Canal

Karel Klíč

September 2, 2012

Contents

Chapter 1

Overview

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

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

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

1.1 Use cases

1.1.1 Analysis of program behaviour

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

1.1.2 Comparison with a specification

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

1.1.3 Conformance to environment constraints

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

Chapter 2

Installation

2.1 Installation from source code on Red Hat Enterprise Linux 6

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

Prerequisites

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

- The core library requires `llvm-devel` and `clang` packages, which can be obtained from Extra Packages for Enterprise Linux (or EPEL) software repository.
- The command-line user interface tool requires `elfutils-devel` and `readline-devel` packages.
- The documentation requires `doxygen`, `graphviz`, and `texlive-latex` packages.

The compiled and installed Canal requires `llvm`, `clang`, `elfutils`, and `readline` packages.

2.2 Installation from source code on Fedora 17

Canal can be built, installed, and developed on a computer with the Fedora 17 operating system.

Prerequisites

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

- The core library requires `llvm-devel` and `clang` packages.
- The command-line user interface tool requires `elfutils-devel` and `readline-devel` packages.
- The documentation requires `doxygen`, `graphviz`, and `texlive-latex` packages.

The compiled and installed Canal requires `llvm`, `clang`, `elfutils`, and `readline` packages.

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 $\bot$.

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 $\bigsqcup A$. 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 $\bigsqcap A$.

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* $(\mathcal{D}, \sqsubseteq, \sqcup, \sqcap, \bot, \top)$ 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 $\bigsqcup \mathcal{D}$, and the lowest element $\bot$ defined as $\bigsqcap \mathcal{D}$.

A function $F \in \mathcal{D}_1 \to \mathcal{D}_2$ 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 $F \in \mathcal{D}_1 \to \mathcal{D}_2$ is *strict* if $F(\bot_1) = \bot_2$. A function $F \in \mathcal{D}_1 \to \mathcal{D}_2$ is *continuous* if it preserves the existing limits of increasing chains $(X_i)_{i \in I}$: $F(\bigsqcup_1 \{X_i \mid i \in I\}) = \bigsqcup_2 \{F(X_i) \mid i \in I\}$ whenever $\bigsqcup_1 \{X_i \mid i \in I\}$ exists.

A *fixpoint* of a function $F : \mathcal{D} \to \mathcal{D}$ 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 $\mathrm{fp}(F)$. A set of all prefixpoints is denoted by $\mathrm{prefp}(F)$. A set of all postfixpoints is denoted by $\mathrm{postfp}(F)$. The *least fixpoint* or *lfp* of a fuction F on a poset $(\mathcal{D}, \sqsubseteq)$ satisfies $\mathrm{lfp} \in \mathrm{fp}(F)$ and $\forall p \in \mathrm{fp}(F) : \mathrm{lfp} \sqsubseteq p$.

A *Galois connection* is a pair of two functions $\alpha : \mathcal{D}_1 \to \mathcal{D}_2$ and $\gamma : \mathcal{D}_2 \to \mathcal{D}_1$ 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

$(\mathcal{D}_1, \sqsubseteq_1) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (\mathcal{D}_2, \sqsubseteq_2).$

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 $\mathbf{i}sz \mid sz \in \mathbb{N}^*$, such as $\mathbf{i}1$, $\mathbf{i}8$, $\mathbf{i}32$, $\mathbf{i}64$. They also include floating point types *fp*. The **void** type does not represent any value and has no size. Pointers $typ*$ are used to specify memory locations. Arrays $[sz \times typ]$ have statically known size *sz*. Structures $\{\overline{typ_j}^j\}$ are defined as a list of types. Functions $typ\ \overline{typ_j}^j$ consist of a return type and a list of parameter types. Types can also be named by identifiers *id*, which is useful for the definition of recursive types. The **label** type represents code labels. The **metadata** type represents embedded metadata.

$$
\begin{array}{llcl}
\text{Modules} & \mathit{mod} & ::= & \overline{\mathit{layout}}\ \overline{\mathit{asm}}\ \overline{\mathit{namedt}}\ \overline{\mathit{namedm}}\ \overline{\mathit{alias}}\ \overline{\mathit{prod}} \\
\text{Layouts} & \mathit{layout} & ::= & \mathbf{bigendian} \mid \mathbf{littleendian} \mid \mathbf{ptr}\ \mathit{sz}\ \mathit{align}_{\mathit{abi}}\ \mathit{align}_{\mathit{pref}} \\
 & & \mid & \mathbf{int}\ \mathit{sz}\ \mathit{align}_{\mathit{abi}}\ \mathit{align}_{\mathit{pref}} \mid \mathbf{float}\ \mathit{sz}\ \mathit{align}_{\mathit{abi}}\ \mathit{align}_{\mathit{pref}} \\
 & & \mid & \mathbf{aggr}\ \mathit{sz}\ \mathit{align}_{\mathit{abi}}\ \mathit{align}_{\mathit{pref}} \mid \mathbf{vec}\ \mathit{sz}\ \mathit{align}_{\mathit{abi}}\ \mathit{align}_{\mathit{pref}} \\
 & & \mid & \mathbf{stack}\ \mathit{sz}\ \mathit{align}_{\mathit{abi}}\ \mathit{align}_{\mathit{pref}} \\
\text{Products} & \mathit{prod} & ::= & \mathit{id} = \mathbf{global}\ \mathit{typ}\ \mathit{const}\ \mathit{align} \mid \mathbf{define}\ \mathit{typ}\ \mathit{id}(\overline{\mathit{arg}})\{\overline{b}\} \mid \mathbf{declare}\ \mathit{typ}\ \mathit{id}(\overline{\mathit{arg}}) \\
\text{Floats} & \mathit{fp} & ::= & \mathbf{half} \mid \mathbf{float} \mid \mathbf{double} \mid \mathbf{x86_fp80} \mid \mathbf{fp128} \mid \mathbf{ppc_fp128} \\
\text{Vec types} & \mathit{vtyp} & ::= & \mathit{fp} \mid \mathbf{i}\mathit{sz} \mid \mathit{fp}* \mid \mathbf{i}\mathit{sz}* \\
\text{Types} & \mathit{typ} & ::= & \mathbf{i}\mathit{sz} \mid \mathit{fp} \mid \mathbf{void} \mid \mathit{typ}* \mid [\mathit{sz} \times \mathit{typ}] \mid [\mathit{sz} \times \mathit{vtyp}] \mid \{\overline{\mathit{typ}_j}^{j}\} \mid \mathit{typ}\ \overline{\mathit{typ}_j}^{j} \\
 & & \mid & \mathit{id} \mid \mathbf{label} \mid \mathbf{metadata} \\
\text{Values} & \mathit{val} & ::= & \mathit{id} \mid \mathit{cnst} \\
\text{Binops} & \mathit{bop} & ::= & \mathbf{add} \mid \mathbf{sub} \mid \mathbf{mul} \mid \mathbf{udiv} \mid \mathbf{sdiv} \mid \mathbf{urem} \mid \mathbf{srem} \mid \mathbf{shl} \mid \mathbf{lshr} \mid \mathbf{ashr} \\
 & & \mid & \mathbf{and} \mid \mathbf{or} \mid \mathbf{xor} \\
\text{Float ops} & \mathit{fbop} & ::= & \mathbf{fadd} \mid \mathbf{fsub} \mid \mathbf{fmul} \mid \mathbf{fdiv} \mid \mathbf{frem} \\
\text{Extension} & \mathit{eop} & ::= & \mathbf{zext} \mid \mathbf{sext} \mid \mathbf{fpext} \\
\text{Cast ops} & \mathit{cop} & ::= & \mathbf{fptoui} \mid \mathbf{ptrtoint} \mid \mathbf{inttoptr} \mid \mathbf{bitcast} \\
\text{Trunc ops} & \mathit{trop} & ::= & \mathbf{trunc}_{\mathit{int}} \mid \mathbf{trunc}_{\mathit{fp}} \\
\text{Constants} & \mathit{cnst} & ::= & \mathbf{i}\mathit{sz}\ \mathit{Int} \mid \mathit{fp}\ \mathit{Float} \mid \mathit{typ} * \mathit{id} \mid (\mathit{typ}*)\ \mathbf{null} \mid \mathit{typ}\ \mathbf{zeroinitializer} \\
 & & \mid & \mathit{typ}[\overline{\mathit{cnst}_j}^{j}] \mid \{\overline{\mathit{cnst}_j}^{j}\} \mid \mathit{typ}\ \mathbf{undef} \mid \mathit{bop}\ \mathit{cnst}_1\ \mathit{cnst}_2 \mid \mathit{fbop}\ \mathit{cnst}_1\ \mathit{cnst}_2 \\
 & & \mid & \mathit{trop}\ \mathit{cnst}\ \mathbf{to}\ \mathit{typ} \mid \mathit{eop}\ \mathit{cnst}\ \mathbf{to}\ \mathit{typ} \mid \mathit{cop}\ \mathit{cnst}\ \mathbf{to}\ \mathit{typ} \\
 & & \mid & \mathbf{getelementptr}\ \mathit{cnst}\ \overline{\mathit{cnst}_j}^{j} \mid \mathbf{select}\ \mathit{cnst}_0\ \mathit{cnst}_1\ \mathit{cnst}_2 \\
 & & \mid & \mathbf{icmp}\ \mathit{cond}\ \mathit{cnst}_1\ \mathit{cnst}_2 \mid \mathbf{fcmp}\ \mathit{fcond}\ \mathit{cnst}_1\ \mathit{cnst}_2 \\
\text{Blocks} & b & ::= & l\ \overline{\phi}\ \overline{c}\ \mathit{tmn} \\
\phi\ \text{nodes} & \phi & ::= & \mathit{id} = \mathbf{phi}\ \mathit{typ}\ \overline{[\mathit{val}_j, l_j]}^{j} \\
\text{Tmns} & \mathit{tmn} & ::= & \mathbf{br}\ \mathit{val}\ l_1\ l_2 \mid \mathbf{br}\ l \mid \mathbf{ret}\ \mathit{typ}\ \mathit{val} \mid \mathbf{ret\ void} \mid \mathbf{unreachable} \\
\text{Commands} & c & ::= & \mathit{id} = \mathit{bop}\ (\mathbf{int}\ \mathit{sz})\ \mathit{val}_1\ \mathit{val}_2 \mid \mathit{id} = \mathit{fbop}\ \mathit{fp}\ \mathit{val}_1\ \mathit{val}_2 \\
 & & \mid & \mathbf{store}\ \mathit{typ}\ \mathit{val}_1\ \mathit{val}_2\ \mathit{align} \mid \mathit{id} = \mathbf{malloc}\ \mathit{typ}\ \mathit{val}\ \mathit{align} \mid \mathbf{free}\ (\mathit{typ}*)\ \mathit{val} \\
 & & \mid & \mathit{id} = \mathbf{alloca}\ \mathit{typ}\ \mathit{val}\ \mathit{align} \mid \mathit{id} = \mathit{trop}\ \mathit{typ}_1\ \mathit{val}\ \mathbf{to}\ \mathit{typ}_2 \\
 & & \mid & \mathit{id} = \mathit{eop}\ \mathit{typ}_1\ \mathit{val}\ \mathbf{to}\ \mathit{typ}_2 \mid \mathit{id} = \mathit{cop}\ \mathit{typ}_1\ \mathit{val}\ \mathbf{to}\ \mathit{typ}_2 \\
 & & \mid & \mathit{id} = \mathbf{select}\ \mathit{val}_0\ \mathit{typ}\ \mathit{val}_1\ \mathit{val}_2 \mid \mathit{option}\ \mathit{id} = \mathbf{call}\ \mathit{typ}_0\ \mathit{val}_0\ \overline{\mathit{param}} \\
 & & \mid & \mathit{id} = \mathbf{icmp}\ \mathit{cond}\ \mathit{typ}\ \mathit{val}_1\ \mathit{val}_2 \mid \mathit{id} = \mathbf{fcmp}\ \mathit{fcond}\ \mathit{fp}\ \mathit{val}_1\ \mathit{val}_2 \\
 & & \mid & \mathit{id} = \mathbf{getelementptr}\ (\mathit{typ}*)\ \mathit{val}\ \overline{\mathit{val}_j}^{j} \mid \mathit{id} = \mathbf{load}\ (\mathit{typ}*)\ \mathit{val}\ \mathit{align} \\
 & & \mid & \mathit{id} = \mathbf{extractelement}\ [\mathit{sz} \times \mathit{vtyp}]\ \mathit{val}_1\ \mathit{val}_2 \\
 & & \mid & \mathit{id} = \mathbf{insertelement}\ [\mathit{sz} \times \mathit{vtyp}]\ \mathit{val}_1\ \mathit{val}_2\ \mathit{val}_3
\end{array}
$$

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].

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

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 (*z*ero) and to 1 (*o*ne).

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 Fixpoint Recalculation

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

13.6 Multi Threading

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

13.7 Symbolic Abstract Domains

13.8 Weakly-Relational Abstract Domains

Implement weakly relational integer and floating-point abstract domains.

