Abstract Interpretation and Properties of C Programs
EJCP 2018

Virgile Prevosto
virgile.prevosto@cea.fr

June 26th, 2018
Context

Overview of Static Analysis

Analyzing C code with Frama-C

EVA Plugin
Introduction

- Software is more and more pervasive in embedded systems...
- ...and keeps getting larger
- Tests and code review too costly beyond a certain size and coverage criterion
- Need for **correct** tools
  - ✓ Detect all potential issues
  - ✗ May issue spurious warnings
  - ✗ Impossible for an automated tool to warn for all real issues and only for them (Rice theorem)
Software is more and more pervasive in embedded systems...
...and keeps getting larger
Tests and code review too costly beyond a certain size and coverage criterion
Need for correct tools
- Detect all potential issues
- May issue spurious warnings
- Impossible for an automated tool to warn for all real issues and only for them (Rice theorem)
Context

Overview of Static Analysis

Analyzing C code with Frama-C

EVA Plugin
Abstract interpretation is about

- abstracting away information
- ensuring answer in a reasonable time
- while retaining adequate precision
- and guaranteeing correct answers
Abstract Interpretation in two pictures

Abstract interpretation is about

- abstracting away information
- ensuring answer in a reasonable time
- while retaining adequate precision
- and guaranteeing correct answers
Abstract interpretation is about

- abstracting away information
- ensuring answer in a reasonable time
- while retaining adequate precision
- and guaranteeing correct answers
Abstract interpretation is about

- abstracting away information
- ensuring answer in a reasonable time
- while retaining adequate precision
- and guaranteeing correct answers
Abstract Interpretation in two pictures

Abstract interpretation is about:

▶ abstracting away information
▶ ensuring answer in a reasonable time
▶ while retaining adequate precision
▶ and guaranteeing correct answers
Overview of Static Analysis

Context

Overview of Static Analysis
Static Analysis Framework
Abstract Interpretation

Analyzing C code with Frama-C

EVA Plugin
mpz_ptr syracuse(mpz_t res, const mpz_t arg) {
    mpz_t x;
    mpz_init_set_ui(res, 0UL);
    mpz_init_set(x, arg);
    while (mpz_cmp_ui(x, 1UL) > 0) {
        mpz_out_str(stdout, 10, x);
        putchar('
');
        if (mpz_odd_p(x)) {
            mpz_mul_ui(x, x, 3UL);
            mpz_add_ui(x, x, 1UL);
        } else {
            mpz_cdiv_q_ui(x, x, 2UL);
        }
        mpz_add_ui(res, res, 1UL);
    }
    mpz_clear(x);
    return res;
}
Overview of Static Analysis
Static Analysis Framework

Trace Semantics

- Initial state on start node
- Transfer functions across edges
  - infinite number of traces
  - some traces might be infinite
Overview of Static Analysis
Static Analysis Framework

Trace Semantics

- Initial state on start node
- Transfer functions across edges
- Infinite number of traces
- Some traces might be infinite
Overview of Static Analysis
Static Analysis Framework

Trace Semantics

- Initial state on start node
- Transfer functions across edges
- infinite number of traces
- some traces might be infinite

\[
\begin{align*}
\text{odd}(x) & : x = 3 \times x + 1 \\
\text{even}(x) & : x = x / 2 \\
\end{align*}
\]
Overview of Static Analysis
Static Analysis Framework

Trace Semantics

- Initial state on start node
- Transfer functions across edges
- Infinite number of traces
- Some traces might be infinite

\[ s_0 \]
\begin{align*}
\text{arg} & \mapsto 5 \\
x & \mapsto 5
\end{align*}

\[ s_1 \]
\begin{align*}
x & \mapsto 5 \\
\text{res} & \mapsto 0
\end{align*}

\[ s_2 \]
\begin{align*}
x & \mapsto 5 \\
\text{res} & \mapsto 0
\end{align*}
Overview of Static Analysis
Static Analysis Framework

Trace Semantics

- Initial state on start node
- Transfer functions across edges
- Infinite number of traces
- Some traces might be infinite

\[
\begin{align*}
S_0: & \quad \text{arg} \mapsto 5 \\
S_1: & \quad x \mapsto 5, \quad \text{res} \mapsto 0 \\
S_2: & \quad \text{odd}(x)? \\
S_3: & \quad x = 3 \times x + 1 \\
S_4: & \quad x \mapsto 5 \\
S_5: & \quad \text{even}(x)? \\
S_6: & \quad x = x/2 \\
S_7: & \quad \text{res} = \text{res} + 1 \\
S_8: & \quad \text{res} \mapsto \text{res} + 1 \\
S_9: & \quad x \mapsto 5, \quad \text{res} \mapsto 0 \\
S_{10}: & \quad x \mapsto 5, \quad \text{res} \mapsto 0
\end{align*}
\]
Overview of Static Analysis
Static Analysis Framework

