

Canal

Karel Klíč

November 8, 2012

Contents

1 Overview	7
2 Installation	9
I Concepts	11
3 Preliminaries	13
4 LLVM	15
5 Abstract Interpretation	17
6 Memory Model	19
7 Array Abstract Domains	21
8 Structure Abstract Domain	23
9 Integer Abstract Domains	25
9.1 Integer Interval Domain \mathcal{D}_i^\sharp	25
9.2 Integer Bitfield Domain \mathcal{D}_b^\sharp	25
10 Abstract Domains for Floating-Point Numbers	27
11 Pointer Abstract Domains	29
12 Abstract Domain Combination	31
13 Wishlist	33
13.1 Reduced Product of Abstract Domains	33
13.2 Widening and Narrowing	33
13.3 String Abstract Domains	33
13.4 Trace Partitioning	33
13.5 Boolean Partitioning	33
13.6 Fixpoint Recalculation	33
13.7 Multi Threading	33
13.8 Symbolic Abstract Domains	34
13.9 Weakly-Relational Abstract Domains	34
13.10 Compositional Analysis	34
13.11 Parallelization	34
13.12 Custom Domains	34

II Implementation	35
14 Overview	37
15 Library Class Index	39
15.1 Class Hierarchy	39
15.2 Class List	40
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
16.4.1 Member Function Documentation	52
16.5 Canal::Integer::Container Class Reference	52
16.5.1 Constructor & Destructor Documentation	55
16.5.2 Member Function Documentation	55
16.6 Canal::Widening::DataInterface Class Reference	57
16.7 Canal::Widening::DataIterationCount Class Reference	58
16.8 Canal::Domain Class Reference	59
16.8.1 Detailed Description	61
16.8.2 Constructor & Destructor Documentation	62
16.8.3 Member Function Documentation	62
16.9 Canal::Integer::Enumeration Class Reference	63
16.9.1 Member Function Documentation	65
16.10 Canal::Environment Class Reference	67
16.11 Canal::Array::ExactSize Class Reference	67
16.11.1 Detailed Description	70
16.11.2 Constructor & Destructor Documentation	70
16.11.3 Member Function Documentation	70
16.12 Canal::InterpreterBlock::Function Class Reference	71
16.12.1 Member Function Documentation	72
16.13 Canal::FunctionModel Class Reference	72
16.14 Canal::FunctionModelManager Class Reference	73
16.15 Canal::Array::Interface Class Reference	73
16.15.1 Member Function Documentation	73
16.16 Canal::Widening::Interface Class Reference	75
16.17 Canal::InterpreterBlock::Interpreter Class Reference	75
16.17.1 Constructor & Destructor Documentation	76
16.18 Canal::Float::Interval Class Reference	77
16.18.1 Member Function Documentation	79
16.19 Canal::Integer::Interval Class Reference	79
16.19.1 Detailed Description	82
16.19.2 Constructor & Destructor Documentation	82
16.19.3 Member Function Documentation	82
16.19.4 Member Data Documentation	84
16.20 Canal::InterpreterBlock::Iterator Class Reference	84
16.20.1 Detailed Description	85
16.20.2 Member Function Documentation	85

16.20.3 Member Data Documentation	85
16.21 Canal::InterpreterBlock::IteratorCallback Class Reference	86
16.22 Canal::Widening::Manager Class Reference	86
16.23 Canal::InterpreterBlock::Module Class Reference	88
16.24 Canal::Widening::NumericalInfinity Class Reference	89
16.25 Canal::Operations Class Reference	90
16.25.1 Detailed Description	93
16.25.2 Member Function Documentation	94
16.26 Canal::InterpreterBlock::OperationsCallback Class Reference	96
16.26.1 Member Function Documentation	97
16.27 Canal::OperationsCallback Class Reference	98
16.27.1 Member Function Documentation	98
16.28 Canal::Pointer::Pointer Class Reference	99
16.28.1 Detailed Description	101
16.28.2 Constructor & Destructor Documentation	101
16.28.3 Member Function Documentation	101
16.28.4 Member Data Documentation	102
16.29 Canal::Widening::Pointers Class Reference	103
16.30 Canal::APIntUtils::SCompare Struct Reference	104
16.31 Canal::Array::SingleItem Class Reference	104
16.31.1 Detailed Description	107
16.31.2 Member Function Documentation	107
16.31.3 Member Data Documentation	107
16.32 Canal::SlotTracker Class Reference	107
16.32.1 Detailed Description	108
16.32.2 Member Function Documentation	108
16.33 Canal::State Class Reference	109
16.33.1 Detailed Description	110
16.33.2 Member Function Documentation	110
16.34 Canal::StateMap Class Reference	111
16.35 Canal::Structure Class Reference	111
16.35.1 Member Function Documentation	113
16.36 Canal::Pointer::Target Class Reference	113
16.36.1 Detailed Description	115
16.36.2 Constructor & Destructor Documentation	115
16.36.3 Member Function Documentation	115
16.36.4 Member Data Documentation	116
16.37 Canal::APIntUtils::UCompare Struct Reference	116
16.38 Canal::Pointer::Utils Class Reference	116
16.39 Canal::VariableArguments Class Reference	117
16.39.1 Member Function Documentation	117
16.40 Canal::VariablePrecisionDomain Class Reference	117
16.40.1 Detailed Description	117
16.40.2 Member Function Documentation	117
17 Tool Class Index	119
17.1 Class Hierarchy	119
17.2 Class List	119
18 Tool Class Documentation	121
18.1 Arguments Class Reference	121
18.2 Command Class Reference	123
18.3 CommandBreak Class Reference	125
18.4 CommandCd Class Reference	126

18.5 CommandContinue Class Reference	128
18.6 CommandDump Class Reference	129
18.7 CommandFile Class Reference	131
18.8 CommandFinish Class Reference	132
18.9 CommandHelp Class Reference	134
18.10 CommandInfo Class Reference	135
18.11 CommandPrint Class Reference	137
18.12 CommandPwd Class Reference	138
18.13 CommandQuit Class Reference	140
18.14 CommandRun Class Reference	141
18.15 Commands Class Reference	143
18.16 CommandShell Class Reference	144
18.17 CommandShow Class Reference	146
18.18 CommandStart Class Reference	147
18.19 CommandStep Class Reference	149
18.20 IteratorCallback Class Reference	150
18.21 State Class Reference	151
19 Known Bugs	153
20 Wishlist	155
20.1 Callbacks Interface	155
20.2 Models	155
20.3 Support of Multiple Platforms	155
20.4 Automatic Tests	155
20.5 Graphical User Interface	155
Bibliography	157
Index	159

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 static analysis tool designed to analyze behaviour of application programs. It is based on the theoretical framework of abstract interpretation, with focus on the scalability to large programs and proper handling of real-world source code.

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.

Canal comes with two build systems: Autotools (also known as GNU build system) and CMake. The systems are equivalent in terms of features, but they come with different advantages and disadvantages. Autotools system works on Unix-like operating systems only, and CMake also works on other operating systems such as Microsoft Windows. CMake tool is required to be installed to build Canal via CMake while Autotools require no tool to be present on the system. Supporting multiple versions of the CMake tool is not straightforward, but it is something that needs to be done when supporting CMake for multiple operating systems.

Building and installing Canal with Autotools

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. With default settings, the build system is configured to optimize the resulting code to make Canal run faster. If you need to debug Canal, set compiler flags to avoid any optimization:

```
./configure CXXFLAGS="-g -O0"
```

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 privileges might be needed. To install Canal, run

```
make install
```

Building and installing Canal with CMake

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 `cmake` program:

```
cmake .
```

It prints messages telling which dependencies it found.

Once the `cmake` tool 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 test
```

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.

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. If `elfutils-devel` is not present, the tool is built without the ELF support.
- The `doxygen`, `graphviz`, and `texlive-latex` packages. These are needed to build the documentation. If not present, the documentation is not built.

Both Autotools and CMake can be used to build Canal on Fedora and Red Hat Enterprise Linux.

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 $\mathbf{isz} \mid sz \in \mathbb{N}^*$, 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 $\{\overline{typ}_j\}$ are defined as a list of types. Functions *typ* \overline{typ}_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	<i>mod</i>	::= <i>layout</i> <i>asm</i> <i>namedt</i> <i>namedm</i> <i>alias</i> <i>prod</i>
Layouts	<i>layout</i>	::= bigendian little endian ptr <i>sz align_{abi} align_{pref}</i> int <i>sz align_{abi} align_{pref}</i> float <i>sz align_{abi} align_{pref}</i> aggr <i>sz align_{abi} align_{pref}</i> vec <i>sz align_{abi} align_{pref}</i> stack <i>sz align_{abi} align_{pref}</i>
Products	<i>prod</i>	::= <i>id = global typ const align</i> define <i>typ id{arg}</i> { <i>b</i> } declare <i>typ id{arg}</i>
Floats	<i>fp</i>	::= half float double x86_fp80 fp128 ppc_fp128
Vec types	<i>vtyp</i>	::= <i>fp</i> isz <i>fp*</i> isz*
Types	<i>typ</i>	::= isz <i>fp</i> void <i>typ*</i> [sz × typ] [sz × vtyp] {typ_j^j} <i>typ typ_j^j</i> <i>id</i> label metadata
Values	<i>val</i>	::= <i>id</i> <i>cnst</i>
Binops	<i>bop</i>	::= add sub mul udiv sdiv urem srem shl lshr ashr and or xor
Float ops	<i>fbop</i>	::= fadd fsub fmul fdiv frem
Extension	<i>eop</i>	::= zext sext fpext
Cast ops	<i>cop</i>	::= fptoui ptrtoint inttoptr bitcast
Trunc ops	<i>trop</i>	::= trunc_{int} trunc_{fp}
Constants	<i>cnst</i>	::= isz Int fp Float typ * id (typ*) null typ zeroinitializer typ[cnst_j^j] {cnst_j^j} typ undef bop cnst₁ cnst₂ fbop cnst₁ cnst₂ trop cnst to typ eop cnst to typ cop cnst to typ getelementptr cnst cnst_j^j select cnst₀ cnst₁ cnst₂ icmp cond cnst₁ cnst₂ fcmp fcond cnst₁ cnst₂
Blocks	<i>b</i>	::= <i>l φ c tmn</i>
φ nodes	<i>φ</i>	::= <i>id = phi typ [val_j, l_j]^j</i>
Tmns	<i>tmn</i>	::= br val l₁ l₂ br l ret typ val ret void unreachable
Commands	<i>c</i>	::= <i>id = bop (int sz) val₁ val₂</i> <i>id = fbop fp val₁ val₂</i> store typ val₁ val₂ align <i>id = malloc typ val align</i> free (typ*) val <i>id = alloca typ val align</i> <i>id = trop typ₁ val to typ₂</i> <i>id = eop typ₁ val to typ₂</i> <i>id = cop typ₁ val to typ₂</i> <i>id = select val₀ typ val₁ val₂</i> option id = call typ₀ val₀ param <i>id = icmp cond typ val₁ val₂</i> <i>id = fcmp fcond fp val₁ val₂</i> <i>id = getelementptr (typ*) val val_j^j</i> <i>id = load (typ*) val align</i> <i>id = extractelement [sz × vtyp] val₁ val₂</i> <i>id = insertelement [sz × vtyp] val₁ val₂ val₃</i>

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-sensite 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^\sharp

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

$$\mathcal{D}_i^\sharp \stackrel{\text{def}}{=} \{[l, h] \mid l, h \in \mathbb{Z} \cup \{\pm\infty\}\}$$

9.2 Integer Bitfield Domain \mathcal{D}_b^\sharp

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, privileges 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 reimplementation 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	77
Canal::Integer::Bitfield	46
Canal::Integer::Container	52
Canal::Integer::Enumeration	63
Canal::Integer::Interval	79
Canal::InterpreterBlock::BasicBlock	45
Canal::Constructors	51
Canal::Widening::DataInterface	57
Canal::Widening::DataIterationCount	58
Canal::Domain	59
Canal::Array::ExactSize	67
Canal::Array::SingleItem	104
Canal::Float::Interval	77
Canal::Integer::Bitfield	46
Canal::Integer::Container	52
Canal::Integer::Enumeration	63
Canal::Integer::Interval	79
Canal::Pointer::Pointer	99
Canal::Structure	111
Canal::Environment	67
Canal::InterpreterBlock::Function	71
Canal::FunctionModel	72
Canal::FunctionModelMAnager	73
Canal::Array::Interface	73
Canal::Array::ExactSize	67
Canal::Array::SingleItem	104
Canal::Structure	111
Canal::Widening::Interface	75
Canal::Widening::NumericalInfinity	89
Canal::Widening::Pointers	103
Canal::InterpreterBlock::Interpreter	75

Canal::InterpreterBlock::Iterator	84
Canal::InterpreterBlock::IteratorCallback	86
Canal::Widening::Manager	86
Canal::InterpreterBlock::Module	88
Canal::Operations	90
Canal::OperationsCallback	98
Canal::InterpreterBlock::OperationsCallback	96
Canal::APIntUtils::SCompare	104
Canal::SlofTracker	107
Canal::State	109
Canal::StateMap	111
Canal::Pointer::Target	113
Canal::APIntUtils::UCompare	116
Canal::Pointer::Utils	116
Canal::VariableArguments	117
Canal::VariablePrecisionDomain	117

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::Widening::DataInterface	57
Canal::Widening::DataIterationCount	58
Canal::Domain	
Base class for all abstract domains	59
Canal::Integer::Enumeration	63
Canal::Environment	67
Canal::Array::ExactSize	67
Canal::InterpreterBlock::Function	71
Canal::FunctionModel	72
Canal::FunctionModelMAnager	73
Canal::Array::Interface	73
Canal::Widening::Interface	75
Canal::InterpreterBlock::Interpreter	75
Canal::Float::Interval	77
Canal::Integer::Interval	
Abstracts integer values as a interval min - max	79
Canal::InterpreterBlock::Iterator	84
Canal::InterpreterBlock::IteratorCallback	86
Canal::Widening::Manager	86
Canal::InterpreterBlock::Module	88
Canal::Widening::NumericalInfinity	89
Canal::Operations	90
Canal::InterpreterBlock::OperationsCallback	96
Canal::OperationsCallback	98

Canal::Pointer::Pointer	
Inclusion-based flow-insensitive abstract pointer	99
Canal::Widening::Pointers	103
Canal::APIntUtils::SCompare	104
Canal::Array::SingleItem	
This array type is very imprecise	104
Canal::SlotTracker	107
Canal::State	
Abstract memory state	109
Canal::StateMap	111
Canal::Structure	111
Canal::Pointer::Target	113
Canal::APIntUtils::UCompare	116
Canal::Pointer::Utils	116
Canal::VariableArguments	117
Canal::VariablePrecisionDomain	
Base class for abstract domains that can lower the precision and memory requirements on demand	117

Chapter 16

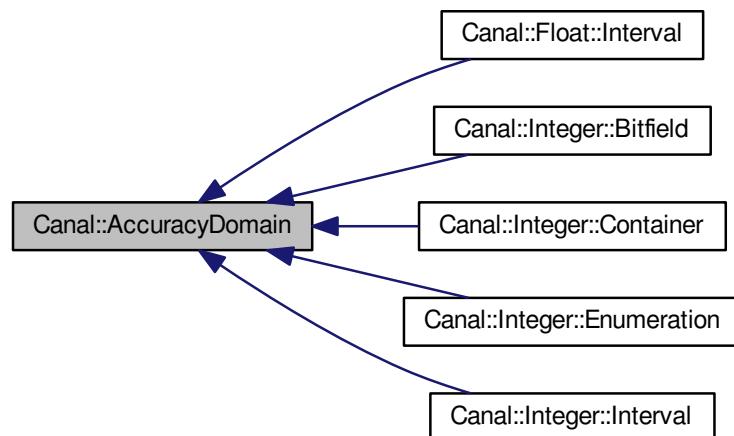
Library Class Documentation

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).

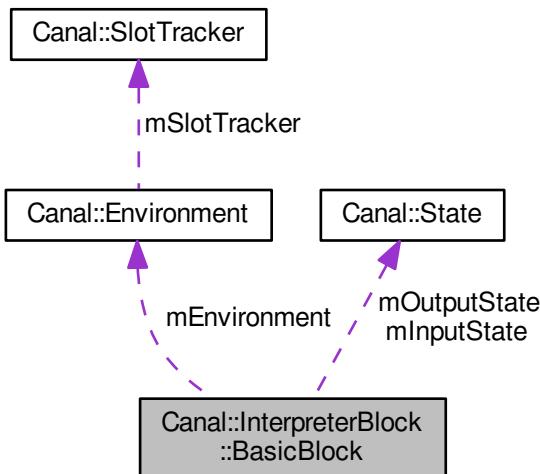
Reimplemented in Canal::Integer::Bitfield, Canal::Integer::Interval, Canal::Integer::Container, Canal::Integer::Enumeration, and Canal::Float::Interval.

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 & **getLlvmBasicBlock** () const
- llvm::BasicBlock::const_iterator **begin** () const
- llvm::BasicBlock::const_iterator **end** () const
- State & **getInputState** ()
- State & **getOutputState** ()
- std::string **toString** () const

Protected Attributes

- const llvm::BasicBlock & **mBasicBlock**
- const Environment & **mEnvironment**
- State **mInputState**
- State **mOutputState**

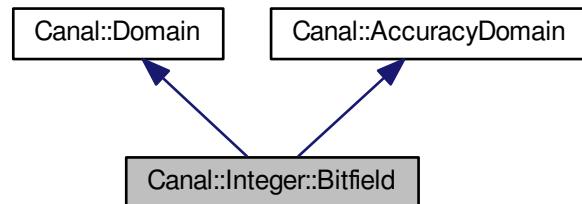
The documentation for this class was generated from the following files:

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

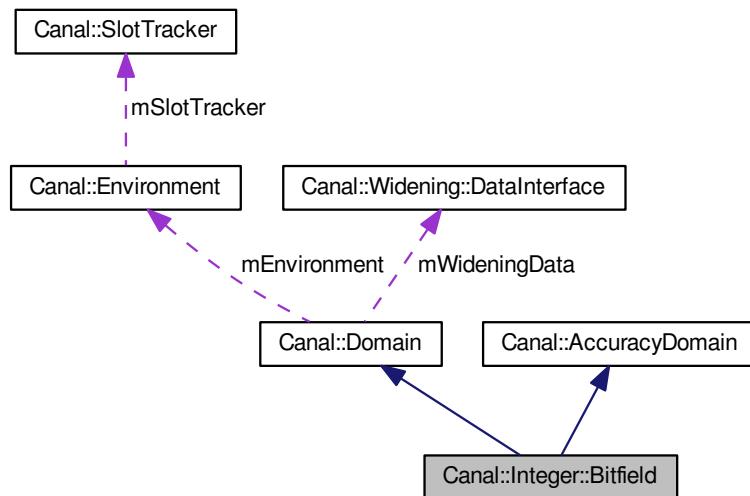
16.3 Canal::Integer::Bitfield Class Reference