13.9 Compositional Analysis

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

13.10 Parallelization

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

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:

15.2 Class List

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

Chapter 16

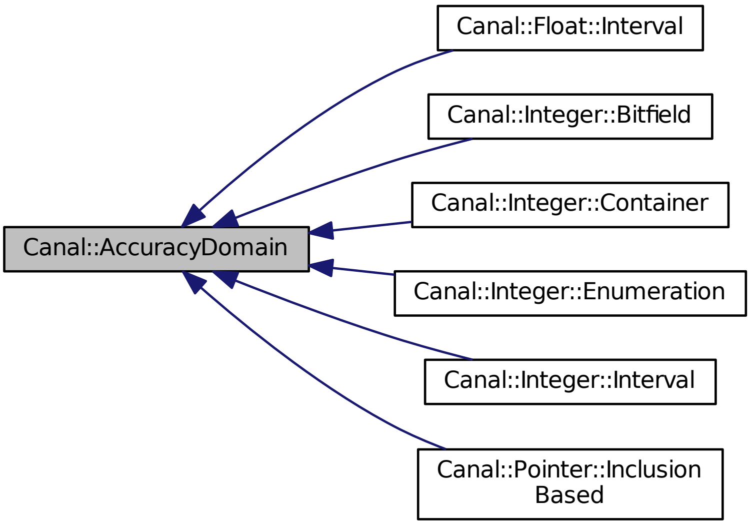
Library Class Documentation

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

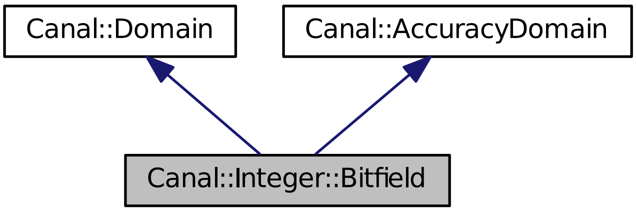
The documentation for this class was generated from the following files:

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

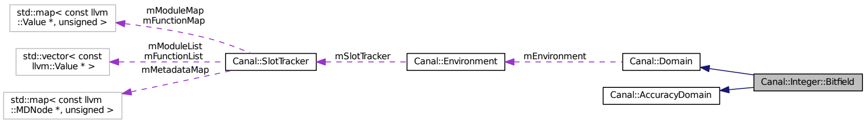
16.2 Canal::Integer::Bitfield Class Reference

`#include <IntegerBitfield.h>`

Inheritance diagram for Canal::Integer::Bitfield:



Collaboration diagram for Canal::Integer::Bitfield:



Public Member Functions

- Bitfield (const Environment &environment, unsigned bitWidth)

  Initializes to the lowest value.
- Bitfield (const Environment &environment, const llvm::APInt &number)

  Initializes to the given value.
- unsigned getBitWidth () const

  Return the number of bits of the represented number.
- int getBitValue (unsigned pos) const
- void setBitValue (unsigned pos, int value)
- bool signedMin (llvm::APInt &result) const
- bool signedMax (llvm::APInt &result) const
- bool unsignedMin (llvm::APInt &result) const
- bool unsignedMax (llvm::APInt &result) const
- bool isSingleValue () const

  Does these bits represent single value?
- virtual Bitfield ∗ clone () const
- virtual Bitfield ∗ cloneCleaned () const
- virtual bool operator== (const Domain &value) const

  Implementation of Domain::operator==().
- virtual void merge (const Domain &value)

  Implementation of Domain::merge().
- virtual size_t memoryUsage () const

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

  Implementation of Domain::toString().
- virtual bool matchesString (const std::string &text, std::string &rationale) const

  Implementation of Domain::matchesString().
- virtual void add (const Domain &a, const Domain &b)

  Implementation of Domain::add().
- virtual void sub (const Domain &a, const Domain &b)

  Implementation of Domain::sub().
- virtual void mul (const Domain &a, const Domain &b)

  Implementation of Domain::mul().
- virtual void udiv (const Domain &a, const Domain &b)

  Implementation of Domain::udiv().
- virtual void sdiv (const Domain &a, const Domain &b)

  Implementation of Domain::sdiv().

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

  Implementation of Domain::urem().
- virtual void srem (const Domain &a, const Domain &b)

  Implementation of Domain::srem().
- virtual void shl (const Domain &a, const Domain &b)

  Implementation of Domain::shl().
- virtual void lshr (const Domain &a, const Domain &b)

  Implementation of Domain::lshr().
- virtual void ashr (const Domain &a, const Domain &b)

  Implementation of Domain::ashr().
- virtual void and_ (const Domain &a, const Domain &b)

  Implementation of Domain::and_().
- virtual void or_ (const Domain &a, const Domain &b)

  Implementation of Domain::or_().
- virtual void xor_ (const Domain &a, const Domain &b)

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

  Implementation of Domain::icmp().
- virtual float accuracy () const

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

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

  Implementation of AccuracyDomain::setBottom().
- virtual bool isTop () const

  Implementation of AccuracyDomain::isTop().
- virtual void setTop ()

  Implementation of AccuracyDomain::setTop().

Public Attributes

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

Additional Inherited Members

16.2.1 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)

16.2.2 Member Function Documentation

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

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

Implements Canal::Domain.

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

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

Implements Canal::Domain.

16.2.2.3 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.

16.2.2.4 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.

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

Highest signed number represented by this abstract domain.

Parameters

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

Returns

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

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

Lowest signed number represented by this abstract domain.

Parameters

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

Returns

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

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

Highest unsigned number represented by this abstract domain.

Parameters

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

Returns

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

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

Lowest unsigned number represented by this abstract domain.

Parameters

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

Returns

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

16.2.3 Member Data Documentation

16.2.3.1 llvm::APInt Canal::Integer::Bitfield::mOnes

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

16.2.3.2 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

16.3 Canal::Callbacks Class Reference

```
#include <Callbacks.h>
```

Public Member Functions

- virtual void **beforeStore** (const llvm::StoreInst &instruction, State &state, const Environment &environment, Domain ∗pointer, Domain ∗value)
- virtual void **afterStore** (const llvm::StoreInst &instruction, State &state, const Environment &environment, Domain ∗pointer, Domain ∗value)

16.3.1 Detailed Description

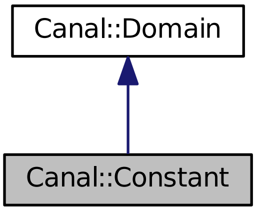
Universal class for program behavior monitoring. It can be used via inheritance to detect invalid operations, to find static analysis precision bottlenecks, to identify situations that reach boundaries of implemented techniques, and to drive custom static analysis techniques.

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

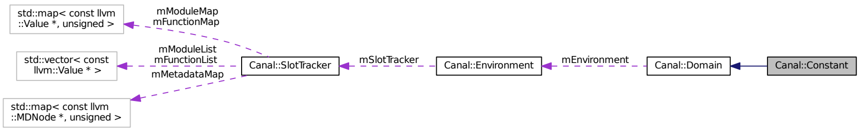
- lib/Callbacks.h

16.4 Canal::Constant Class Reference

Inheritance diagram for Canal::Constant:



Collaboration diagram for Canal::Constant:



Public Member Functions

- **Constant** (const Environment &environment, const llvm::Constant ∗constant)
- bool **isAPInt** () const
- const llvm::APInt & **getAPInt** () const
- bool **isNullPtr** () const
- bool **isGetElementPtr** () const
- Domain ∗ **toModifiableValue** () const
- virtual Constant ∗ clone () const

  Create a copy of this value.
- virtual Constant ∗ cloneCleaned () const

- virtual bool operator== (const Domain &value) const
- virtual size_t memoryUsage () const

  Get memory usage (used byte count) of this abstract value.
- virtual std::string toString () const

  Create a string representation of the abstract value.
- virtual bool matchesString (const std::string &text, std::string &rationale) const
- virtual void merge (const Domain &value)

  Merge another value into this one.

Public Attributes

- const llvm::Constant ∗ **mConstant**

Additional Inherited Members

16.4.1 Member Function Documentation

16.4.1.1 Constant ∗ Canal::Constant::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.

16.4.1.2 bool Canal::Constant::matchesString (const std::string & *text,* std::string & *rationale*) const [virtual]

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

Implements Canal::Domain.

16.4.1.3 bool Canal::Constant::operator== (const Domain & *value*) const [virtual]

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

Implements Canal::Domain.

16.4.1.4 std::string Canal::Constant::toString ( ) const [virtual]

Create a string representation of the abstract value.

An idea for different memory interpretation. virtual Domain ∗castTo(const llvm::Type ∗itemType, int offset) const = 0;

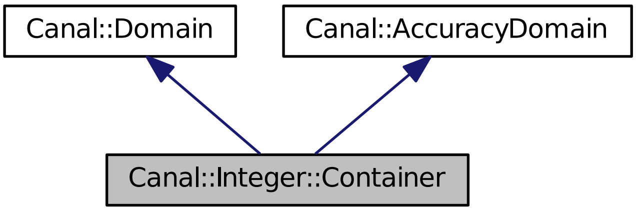
Implements Canal::Domain.

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

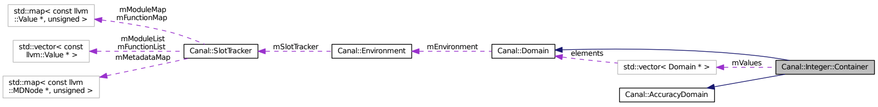
- lib/Constant.h
- lib/Constant.cpp

16.5 Canal::Integer::Container Class Reference

Inheritance diagram for Canal::Integer::Container:



Collaboration diagram for Canal::Integer::Container:



Public Member Functions

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

  Copy constructor. Creates independent copy of the container.
- virtual ~Container ()

  Destructor. Deletes the contents of the container.
- unsigned **getBitWidth** () const
- Bitfield & **getBitfield** ()
- const Bitfield & **getBitfield** () const
- Enumeration & **getEnumeration** ()
- const Enumeration & **getEnumeration** () const
- Interval & **getInterval** ()
- const Interval & **getInterval** () const
- bool signedMin (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)

  Implementation of Domain::merge().
- virtual size_t memoryUsage () const

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

  Implementation of Domain::toString().
- virtual bool matchesString (const std::string &text, std::string &rationale) const
- virtual void add (const Domain &a, const Domain &b)

  Implementation of Domain::add().
- virtual void sub (const Domain &a, const Domain &b)

  Implementation of Domain::sub().
- virtual void mul (const Domain &a, const Domain &b)

  Implementation of Domain::mul().
- virtual void udiv (const Domain &a, const Domain &b)

  Implementation of Domain::udiv().
- virtual void sdiv (const Domain &a, const Domain &b)

  Implementation of Domain::sdiv().
- virtual void urem (const Domain &a, const Domain &b)

  Implementation of Domain::urem().
- virtual void srem (const Domain &a, const Domain &b)

  Implementation of Domain::srem().
- virtual void shl (const Domain &a, const Domain &b)

  Implementation of Domain::shl().
- virtual void lshr (const Domain &a, const Domain &b)

  Implementation of Domain::lshr().
- virtual void ashr (const Domain &a, const Domain &b)

  Implementation of Domain::ashr().
- virtual void and_ (const Domain &a, const Domain &b)

  Implementation of Domain::and_().
- virtual void or_ (const Domain &a, const Domain &b)

  Implementation of Domain::or_().
- virtual void xor_ (const Domain &a, const Domain &b)

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

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

  Implementation of Domain::fcmp().
- virtual float accuracy () const

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

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

  Implementation of AccuracyDomain::setBottom().
- virtual bool isTop () const

  Implementation of AccuracyDomain::isTop().
- virtual void setTop ()

Implementation of AccuracyDomain::setTop().

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

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

16.5.2 Member Function Documentation

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

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

Implements Canal::Domain.

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

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

Implements Canal::Domain.

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

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

Implements Canal::Domain.

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

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

Parameters

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

Returns

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

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

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

Parameters

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

Returns

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

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

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

Parameters

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

Returns

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

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

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

Parameters

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

Returns

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

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

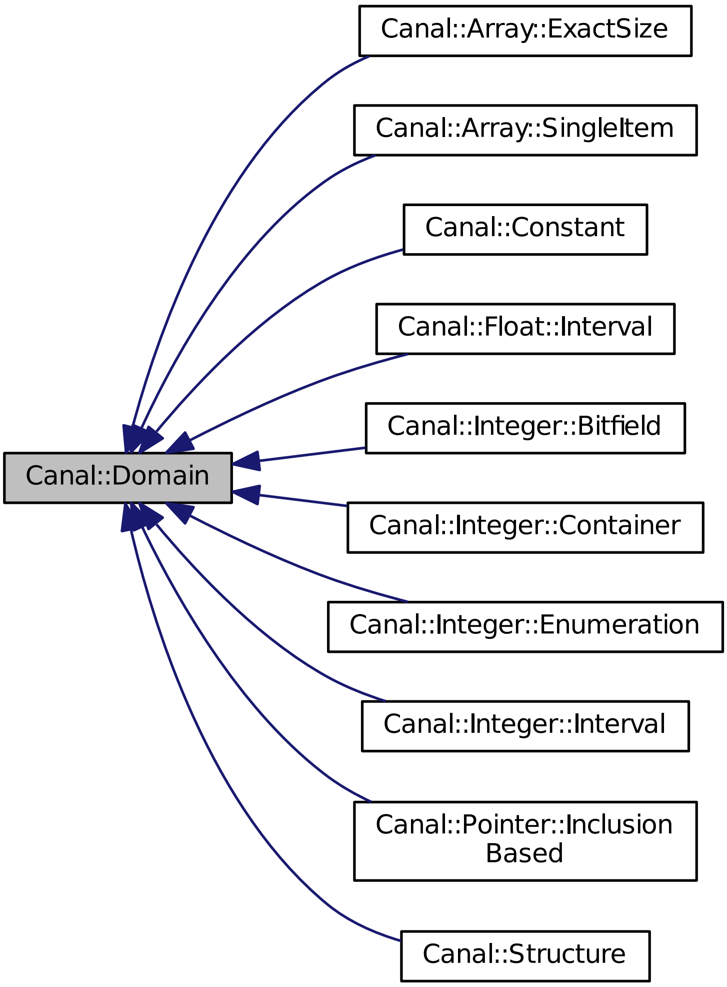
- lib/IntegerContainer.h
- lib/IntegerContainer.cpp

16.6 Canal::Domain Class Reference

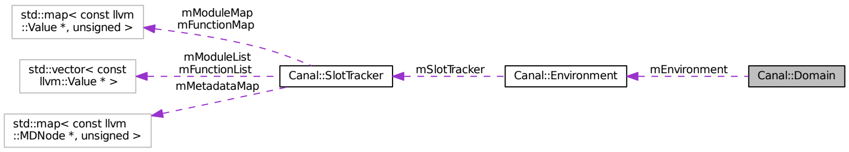
Base class for all abstract domains.

