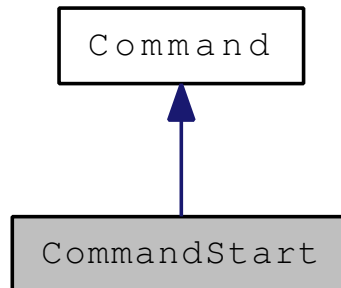
- **CommandShow** (`Commands &commands`)
- virtual void **run** (`const std::vector< std::string > &args`)

The documentation for this class was generated from the following files:

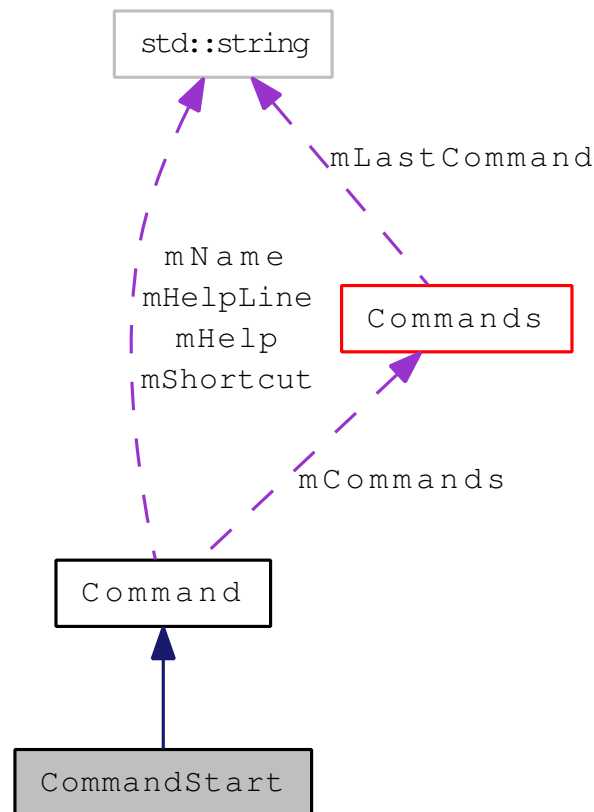
- `tool/CommandShow.h`
- `tool/CommandShow.cpp`

18.17 CommandStart Class Reference

Inheritance diagram for CommandStart:



Collaboration diagram for CommandStart:



Public Member Functions

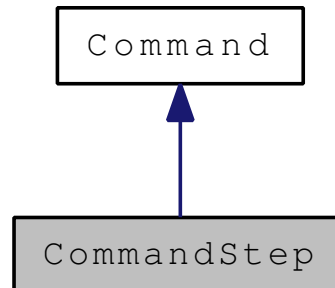
- **CommandStart** (Commands &commands)
- virtual void **run** (const std::vector< std::string > &args)

The documentation for this class was generated from the following files:

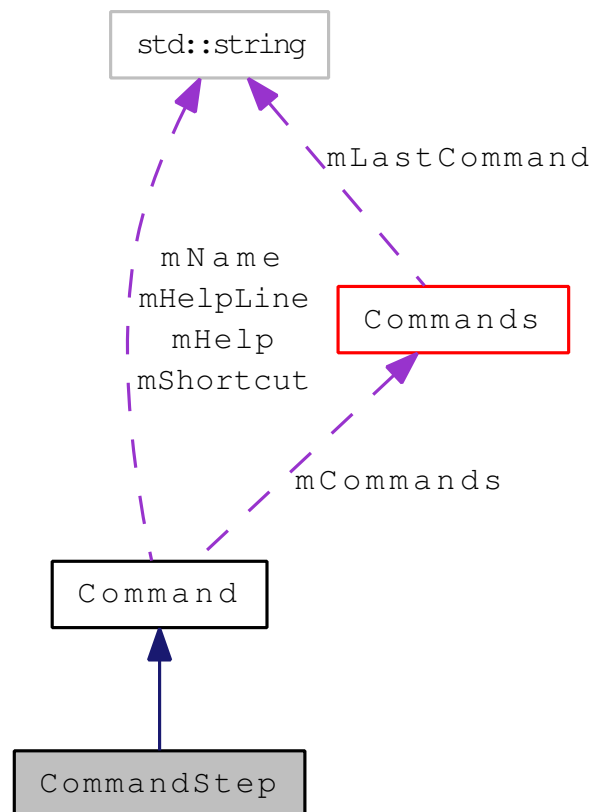
- `tool/CommandStart.h`
- `tool/CommandStart.cpp`