```
#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.

- Bitfield (const Bitfield &value)
Copy constructor.
- 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 void setZero (const llvm::Value *place)
Implementation of Domain::setZero().
- 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 void trunc (const Domain &value)
- virtual void zext (const Domain &value)
- virtual void sext (const Domain &value)
- virtual void fptoui (const Domain &value)
- virtual void fptosi (const Domain &value)
- 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

Additional Inherited Members

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

<i>result</i>	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

<i>result</i>	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

<i>result</i>	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

<i>result</i>	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

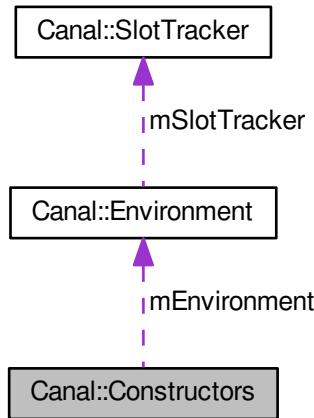
When a bit in mZeroes is 1, the value is known to contain zero at this position.

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)
- const Environment & **getEnvironment** () const
- Domain * **create** (const llvm::Type &type) const
- Domain * **create** (const llvm::Constant &value, const llvm::Value &place, const State *state) const
- Domain * **createInteger** (unsigned bitWidth) const
- Domain * **createInteger** (const llvm::APInt &number) const
- Domain * **createFloat** (const llvm::fltSemantics &semantics) const
- Domain * **createFloat** (const llvm::APFloat &number) const
- Domain * **createArray** (Domain *size, Domain *value) const
- Domain * **createArray** (uint64_t size, const Domain &value) const
- Domain * **createArray** (const std::vector< Domain * > &values) const
- Domain * **createPointer** (const llvm::Type &type) const

Static Public Member Functions

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

Protected Member Functions

- Domain * **createGetElementPtr** (const llvm::ConstantExpr &value, const Domain &variable, const llvm::Value &place) const
- Domain * **createBitCast** (const llvm::ConstantExpr &value, const Domain &variable, const llvm::Value &place) const

Protected Attributes

- const Environment & **mEnvironment**

16.4.1 Member Function Documentation

16.4.1.1 Domain * Canal::Constructors::create (const llvm::Constant & value, const llvm::Value & place, const State * state) const

Parameters

<i>state</i>	State is used only for constant expressions such as getelementptr and bitcast. For other types of constants it might be NULL.
--------------	---

Returns

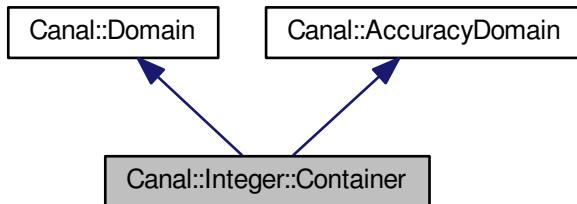
Returns a newly allocated value or NULL. Caller takes ownership of the returned value.

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

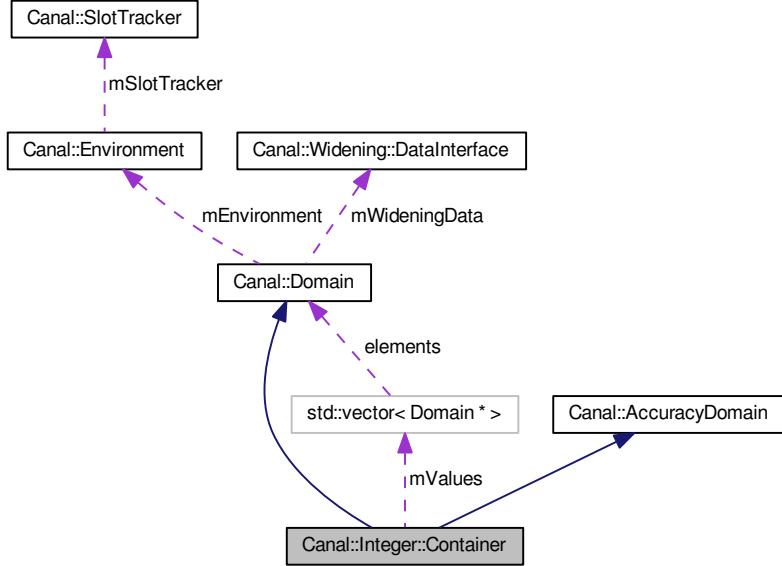
- lib/Constructors.h
- lib/Constructors.cpp

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 &value)

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 (llvm::APInt &result) const
- bool signedMax (llvm::APInt &result) const
- bool unsignedMin (llvm::APInt &result) const
- bool unsignedMax (llvm::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)

- virtual size_t memoryUsage () const

Implementation of Domain::memoryUsage().
- virtual std::string toString () const

Implementation of Domain::toString().
- virtual void setZero (const llvm::Value *place)

Implementation of Domain::setZero().
- 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 ashtr (const Domain &a, const Domain &b)

Implementation of Domain::ashtr().
- 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 void trunc (const Domain &value)
- virtual void zext (const Domain &value)
- virtual void sext (const Domain &value)
- virtual void fptoui (const Domain &value)
- virtual void fptosi (const Domain &value)
- virtual float accuracy () const

Implementation of AccuracyDomain::accuracy().
- virtual bool isBottom () const

Implementation of AccuracyDomain::isBottom().
- virtual void setBottom ()

Implementation of AccuracyDomain::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**

Additional Inherited Members

16.5.1 Constructor & Destructor Documentation

16.5.1.1 **Canal::Integer::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::signedMax (llvm::APInt & result) const**

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

Parameters

<i>result</i>	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.

16.5.2.4 bool Canal::Integer::Container::signedMin (`Ilvm::APInt & result`) const

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

Parameters

<code>result</code>	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.

16.5.2.5 bool Canal::Integer::Container::unsignedMax (`Ilvm::APInt & result`) const

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

Parameters

<code>result</code>	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.

16.5.2.6 bool Canal::Integer::Container::unsignedMin (`Ilvm::APInt & result`) const

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

Parameters

<code>result</code>	Filled by the minimum value if it is known. Otherwise, the value is undefined.
---------------------	--

Returns

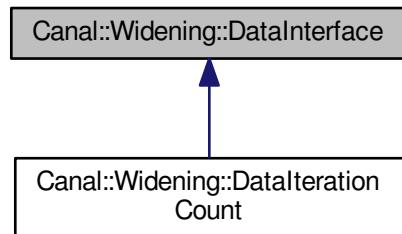
True if the result is known and the parameter was set to correct value.

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

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

16.6 Canal::Widening::DataInterface Class Reference

Inheritance diagram for Canal::Widening::DataInterface:



Public Member Functions

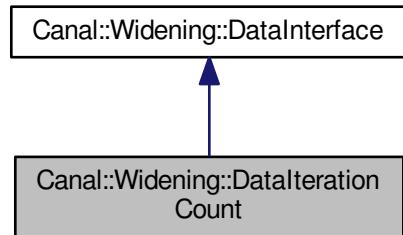
- virtual DataInterface * **clone** () const =0

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

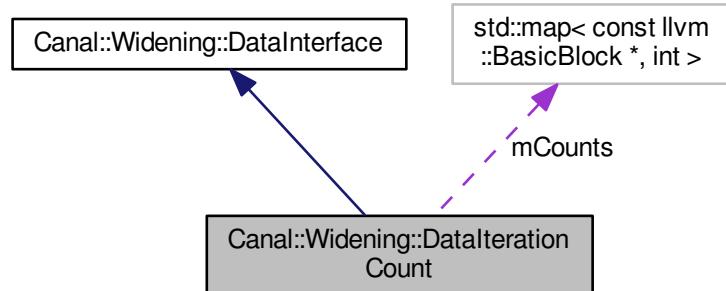
- lib/WideningDataInterface.h

16.7 Canal::Widening::DataIterationCount Class Reference

Inheritance diagram for Canal::Widening::DataIterationCount:



Collaboration diagram for Canal::Widening::DataIterationCount:



Public Member Functions

- void **increase** (const llvm::BasicBlock &block)
- int **count** (const llvm::BasicBlock &block) const
- virtual DataIterationCount * **clone** () const

Protected Attributes

- std::map<const llvm::BasicBlock *, int > **mCounts**

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

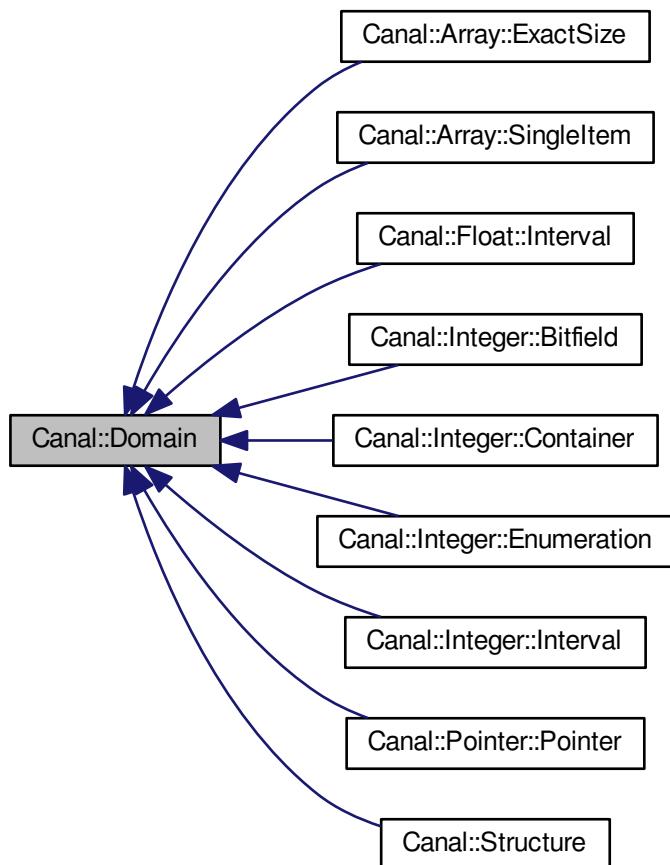
- lib/WideningDataIterationCount.h
- lib/WideningDataIterationCount.cpp