`#include <Domain.h>`

Inheritance diagram for Canal::Domain:



Collaboration diagram for Canal::Domain:



Public Types

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

Public Member Functions

- Domain (const Environment &environment)

  Standard constructor.
- virtual Domain ∗ clone () const =0

  Create a copy of this value.
- virtual Domain ∗ cloneCleaned () const =0
- virtual bool operator== (const Domain &value) const =0
- virtual bool operator!= (const Domain &value) const

  Inequality is implemented by calling the equality operator.
- virtual void merge (const Domain &value)

  Merge another value into this one.
- virtual size_t memoryUsage () const =0

  Get memory usage (used byte count) of this abstract value.
- virtual std::string toString () const =0

  Create a string representation of the abstract value.
- virtual bool matchesString (const std::string &text, std::string &rationale) const =0
- virtual void add (const Domain &a, const Domain &b)

  Implementation of instructions operating on values.
- virtual void **fadd** (const Domain &a, const Domain &b)
- virtual void **sub** (const Domain &a, const Domain &b)
- virtual void **fsub** (const Domain &a, const Domain &b)
- virtual void **mul** (const Domain &a, const Domain &b)
- virtual void **fmul** (const Domain &a, const Domain &b)
- virtual void udiv (const Domain &a, const Domain &b)

  Unsigned division.
- virtual void sdiv (const Domain &a, const Domain &b)

  Signed division.
- virtual void fdiv (const Domain &a, const Domain &b)

  Floating point division.

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

Static Public Member Functions

- static Domain ∗ handleMergeConstants (Domain ∗what, const Domain ∗target)

  Prepare value so that merge will not fail on assert when what is Constant.

Public Attributes

- const Environment & **mEnvironment**

16.6.1 Detailed Description

Base class for all abstract domains.

16.6.2 Member Function Documentation

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

Implementation of instructions operating on values.

Load the abstract value state from a string representation.

Parameters

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

Returns

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

Reimplemented in Canal::Integer::Container, Canal::Integer::Bitfield, Canal::Integer::Interval, Canal::Integer::Enumeration, Canal::Array::SingleItem, and Canal::Array::ExactSize.

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

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

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

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

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

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

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

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

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

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

Create a string representation of the abstract value.

An idea for different memory interpretation. virtual Domain ∗castTo(const llvm::Type ∗itemType, int offset) const = 0;

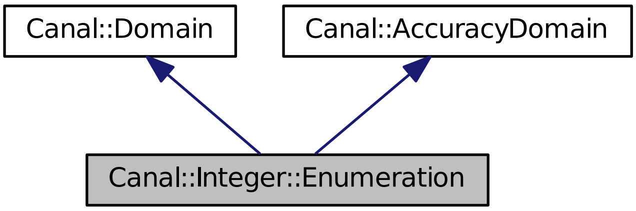
Implemented in Canal::Pointer::InclusionBased, Canal::Integer::Bitfield, Canal::Integer::Container, Canal::Integer::Interval, Canal::Integer::Enumeration, Canal::Float::Interval, Canal::Constant, Canal::Array::SingleItem, Canal::Array::ExactSize, and Canal::Structure.

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

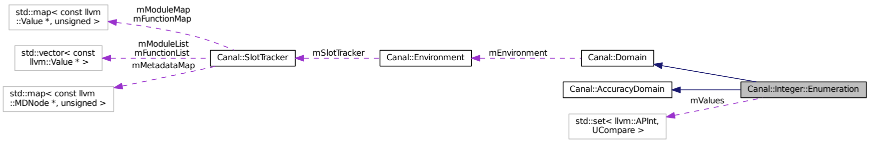
- lib/Domain.h
- lib/Domain.cpp

16.7 Canal::Integer::Enumeration Class Reference

Inheritance diagram for Canal::Integer::Enumeration:



Collaboration diagram for Canal::Integer::Enumeration:


std::map< const llvm ::Value *, unsigned >
mModuleMap
mFunctionMap
std::vector< const llvm::Value * >
mModuleList
mFunctionList
mMetadataMap
std::map< const llvm ::MDNode *, unsigned >
Canal::SlotTracker
mSlotTracker
Canal::Environment
mEnvironment
Canal::Domain
Canal::AccuracyDomain
Canal::Integer::Enumeration
mValues
std::set< llvm::APInt, UCompare >

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

  Does this enumeration represent single value?
- virtual Enumeration ∗ clone () const
- virtual Enumeration ∗ cloneCleaned () const
- virtual bool operator== (const Domain &value) const

  Implementation of Domain::operator==().
- virtual void merge (const Domain &value)

  Implementation of Domain::merge().
- virtual size_t memoryUsage () const

Implementation of Domain::memoryUsage().

- virtual std::string toString () const

  Implementation of Domain::toString().
- virtual bool matchesString (const std::string &text, std::string &rationale) const
- virtual void add (const Domain &a, const Domain &b)

  Implementation of Domain::add().
- virtual void sub (const Domain &a, const Domain &b)

  Implementation of Domain::sub().
- virtual void mul (const Domain &a, const Domain &b)

  Implementation of Domain::mul().
- virtual void udiv (const Domain &a, const Domain &b)

  Implementation of Domain::udiv().
- virtual void sdiv (const Domain &a, const Domain &b)

  Implementation of Domain::sdiv().
- virtual void urem (const Domain &a, const Domain &b)

  Implementation of Domain::urem().
- virtual void srem (const Domain &a, const Domain &b)

  Implementation of Domain::srem().
- virtual void shl (const Domain &a, const Domain &b)

  Implementation of Domain::shl().
- virtual void lshr (const Domain &a, const Domain &b)

  Implementation of Domain::lshr().
- virtual void ashr (const Domain &a, const Domain &b)

  Implementation of Domain::ashr().
- virtual void and_ (const Domain &a, const Domain &b)

  Implementation of Domain::and_().
- virtual void or_ (const Domain &a, const Domain &b)

  Implementation of Domain::or_().
- virtual void xor_ (const Domain &a, const Domain &b)

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

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

  Implementation of Domain::fcmp().
- virtual float accuracy () const

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

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

  Implementation of AccuracyDomain::setBottom().
- virtual bool isTop () const

  Implementation of AccuracyDomain::isTop().
- virtual void setTop ()

  Implementation of AccuracyDomain::setTop().

Public Attributes

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

Protected Member Functions

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

Additional Inherited Members

16.7.1 Member Function Documentation

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

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

Implements Canal::Domain.

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

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

Implements Canal::Domain.

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

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

Implements Canal::Domain.

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

Highest signed number represented by this abstract domain.

Parameters

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

Returns

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

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

Lowest signed number represented by this abstract domain.

Parameters

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

Returns

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

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

Highest unsigned number represented by this abstract domain.

Parameters

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

Returns

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

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

Lowest unsigned number represented by this abstract domain.

Parameters

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

Returns

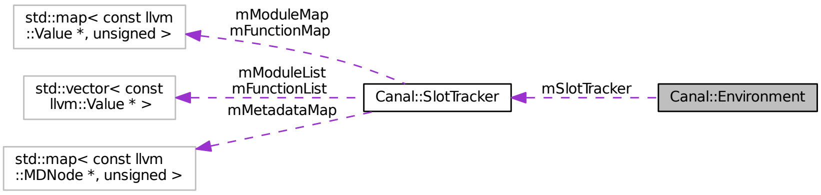
True if the result is known and the parameter was set to correct value.

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

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

16.8 Canal::Environment Class Reference

Collaboration diagram for Canal::Environment:



Public Member Functions

- **Environment** (const llvm::Module &module)
- llvm::LLVMContext & **getContext** () const

Public Attributes

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

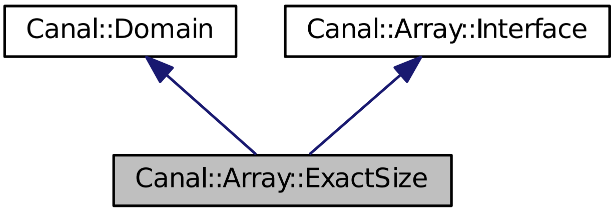
The documentation for this class was generated from the following files:

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

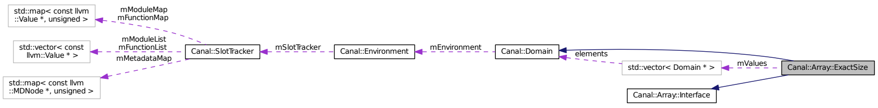
16.9 Canal::Array::ExactSize Class Reference

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

Inheritance diagram for Canal::Array::ExactSize:



Collaboration diagram for Canal::Array::ExactSize:



Public Member Functions

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

  Implementation of Domain::operator==().
- virtual void merge (const Domain &value)

  Implementation of Domain::merge().
- virtual size_t memoryUsage () const

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

  Implementation of Domain::toString().
- virtual bool matchesString (const std::string &text, std::string &rationale) const

  Implementation of Domain::matchesString().
- virtual void add (const Domain &a, const Domain &b)

  Implementation of Domain::add().
- virtual void fadd (const Domain &a, const Domain &b)

  Implementation of Domain::fadd().
- virtual void sub (const Domain &a, const Domain &b)

  Implementation of Domain::sub().
- virtual void fsub (const Domain &a, const Domain &b)

  Implementation of Domain::fsub().
- virtual void mul (const Domain &a, const Domain &b)

  Implementation of Domain::mul().
- virtual void fmul (const Domain &a, const Domain &b)

  Implementation of Domain::fmul().
- virtual void udiv (const Domain &a, const Domain &b)

  Implementation of Domain::udiv().
- virtual void sdiv (const Domain &a, const Domain &b)

  Implementation of Domain::sdiv().
- virtual void fdiv (const Domain &a, const Domain &b)

  Implementation of Domain::fdiv().
- virtual void urem (const Domain &a, const Domain &b)

  Implementation of Domain::urem().
- virtual void srem (const Domain &a, const Domain &b)

  Implementation of Domain::srem().

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

  Implementation of Domain::frem().
- virtual void shl (const Domain &a, const Domain &b)

  Implementation of Domain::shl().
- virtual void lshr (const Domain &a, const Domain &b)

  Implementation of Domain::lshr().
- virtual void ashr (const Domain &a, const Domain &b)

  Implementation of Domain::ashr().
- virtual void and_ (const Domain &a, const Domain &b)

  Implementation of Domain::and_().
- virtual void or_ (const Domain &a, const Domain &b)

  Implementation of Domain::or_().
- virtual void xor_ (const Domain &a, const Domain &b)

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

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

  Implementation of Domain::fcmp().
- virtual std::vector< Domain ∗ > getItem (const Domain &offset) const

  Implementation of Array::Interface::getItem().
- virtual Domain ∗ getItem (uint64_t offset) const

  Implementation of Array::Interface::getItem().
- virtual void setItem (const Domain &offset, const Domain &value)

  Implementation of Array::Interface::setItem().
- virtual void setItem (uint64_t offset, const Domain &value)

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

Public Attributes

- std::vector< Domain ∗ > **mValues**

Additional Inherited Members

16.9.1 Detailed Description

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

16.9.2 Member Function Documentation

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

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

Implements Canal::Domain.

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

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

Implements Canal::Domain.

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

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

16.10 Canal::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

16.11 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:

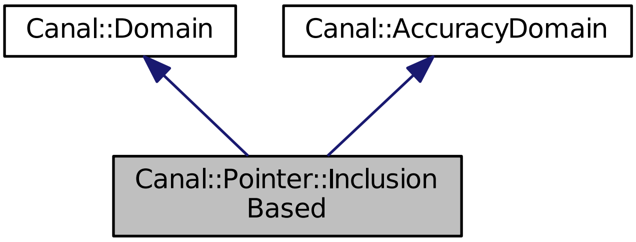
- lib/FunctionModelManager.h

16.12 Canal::Pointer::InclusionBased Class Reference

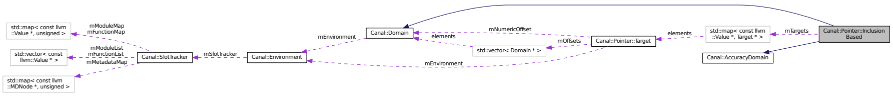
Inclusion-based flow-insensitive abstract pointer.