18.18 CommandStep Class Reference

Inheritance diagram for CommandStep:



Collaboration diagram for CommandStep:



Public Member Functions

- **CommandStep** (`Commands &commands`)
- virtual void **run** (`const std::vector< std::string > &args`)

The documentation for this class was generated from the following files:

- `tool/CommandStep.h`
- `tool/CommandStep.cpp`

18.19 IteratorCallback Class Reference

Public Member Functions

- virtual void **onFixpointReached** ()
- virtual void **onFunctionEnter** (Canal::InterpreterBlock::Function &function)
- virtual void **onBasicBlockEnter** (Canal::InterpreterBlock::BasicBlock &basicBlock)
- virtual void **onInstructionExit** (const llvm::Instruction &instruction)
- bool **isFixpointReached** () const
- bool **isFunctionEnter** ()
- bool **isBasicBlockEnter** ()

Protected Attributes

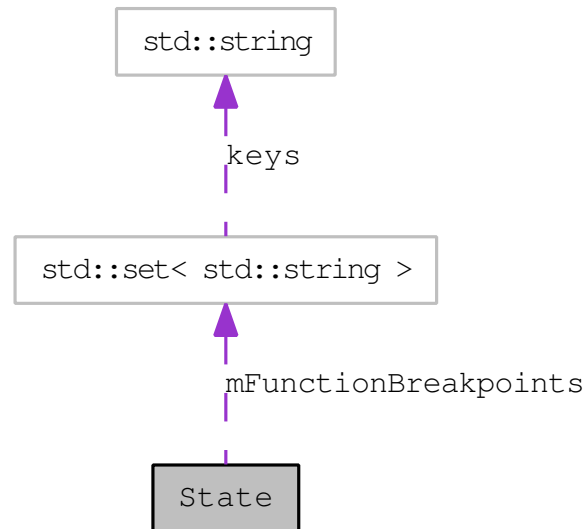
- bool **mFixpointReached**
- bool **mFunctionEnter**
- bool **mBasicBlockEnter**

The documentation for this class was generated from the following file:

- tool/IteratorCallback.h

18.20 State Class Reference

Collaboration diagram for State:



Public Member Functions

- **State** (const llvm::Module *module)
- Canal::InterpreterBlock::Interpreter & **getInterpreter** ()
- const Canal::InterpreterBlock::Interpreter & **getInterpreter** () const
- const Canal::Environment & **getEnvironment** () const
- const llvm::Module & **getModule** () const
- Canal::SlotTracker & **getSlotTracker** () const
- bool **isInterpreting** () const
- void **start** ()
- void **run** ()
- void **step** (int count)
- void **finish** ()
- void **addFunctionBreakpoint** (const std::string &functionName)

Protected Member Functions

- bool **reachedBreakpoint** ()

Protected Attributes

- Canal::InterpreterBlock::Interpreter **mInterpreter**
- std::set< std::string > **mFunctionBreakpoints**
- IteratorCallback **mIteratorCallback**

The documentation for this class was generated from the following files:

- `tool/State.h`
- `tool/State.cpp`

Chapter 19

Known Bugs

Pointers should have the possibility to be set to top.

Chapter 20

Wishlist

20.1 Callbacks Interface

20.2 Models

Models of functions and modules. Model of environment.

20.3 Support of Multiple Platforms

Support Microsoft Windows and Mac OS X natively.

20.4 Automatic Tests

Unit tests and integration tests.

20.5 Graphical User Interface

Extend Eclipse to provide an user interface to Canal.