16.8 Canal::Domain Class Reference

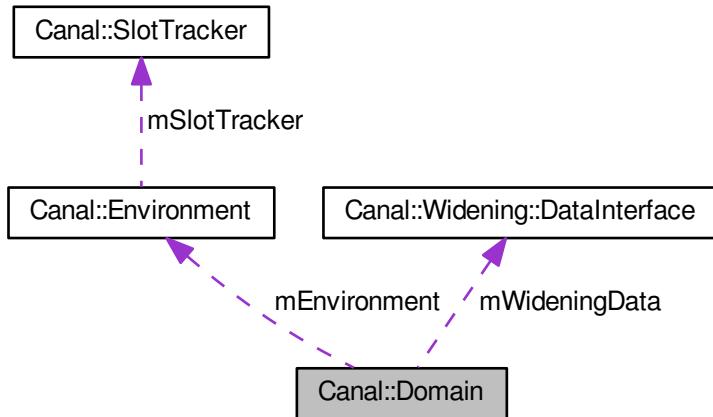
Base class for all abstract domains.

```
#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.
- `Domain (const Domain &value)`
- `virtual ~Domain ()`
Virtual destructor.
- `const Environment & getEnvironment () const`
- `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)=0`
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 void setZero (const llvm::Value *place)=0
- virtual Widening::DataInterface * **getWideningData** () const
- virtual void setWideningData (Widening::DataInterface *wideningData)

This class takes ownership of the wideningData memory.

- 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 **fptoui** (const Domain &value)
- virtual void **fptosi** (const Domain &value)
- virtual void **uitofp** (const Domain &value)
- virtual void **sitofp** (const Domain &value)

Public Attributes

- const Environment & **mEnvironment**
- Widening::DataInterface * **mWideningData**

16.8.1 Detailed Description

Base class for all abstract domains.

16.8.2 Constructor & Destructor Documentation

16.8.2.1 `Canal::Domain::Domain (const Domain & value)`

Copy constructor. Careful! Copy constructor of base class is not called by automatically generated copy constructor of an inherited class.

16.8.3 Member Function Documentation

16.8.3.1 `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::Pointer::Pointer`, `Canal::Integer::Bitfield`, `Canal::Integer::Interval`, `Canal::Integer::Container`, `Canal::Integer::Enumeration`, `Canal::Float::Interval`, `Canal::Array::ExactSize`, `Canal::Array::SingleItem`, and `Canal::Structure`.

16.8.3.2 `virtual bool Canal::Domain::operator== (const Domain & value) const [pure virtual]`

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

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

16.8.3.3 `virtual void Canal::Domain::setZero (const llvm::Value * place) [pure virtual]`

Set value of this domain to represent zeroed memory. Needed for constants with zero initializer.

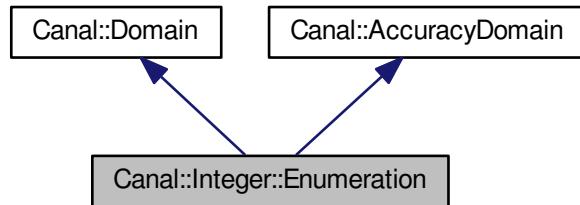
Implemented in `Canal::Pointer::Pointer`, `Canal::Integer::Bitfield`, `Canal::Integer::Interval`, `Canal::Integer::Container`, `Canal::Integer::Enumeration`, `Canal::Float::Interval`, `Canal::Array::ExactSize`, `Canal::Array::SingleItem`, and `Canal::Structure`.

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

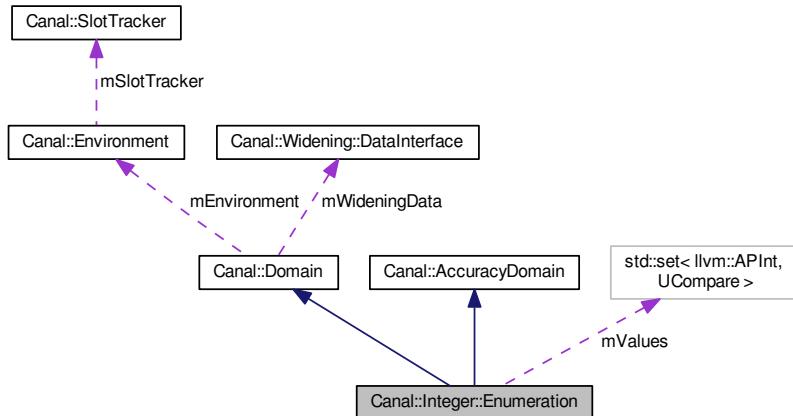
- lib/Domain.h
- lib/Domain.cpp

16.9 Canal::Integer::Enumeration Class Reference

Inheritance diagram for Canal::Integer::Enumeration:



Collaboration diagram for Canal::Integer::Enumeration:



Public Member Functions

- **Enumeration (const Environment &environment, unsigned bitWidth)**
Initializes to the lowest value.
- **Enumeration (const Environment &environment, const llvm::APInt &number)**
Initializes to the given value.
- **Enumeration (const Enumeration &value)**
Copy 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

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 void setZero (const llvm::Value *place)

Implementation of Domain::setZero().
- 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 void **trunc** (const Domain &value)
- virtual void **zext** (const Domain &value)
- virtual void **sext** (const Domain &value)
- virtual void **fptoui** (const Domain &value)
- virtual void **fptosi** (const Domain &value)
- 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)

Additional Inherited Members

16.9.1 Member Function Documentation

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

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

Implements Canal::Domain.

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

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

Implements Canal::Domain.

16.9.1.3 bool Canal::Integer::Enumeration::signedMax (`Ilvm::APInt & result`) const

Highest signed number represented by this abstract domain.

Parameters

<code>result</code>	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.

16.9.1.4 bool Canal::Integer::Enumeration::signedMin (`Ilvm::APInt & result`) const

Lowest signed number represented by this abstract domain.

Parameters

<code>result</code>	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.

16.9.1.5 bool Canal::Integer::Enumeration::unsignedMax (`Ilvm::APInt & result`) const

Highest unsigned number represented by this abstract domain.

Parameters

<code>result</code>	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.

16.9.1.6 bool Canal::Integer::Enumeration::unsignedMin (`Ilvm::APInt & result`) const

Lowest unsigned number represented by this abstract domain.

Parameters

<code>result</code>	Filled by the minimum value if it is known. Otherwise, the value is undefined.
---------------------	--

Returns

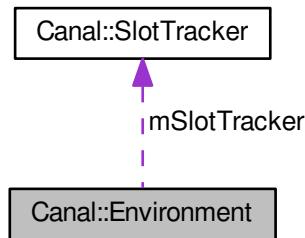
True if the result is known and the parameter was set to correct value.

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

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

16.10 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**

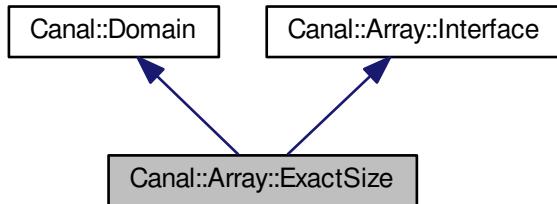
The documentation for this class was generated from the following files:

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

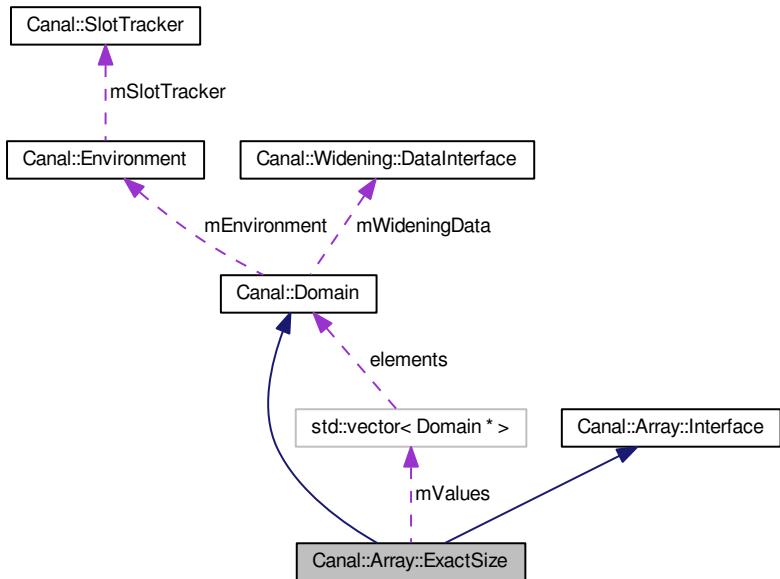
16.11 Canal::Array::ExactSize Class Reference

```
#include <ArrayExactSize.h>
```

Inheritance diagram for Canal::Array::ExactSize:



Collaboration diagram for Canal::Array::ExactSize:



Public Member Functions

- `ExactSize (const Environment &environment, const uint64_t size, const Domain &value)`
- `ExactSize (const Environment &environment, const std::vector< Domain * > &values)`
- `ExactSize (const ExactSize &value)`

Copy constructor.

- `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 void setZero (const llvm::Value *place)
 - Implementation of Domain::setZero()*
- 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)

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

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

Implementation of Domain::icmp().
- 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**

Additional Inherited Members

16.11.1 Detailed Description

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

16.11.2 Constructor & Destructor Documentation

16.11.2.1 **Canal::Array::ExactSize::ExactSize (const Environment & environment, const uint64_t size, const Domain & value)**

Parameters

<i>value</i>	This class does not take ownership of this value.
--------------	---

16.11.2.2 **Canal::Array::ExactSize::ExactSize (const Environment & environment, const std::vector< Domain * > & values)**

Parameters

<i>values</i>	This class takes ownership of the values.
---------------	---

16.11.3 Member Function Documentation

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

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

Implements Canal::Domain.

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

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

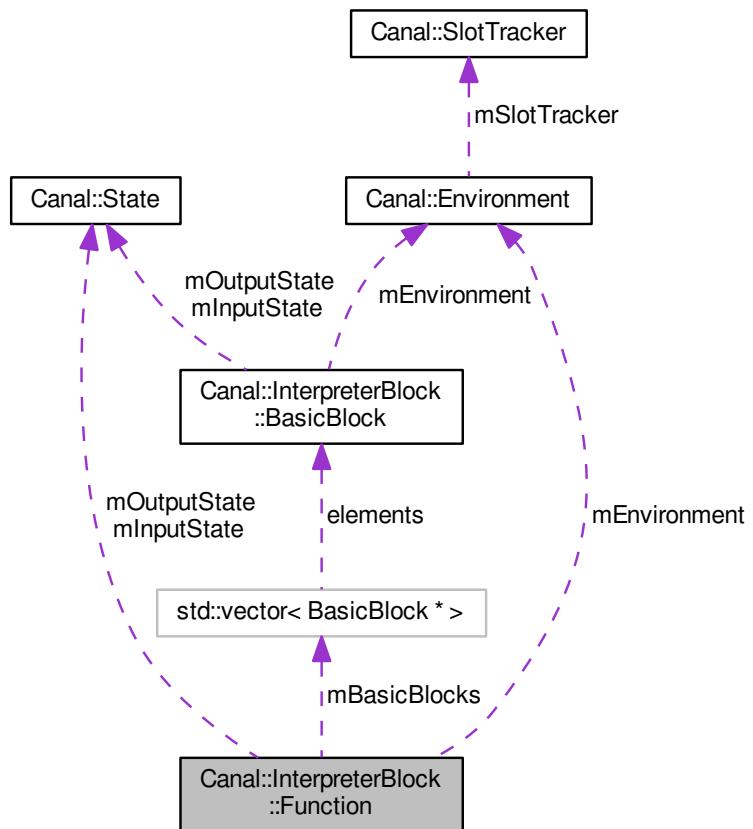
Implements Canal::Domain.

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

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

16.12 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 & **getLlvmFunction** () const
- const llvm::BasicBlock & **getLlvmEntryBlock** () 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.
- std::string **toString** () const

Protected Attributes

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

16.12.1 Member Function Documentation

16.12.1.1 void Canal::InterpreterBlock::Function::updateBasicBlockInputState (BasicBlock & *basicBlock*)

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

Parameters

<i>basicBlock</i>	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

16.13 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

16.14 Canal::FunctionModelManager Class Reference

Public Member Functions

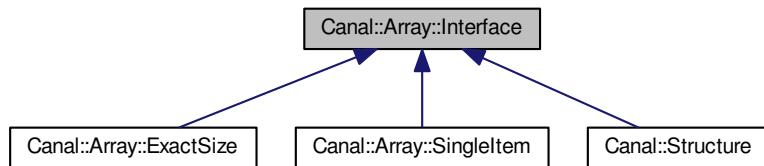
- bool **canHandle** (llvm::Function *function, bool implementationAvailable)
- void **handle** (llvm::Function *function, State &state)

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

- lib/FunctionModelManager.h

16.15 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

16.15.1 Member Function Documentation

16.15.1.1 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.

16.15.1.2 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.

16.15.1.3 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.

16.15.1.4 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.

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

Parameters

<i>value</i>	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.

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

Parameters

<i>value</i>	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.

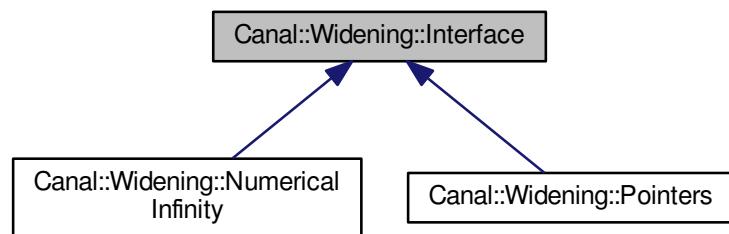
Implemented in Canal::Array::ExactSize, Canal::Array::SingleItem, and Canal::Structure.

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

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

16.16 Canal::Widening::Interface Class Reference

Inheritance diagram for Canal::Widening::Interface:



Public Member Functions

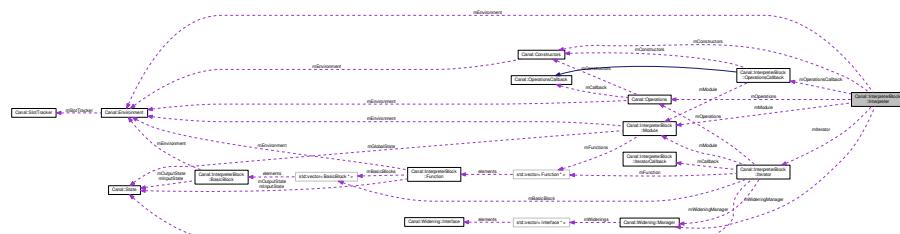
- virtual void **widen** (const llvm::BasicBlock &wideningPoint, Domain &first, const Domain &second)=0

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

- lib/WideningInterface.h

16.17 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
- std::string **toString** () const

Protected Attributes

- Environment **mEnvironment**
- Constructors **mConstructors**
- Module **mModule**
- OperationsCallback **mOperationsCallback**
- Operations **mOperations**
- Widening::Manager **mWideningManager**
- Iterator **mIterator**

16.17.1 Constructor & Destructor Documentation

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

Parameters

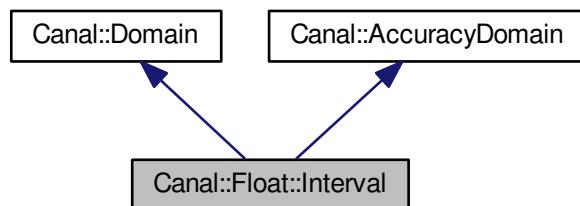
<i>module</i>	Interpreter takes ownership of the module.
---------------	--

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

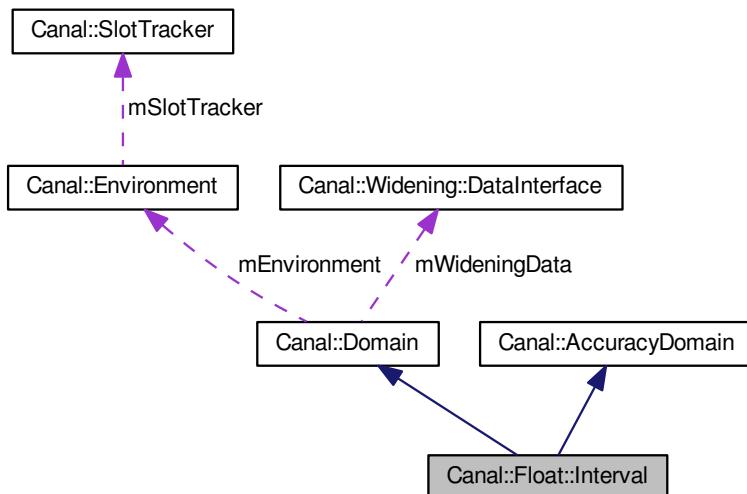
- lib/InterpreterBlock.h
- lib/InterpreterBlock.cpp

16.18 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)
- **Interval** (const Environment &environment, const llvm::APFloat &number)
- **Interval** (const Interval &value)

Copy constructor.

- 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)

Merge another value into this one.

- 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 void **setZero** (const llvm::Value *place)

Implementation of Domain::setZero().

- 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.

- virtual void **fadd** (const Domain &a, const Domain &b)
- virtual void **fsub** (const Domain &a, const Domain &b)
- virtual void **fmul** (const Domain &a, const Domain &b)
- virtual void **fdiv** (const Domain &a, const Domain &b)

Floating point division.

- virtual void **frem** (const Domain &a, const Domain &b)
- virtual void **uitofp** (const Domain &value)
- virtual void **sitofp** (const Domain &value)

Public Attributes

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

Additional Inherited Members

16.18.1 Member Function Documentation

16.18.1.1 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.

16.18.1.2 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.

16.18.1.3 bool Canal::Float::Interval::operator==(const Domain & value) const [virtual]

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

Implements Canal::Domain.

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

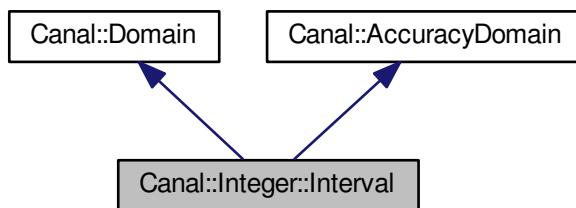
- lib/FloatInterval.h
- lib/FloatInterval.cpp

16.19 Canal::Integer::Interval Class Reference

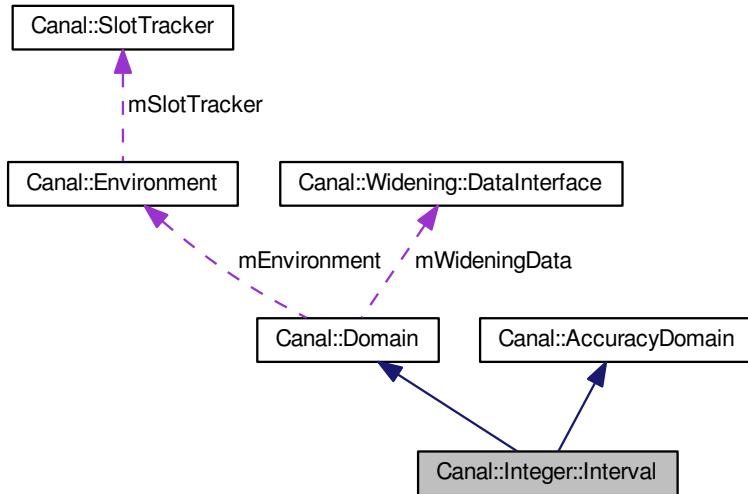
Abstracts integer values as a interval min - max.