`#include <Pointer.h>`

Inheritance diagram for Canal::Pointer::InclusionBased:



Collaboration diagram for Canal::Pointer::InclusionBased:



Public Member Functions

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

  Standard constructor.
- InclusionBased (const InclusionBased &second)

  Copy constructor.
- InclusionBased (const InclusionBased &second, const llvm::Type &newType)
- virtual ~InclusionBased ()

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

  Creates a copy of this object with a different pointer type.
- InclusionBased ∗ getElementPtr (const std::vector< Domain ∗ > &offsets, const llvm::Type &type) const
- void **store** (const Domain &value, State &state)
- virtual InclusionBased ∗ clone () const
- virtual InclusionBased ∗ cloneCleaned () const
- virtual bool operator== (const Domain &value) const

  Implementation of Domain::operator==().
- bool isSingleTarget () const

  Does this pointer point to single target?
- virtual void merge (const Domain &value)

Implementation of Domain::merge().

- virtual size_t memoryUsage () const

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

  Implementation of Domain::toString().
- virtual bool matchesString (const std::string &text, std::string &rationale) const

  Implementation of Domain::matchesString().
- virtual float accuracy () const

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

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

  Implementation of AccuracyDomain::setBottom().
- virtual bool isTop () const

  Implementation of AccuracyDomain::isTop().
- virtual void setTop ()

  Implementation of AccuracyDomain::setTop().

Public Attributes

- PlaceTargetMap mTargets
- const llvm::Type & mType

  The type object is owned by the LLVM framework.
- bool mTop

  If true, this pointer can point anywhere.

Additional Inherited Members

16.12.1 Detailed Description

Inclusion-based flow-insensitive abstract pointer.

16.12.2 Constructor & Destructor Documentation

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

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

16.12.3 Member Function Documentation

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

Add a new target to the pointer.

Parameters

type	Type of the referenced memory.
instruction	Place where the pointer target is added.
target	Represents the target memory block. If type is Constant, it must be NULL. Otherwise, it must be a valid pointer to an instruction. This is a key to State::mFunctionBlocks, State::mFunctionVariables, State::mGlobalBlocks, or State::mGlobalVariables, depending on the type.
offsets	Offsets in the getelementptr style. The provided vector might be empty. The newly created pointer target becomes the owner of the objects in the vector.
numericOffset	Numerical offset that is used in addition to the getelementptr style offset and after they have been applied. It might be NULL, which indicates the offset 0. The newly created pointer target becomes the owner of the numerical offset when it's provided. This parameter is mandatory for pointers of Constant type, because it contains the constant.

16.12.3.2 InclusionBased ∗ Canal::Pointer::InclusionBased::clone ( ) const [virtual]

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

Implements Canal::Domain.

16.12.3.3 InclusionBased ∗ Canal::Pointer::InclusionBased::cloneCleaned ( ) const [virtual]

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

Implements Canal::Domain.

16.12.3.4 Domain ∗ Canal::Pointer::InclusionBased::dereferenceAndMerge (const State & *state*) const

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

Returns

It might return NULL.

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

Creates a copy of this object pointing to subtargets.

Parameters

offsets	Pointer takes ownership of the values inside the vector.

16.12.4 Member Data Documentation

16.12.4.1 PlaceTargetMap Canal::Pointer::InclusionBased::mTargets

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

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

The type object is owned by the LLVM framework.

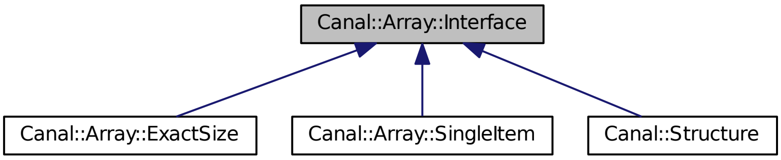
Type of the object the pointer is pointing to. It might be incompatible with the type of the actual abstract value. Conversion is needed during store and load operations in such a case.

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

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

16.13 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

16.13.1 Member Function Documentation

16.13.1.1 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::SingleItem, Canal::Array::ExactSize, and Canal::Structure.

16.13.1.2 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::SingleItem, Canal::Array::ExactSize, and Canal::Structure.

16.13.1.3 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.

16.13.1.4 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.

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

Parameters

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

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

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

Parameters

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

Note

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

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

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

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

16.14 Canal::Interpreter Class Reference

```
#include <Interpreter.h>
```

Public Member Functions

- void addGlobalVariables (State &state, const Environment &environment)
- virtual bool step (Stack &stack, const Environment &environment)
- void interpretInstruction (Stack &stack, const Environment &environment)

  Interprets current instruction.

Protected Member Functions

- virtual void ret (const llvm::ReturnInst &instruction, State &state, const Environment &environment)
- virtual void br (const llvm::BranchInst &instruction, State &state, const Environment &environment)
- virtual void switch_ (const llvm::SwitchInst &instruction, State &state, const Environment &environment)
- virtual void indirectbr (const llvm::IndirectBrInst &instruction, State &state, const Environment &environment)
- virtual void invoke (const llvm::InvokeInst &instruction, Stack &stack, const Environment &environment)
- virtual void unreachable (const llvm::UnreachableInst &instruction, State &state, const Environment &environment)
- virtual void add (const llvm::BinaryOperator &instruction, State &state, const Environment &environment)

  Sum of two operands. It's a binary operator.
- virtual void fadd (const llvm::BinaryOperator &instruction, State &state, const Environment &environment)
- virtual void sub (const llvm::BinaryOperator &instruction, State &state, const Environment &environment)

  Difference of two operands. It's a binary operator.
- virtual void fsub (const llvm::BinaryOperator &instruction, State &state, const Environment &environment)
- virtual void mul (const llvm::BinaryOperator &instruction, State &state, const Environment &environment)

  Product of two operands. It's a binary operator.
- virtual void fmul (const llvm::BinaryOperator &instruction, State &state, const Environment &environment)

  Product of two operands. It's a binary operator.
- virtual void udiv (const llvm::BinaryOperator &instruction, State &state, const Environment &environment)
- virtual void sdiv (const llvm::BinaryOperator &instruction, State &state, const Environment &environment)
- virtual void fdiv (const llvm::BinaryOperator &instruction, State &state, const Environment &environment)
- virtual void urem (const llvm::BinaryOperator &instruction, State &state, const Environment &environment)

  Unsigned division remainder. It's a binary operator.

- virtual void srem (const llvm::BinaryOperator &instruction, State &state, const Environment &environment)

  Signed division remainder. It's a binary operator.
- virtual void frem (const llvm::BinaryOperator &instruction, State &state, const Environment &environment)

  Floating point remainder. It's a binary operator.
- virtual void shl (const llvm::BinaryOperator &instruction, State &state, const Environment &environment)

  It's a bitwise binary operator.
- virtual void lshr (const llvm::BinaryOperator &instruction, State &state, const Environment &environment)

  It's a bitwise binary operator.
- virtual void ashr (const llvm::BinaryOperator &instruction, State &state, const Environment &environment)

  It's a bitwise binary operator.
- virtual void and_ (const llvm::BinaryOperator &instruction, State &state, const Environment &environment)

  It's a bitwise binary operator.
- virtual void or_ (const llvm::BinaryOperator &instruction, State &state, const Environment &environment)

  It's a bitwise binary operator.
- virtual void xor_ (const llvm::BinaryOperator &instruction, State &state, const Environment &environment)

  It's a bitwise binary operator.
- virtual void extractelement (const llvm::ExtractElementInst &instruction, State &state, const Environment &environment)

  It's a vector operation.
- virtual void insertelement (const llvm::InsertElementInst &instruction, State &state, const Environment &environment)

  It's a vector operation.
- virtual void shufflevector (const llvm::ShuffleVectorInst &instruction, Stack &stack, const Environment &environment)

  It's a vector operation.
- virtual void extractvalue (const llvm::ExtractValueInst &instruction, State &state, const Environment &environment)

  It's an aggregate operation.
- virtual void insertvalue (const llvm::InsertValueInst &instruction, State &state, const Environment &environment)

  It's an aggregate operation.
- virtual void alloca_ (const llvm::AllocaInst &instruction, Stack &stack, const Environment &environment)

  It's a memory access operation.
- virtual void load (const llvm::LoadInst &instruction, State &state, const Environment &environment)

  It's a memory access operation.
- virtual void store (const llvm::StoreInst &instruction, State &state, const Environment &environment)

  It's a memory access operation.

- virtual void getelementptr (const llvm::GetElementPtrInst &instruction, Stack &stack, const Environment &environment)

  It's a memory addressing operation.
- virtual void trunc (const llvm::TruncInst &instruction, State &state, const Environment &environment)

  It's a conversion operation.
- virtual void zext (const llvm::ZExtInst &instruction, State &state, const Environment &environment)

  It's a conversion operation.
- virtual void sext (const llvm::SExtInst &instruction, State &state, const Environment &environment)

  It's a conversion operation.
- virtual void fptrunc (const llvm::FPTruncInst &instruction, State &state, const Environment &environment)

  It's a conversion operation.
- virtual void fpext (const llvm::FPExtInst &instruction, State &state, const Environment &environment)

  It's a conversion operation.
- virtual void fptoui (const llvm::FPToUIInst &instruction, State &state, const Environment &environment)

  It's a conversion operation.
- virtual void fptosi (const llvm::FPToSIInst &instruction, State &state, const Environment &environment)

  It's a conversion operation.
- virtual void uitofp (const llvm::UIToFPInst &instruction, State &state, const Environment &environment)

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

  It's a conversion operation.
- virtual void ptrtoint (const llvm::PtrToIntInst &instruction, State &state, const Environment &environment)

  It's a conversion operation.
- virtual void inttoptr (const llvm::IntToPtrInst &instruction, State &state, const Environment &environment)

  It's a conversion operation.
- virtual void bitcast (const llvm::BitCastInst &instruction, State &state, const Environment &environment)

  It's a conversion operation.
- virtual void **icmp** (const llvm::ICmpInst &instruction, State &state, const Environment &environment)
- virtual void **fcmp** (const llvm::FCmpInst &instruction, State &state, const Environment &environment)
- virtual void **phi** (const llvm::PHINode &instruction, State &state, const Environment &environment)
- virtual void **select** (const llvm::SelectInst &instruction, State &state, const Environment &environment)
- virtual void **call** (const llvm::CallInst &instruction, Stack &stack, const Environment &environment)
- virtual void **va_arg** (const llvm::VAArgInst &instruction, State &state, const Environment &environment)

16.14.1 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.

16.14.2 Member Function Documentation

16.14.2.1 void Canal::Interpreter::addGlobalVariables (State & *state*, const Environment & *environment*)

Adds all global variables and constants from a module to the state.

16.14.2.2 void Canal::Interpreter::br (const llvm::BranchInst & *instruction*, State & *state*, const Environment & *environment*) [protected], [virtual]

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

16.14.2.3 void Canal::Interpreter::fadd (const llvm::BinaryOperator & *instruction*, State & *state*, const Environment & *environment*) [protected], [virtual]

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

16.14.2.4 void Canal::Interpreter::fdiv (const llvm::BinaryOperator & *instruction*, State & *state*, const Environment & *environment*) [protected], [virtual]

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

16.14.2.5 void Canal::Interpreter::fsub (const llvm::BinaryOperator & *instruction*, State & *state*, const Environment & *environment*) [protected], [virtual]

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

16.14.2.6 void Canal::Interpreter::indirectbr (const llvm::IndirectBrInst & *instruction*, State & *state*, const Environment & *environment*) [protected], [virtual]

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

16.14.2.7 void Canal::Interpreter::invoke (const llvm::InvokeInst & *instruction*, Stack & *stack*, const Environment & *environment*) [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.

16.14.2.8 void Canal::Interpreter::ret (const llvm::ReturnInst & *instruction*, State & *state*, const Environment & *environment*) [protected], [virtual]

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

16.14.2.9 void Canal::Interpreter::sdiv (const llvm::BinaryOperator & *instruction*, State & *state*, const Environment & *environment*) [protected], [virtual]

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

16.14.2.10 bool Canal::Interpreter::step (Stack & *stack*, const Environment & *environment*) [virtual]

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

Returns

True if next step is possible. False on the end of the program.

16.14.2.11 void Canal::Interpreter::switch_ (const llvm::SwitchInst & *instruction*, State & *state*, const Environment & *environment*) [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.

16.14.2.12 void Canal::Interpreter::udiv (const llvm::BinaryOperator & *instruction*, State & *state*, const Environment & *environment*) [protected], [virtual]

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

16.14.2.13 void Canal::Interpreter::unreachable (const llvm::UnreachableInst & *instruction*, State & *state*, const Environment & *environment*) [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.

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

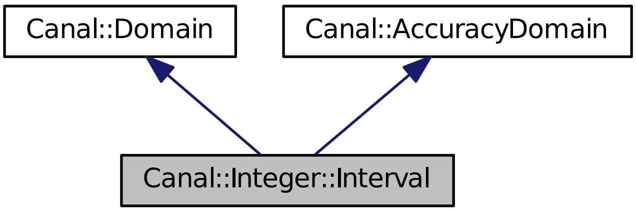
- lib/Interpreter.h
- lib/Interpreter.cpp

16.15 Canal::Integer::Interval Class Reference

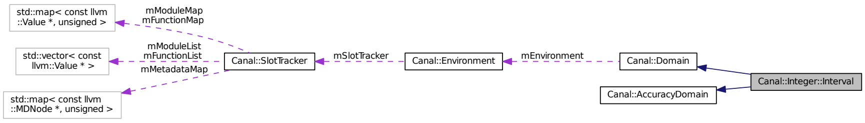
Abstracts integer values as a interval min - max.