Trace Semantics

- Initial state on start node
- Transfer functions across edges
- Infinite number of traces
- Some traces might be infinite
Overview of Static Analysis
Static Analysis Framework

Trace Semantics

- Initial state on start node
- Transfer functions across edges
- Infinite number of traces
- Some traces might be infinite
Overview of Static Analysis

Static Analysis Framework

Trace Semantics

- Initial state on start node
- Transfer functions across edges
- infinite number of traces
- some traces might be infinite
Overview of Static Analysis
Static Analysis Framework

Trace Semantics

- Initial state on start node
- Transfer functions across edges
- Infinite number of traces
- Some traces might be infinite
Overview of Static Analysis
Static Analysis Framework

Trace Semantics

Initial state on start node
- Transfer functions across edges
- infinite number of traces
- some traces might be infinite
Overview of Static Analysis

Static Analysis Framework

Trace Semantics

- Initial state on start node
- Transfer functions across edges
- Infinite number of traces
- Some traces might be infinite
Overview of Static Analysis
Static Analysis Framework

Trace Semantics

- Initial state on start node
- Transfer functions across edges
- Infinite number of traces
- Some traces might be infinite
Overview of Static Analysis
Static Analysis Framework

Trace Semantics

- Initial state on start node
- Transfer functions across edges
  - infinite number of traces
  - some traces might be infinite
Collecting Semantics

From the set of all traces to set of all states
• multiple predecessors: take union
• lose “temporal” relations
• fixpoint computation
• may not terminate
Collecting Semantics

From the set of all traces to set of all states

- multiple predecessors: take union
- lose “temporal” relations
- fixpoint computation
- may not terminate

\[ S_0 \leadsto \text{arg} \in \mathbb{Z} \]

\[
\begin{align*}
\text{res} &= 0, \quad x = \text{arg} \\
\text{x} &> 1? \\
\text{x} &\leq 1? \\
\text{odd}(x)? &\quad \text{even}(x)? \\
\text{x} &= 3 \times x + 1, & \text{x} &= x/2 \\
\text{res} &= \text{res} + 1
\end{align*}
\]
Overview of Static Analysis

Static Analysis Framework

Collecting Semantics

\[ \text{res} = 0, \text{x} = \text{arg} \]

\[ \text{x} > 1? \]

\[ \text{odd(x) ?} \quad \text{even(x) ?} \]

\[ x = 3 \times x + 1 \quad x = x / 2 \]

\[ \text{res} = \text{res} + 1 \]

\[ S_3 \mapsto \{(x, \text{res}) | x = 2k + 1, \text{res} = 0\} \]

\[ S_4 \mapsto \{(x, \text{res}) | x = 6k + 2, \text{res} = 0\} \]

- From the set of all traces to set of all states
- multiple predecessors: take union
- lose “temporal” relations
- fixpoint computation
- may not terminate
Collecting Semantics

From the set of all traces to set of all states

- multiple predecessors: take union
- lose “temporal” relations
- fixpoint computation
- may not terminate

\[
S_4 \mapsto \{(x, \text{res}) | x = 6k + 2, \text{res} = 0\}
\]

\[
S_6 \mapsto \{(x, \text{res}) | x = k, \text{res} = 0\}
\]

\[
S_7 \mapsto \{(x, \text{res}) | x = k, \text{res} = 0\}
\]
Collecting Semantics

- From the set of all traces to set of all states
- multiple predecessors: take union
- lose "temporal" relations
- fixpoint computation
- may not terminate
Collecting Semantics

From the set of all traces to set of all states

- multiple predecessors: take union
- lose “temporal” relations
- fixpoint computation
- may not terminate
Collecting Semantics

From the set of all traces to set of all states

- multiple predecessors: take union
- lose “temporal” relations
- fixpoint computation
- may not terminate

\[ S_1 \mapsto \{(x, res)|x = k, res = 0\} \]
\[ S_8 \mapsto \{(x, res)|x = k, res = 1\} \]
\[ S_1 \mapsto \{(x, res)|x = k, res \in \{0, 1\}\} \]
Replace set of states ...