```
#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.
- `Interval (const Interval &value)`

Copy 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`

- virtual std::string **toString** () const
 - Implementation of Domain::toString().*
- virtual void **setZero** (const llvm::Value *place)
 - Implementation of Domain::setZero().*
- 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 void **trunc** (const Domain &value)
- virtual void **zext** (const Domain &value)
- virtual void **sext** (const Domain &value)
- virtual void **fptoui** (const Domain &value)
- virtual void **fptosi** (const Domain &value)
- 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.

Additional Inherited Members

16.19.1 Detailed Description

Abstracts integer values as a interval min - max.

16.19.2 Constructor & Destructor Documentation

16.19.2.1 `Canal::Integer::Interval::Interval (const Environment & environment, unsigned bitWidth)`

Standard constructor.

Initializes an empty interval.

16.19.3 Member Function Documentation

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

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

Implements Canal::Domain.

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

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

Implements Canal::Domain.

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

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

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

Highest signed number represented by this abstract domain.

Parameters

<code>result</code>	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.

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

Lowest signed number represented by this abstract domain.

Parameters

<code>result</code>	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.

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

Highest unsigned number represented by this abstract domain.

Parameters

<code>result</code>	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.

16.19.3.7 bool Canal::Integer::Interval::unsignedMin (`llvm::APInt & result`) const

Lowest unsigned number represented by this abstract domain.

Parameters

<code>result</code>	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.

16.19.4 Member Data Documentation

16.19.4.1 bool Canal::Integer::Interval::mEmpty

Indicates an empty interval.

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

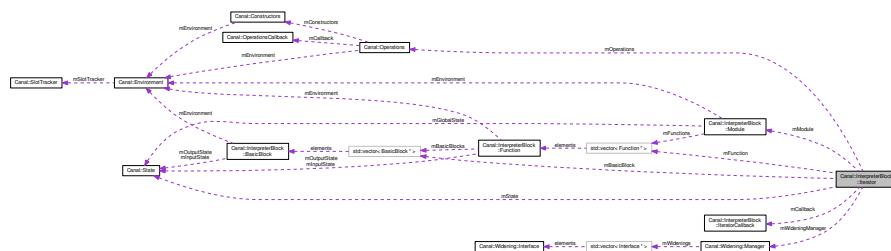
The documentation for this class was generated from the following files:

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

16.20 Canal::InterpreterBlock::Iterator Class Reference

```
#include <InterpreterBlockIterator.h>
```

Collaboration diagram for Canal::InterpreterBlock::Iterator:



Public Member Functions

- **Iterator** (Module &module, Operations &operations, Widening::Manager &wideningManager)
 - 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
 - std::string **toString** () const

Protected Member Functions

- void **nextInstruction** ()

Protected Attributes

- Module & mModule

- Operations & **mOperations**
- Widening::Manager & **mWideningManager**
- 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.

16.20.1 Detailed Description

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

16.20.2 Member Function Documentation

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

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

16.20.3 Member Data Documentation

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

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

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

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

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

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

16.20.3.4 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

16.21 Canal::InterpreterBlock::IteratorCallback Class Reference

Public Member Functions

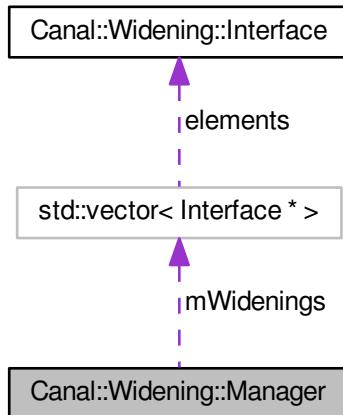
- virtual void **onFixpointReached** ()
- virtual void **onModuleEnter** ()
- virtual void **onModuleExit** ()
- virtual void **onFunctionEnter** (Function &function)
- virtual void **onFunctionExit** (Function &function)
- virtual void **onBasicBlockEnter** (BasicBlock &basicBlock)
- virtual void **onBasicBlockExit** (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

16.22 Canal::Widening::Manager Class Reference

Collaboration diagram for Canal::Widening::Manager:



Public Member Functions

- void **widen** (const llvm::BasicBlock &wideningPoint, State &first, const State &second) const

Protected Member Functions

- void **widen** (const llvm::BasicBlock &wideningPoint, StateMap &first, const StateMap &second)
const

- void **widen** (const llvm::BasicBlock &wideningPoint, Domain &first, const Domain &second) const

Protected Attributes

- std::vector< Interface * > **mWidenings**

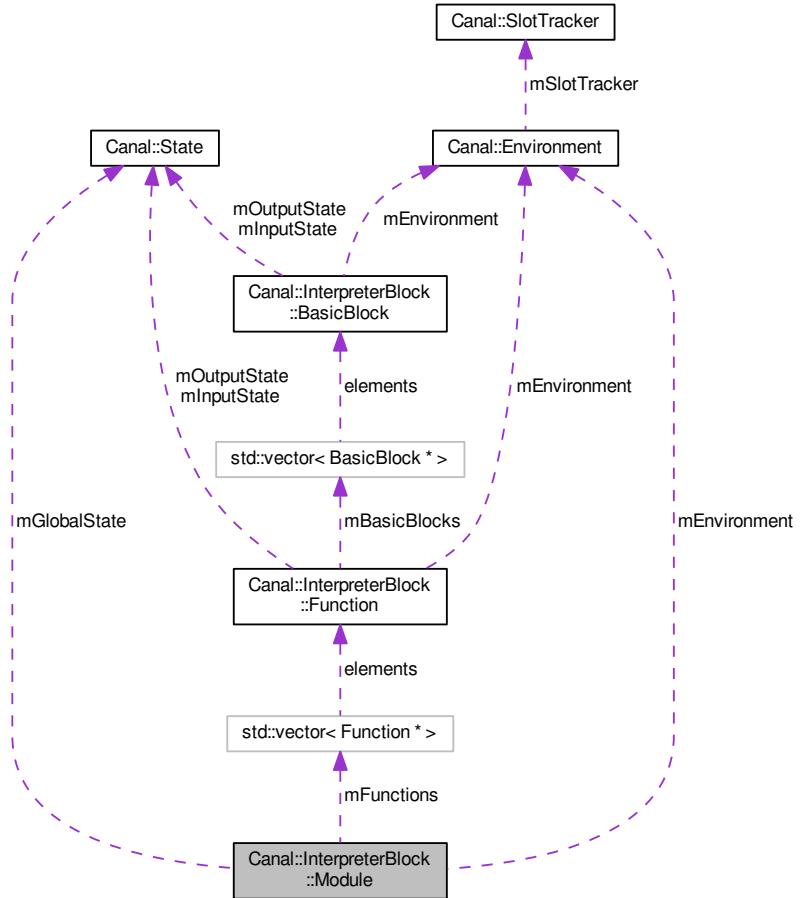
The documentation for this class was generated from the following files:

- lib/WideningManager.h

- lib/WideningManager.cpp

16.23 Canal::InterpreterBlock::Module Class Reference

Collaboration diagram for Canal::InterpreterBlock::Module:



Classes

- struct **tsortValue**

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
- std::string **toString** () const
- void **updateGlobalState** ()

Protected Attributes

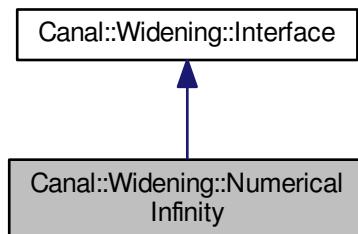
- const llvm::Module & **mModule**
- const Environment & **mEnvironment**
- std::vector< Function * > **mFunctions**
- State **mGlobalState**

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

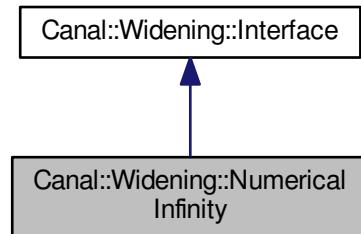
- lib/InterpreterBlockModule.h
- lib/InterpreterBlockModule.cpp

16.24 Canal::Widening::NumericalInfinity Class Reference

Inheritance diagram for Canal::Widening::NumericalInfinity:



Collaboration diagram for Canal::Widening::NumericalInfinity:



Public Member Functions

- virtual void **widen** (const llvm::BasicBlock &wideningPoint, Domain &first, const Domain &second)

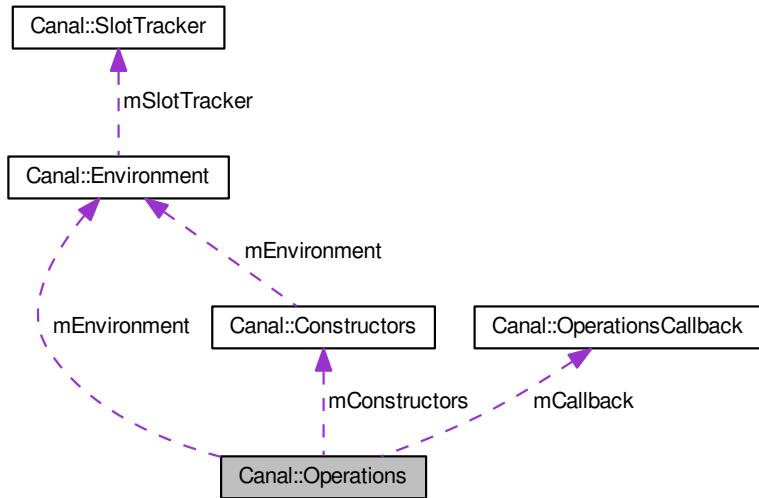
The documentation for this class was generated from the following files:

- lib/WideningNumericalInfinity.h
- lib/WideningNumericalInfinity.cpp

16.25 Canal::Operations Class Reference

```
#include <Operations.h>
```

Collaboration diagram for Canal::Operations:



Public Member Functions

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

Interprets current instruction.

Protected Member Functions

- Domain * **variableOrConstant** (const llvm::Value &place, State &state, const llvm::Instruction &instruction, 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 llvm::Value &place, 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)

- 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 access operation.
- virtual void trunc (const llvm::TruncInst &instruction, State &state)

It's a memory addressing 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::UIToFPIInst &instruction, State &state)

It's a conversion operation.
- virtual void sitofp (const llvm::SIToFPIInst &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::VAArgInst &instruction, State &state)

Protected Attributes

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

16.25.1 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.

16.25.2 Member Function Documentation

16.25.2.1 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.

16.25.2.2 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.

16.25.2.3 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.

16.25.2.4 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.

16.25.2.5 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.

16.25.2.6 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.

16.25.2.7 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.

16.25.2.8 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.

16.25.2.9 `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.

16.25.2.10 `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.

16.25.2.11 `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.

16.25.2.12 `Domain * Canal::Operations::variableOrConstant (const llvm::Value & place, State & state, const llvm::Instruction & instruction, 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

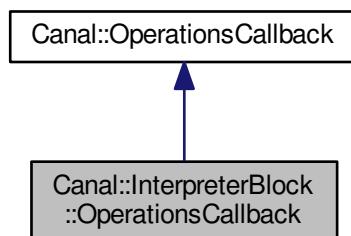
Returns a pointer to the variable if it is found in the state. Returns a pointer to the provided constant if the place contains a constant. Otherwise, it returns NULL.