`#include <IntegerInterval.h>`

Inheritance diagram for Canal::Integer::Interval:



Collaboration diagram for Canal::Integer::Interval:



Public Member Functions

- Interval (const Environment &environment, unsigned bitWidth)

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

  Standard constructor.
- unsigned **getBitWidth** () const
- bool signedMin (llvm::APInt &result) const
- bool signedMax (llvm::APInt &result) const
- bool unsignedMin (llvm::APInt &result) const
- bool unsignedMax (llvm::APInt &result) const
- bool isSingleValue () const
- virtual Interval ∗ clone () const
- virtual Interval ∗ cloneCleaned () const
- virtual bool operator== (const Domain &value) const

  Implementation of Domain::operator==().
- virtual void merge (const Domain &value)

  Implementation of Domain::merge().
- virtual size_t memoryUsage () const

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

  Implementation of Domain::toString().
- virtual bool matchesString (const std::string &text, std::string &rationale) const

Implementation of Domain::matchesString().

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

  Implementation of Domain::add().

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

  Implementation of Domain::sub().

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

  Implementation of Domain::mul().

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

  Implementation of Domain::udiv().

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

  Implementation of Domain::sdiv().

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

  Implementation of Domain::urem().

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

  Implementation of Domain::srem().

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

  Implementation of Domain::shl().

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

  Implementation of Domain::lshr().

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

  Implementation of Domain::ashr().

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

  Implementation of Domain::and_().

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

  Implementation of Domain::or_().

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

  Implementation of Domain::xor_().

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

  Implementation of Domain::icmp().

- virtual float accuracy () const

  Implementation of AccuracyDomain::accuracy().

- virtual bool isBottom () const

  Implementation of AccuracyDomain::isBottom().

- virtual void setBottom ()

  Implementation of AccuracyDomain::setBottom().

- virtual bool isTop () const

  Implementation of AccuracyDomain::isTop().

- virtual void setTop ()

  Implementation of AccuracyDomain::setTop().

Public Attributes

- bool mEmpty

  Indicates an empty interval.
- bool **mSignedTop**
- llvm::APInt mSignedFrom

  The number is included in the interval.
- llvm::APInt mSignedTo

  The number is included in the interval.
- bool **mUnsignedTop**
- llvm::APInt mUnsignedFrom

  The number is included in the interval.
- llvm::APInt mUnsignedTo

  The number is included in the interval.

Additional Inherited Members

16.15.1 Detailed Description

Abstracts integer values as a interval min - max.

16.15.2 Constructor & Destructor Documentation

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

Standard constructor.

Initializes an empty interval.

16.15.3 Member Function Documentation

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

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

Implements Canal::Domain.

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

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

Implements Canal::Domain.

16.15.3.3 bool Canal::Integer::Interval::isSingleValue ( ) const

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

16.15.3.4 bool Canal::Integer::Interval::signedMax (llvm::APInt & *result*) const

Highest signed number represented by this abstract domain.

Parameters

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

Returns

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

16.15.3.5 bool Canal::Integer::Interval::signedMin (llvm::APInt & *result*) const

Lowest signed number represented by this abstract domain.

Parameters

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

Returns

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

16.15.3.6 bool Canal::Integer::Interval::unsignedMax (llvm::APInt & *result*) const

Highest unsigned number represented by this abstract domain.

Parameters

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

Returns

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

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

Lowest unsigned number represented by this abstract domain.

Parameters

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

Returns

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

16.15.4 Member Data Documentation

16.15.4.1 bool Canal::Integer::Interval::mEmpty

Indicates an empty interval.

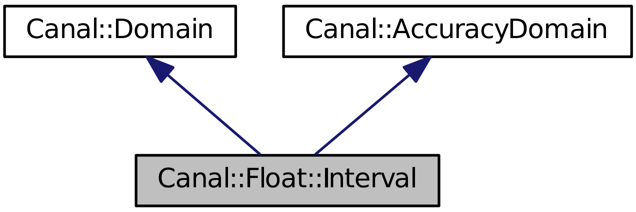
When it is set to true, other members' values are not considered as valid.

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

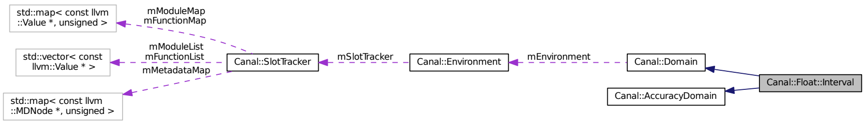
- lib/IntegerInterval.h
- lib/IntegerInterval.cpp

16.16 Canal::Float::Interval Class Reference

Inheritance diagram for Canal::Float::Interval:



Collaboration diagram for Canal::Float::Interval:



Public Member Functions

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

- llvm::APFloat **getMin** () const
- virtual Interval ∗ clone () const

  Create a copy of this value.
- virtual Interval ∗ cloneCleaned () const
- virtual bool operator== (const Domain &value) const
- virtual void merge (const Domain &value)

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

  Is it the lowest value.
- virtual void setBottom ()

  Set to the lowest value.
- virtual bool isTop () const

  Is it the highest value.
- virtual void setTop ()

  Set it to the top value of lattice.

Public Attributes

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

Additional Inherited Members

16.16.1 Member Function Documentation

16.16.1.1 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.

16.16.1.2 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.

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

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

Implements Canal::Domain.

16.16.1.4 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.

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

Create a string representation of the abstract value.

An idea for different memory interpretation. virtual Domain ∗castTo(const llvm::Type ∗itemType, int offset) const = 0;

Implements Canal::Domain.

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

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

16.17 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

16.18 Canal::SELinuxModulePass Class Reference

Public Member Functions

- virtual bool **runOnModule** (llvm::Module &module)
- void **interpretFunction** (const llvm::Function &F, const std::vector< Domain > &Arguments)
- virtual void **getAnalysisUsage** (llvm::AnalysisUsage &AU) const

Static Public Attributes

- static char **ID** = 0

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

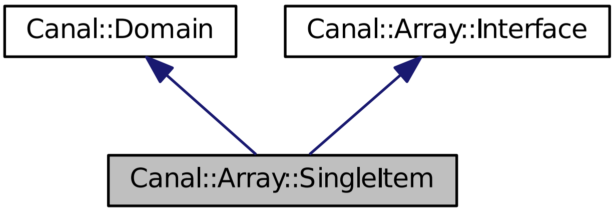
- lib/SELinuxModulePass.cpp

16.19 Canal::Array::SingleItem Class Reference

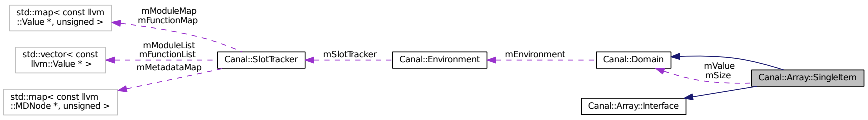
This array type is very imprecise.

`#include <ArraySingleItem.h>`

Inheritance diagram for Canal::Array::SingleItem:



Collaboration diagram for Canal::Array::SingleItem:



Public Member Functions

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

  Implementation of Domain::operator==().
- virtual void merge (const Domain &value)

  Implementation of Domain::merge().
- virtual size_t memoryUsage () const

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

  Implementation of Domain::toString().

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

  Implementation of Domain::matchesString().
- virtual void add (const Domain &a, const Domain &b)

  Implementation of Domain::add().
- virtual void fadd (const Domain &a, const Domain &b)

  Implementation of Domain::fadd().
- virtual void sub (const Domain &a, const Domain &b)

  Implementation of Domain::sub().
- virtual void fsub (const Domain &a, const Domain &b)

  Implementation of Domain::fsub().
- virtual void mul (const Domain &a, const Domain &b)

  Implementation of Domain::mul().
- virtual void fmul (const Domain &a, const Domain &b)

  Implementation of Domain::fmul().
- virtual void udiv (const Domain &a, const Domain &b)

  Implementation of Domain::udiv().
- virtual void sdiv (const Domain &a, const Domain &b)

  Implementation of Domain::sdiv().
- virtual void fdiv (const Domain &a, const Domain &b)

  Implementation of Domain::fdiv().
- virtual void urem (const Domain &a, const Domain &b)

  Implementation of Domain::urem().
- virtual void srem (const Domain &a, const Domain &b)

  Implementation of Domain::srem().
- virtual void frem (const Domain &a, const Domain &b)

  Implementation of Domain::frem().
- virtual void shl (const Domain &a, const Domain &b)

  Implementation of Domain::shl().
- virtual void lshr (const Domain &a, const Domain &b)

  Implementation of Domain::lshr().
- virtual void ashr (const Domain &a, const Domain &b)

  Implementation of Domain::ashr().
- virtual void and_ (const Domain &a, const Domain &b)

  Implementation of Domain::and_().
- virtual void or_ (const Domain &a, const Domain &b)

  Implementation of Domain::or_().
- virtual void xor_ (const Domain &a, const Domain &b)

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

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

  Implementation of Domain::fcmp().
- virtual std::vector< Domain ∗ > getItem (const Domain &offset) const

  Implementation of Array::Interface::getItem().
- virtual Domain ∗ getItem (uint64_t offset) const

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

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

  Implementation of Array::Interface::setItem().
- virtual void setItem (uint64_t offset, const Domain &value)

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

Public Attributes

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

Additional Inherited Members

16.19.1 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.

16.19.2 Member Function Documentation

16.19.2.1 SingleItem ∗ Canal::Array::SingleItem::clone ( ) const [virtual]

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

Implements Canal::Domain.

16.19.2.2 SingleItem ∗ Canal::Array::SingleItem::cloneCleaned ( ) const [virtual]

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

Implements Canal::Domain.

16.19.3 Member Data Documentation

16.19.3.1 Domain∗ Canal::Array::SingleItem::mSize

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

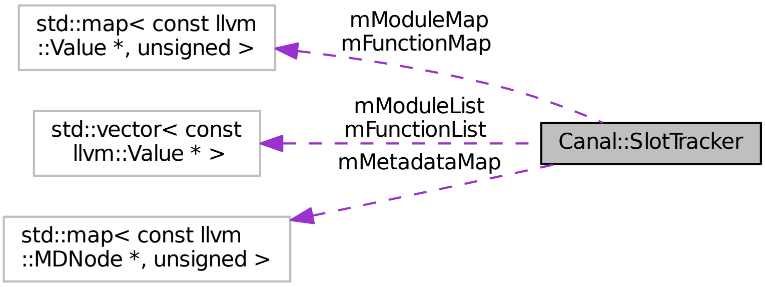
The documentation for this class was generated from the following files:

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

16.20 Canal::SlotTracker Class Reference

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

Collaboration diagram for Canal::SlotTracker:


std::map< const llvm ::Value *, unsigned >
std::vector< const llvm::Value * >
std::map< const llvm ::MDNode *, unsigned >
mModuleMap
mFunctionMap
mModuleList
mFunctionList
mMetadataMap
Canal::SlotTracker

Public Types

- typedef std::map< const llvm::MDNode ∗, unsigned > ::iterator mdn_iterator

  MDNode map iterators.

Public Member Functions

- SlotTracker (const llvm::Module &module)

  Construct from a module.
- void setActiveFunction (const llvm::Function &function)
- int getLocalSlot (const llvm::Value &value)
- const llvm::Value ∗ **getLocalSlot** (int num)
- int getGlobalSlot (const llvm::Value &value)

  Get the slot number of a global value.
- const llvm::Value ∗ **getGlobalSlot** (int num)
- int getMetadataSlot (const llvm::MDNode &node)

  Get the slot number of a MDNode.
- mdn_iterator **mdn_begin** ()
- mdn_iterator **mdn_end** ()
- unsigned **mdn_size** () const
- bool **mdn_empty** () const

Protected Types

- typedef std::map< const llvm::Value ∗, unsigned > ValueMap

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

Protected Member Functions

- void initialize ()

  This function does the actual initialization.
- void createFunctionSlot (const llvm::Value &value)

  Insert the specified Value∗ into the slot table.
- void createModuleSlot (const llvm::GlobalValue &value)

  Insert the specified GlobalValue∗ into the slot table.
- void createMetadataSlot (const llvm::MDNode &node)

  Insert the specified MDNode∗ into the slot table.
- void processModule ()
- void processFunction ()

Protected Attributes

- const llvm::Module & mModule

  The module for which we are holding slot numbers.
- bool **mModuleProcessed**
- const llvm::Function ∗ mFunction

  The function for which we are holding slot numbers.
- bool **mFunctionProcessed**
- ValueMap mModuleMap

  The slot map for the module level data.
- ValueList **mModuleList**
- unsigned **mModuleNext**
- ValueMap mFunctionMap

  The slot map for the function level data.
- ValueList **mFunctionList**
- unsigned **mFunctionNext**
- std::map< const llvm::MDNode
  ∗, unsigned > mMetadataMap

  The slot map for MDNodes.
- unsigned **mMetadataNext**

16.20.1 Detailed Description

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

16.20.2 Member Function Documentation

16.20.2.1 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.