Bibliography

- [1] Patrick Cousot and Radhia Cousot. Static Determination of Dynamic Properties of Programs. In *Proceedings of the Second International Symposium on Programming*, 1976.
- [2] Patrick Cousot and Radhia Cousot. Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *POPL '77: Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, 1977.
- [3] Patrick Cousot and Radhia Cousot. Systematic Design of Program Analysis Frameworks. In *POPL '79: Proceedings of the 6th ACM SIGACT-SIGPLAN symposium on Principles of Programming Languages*, 1979.
- [4] Patrick Cousot and Radhia Cousot. Compositional Separate Modular Static Analysis of Programs by Abstract Interpretation. In *SSGRR '01: Proceedings of the Second International Conference on Advances in Infrastructure for E-Business, E-Science and E-Education on the Internet*, 2001.
- [5] Seth Hallem, Benjamin Chelf, Yichen Xie, and Dawson Engler. A System and Language for Building System-Specific, Static Analyses. In *PLDI '02: Proceedings of the ACM SIGPLAN 2002 Conference on Programming language design and implementation*, 2002.
- [6] Patrick Cousot and Radhia Cousot. Modular Static Program Analysis. In *CC '02: International Conference on Compiler Construction*, 2002.
- [7] Brian Albert Davey and Hilary Ann Priestley. Introduction to Lattices and Order. 2nd ed. Cambridge University Press, 2002.
- [8] Roland Backhouse, Roy Crole, and Jeremy Gibbons, eds. Algebraic and Coalgebraic Methods in the Mathematics of Program Construction. Springer-Verlag, 2002.
- [9] Antoine Miné. Weakly relational numerical abstract domains. Ph.D report. 2004.
- [10] Chris Lattner and Vikram Adve. LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation. In *CGO '04: Proceedings of the International Symposium on Code Generation and Optimization: Feedback-directed and Runtime Optimization*, 2004.
- [11] Laurent Mauborgne and Xavier Rival. Trace Partitioning in Abstract Interpretation Based Static Analyzers. In *ESOP '05: European Symposium on Programming*, 2005.
- [12] David Monniaux. The parallel implementation of the Astrée static analyzer. 2005.
- [13] Antoine Miné. Field-Sensitive Value Analysis of Embedded C Programs with Union Types and Pointer Arithmetics. In *LCTES '06: Proceedings of the 2006 ACM SIGPLAN/SIGBED conference on Language, compilers, and tool support for embedded systems*, 2006.
- [14] Patrick Cousot, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, Antoine Miné, David Monniaux, and Xavier Rival. Combination of Abstractions in the ASTRÉE Static Analyzer. In *ASIAN '06: 11th Annual Asian Computing Science Conference*, 2006.

- [15] Antoine Miné. Static Analysis of Run-time Errors in Embedded Critical Parallel C Programs. In *ESOP '11: Proceedings of The 20th European Symposium on Programming*, 2011.
- [16] Antoine Miné. Abstract Domains for Bit-Level Machine Integer and Floating-point Operations. In *WING '12: Proceedings of The 4th International Workshop on Invariant Generation*, 2012.
- Boxes: A Symbolic Abstract Domain of Boxes Arie Gurfinkel and Sagar Chaki
- SubPolyhedra: A family of numerical abstract domains for the (more) scalable inference of linear inequalities Vincent Laviol , Francesco Logozzo
- The Calculational Design of a Generic Abstract Interpreter Patrick COUSOT
- SAS 2012 The 19th International Static Analysis Symposium 11-13 September 2012, Deauville, France
- Patrick Cousot, Radhia Cousot and Laurent Mauborgne. A Scalable Segmented Decision Tree Abstract Domain.
- [17] Jianzhou Zhao, Santosh Nagarakatte, Milo M. K. Martin, and Steve Zdancewic. Formalizing the LLVM Intermediate Representation for Verified Program Transformations. In *POPL '12: Proceedings of the 39th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, 2012.