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

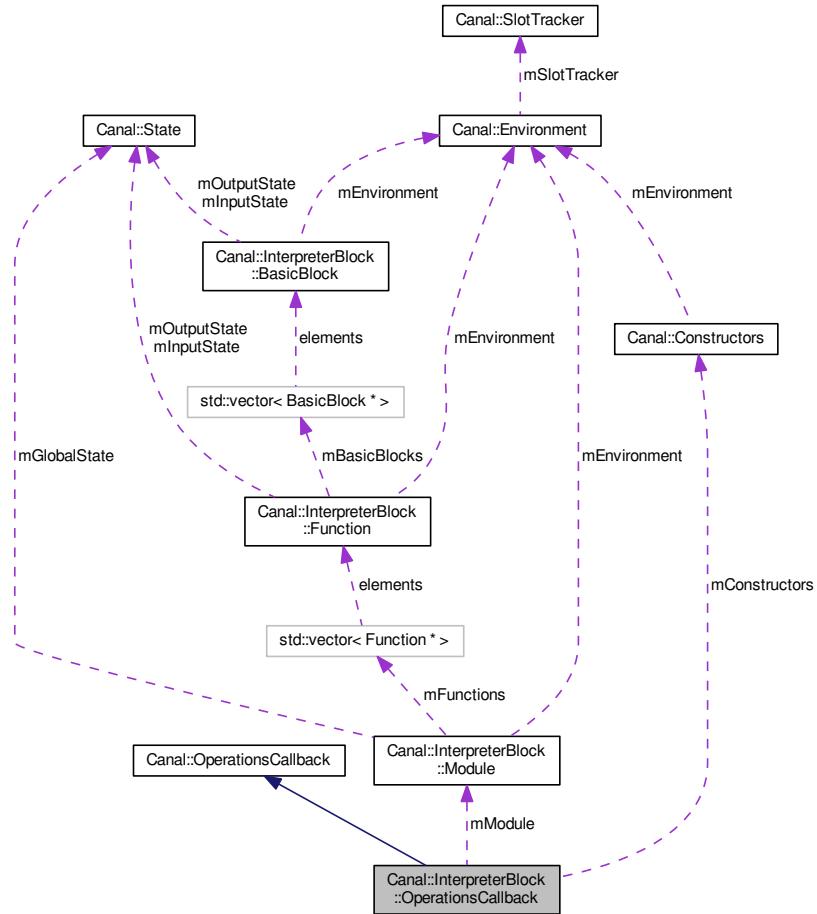
- lib/Operations.h
- lib/Operations.cpp

16.26 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**

16.26.1 Member Function Documentation

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

Parameters

<i>callState</i>	Contains function arguments and referenced memory blocks.
<i>resultState</i>	A state where the changes caused by the function can be merged.
<i>resultPlace</i>	A place in the <i>resultState</i> where the returned value should be merged.

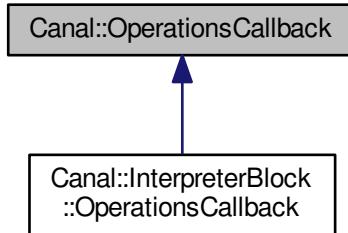
Implements Canal::OperationsCallback.

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

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

16.27 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

16.27.1 Member Function Documentation

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

Parameters

<i>callState</i>	Contains function arguments and referenced memory blocks.
<i>resultState</i>	A state where the changes caused by the function can be merged.
<i>resultPlace</i>	A place in the <i>resultState</i> where the returned value should be merged.

Implemented in Canal::InterpreterBlock::OperationsCallback.

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

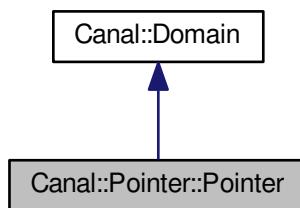
- lib/OperationsCallback.h

16.28 Canal::Pointer::Pointer Class Reference

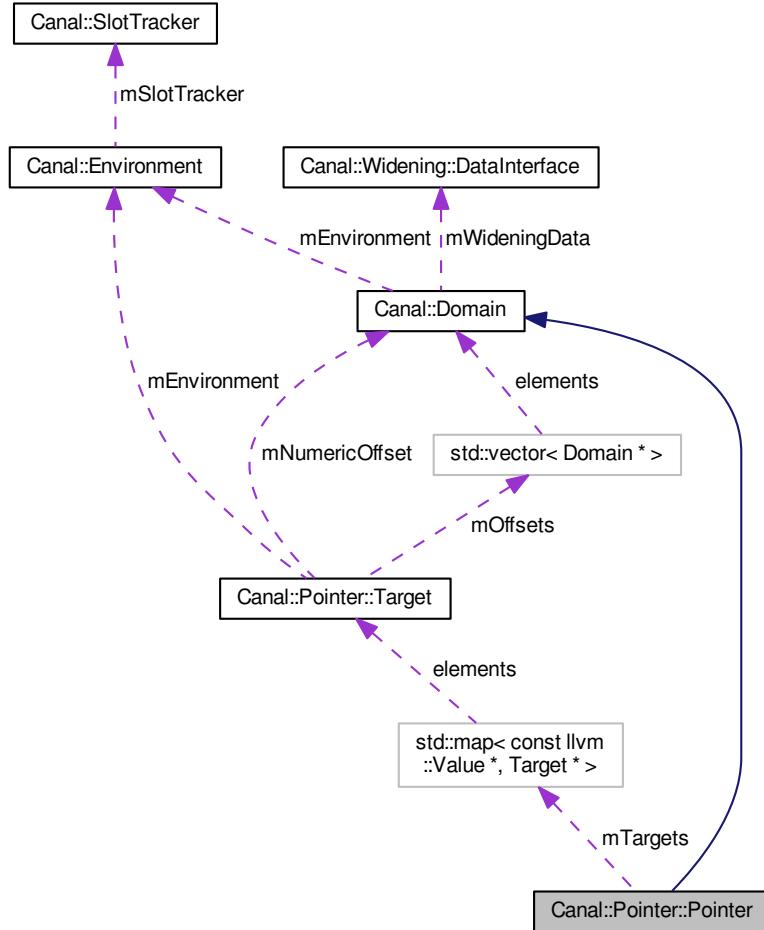
Inclusion-based flow-insensitive abstract pointer.

```
#include <Pointer.h>
```

Inheritance diagram for Canal::Pointer::Pointer:



Collaboration diagram for Canal::Pointer::Pointer:



Public Member Functions

- `Pointer (const Environment &environment, const llvm::Type &type)`

Standard constructor.
- `Pointer (const Pointer &value)`

Copy constructor.
- `Pointer (const Pointer &value, const llvm::Type &newType)`
- `virtual ~Pointer ()`

Standard destructor.
- `void addTarget (Target::Type type, const llvm::Value *place, const llvm::Value *target, const std::vector< Domain * > &offsets, Domain *numericOffset)`
- `Domain * dereferenceAndMerge (const State &state) const`
- `Pointer * bitcast (const llvm::Type &type) const`

Creates a copy of this object with a different pointer type.

- Pointer * getElementPtr (const std::vector< Domain * > &offsets, const llvm::Type &type) const
- void **store** (const Domain &value, State &state)
- virtual Pointer * clone () const
- virtual Pointer * 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 void setZero (const llvm::Value *place)
Implementation of Domain::setZero().

Public Attributes

- PlaceTargetMap mTargets
- const llvm::Type & mType

The type object is owned by the LLVM framework.

Additional Inherited Members

16.28.1 Detailed Description

Inclusion-based flow-insensitive abstract pointer.

16.28.2 Constructor & Destructor Documentation

16.28.2.1 Canal::Pointer::Pointer (const Pointer & value, const llvm::Type & newType)

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

16.28.3 Member Function Documentation

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

Add a new target to the pointer.

Parameters

<i>type</i>	Type of the referenced memory.
<i>place</i>	Place where the pointer target is added.
<i>target</i>	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.
<i>offsets</i>	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.
<i>numericOffset</i>	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.

16.28.3.2 Pointer * Canal::Pointer::Pointer::clone() const [virtual]

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

Implements Canal::Domain.

16.28.3.3 Pointer * Canal::Pointer::Pointer::cloneCleaned() const [virtual]

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

Implements Canal::Domain.

16.28.3.4 Domain * Canal::Pointer::Pointer::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.

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

Creates a copy of this object pointing to subtargets.

Parameters

<i>offsets</i>	Pointer takes ownership of the values inside the vector. The offsets must be converted to 64-bit integers before calling getElementPtr!
----------------	---

16.28.4 Member Data Documentation

16.28.4.1 PlaceTargetMap Canal::Pointer::Pointer::mTargets

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

16.28.4.2 const llvm::Type& Canal::Pointer::Pointer::mType

The type object is owned by the LLVM framework.

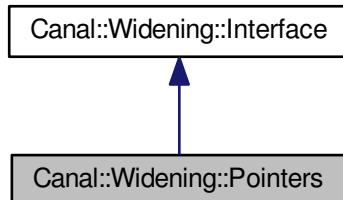
Type of the object the pointer is pointing to. It might be incompatible with the type of the actual abstract value. Conversion is needed during store and load operations in such a case.

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

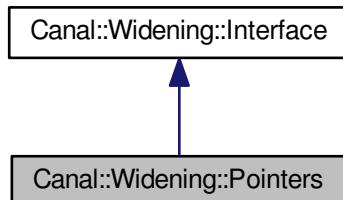
- lib/Pointer.h
- lib/Pointer.cpp

16.29 Canal::Widening::Pointers Class Reference

Inheritance diagram for Canal::Widening::Pointers:



Collaboration diagram for Canal::Widening::Pointers:



Public Member Functions

- virtual void **widen** (const llvm::BasicBlock &wideningPoint, Domain &first, const Domain &second)

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

- lib/WideningPointers.h
- lib/WideningPointers.cpp

16.30 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:

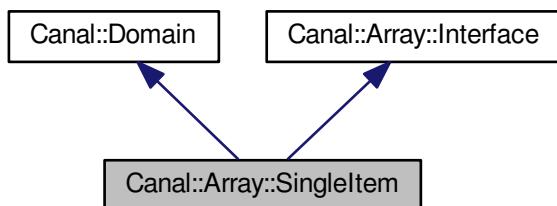
- lib/APIntUtils.h

16.31 Canal::Array::SingleItem Class Reference

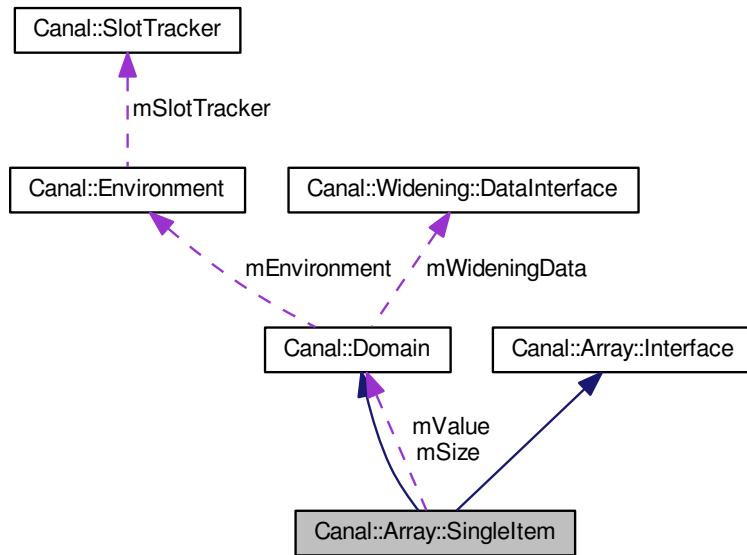
This array type is very imprecise.

```
#include <ArraySingleItem.h>
```

Inheritance diagram for Canal::Array::SingleItem:



Collaboration diagram for Canal::Array::SingleItem:



Public Member Functions

- **SingleItem** (const Environment &environment, Domain *size, Domain *value)
- **SingleItem** (const SingleItem &value)
- 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 void setZero (const llvm::Value *place)

Implementation of Domain::setZero().
- 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)

- 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

Additional Inherited Members

16.31.1 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.

16.31.2 Member Function Documentation

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

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

Implements Canal::Domain.

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

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

Implements Canal::Domain.

16.31.3 Member Data Documentation

16.31.3.1 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

16.32 Canal::SlotTracker Class Reference

```
#include <SlotTracker.h>
```

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** (unsigned num)
- int getGlobalSlot (const llvm::Value &value)

Get the slot number of a global value.
- const llvm::Value * **getGlobalSlot** (unsigned 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 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 ()

16.32.1 Detailed Description

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

16.32.2 Member Function Documentation

16.32.2.1 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 it's type plane. If something is not in the SlotTracker, return -1.

16.32.2.2 void Canal::SlotTracker::processFunction () [protected]

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

16.32.2.3 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.

16.32.2.4 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

16.33 Canal::State Class Reference

Abstract memory state.

```
#include <State.h>
```

Public Member Functions

- **State** (const State &state)
- bool **operator==** (const State &state) const
- bool **operator!=** (const State &state) const
- 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.
- void **setReturnedValue** (Domain *value)
- void **mergeToReturnedValue** (const Domain &value)
- const Domain * **getReturnedValue** () const
- void **addVariableArgument** (const llvm::Instruction &place, Domain *argument)
- const StateMap & **getGlobalVariables** () const
- StateMap & **getGlobalVariables** ()
- const StateMap & **getGlobalBlocks** () const
- StateMap & **getGlobalBlocks** ()
- const StateMap & **getFunctionVariables** () const
- StateMap & **getFunctionVariables** ()
- const StateMap & **getFunctionBlocks** () const
- StateMap & **getFunctionBlocks** ()
- Domain * **findVariable** (const llvm::Value &place) const
- Domain * **findBlock** (const llvm::Value &place) const
- std::string **toString** (const llvm::Value &place, SlotTracker &slotTracker) const

16.33.1 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.

16.33.2 Member Function Documentation

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

Adds a register-type value to the stack.

Parameters

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

See also

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

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

Parameters

<i>place</i>	Represents a place in the program where the global variable is defined and assigned.
--------------	--

16.33.2.3 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.

16.33.2.4 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.

16.33.2.5 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.