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

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

16.20.2.3 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.

16.20.2.4 void Canal::SlotTracker::setActiveFunction (const llvm::Function & *function*)

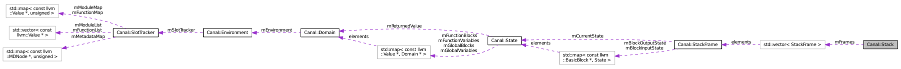
If you'd like to deal with a function instead of just a module, use this method to get its data into the SlotTracker.

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

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

16.21 Canal::Stack Class Reference

Collaboration diagram for Canal::Stack:



Public Member Functions

- bool **nextInstruction** ()
- bool **hasEnteredNewFrame** () const
- bool **hasReturnedFromFrame** () const
- std::vector< StackFrame > & **getFrames** ()
- const std::vector< StackFrame > & **getFrames** () const
- const llvm::Instruction & **getCurrentInstruction** () const
- State & **getCurrentState** ()
- const llvm::Function & **getCurrentFunction** () const
- void addFrame (const llvm::Function &function, const State &initialState)

Protected Attributes

- std::vector< StackFrame > **mFrames**
- bool **mHasEnteredNewFrame**
- bool **mHasReturnedFromFrame**

16.21.1 Member Function Documentation

16.21.1.1 void Canal::Stack::addFrame (const llvm::Function & *function,* const State & *initialState*)

Parameters

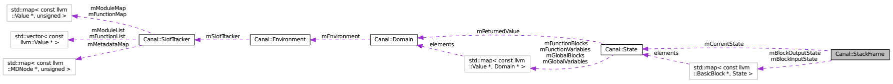
function	Function to be interpreted. Its instructions will be applied in abstract domain on the provided input state.
initialState	Initial state when entering the function. It includes global variables and function arguments.

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

- lib/Stack.h
- lib/Stack.cpp

16.22 Canal::StackFrame Class Reference

Collaboration diagram for Canal::StackFrame:



Public Member Functions

- **StackFrame** (const llvm::Function ∗function, const State &initialState)
- bool nextInstruction ()
- Domain ∗ **getReturnedValue** () const
- void **mergeGlobalVariables** (State &target) const

Public Attributes

- const llvm::Function ∗ mFunction
- std::map< const llvm::BasicBlock ∗, State > **mBlockInputState**
- std::map< const llvm::BasicBlock ∗, State > **mBlockOutputState**
- llvm::Function::const_iterator **mCurrentBlock**
- State **mCurrentState**
- llvm::BasicBlock::const_iterator **mCurrentInstruction**
- bool **mChanged**

16.22.1 Member Function Documentation

16.22.1.1 bool Canal::StackFrame::nextInstruction ( )

Returns true if next instruction should be interpreted for this frame, and false when fixpoint has been reached.

16.22.2 Member Data Documentation

16.22.2.1 const llvm::Function∗ Canal::StackFrame::mFunction

Never is NULL. It is a pointer just to allow storing instances of this class in std::vector.

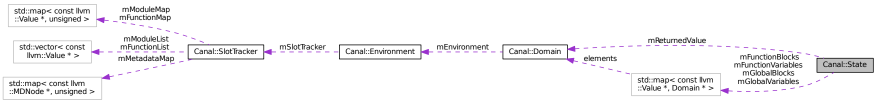
The documentation for this class was generated from the following files:

- lib/Stack.h
- lib/Stack.cpp

16.23 Canal::State Class Reference

`#include <State.h>`

Collaboration diagram for Canal::State:


std::map< const llvm ::Value *, unsigned >
mModuleMap mFunctionMap
std::vector< const llvm::Value * >
mModuleList mFunctionList
mMetadataMap
std::map< const llvm ::MDNode *, unsigned >
Canal::SlotTracker
mSlotTracker
Canal::Environment
mEnvironment
Canal::Domain
mReturnedValue
elements
std::map< const llvm ::Value *, Domain * >
mFunctionBlocks mFunctionVariables mGlobalBlocks mGlobalVariables
Canal::State

Public Member Functions

- **State** (const State &state)
- State & **operator=** (const State &state)
- bool **operator==** (const State &state) const
- bool **operator!=** (const State &state) const
- void clear ()

  Clears everything. Releases all memory.
- void clearFunctionLevel ()

  Clears function variables, blocks and returned value.
- void **merge** (const State &state)
- void mergeGlobalLevel (const State &state)
- void addGlobalVariable (const llvm::Value &place, Domain ∗value)
- void addFunctionVariable (const llvm::Value &place, Domain ∗value)
- void **addGlobalBlock** (const llvm::Value &place, Domain ∗value)
- void addFunctionBlock (const llvm::Value &place, Domain ∗value)

  Adds a value created by alloca to the stack.
- const PlaceValueMap & **getGlobalVariables** () const
- const PlaceValueMap & **getGlobalBlocks** () const
- const PlaceValueMap & **getFunctionVariables** () const
- const PlaceValueMap & **getFunctionBlocks** () const
- Domain ∗ findVariable (const llvm::Value &place) const
- Domain ∗ findBlock (const llvm::Value &place) const

Public Attributes

- Domain ∗ mReturnedValue

  Value returned from function.

Protected Attributes

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

16.23.1 Detailed Description

Includes global variables and heap. Includes function-level memory and variables (=stack).

16.23.2 Member Function Documentation

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

Adds a register-type value to the stack.

Parameters

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

See also

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

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

Parameters

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

16.23.2.3 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.

16.23.2.4 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.

16.23.2.5 void Canal::State::mergeGlobalLevel (const State & *state*)

Merge only global variables and global memory blocks of the provided state. 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 function level state is not relevant.

16.23.3 Member Data Documentation

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

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

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

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

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

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

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

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

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

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

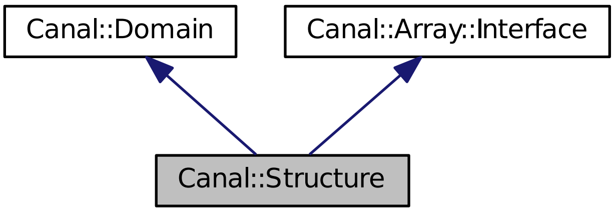
The key (llvm::Value∗) is not owned by this class. It is not deleted. The value (Domain∗) memory is owned by this class, so it is deleted in state destructor.

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

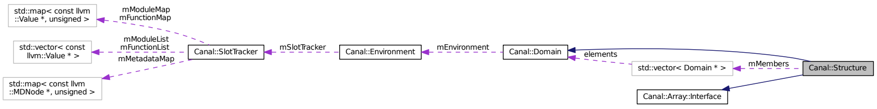
- lib/State.h
- lib/State.cpp

16.24 Canal::Structure Class Reference

Inheritance diagram for Canal::Structure:



Collaboration diagram for Canal::Structure:



Public Member Functions

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

  Implementation of Domain::operator==().
- virtual void merge (const Domain &value)

  Implementation of Domain::merge().
- virtual size_t memoryUsage () const

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

  Implementation of Domain::toString().
- virtual bool matchesString (const std::string &text, std::string &rationale) const

  Implementation of Domain::matchesString().
- virtual std::vector< Domain ∗ > getItem (const Domain &offset) const

  Implementation of Array::Interface::getItem().
- virtual Domain ∗ getItem (uint64_t offset) const

  Implementation of Array::Interface::getItem().
- virtual void setItem (const Domain &offset, const Domain &value)

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

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

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

Public Attributes

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

Additional Inherited Members

16.24.1 Member Function Documentation

16.24.1.1 Structure ∗ Canal::Structure::clone ( ) const [virtual]

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

Implements Canal::Domain.

16.24.1.2 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:

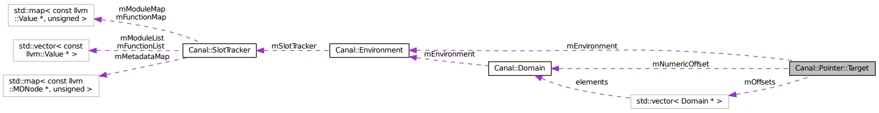
- lib/Structure.h
- lib/Structure.cpp

16.25 Canal::Pointer::Target Class Reference

TODO: Pointers to functions.

`#include <PointerTarget.h>`

Collaboration diagram for Canal::Pointer::Target:



Public Types

- enum **Type** {
  Constant, **FunctionBlock**, **FunctionVariable**, **GlobalBlock**,
  GlobalVariable }

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.
- 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 ∗ mInstruction
- std::vector< Domain ∗ > mOffsets
- Domain ∗ mNumericOffset

16.25.1 Detailed Description

TODO: Pointers to functions.

Pointer target – where the pointer points to. Pointer can:

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

16.25.2 Constructor & Destructor Documentation

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

Standard constructor.

Parameters

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

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

16.25.3 Member Function Documentation

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

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

16.25.4 Member Data Documentation

16.25.4.1 const llvm::Value∗ Canal::Pointer::Target::mInstruction

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

16.25.4.2 Domain∗ Canal::Pointer::Target::mNumericOffset

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

16.25.4.3 std::vector<Domain∗> Canal::Pointer::Target::mOffsets

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

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

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

16.26 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

16.27 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)

16.27.1 Detailed Description

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

16.27.2 Member Function Documentation

16.27.2.1 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:

17.2 Class List

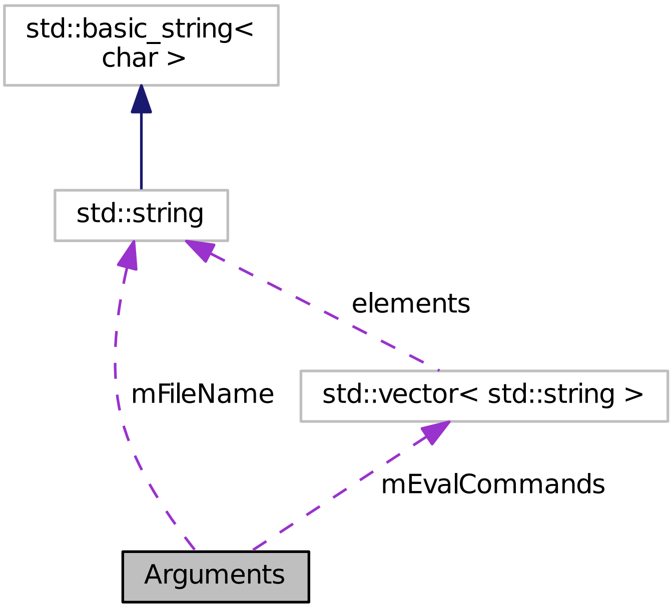
Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 18

Tool Class Documentation

18.1 Arguments Class Reference

Collaboration diagram for Arguments:



Public Attributes

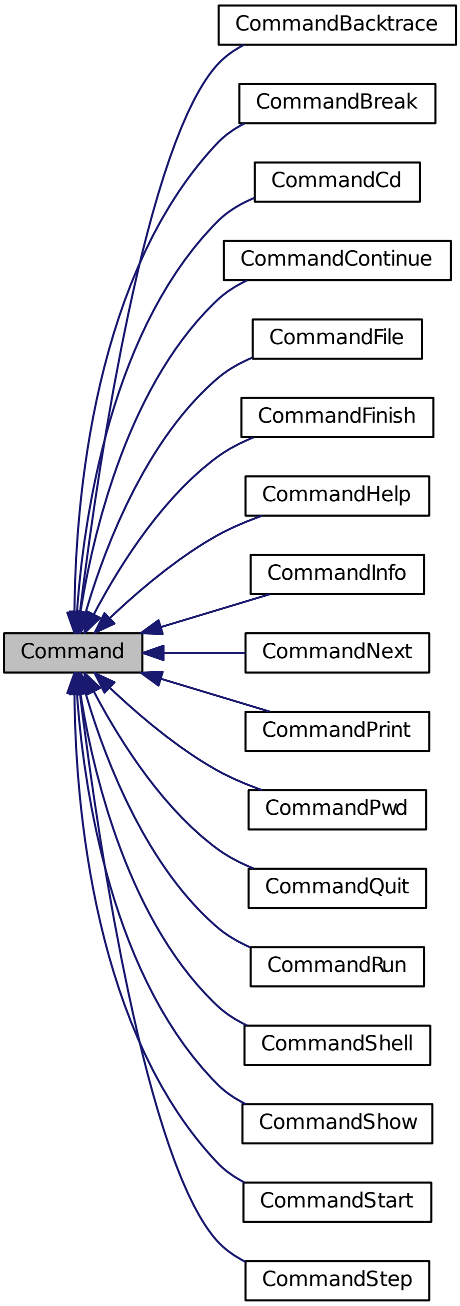
- std::vector< std::string > **mEvalCommands**
- std::string **mFileName**

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

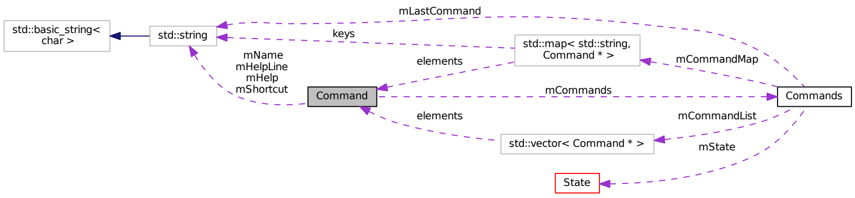
- tool/Canal.cpp

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)
- 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