Index

- accuracy
 - Canal::AccuracyDomain, 40
 - Canal::Float::Interval, 81
- add
 - Canal::Domain, 55
- addFunctionVariable
 - Canal::State, 102
- addGlobalVariable
 - Canal::State, 102
- addTarget
 - Canal::Pointer::InclusionBased, 70
- Arguments, 113
- br
 - Canal::Operations, 89
- Canal::AccuracyDomain, 39
 - accuracy, 40
- Canal::APIIntUtils::SCompare, 93
- Canal::APIIntUtils::UCompare, 109
- Canal::Array::ExactSize, 62
 - clone, 64
 - cloneCleaned, 64
- Canal::Array::Interface, 72
 - getItem, 72
 - getValue, 72, 73
 - setItem, 73
- Canal::Array::SingleItem, 94
 - clone, 96
 - cloneCleaned, 96
 - mSize, 96
- Canal::Constructors, 47
- Canal::Domain, 53
 - add, 55
 - cloneCleaned, 55
 - matchesString, 55
 - operator==, 56
 - toString, 56
- Canal::Environment, 61
- Canal::Float::Interval, 80
 - accuracy, 81
 - cloneCleaned, 81
 - matchesString, 81
 - toString, 81
- Canal::FunctionModel, 66
- Canal::FunctionModelManager, 67
- Canal::Integer::Bitfield, 42
 - clone, 44
 - cloneCleaned, 44
 - getBitValue, 44
 - mOnes, 46
 - mZeroes, 46
 - setBitValue, 44
 - signedMax, 45
 - signedMin, 45
 - unsignedMax, 45
 - unsignedMin, 45
- Canal::Integer::Container, 48
 - clone, 50
 - cloneCleaned, 50
 - Container, 50
 - matchesString, 50
 - signedMax, 51
 - signedMin, 51
 - unsignedMax, 51
 - unsignedMin, 51
- Canal::Integer::Enumeration, 57
 - clone, 59
 - cloneCleaned, 59
 - matchesString, 59
 - signedMax, 60
 - signedMin, 60
 - unsignedMax, 60
 - unsignedMin, 60
- Canal::Integer::Interval, 75
 - clone, 78
 - cloneCleaned, 78
 - Interval, 77
 - isSingleValue, 78
 - mEmpty, 79
 - signedMax, 78
 - signedMin, 78
 - unsignedMax, 78
 - unsignedMin, 78
- Canal::InterpreterBlock::BasicBlock, 41
- Canal::InterpreterBlock::Function, 65
 - updateBasicBlockInputState, 65
- Canal::InterpreterBlock::Interpreter, 74
 - Interpreter, 74
- Canal::InterpreterBlock::Iterator, 82

- interpretInstruction, 83
- mBasicBlock, 83
- mChanged, 83
- mFunction, 83
- mInitialized, 83
- Canal::InterpreterBlock::IteratorCallback, 84
- Canal::InterpreterBlock::Module, 85
- Canal::InterpreterBlock::OperationsCallback, 92
 - onFunctionCall, 92
- Canal::Operations, 86
 - br, 89
 - fadd, 89
 - fdiv, 89
 - fsub, 89
 - indirectbr, 89
 - invoke, 89
 - ret, 89
 - sdiv, 90
 - switch_, 90
 - udiv, 90
 - unreachable, 90
 - variableOrConstant, 90
- Canal::OperationsCallback, 91
 - onFunctionCall, 91
- Canal::Pointer::InclusionBased, 68
 - addTarget, 70
 - clone, 70
 - cloneCleaned, 70
 - dereferenceAndMerge, 70
 - getElementPtr, 70
 - InclusionBased, 69
 - mTargets, 71
 - mType, 71
- Canal::Pointer::Target, 106
 - dereference, 107
 - mInstruction, 107
 - mNumericOffset, 107
 - mOffsets, 107
 - Target, 107
- Canal::SlotTracker, 98
 - getLocalSlot, 100
 - processFunction, 100
 - processModule, 100
 - setActiveFunction, 100
- Canal::State, 101
 - addFunctionVariable, 102
 - addGlobalVariable, 102
 - findBlock, 102
 - findVariable, 102
 - mergeForeignFunctionBlocks, 102
 - mFunctionBlocks, 103
 - mFunctionVariables, 103
 - mGlobalBlocks, 103
 - mGlobalVariables, 103
- Canal::Structure, 104
 - clone, 105
 - cloneCleaned, 105
- Canal::VariablePrecisionDomain, 110
 - limitMemoryUsage, 110
- clone
 - Canal::Array::ExactSize, 64
 - Canal::Array::SingleItem, 96
 - Canal::Integer::Bitfield, 44
 - Canal::Integer::Container, 50
 - Canal::Integer::Enumeration, 59
 - Canal::Integer::Interval, 78
 - Canal::Pointer::InclusionBased, 70
 - Canal::Structure, 105
- cloneCleaned
 - Canal::Array::ExactSize, 64
 - Canal::Array::SingleItem, 96
 - Canal::Domain, 55
 - Canal::Float::Interval, 81
 - Canal::Integer::Bitfield, 44
 - Canal::Integer::Container, 50
 - Canal::Integer::Enumeration, 59
 - Canal::Integer::Interval, 78
 - Canal::Pointer::InclusionBased, 70
 - Canal::Structure, 105
- Command, 114
- CommandBreak, 116
- CommandCd, 118
- CommandContinue, 120
- CommandFile, 122
- CommandFinish, 124
- CommandHelp, 126
- CommandInfo, 128
- CommandPrint, 130
- CommandPwd, 132
- CommandQuit, 134
- CommandRun, 136
- Commands, 138
- CommandShell, 140
- CommandShow, 142
- CommandStart, 144
- CommandStep, 146
- Container
 - Canal::Integer::Container, 50
- dereference
 - Canal::Pointer::Target, 107
- dereferenceAndMerge
 - Canal::Pointer::InclusionBased, 70
- fadd
 - Canal::Operations, 89
- fdiv
 - Canal::Operations, 89