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

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

16.34 Canal::StateMap Class Reference

Public Types

- `typedef Map::iterator iterator`
- `typedef Map::const_iterator const_iterator`
- `typedef Map::value_type value_type`
- `typedef Map::size_type size_type`
- `typedef Map::key_type key_type`

Public Member Functions

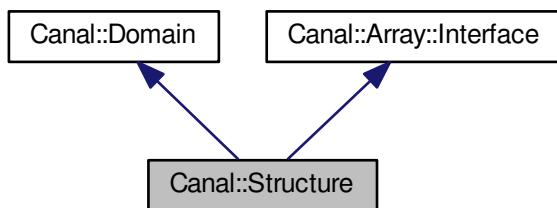
- `StateMap (const StateMap &map)`
- `bool operator== (const StateMap &map) const`
- `iterator begin ()`
- `const_iterator begin () const`
- `iterator end ()`
- `const_iterator end () const`
- `size_type size () const`
- `iterator find (const key_type &x)`
- `const_iterator find (const key_type &x) const`
- `std::pair< iterator, bool > insert (const value_type &x)`
- `void merge (const StateMap &map)`
- `void insert (const llvm::Value &place, Domain *value)`

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

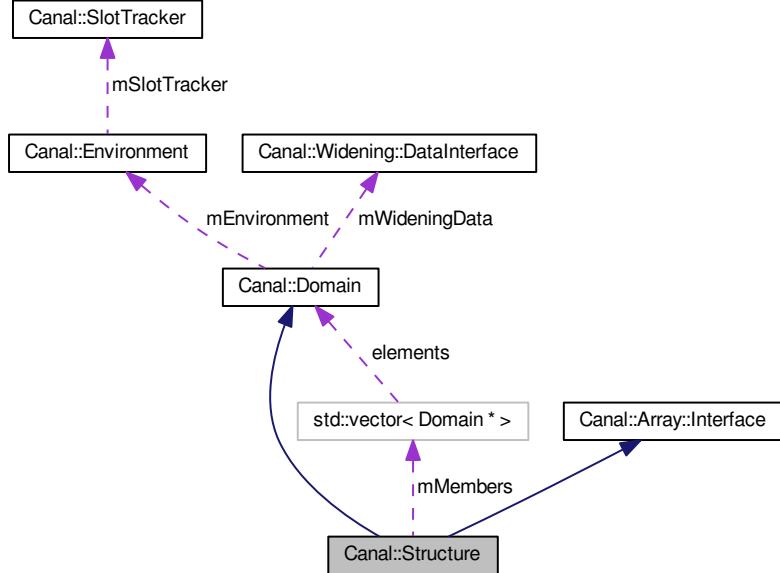
- `lib/StateMap.h`
- `lib/StateMap.cpp`

16.35 Canal::Structure Class Reference

Inheritance diagram for Canal::Structure:



Collaboration diagram for Canal::Structure:



Public Member Functions

- **Structure** (const Environment &environment, const std::vector< Domain * > &members)
- **Structure** (const Structure &value)
- 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 void setZero (const llvm::Value *place)
Implementation of Domain::setZero().
- 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**

Additional Inherited Members

16.35.1 Member Function Documentation

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

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

Implements Canal::Domain.

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

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

Implements Canal::Domain.

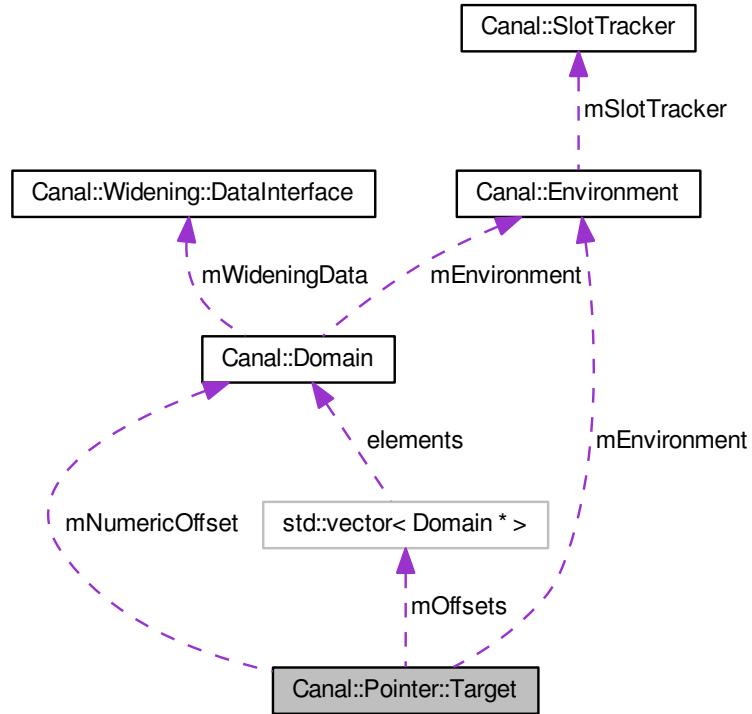
The documentation for this class was generated from the following files:

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

16.36 Canal::Pointer::Target Class Reference

```
#include <PointerTarget.h>
```

Collaboration diagram for Canal::Pointer::Target:



Public Types

- enum **Type** { **Constant**, **Block**, **Function** }

Public Member Functions

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

Copy constructor.
- `~Target ()`

Standard destructor.
- `bool operator== (const Target &target) const`
- `bool operator!= (const Target &target) const`
- `void merge (const Target &target)`

Merge another target into this one.
- `size_t memoryUsage () const`

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

- std::string toString (SlotTracker &slotTracker) const
Get a string representation of the target.
- std::vector< Domain * > dereference (const State &state) const

Public Attributes

- const Environment & **mEnvironment**
- Type mType
Type of the target.
- const llvm::Value * mTarget
- std::vector< Domain * > mOffsets
- Domain * mNumericOffset

16.36.1 Detailed Description

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.

16.36.2 Constructor & Destructor Documentation

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

Standard constructor.

Parameters

<i>type</i>	Type of the referenced memory.
<i>target</i>	Represents the target memory block or function. If type is Constant, it must be NULL. If type is Function, it must be a pointer to a function. Otherwise, it must be a valid pointer to an instruction. The instruction pointer is a key to State::mFunctionBlocks or State::mGlobalBlocks.
<i>offsets</i>	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.
<i>numericOffset</i>	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.

16.36.3 Member Function Documentation

16.36.3.1 std::vector< Domain * > Canal::Pointer::Target::dereferece (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.

16.36.4 Member Data Documentation

16.36.4.1 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.

16.36.4.2 std::vector<Domain*> Canal::Pointer::Target::mOffsets

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

16.36.4.3 const llvm::Value* Canal::Pointer::Target::mTarget

Valid when the target represents a memory block or function. For a 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.

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

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

16.37 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

16.38 Canal::Pointer::Utils Class Reference

Static Public Member Functions

- static void **addTarget** (Domain &pointer, Target::Type type, const llvm::Value *place, const llvm::Value *target, const std::vector< Domain * > &offsets, Domain *numericOffset)

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

- lib/PointerUtils.h
- lib/PointerUtils.cpp

16.39 Canal::VariableArguments Class Reference

Public Member Functions

- VariableArguments ()

Standard constructor.
- VariableArguments (const VariableArguments &arguments)

Copy constructor. Makes a deep copy of all arguments.
- ~VariableArguments ()

Standard destructor. Deletes all arguments.
- bool operator==(const VariableArguments &arguments) const
- void merge (const VariableArguments &arguments)

Merges the arguments per every instruction.
- void addArgument (const llvm::Instruction &place, Domain *argument)

16.39.1 Member Function Documentation

16.39.1.1 void Canal::VariableArguments::addArgument (const llvm::Instruction & place, Domain * argument)

Adds an argument at the end of the argument list for an instruction.

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

- lib/VariableArguments.h
- lib/VariableArguments.cpp

16.40 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)

16.40.1 Detailed Description

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

16.40.2 Member Function Documentation

16.40.2.1 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	121
Command	123
CommandBreak	125
CommandCd	126
CommandContinue	128
CommandDump	129
CommandFile	131
CommandFinish	132
CommandHelp	134
CommandInfo	135
CommandPrint	137
CommandPwd	138
CommandQuit	140
CommandRun	141
CommandShell	144
CommandShow	146
CommandStart	147
CommandStep	149
Commands	143
IteratorCallback[external]	
IteratorCallback	150
State	151

17.2 Class List

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

Arguments	121
Command	123
CommandBreak	125
CommandCd	126
CommandContinue	128

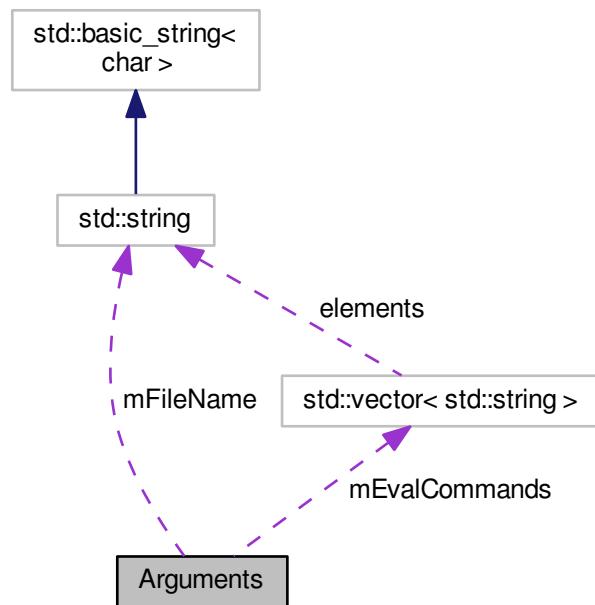
CommandDump	129
CommandFile	131
CommandFinish	132
CommandHelp	134
CommandInfo	135
CommandPrint	137
CommandPwd	138
CommandQuit	140
CommandRun	141
Commands	143
CommandShell	144
CommandShow	146
CommandStart	147
CommandStep	149
IteratorCallback	150
State	151

Chapter 18

Tool Class Documentation

18.1 Arguments Class Reference

Collaboration diagram for Arguments:



Public Attributes

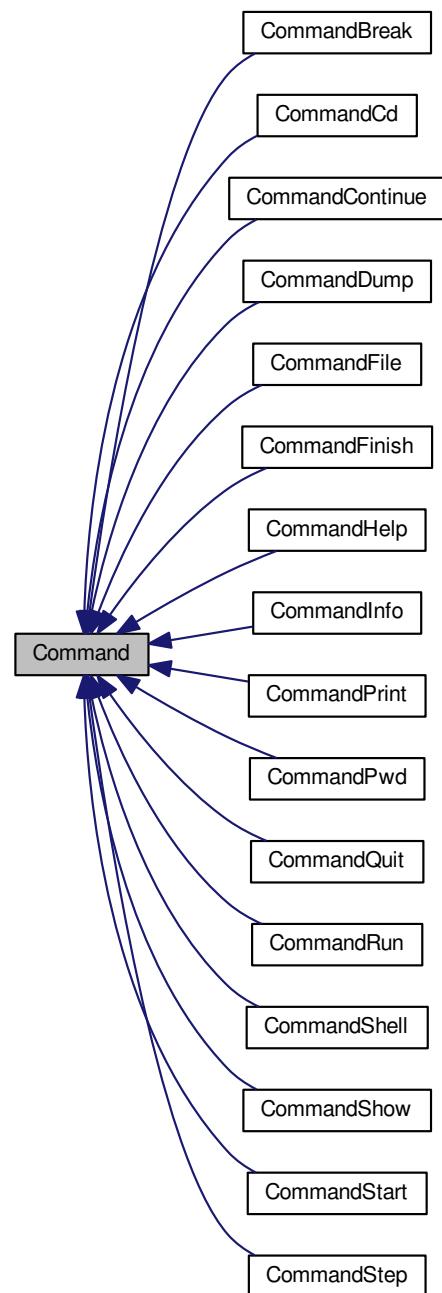
- `std::vector< std::string > mEvalCommands`
- `std::string mFileName`

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

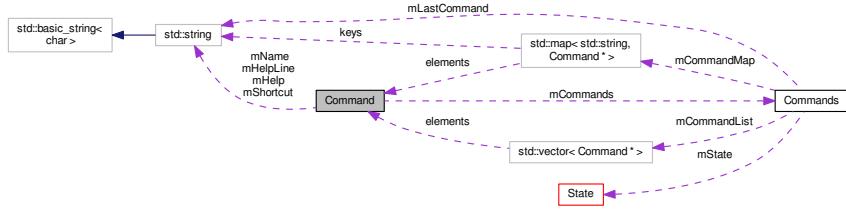
- tool/Canal.cpp

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

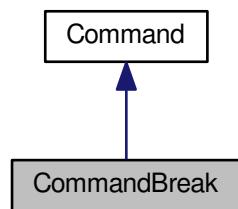
- std::string **mName**
- std::string **mShortcut**
- std::string **mHelpLine**
- std::string **mHelp**
- Commands & **mCommands**

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

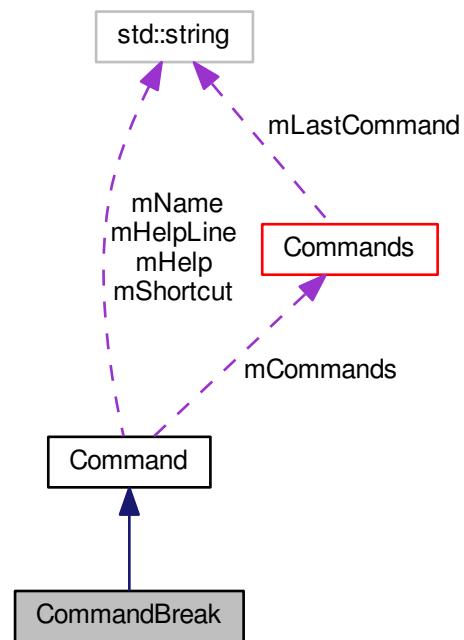
- tool/Command.h
- tool/Command.cpp

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)

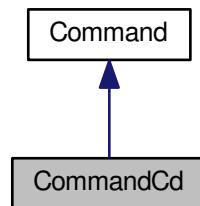
Additional Inherited Members

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

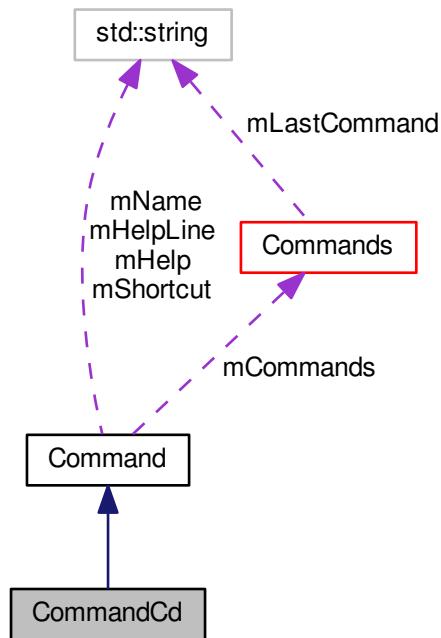
- tool/CommandBreak.h
- tool/CommandBreak.cpp

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)`

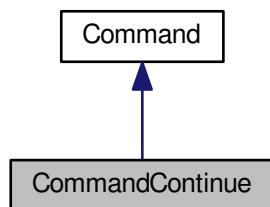
Additional Inherited Members

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

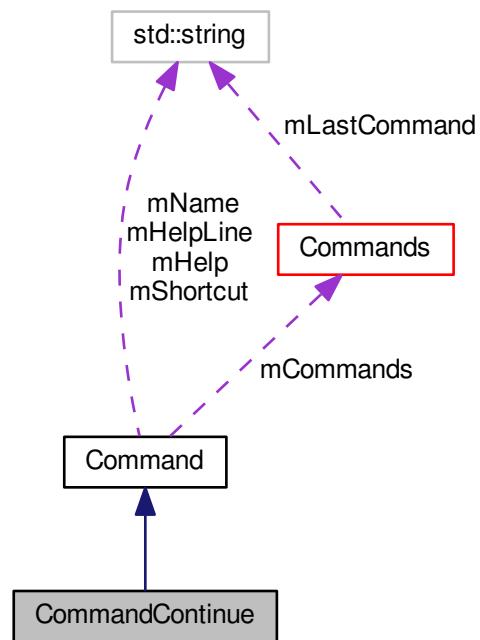
- tool/CommandCd.h
- tool/CommandCd.cpp

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)