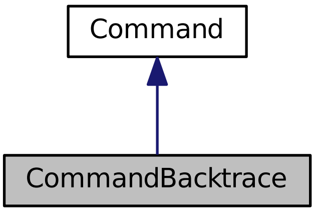
- std::string **mName**
- std::string **mShortcut**
- std::string **mHelpLine**
- std::string **mHelp**
- Commands & **mCommands**

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

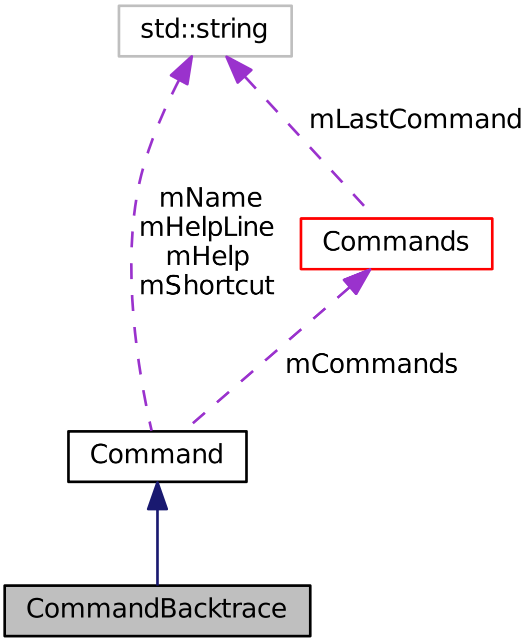
- tool/Command.h
- tool/Command.cpp

18.3 CommandBacktrace Class Reference

Inheritance diagram for CommandBacktrace:



Collaboration diagram for CommandBacktrace:



Public Member Functions

- **CommandBacktrace** (Commands &commands)

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

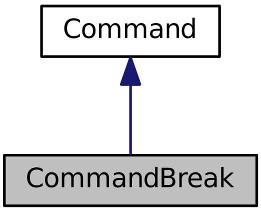
Additional Inherited Members

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

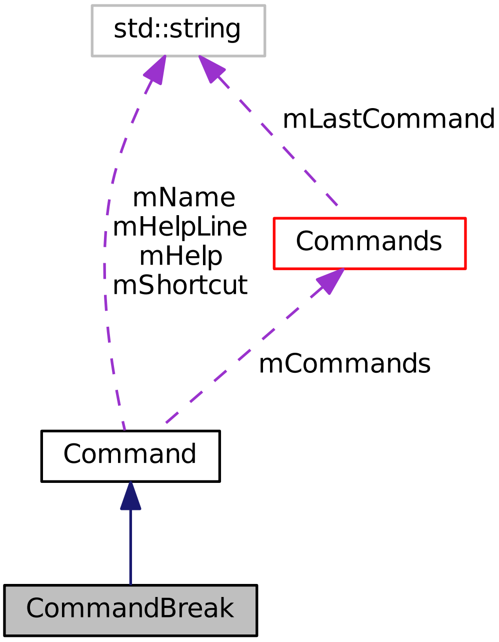
- tool/CommandBacktrace.h
- tool/CommandBacktrace.cpp

18.4 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)

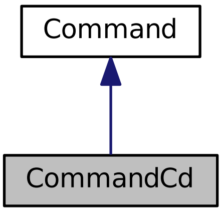
Additional Inherited Members

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

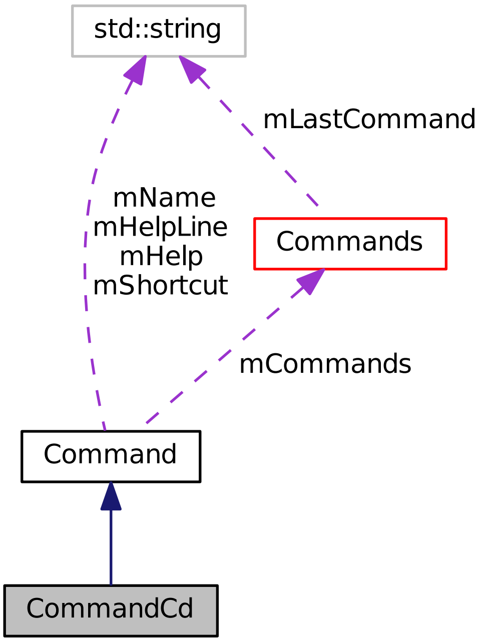
- tool/CommandBreak.h
- tool/CommandBreak.cpp

18.5 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

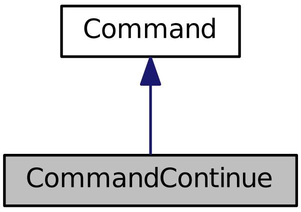
The documentation for this class was generated from the following files:

- tool/CommandCd.h

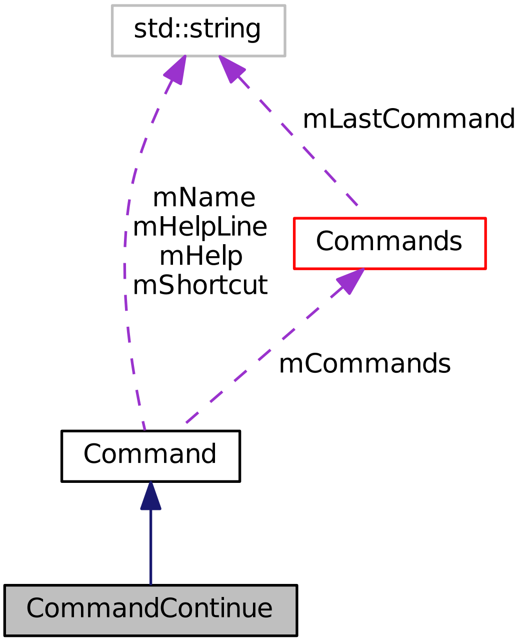
- tool/CommandCd.cpp

18.6 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)

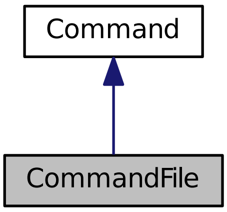
Additional Inherited Members

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

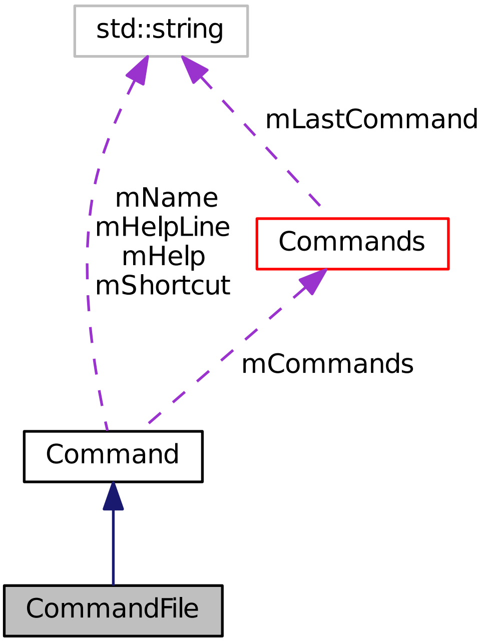
- tool/CommandContinue.h
- tool/CommandContinue.cpp

18.7 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)

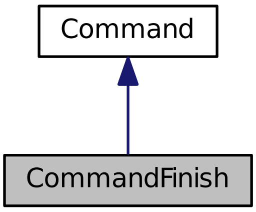
Additional Inherited Members

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

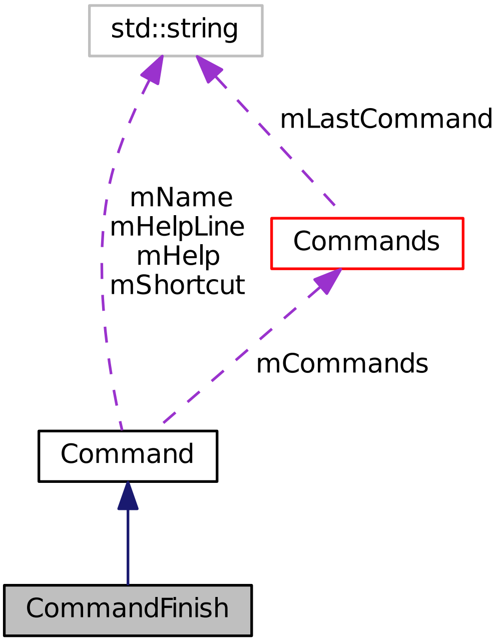
- tool/CommandFile.h
- tool/CommandFile.cpp

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)

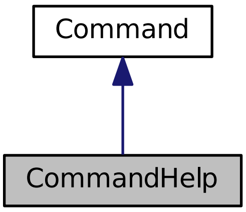
Additional Inherited Members

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

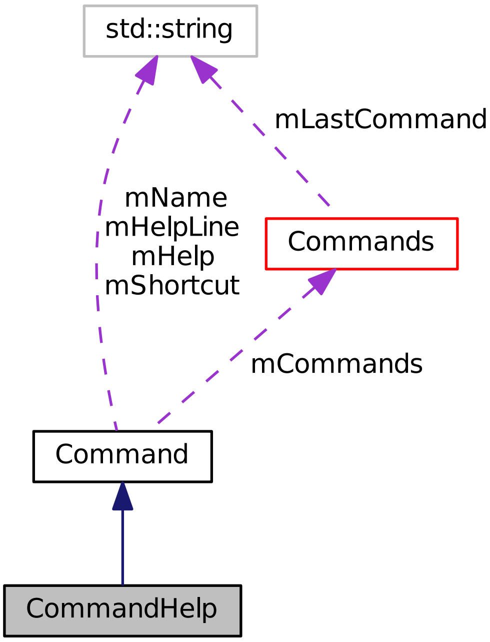
- tool/CommandFinish.h
- tool/CommandFinish.cpp

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** ()

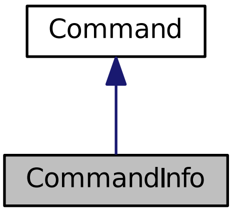
Additional Inherited Members

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

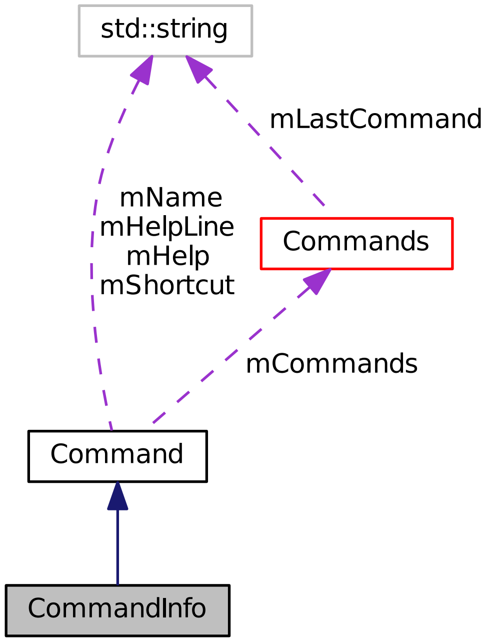
- tool/CommandHelp.h
- tool/CommandHelp.cpp

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** ()

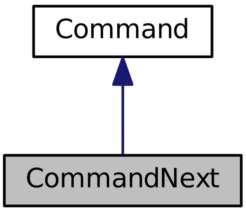
Additional Inherited Members

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

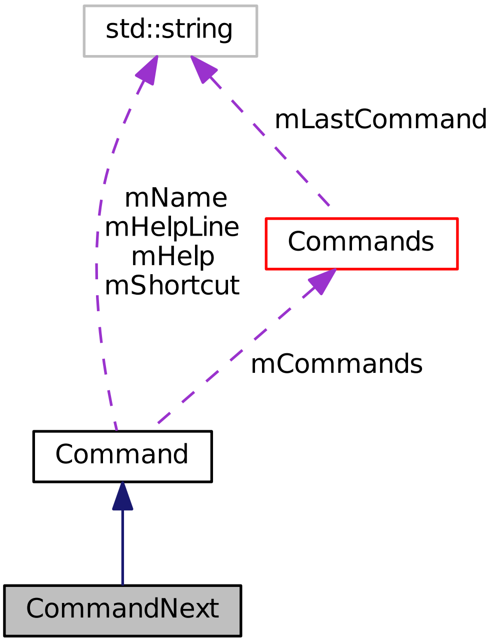
- tool/CommandInfo.h
- tool/CommandInfo.cpp

18.11 CommandNext Class Reference

Inheritance diagram for CommandNext:



Collaboration diagram for CommandNext:



Public Member Functions

- **CommandNext** (Commands &commands)