- findBlock
 - Canal::State, 102
- findVariable
 - Canal::State, 102
- fsub
 - Canal::Operations, 89
- getBitValue
 - Canal::Integer::Bitfield, 44
- getElementPtr
 - Canal::Pointer::InclusionBased, 70
- getItem
 - Canal::Array::Interface, 72
- getLocalSlot
 - Canal::SlotTracker, 100
- getValue
 - Canal::Array::Interface, 72, 73
- InclusionBased
 - Canal::Pointer::InclusionBased, 69
- indirectbr
 - Canal::Operations, 89
- Interpreter
 - Canal::InterpreterBlock::Interpreter, 74
- interpretInstruction
 - Canal::InterpreterBlock::Iterator, 83
- Interval
 - Canal::Integer::Interval, 77
- invoke
 - Canal::Operations, 89
- isSingleValue
 - Canal::Integer::Interval, 78
- IteratorCallback, 148
- limitMemoryUsage
 - Canal::VariablePrecisionDomain, 110
- matchesString
 - Canal::Domain, 55
 - Canal::Float::Interval, 81
 - Canal::Integer::Container, 50
 - Canal::Integer::Enumeration, 59
- mBasicBlock
 - Canal::InterpreterBlock::Iterator, 83
- mChanged
 - Canal::InterpreterBlock::Iterator, 83
- mEmpty
 - Canal::Integer::Interval, 79
- mergeForeignFunctionBlocks
 - Canal::State, 102
- mFunction
 - Canal::InterpreterBlock::Iterator, 83
- mFunctionBlocks
 - Canal::State, 103
- mFunctionVariables
 - Canal::State, 103
- mGlobalBlocks
 - Canal::State, 103
- mGlobalVariables
 - Canal::State, 103
- mInitialized
 - Canal::InterpreterBlock::Iterator, 83
- mInstruction
 - Canal::Pointer::Target, 107
- mNumericOffset
 - Canal::Pointer::Target, 107
- mOffsets
 - Canal::Pointer::Target, 107
- mOnes
 - Canal::Integer::Bitfield, 46
- mSize
 - Canal::Array::SingleItem, 96
- mTargets
 - Canal::Pointer::InclusionBased, 71
- mType
 - Canal::Pointer::InclusionBased, 71
- mZeroes
 - Canal::Integer::Bitfield, 46
- onFunctionCall
 - Canal::InterpreterBlock::OperationsCallback, 92
 - Canal::OperationsCallback, 91
- operator==
 - Canal::Domain, 56
- processFunction
 - Canal::SlotTracker, 100
- processModule
 - Canal::SlotTracker, 100
- ret
 - Canal::Operations, 89
- sdiv
 - Canal::Operations, 90
- setActiveFunction
 - Canal::SlotTracker, 100
- setBitValue
 - Canal::Integer::Bitfield, 44
- setItem
 - Canal::Array::Interface, 73
- signedMax
 - Canal::Integer::Bitfield, 45
 - Canal::Integer::Container, 51
 - Canal::Integer::Enumeration, 60
 - Canal::Integer::Interval, 78
- signedMin

- Canal::Integer::Bitfield, 45
- Canal::Integer::Container, 51
- Canal::Integer::Enumeration, 60
- Canal::Integer::Interval, 78
- State, 149
- switch_
 - Canal::Operations, 90
- Target
 - Canal::Pointer::Target, 107
- toString
 - Canal::Domain, 56
 - Canal::Float::Interval, 81
- udiv
 - Canal::Operations, 90
- unreachable
 - Canal::Operations, 90
- unsignedMax
 - Canal::Integer::Bitfield, 45
 - Canal::Integer::Container, 51
 - Canal::Integer::Enumeration, 60
 - Canal::Integer::Interval, 78
- unsignedMin
 - Canal::Integer::Bitfield, 45
 - Canal::Integer::Container, 51
 - Canal::Integer::Enumeration, 60
 - Canal::Integer::Interval, 78
- updateBasicBlockInputState
 - Canal::InterpreterBlock::Function, 65
- variableOrConstant
 - Canal::Operations, 90