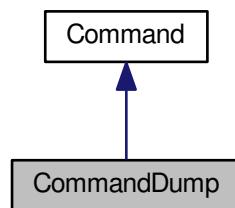
Additional Inherited Members

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

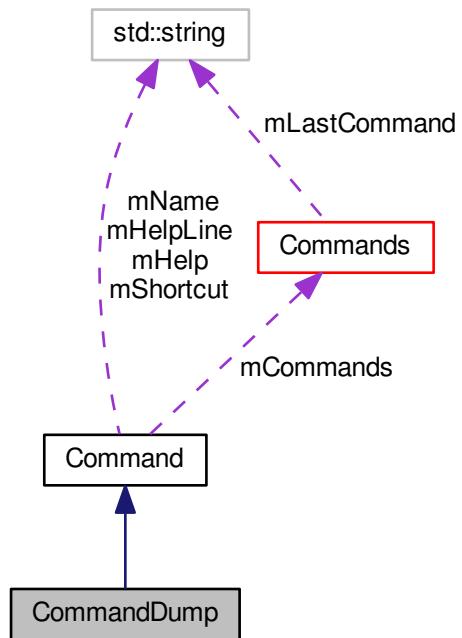
- tool/CommandContinue.h
- tool/CommandContinue.cpp

18.6 CommandDump Class Reference

Inheritance diagram for CommandDump:



Collaboration diagram for CommandDump:



Public Member Functions

- **CommandDump** (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)

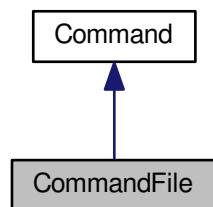
Additional Inherited Members

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

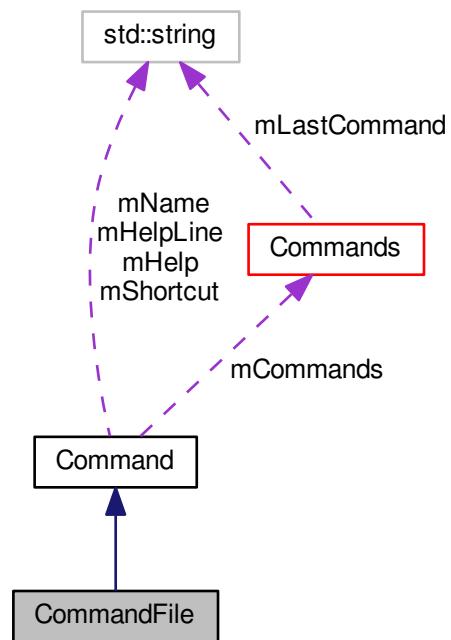
- tool/CommandDump.h
- tool/CommandDump.cpp

18.7 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)

Additional Inherited Members

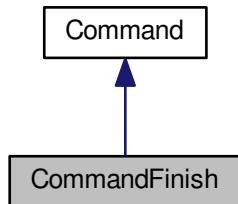
The documentation for this class was generated from the following files:

- tool/CommandFile.h

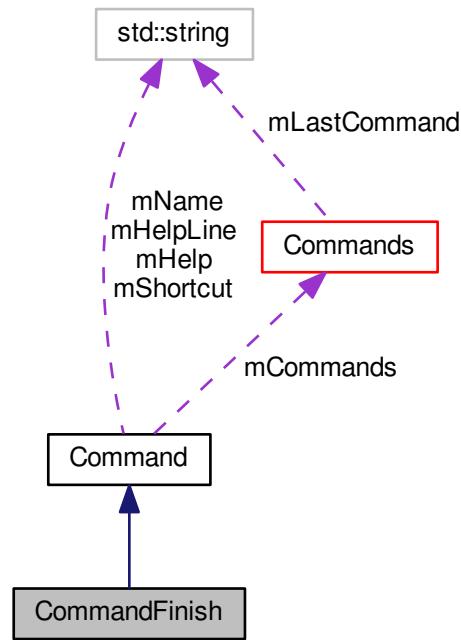
- tool/CommandFile.cpp

18.8 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`)

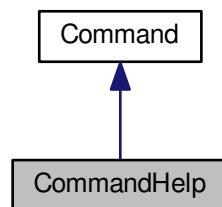
Additional Inherited Members

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

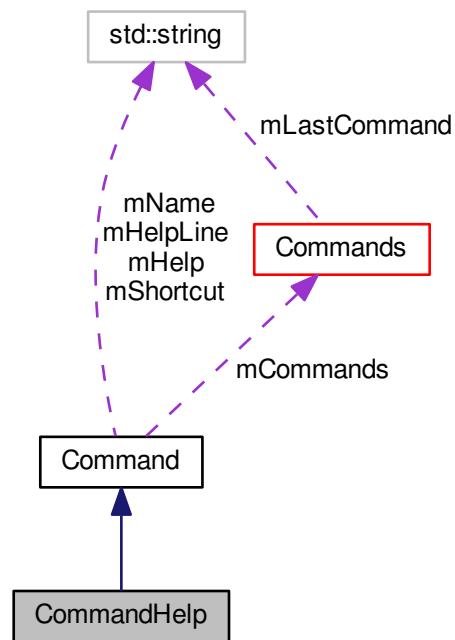
- `tool/CommandFinish.h`
- `tool/CommandFinish.cpp`

18.9 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** ()

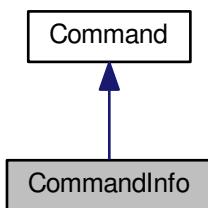
Additional Inherited Members

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

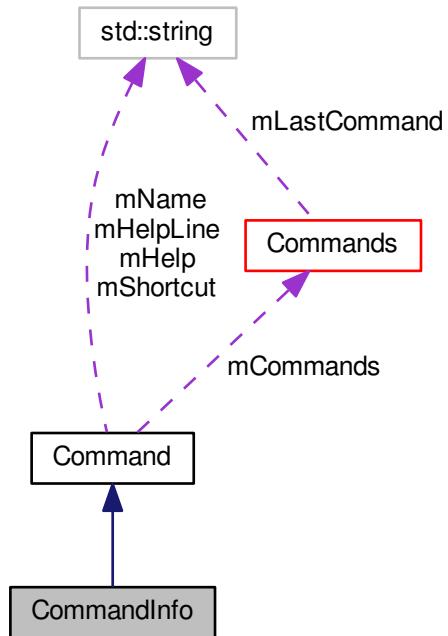
- tool/CommandHelp.h
- tool/CommandHelp.cpp

18.10 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** ()

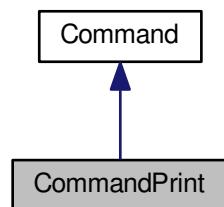
Additional Inherited Members

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

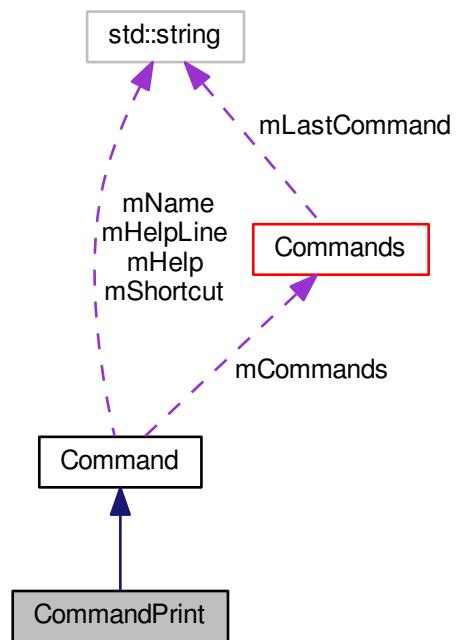
- tool/CommandInfo.h
- tool/CommandInfo.cpp

18.11 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)

Additional Inherited Members

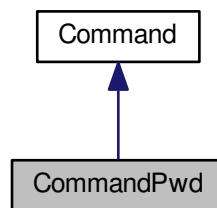
The documentation for this class was generated from the following files:

- tool/CommandPrint.h

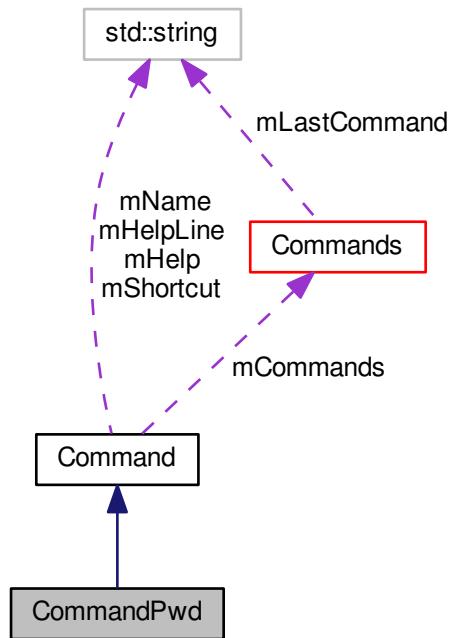
- tool/CommandPrint.cpp

18.12 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)

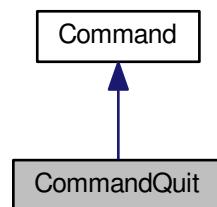
Additional Inherited Members

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

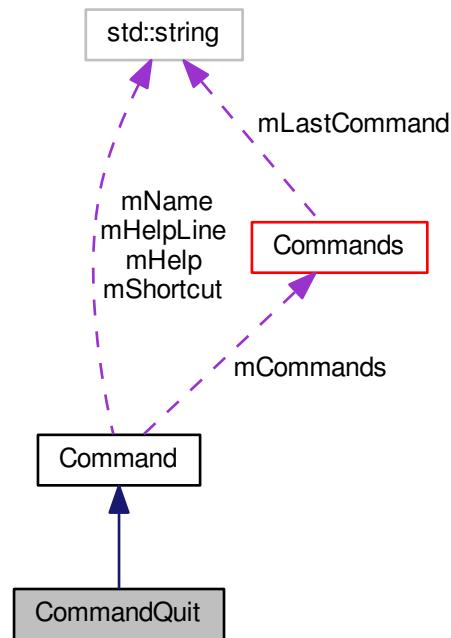
- tool/CommandPwd.h
- tool/CommandPwd.cpp

18.13 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)

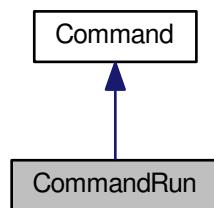
Additional Inherited Members

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

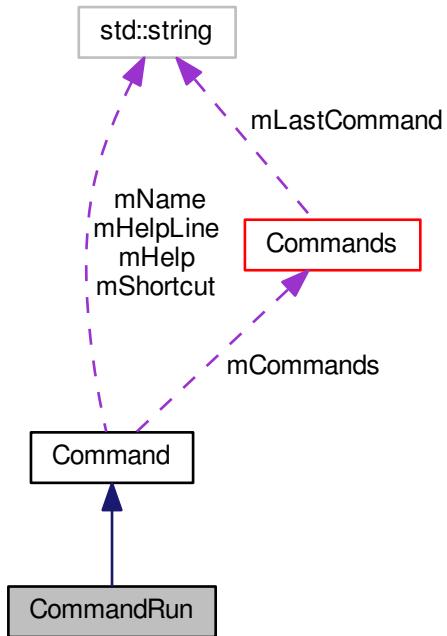
- tool/CommandQuit.h
- tool/CommandQuit.cpp

18.14 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)

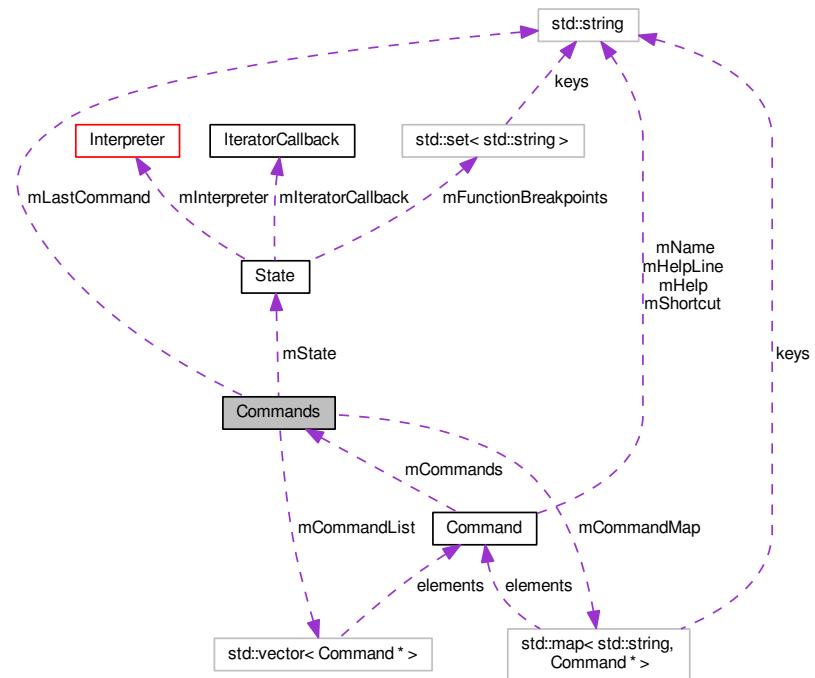
Additional Inherited Members

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

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

18.15 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

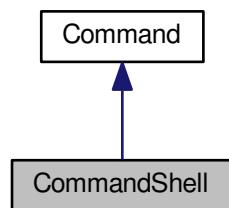
- std::string **mLastCommand**
- State * **mState**

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

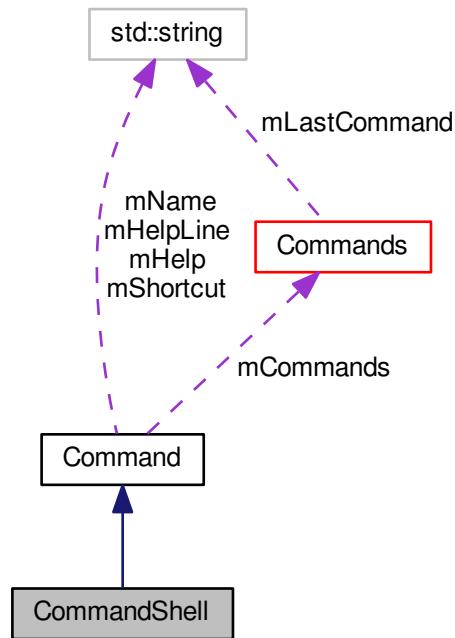
- tool/Commands.h
- tool/Commands.cpp

18.16 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`)