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

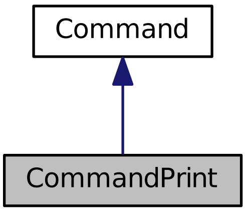
Additional Inherited Members

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

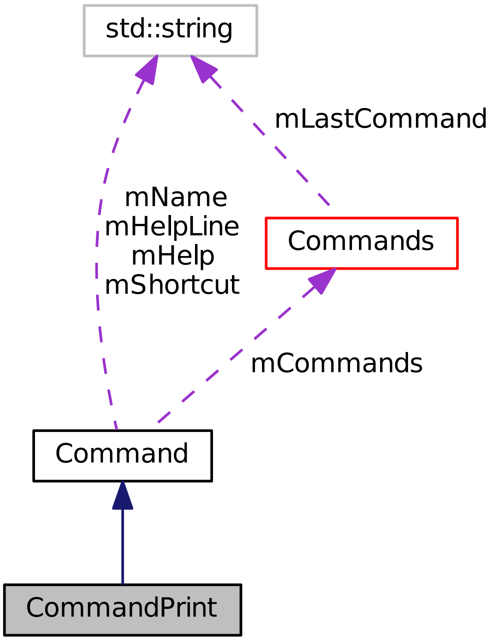
- tool/CommandNext.h
- tool/CommandNext.cpp

18.12 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)

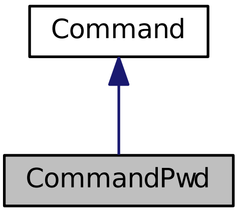
Additional Inherited Members

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

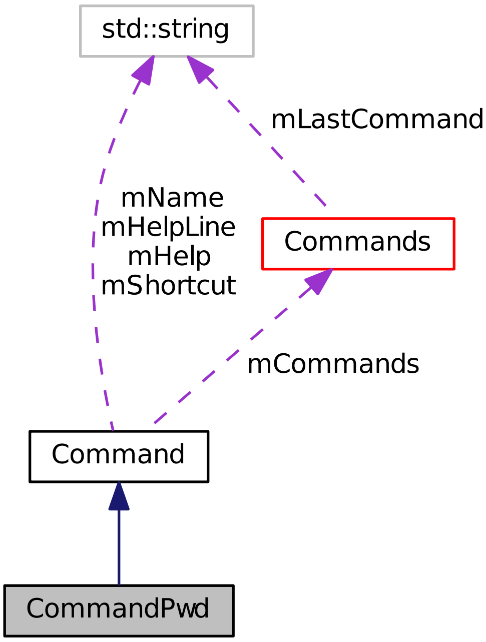
- tool/CommandPrint.h
- tool/CommandPrint.cpp

18.13 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)

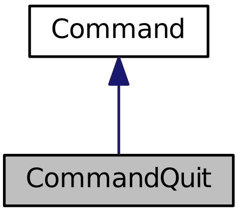
Additional Inherited Members

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

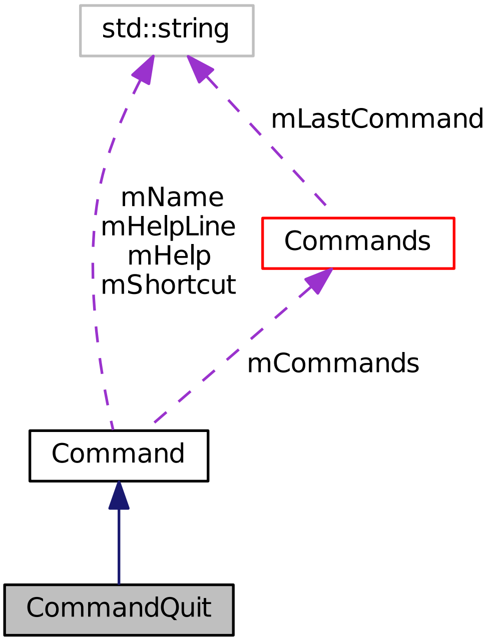
- tool/CommandPwd.h
- tool/CommandPwd.cpp

18.14 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)

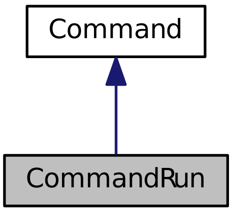
Additional Inherited Members

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

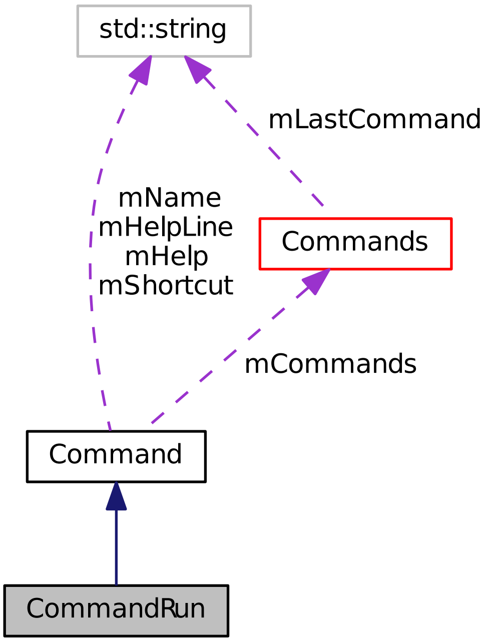
- tool/CommandQuit.h
- tool/CommandQuit.cpp

18.15 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)

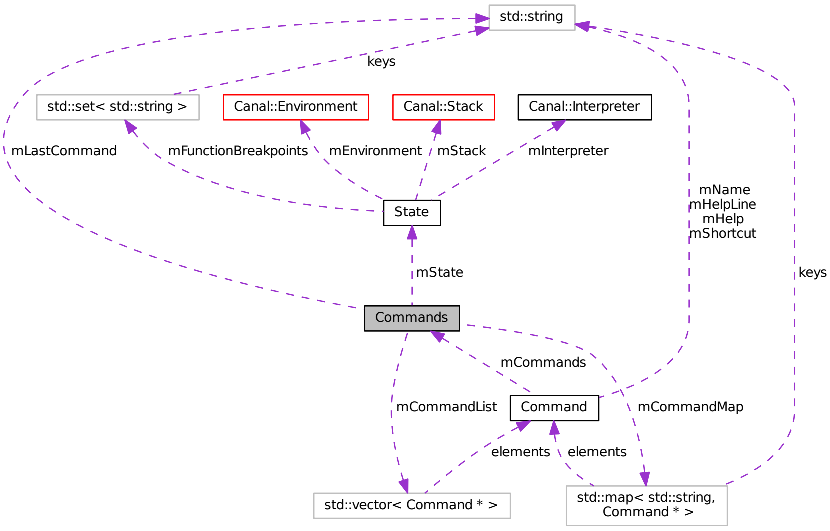
Additional Inherited Members

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

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

18.16 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

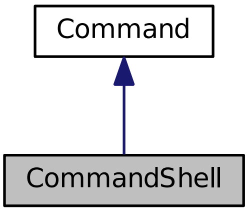
- std::string **mLastCommand**
- State ∗ **mState**

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

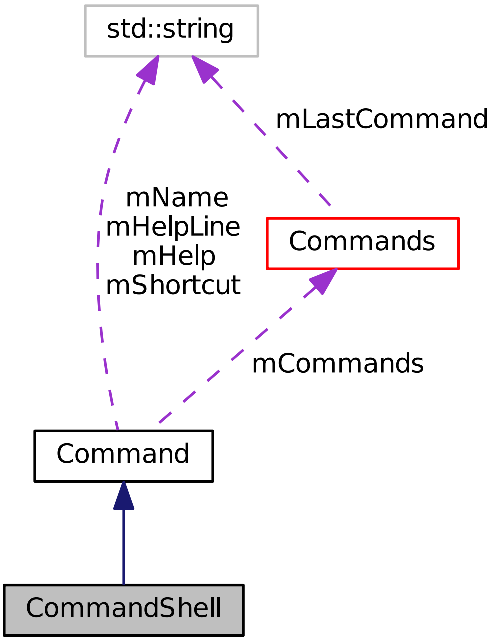
- tool/Commands.h
- tool/Commands.cpp

18.17 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)

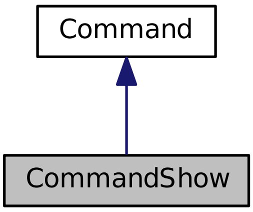
Additional Inherited Members

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

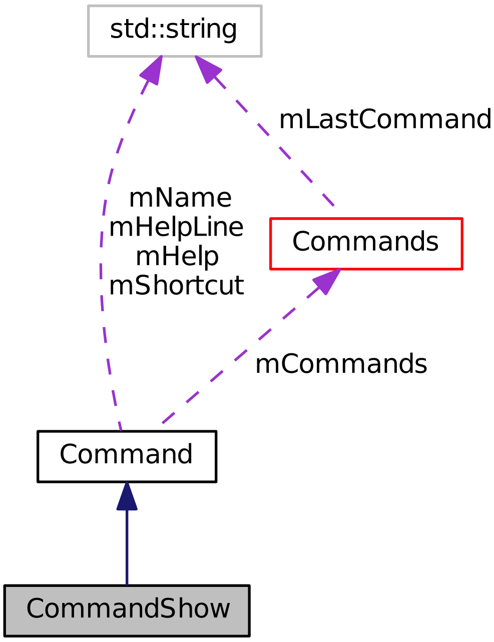
- tool/CommandShell.h
- tool/CommandShell.cpp

18.18 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)

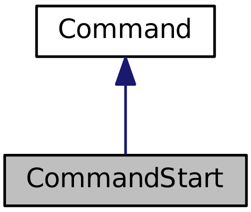
Additional Inherited Members

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

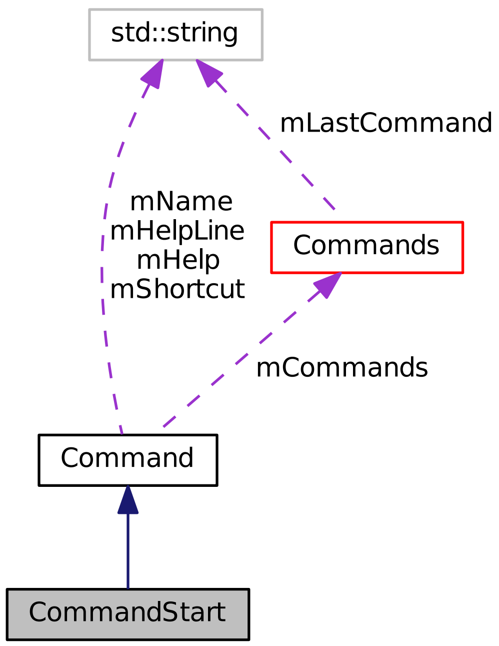
- tool/CommandShow.h
- tool/CommandShow.cpp

18.19 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)

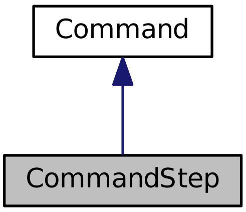
Additional Inherited Members

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

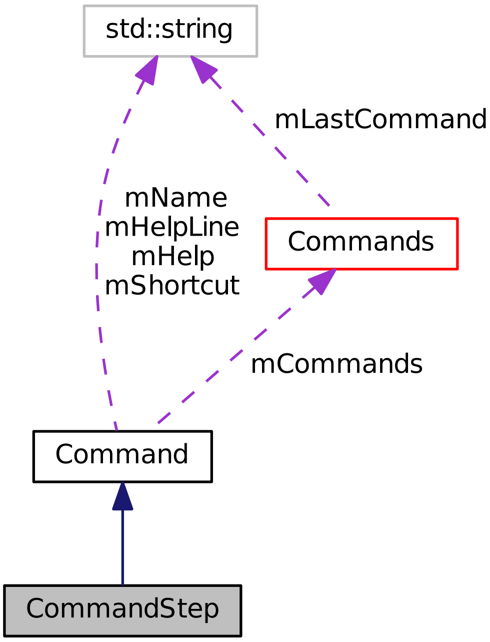
- tool/CommandStart.h
- tool/CommandStart.cpp

18.20 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)

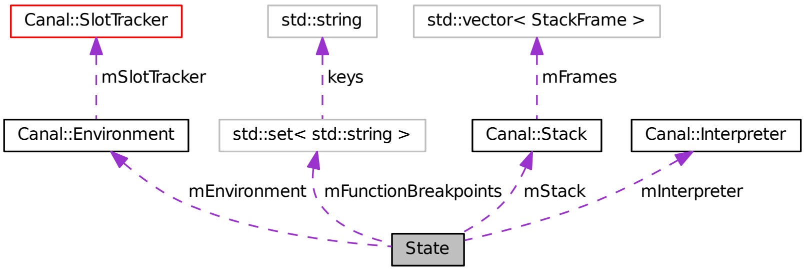
Additional Inherited Members

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

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

18.21 State Class Reference

Collaboration diagram for State:



Public Member Functions

- **State** (const llvm::Module ∗module)
- const llvm::Module & **getModule** () const
- const **Canal::Environment** & **getEnvironment** () const
- **Canal::Stack** & **getStack** ()
- const **Canal::Stack** & **getStack** () const
- **Canal::SlotTracker** & **getSlotTracker** ()
- bool **isInterpreting** () const
- void **run** ()
- void **step** (int count)
- void **next** (int count)
- void **finish** ()
- void **addFunctionBreakpoint** (const std::string &functionName)
- void **addMainFrame** ()

Protected Member Functions

- bool **reachedBreakpoint** ()

Protected Attributes

- const llvm::Module ∗ **mModule**
- **Canal::Environment mEnvironment**
- **Canal::Stack mStack**
- **Canal::Interpreter mInterpreter**
- std::set< std::string > **mFunctionBreakpoints**

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 Support of Multiple Platforms

Support Microsoft Windows and Mac OS X natively.

20.2 Automatic Tests

Unit tests and integration tests.

20.3 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.

[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