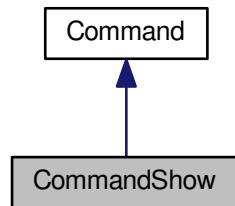
Additional Inherited Members

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

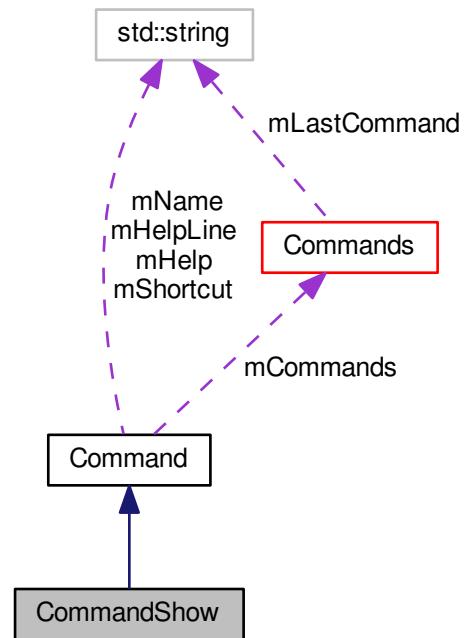
- `tool/CommandShell.h`
- `tool/CommandShell.cpp`

18.17 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)

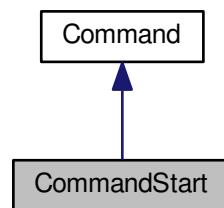
Additional Inherited Members

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

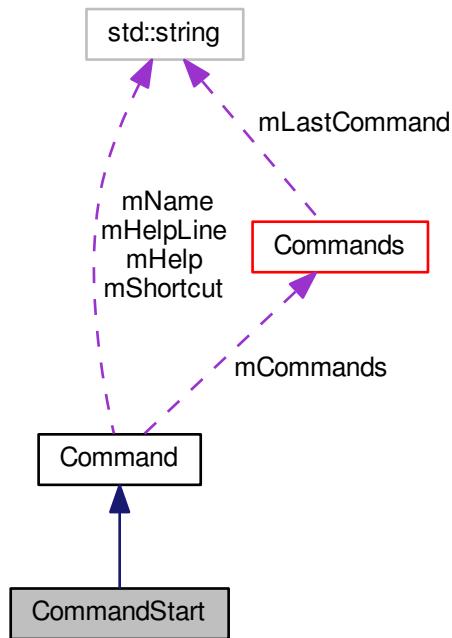
- tool/CommandShow.h
- tool/CommandShow.cpp

18.18 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`)

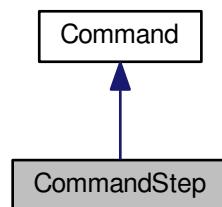
Additional Inherited Members

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

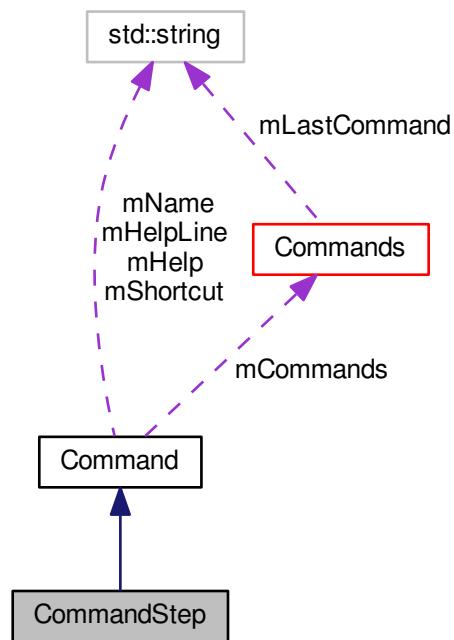
- `tool/CommandStart.h`
- `tool/CommandStart.cpp`

18.19 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)

Additional Inherited Members

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

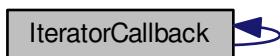
- tool/CommandStep.h
- tool/CommandStep.cpp

18.20 IteratorCallback Class Reference

Inheritance diagram for IteratorCallback:



Collaboration diagram for IteratorCallback:



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

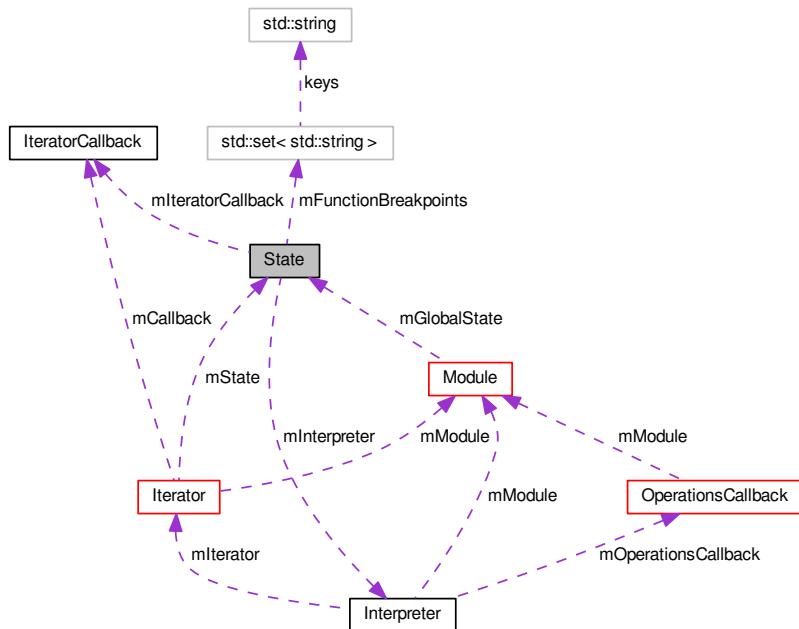
- bool **mFixpointReached**
- bool **mFunctionEnter**
- bool **mBasicBlockEnter**

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

- tool/IteratorCallback.h
- tool/IteratorCallback.cpp

18.21 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 Lviron1 , 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, 75
addArgument
 Canal::VariableArguments, 113
addFunctionVariable
 Canal::State, 106
addGlobalVariable
 Canal::State, 106
addTarget
 Canal::Pointer::Pointer, 97
Arguments, 117

br
 Canal::Operations, 90

Canal::APIntUtils::SCompare, 100
Canal::APIntUtils::UCompare, 112
Canal::AccuracyDomain, 39
 accuracy, 40
Canal::Array::ExactSize, 63
 clone, 66
 cloneCleaned, 66
 ExactSize, 66
Canal::Array::Interface, 69
 getItem, 69
 getValue, 70
 setItem, 70
Canal::Array::SingleItem, 100
 clone, 103
 cloneCleaned, 103
 mSize, 103
Canal::Constructors, 47
 create, 48
Canal::Domain, 55
 cloneCleaned, 58
 Domain, 58
 operator==, 58
 setZero, 58
Canal::Environment, 63
Canal::Float::Interval, 73
 accuracy, 75
 cloneCleaned, 75
 operator==, 75
Canal::FunctionModel, 68

Canal::FunctionModelManager, 69
Canal::Integer::Bitfield, 42
 clone, 44
 cloneCleaned, 44
 getBitValue, 45
 mOnes, 46
 mZeroes, 46
 setBitValue, 45
 signedMax, 45
 signedMin, 45
 unsignedMax, 45
 unsignedMin, 46

Canal::Integer::Container, 48
 clone, 51
 cloneCleaned, 51
 Container, 51
 signedMax, 51
 signedMin, 51
 unsignedMax, 52
 unsignedMin, 52

Canal::Integer::Enumeration, 59
 clone, 61
 cloneCleaned, 61
 signedMax, 61
 signedMin, 62
 unsignedMax, 62
 unsignedMin, 62

Canal::Integer::Interval, 75
 clone, 78
 cloneCleaned, 78
 Interval, 78
 isSingleValue, 78
 mEmpty, 80
 signedMax, 78
 signedMin, 79
 unsignedMax, 79
 unsignedMin, 79

Canal::InterpreterBlock::BasicBlock, 41
Canal::InterpreterBlock::Function, 67
 updateBasicBlockInputState, 68
Canal::InterpreterBlock::Interpreter, 71
 Interpreter, 72
Canal::InterpreterBlock::Iterator, 80
 interpretInstruction, 81

mBasicBlock, 81
 mChanged, 81
 mFunction, 81
 mInitialized, 81
 Canal::InterpreterBlock::IteratorCallback, 82
 Canal::InterpreterBlock::Module, 84
 Canal::InterpreterBlock::OperationsCallback, 92
 onFunctionCall, 93
 Canal::Operations, 86
 br, 90
 fadd, 90
 fdiv, 90
 fsub, 90
 indirectbr, 90
 invoke, 90
 ret, 90
 sdiv, 90
 switch_, 90
 udiv, 91
 unreachable, 91
 variableOrConstant, 91
 Canal::OperationsCallback, 94
 onFunctionCall, 94
 Canal::Pointer::Pointer, 95
 addTarget, 97
 clone, 98
 cloneCleaned, 98
 dereferenceAndMerge, 98
 getElementPtr, 98
 mTargets, 98
 mType, 99
 Pointer, 97
 Canal::Pointer::Target, 109
 dereference, 111
 mNumericOffset, 112
 mOffsets, 112
 mTarget, 112
 Target, 111
 Canal::Pointer::Utils, 112
 Canal::SlotTracker, 103
 getLocalSlot, 104
 processFunction, 104
 processModule, 104
 setActiveFunction, 104
 Canal::State, 105
 addFunctionVariable, 106
 addGlobalVariable, 106
 findBlock, 106
 findVariable, 106
 mergeForeignFunctionBlocks, 106
 Canal::StateMap, 107
 Canal::Structure, 107
 clone, 109
 cloneCleaned, 109
 Canal::VariableArguments, 113
 addArgument, 113
 Canal::VariablePrecisionDomain, 113
 limitMemoryUsage, 113
 Canal::Widening::DataInterface, 53
 Canal::Widening::DataIterationCount, 54
 Canal::Widening::Interface, 71
 Canal::Widening::Manager, 82
 Canal::Widening::NumericalInfinity, 85
 Canal::Widening::Pointers, 99
 clone
 Canal::Array::ExactSize, 66
 Canal::Array::SingleItem, 103
 Canal::Integer::Bitfield, 44
 Canal::Integer::Container, 51
 Canal::Integer::Enumeration, 61
 Canal::Integer::Interval, 78
 Canal::Pointer::Pointer, 98
 Canal::Structure, 109
 cloneCleaned
 Canal::Array::ExactSize, 66
 Canal::Array::SingleItem, 103
 Canal::Domain, 58
 Canal::Float::Interval, 75
 Canal::Integer::Bitfield, 44
 Canal::Integer::Container, 51
 Canal::Integer::Enumeration, 61
 Canal::Integer::Interval, 78
 Canal::Pointer::Pointer, 98
 Canal::Structure, 109
 Command, 119
 CommandBreak, 121
 CommandCd, 122
 CommandContinue, 124
 CommandDump, 125
 CommandFile, 127
 CommandFinish, 128
 CommandHelp, 130
 CommandInfo, 131
 CommandPrint, 133
 CommandPwd, 134
 CommandQuit, 136
 CommandRun, 137
 CommandShell, 140
 CommandShow, 142
 CommandStart, 143
 CommandStep, 145
 Commands, 139
 Container
 Canal::Integer::Container, 51
 create
 Canal::Constructors, 48
 dereference

Canal::Pointer::Target, 111
dereferenceAndMerge
 Canal::Pointer::Pointer, 98
Domain
 Canal::Domain, 58

ExactSize
 Canal::Array::ExactSize, 66

fadd
 Canal::Operations, 90
fdiv
 Canal::Operations, 90
findBlock
 Canal::State, 106
findVariable
 Canal::State, 106
fsub
 Canal::Operations, 90

getBitValue
 Canal::Integer::Bitfield, 45
getElementPtr
 Canal::Pointer::Pointer, 98
getItem
 Canal::Array::Interface, 69
getLocalSlot
 Canal::SlotTracker, 104
getValue
 Canal::Array::Interface, 70

indirectbr
 Canal::Operations, 90
interpretInstruction
 Canal::InterpreterBlock::Iterator, 81
Interpreter
 Canal::InterpreterBlock::Interpreter, 72
Interval
 Canal::Integer::Interval, 78
invoke
 Canal::Operations, 90
isSingleValue
 Canal::Integer::Interval, 78
IteratorCallback, 146

limitMemoryUsage
 Canal::VariablePrecisionDomain, 113

mBasicBlock
 Canal::InterpreterBlock::Iterator, 81
mChanged
 Canal::InterpreterBlock::Iterator, 81
mEmpty
 Canal::Integer::Interval, 80
mFunction
 Canal::Integer::Bitfield, 45
 Canal::Integer::Container, 51
 Canal::Integer::Enumeration, 61
 Canal::Integer::Interval, 78
 Canal::InterpreterBlock::Iterator, 81
mInitialized
 Canal::InterpreterBlock::Iterator, 81
mNumericOffset
 Canal::Pointer::Target, 112
mOffsets
 Canal::Pointer::Target, 112
mOnes
 Canal::Integer::Bitfield, 46
mSize
 Canal::Array::SingleItem, 103
mTarget
 Canal::Pointer::Target, 112
mTargets
 Canal::Pointer::Pointer, 98
mType
 Canal::Pointer::Pointer, 99
mZeroes
 Canal::Integer::Bitfield, 46
mergeForeignFunctionBlocks
 Canal::State, 106

onFunctionCall
 Canal::InterpreterBlock::OperationsCallback, 93
 Canal::OperationsCallback, 94
operator==
 Canal::Domain, 58
 Canal::Float::Interval, 75

Pointer
 Canal::Pointer::Pointer, 97
processFunction
 Canal::SlotTracker, 104
processModule
 Canal::SlotTracker, 104

ret
 Canal::Operations, 90

sdiv
 Canal::Operations, 90
setActiveFunction
 Canal::SlotTracker, 104
setBitValue
 Canal::Integer::Bitfield, 45
setItem
 Canal::Array::Interface, 70
setZero
 Canal::Domain, 58
signedMax
 Canal::Integer::Bitfield, 45
 Canal::Integer::Container, 51
 Canal::Integer::Enumeration, 61
 Canal::Integer::Interval, 78

signedMin
 Canal::Integer::Bitfield, 45
 Canal::Integer::Container, 51
 Canal::Integer::Enumeration, 62
 Canal::Integer::Interval, 79
State, 147
switch_
 Canal::Operations, 90

Target
 Canal::Pointer::Target, 111

udiv
 Canal::Operations, 91

unreachable
 Canal::Operations, 91

unsignedMax
 Canal::Integer::Bitfield, 45
 Canal::Integer::Container, 52
 Canal::Integer::Enumeration, 62
 Canal::Integer::Interval, 79

unsignedMin
 Canal::Integer::Bitfield, 46
 Canal::Integer::Container, 52
 Canal::Integer::Enumeration, 62
 Canal::Integer::Interval, 79

updateBasicBlockInputState
 Canal::InterpreterBlock::Function, 68

variableOrConstant
 Canal::Operations, 91