... by one element in an abstract lattice

Over-approximation and false alarms

Trade-off between precision and computation time
Overview of Static Analysis

Static Analysis Framework

- Replace set of states ...
- ... by one element in an abstract lattice
- Over-approximation and false alarms
- Trade-off between precision and computation time
Static Analysis framework

- Replace set of states ...
- ... by one element in an abstract lattice
- Over-approximation and false alarms
- Trade-off between precision and computation time
Replace set of states ...
... by one element in an abstract lattice
Over-approximation and false alarms
Trade-off between precision and computation time
Abstract transfer functions $F^b$

- Merge abstract states for nodes with multiple predecessors
- **Correction**: Do we include all concrete states in the end?
- **Termination**: Converge in a finite number of steps
- **Abstract interpretation**: A systematic way to build correct and terminating analyses
Correctness and Termination

- Abstract transfer functions $F^\#$
- Merge abstract states for nodes with multiple predecessors
- **Correction**: Do we include all concrete states in the end?
- **Termination**: Converge in a finite number of steps
- **Abstract interpretation**: A systematic way to build correct and terminating analyses
Correctness and Termination

- Abstract transfer functions $F^\#$
- Merge abstract states for nodes with multiple predecessors
- **Correction:** Do we include all concrete states in the end?
- **Termination:** Converge in a finite number of steps
- **Abstract interpretation:** A systematic way to build correct and terminating analyses
Overview of Static Analysis
Static Analysis Framework

Correctness and Termination

Abstract transfer functions $F^\#$
Merge abstract states for nodes with multiple predecessors
**Correction**: Do we include all concrete states in the end?
**Termination**: Converge in a finite number of steps
**Abstract interpretation**: A systematic way to build correct and terminating analyses
Abstract transfer functions $F^{\#}$

Merge abstract states for nodes with multiple predecessors

**Correction:** Do we include all concrete states in the end?

**Termination:** Converge in a finite number of steps

**Abstract interpretation:** A systematic way to build correct and terminating analyses
Abstract transfer functions $F^\#$
Merge abstract states for nodes with multiple predecessors
Correction: Do we include all concrete states in the end?
Termination: Converge in a finite number of steps
Abstract interpretation: A systematic way to build correct and terminating analyses
Correctness and Termination

- Abstract transfer functions $F^\#$
- Merge abstract states for nodes with multiple predecessors
- **Correction**: Do we include all concrete states in the end?
- **Termination**: Converge in a finite number of steps
- **Abstract interpretation**: A systematic way to build correct and terminating analyses
Abstract transfer functions $F^\#$

- Merge abstract states for nodes with multiple predecessors
- **Correction:** Do we include all concrete states in the end?
- **Termination:** Converge in a finite number of steps
- **Abstract interpretation:** A systematic way to build correct and terminating analyses
Abstract transfer functions $F^\#$

Merge abstract states for nodes with multiple predecessors

**Correction:** Do we include all concrete states in the end?

**Termination:** Converge in a finite number of steps

**Abstract interpretation:** A systematic way to build correct and terminating analyses
Correctness and Termination

- Abstract transfer functions $F^\#$
- Merge abstract states for nodes with multiple predecessors
- **Correction**: Do we include all concrete states in the end?
- **Termination**: Converge in a finite number of steps
- **Abstract interpretation**: A systematic way to build correct and terminating analyses
Correctness and Termination

- Abstract transfer functions $F^\#$
- Merge abstract states for nodes with multiple predecessors
- **Correction**: Do we include all concrete states in the end?
- **Termination**: Converge in a finite number of steps
- **Abstract interpretation**: A systematic way to build correct and terminating analyses
Correctness and Termination

- Abstract transfer functions $F^\#$
- Merge abstract states for nodes with multiple predecessors
- **Correction**: Do we include all concrete states in the end?
- **Termination**: Converge in a finite number of steps
- **Abstract interpretation**: A systematic way to build correct and terminating analyses
Correctness and Termination

- Abstract transfer functions $F^\#$
- Merge abstract states for nodes with multiple predecessors
- **Correction:** Do we include all concrete states in the end?
- **Termination:** Converge in a finite number of steps
- **Abstract interpretation:** A systematic way to build correct and terminating analyses
Correctness and Termination

- Abstract transfer functions $F^\#$
- Merge abstract states for nodes with multiple predecessors
- **Correction**: Do we include all concrete states in the end?
- **Termination**: Converge in a finite number of steps
- **Abstract interpretation**: A systematic way to build correct and terminating analyses
Abstract transfer functions $F^\|$  
Merge abstract states for nodes with multiple predecessors  
**Correction:** Do we include all concrete states in the end?  
**Termination:** Converge in a finite number of steps  
**Abstract interpretation:** A systematic way to build correct and terminating analyses
Abstract transfer functions $F^\#$

- Merge abstract states for nodes with multiple predecessors
- **Correction:** Do we include all concrete states in the end?
- **Termination:** Converge in a finite number of steps
- **Abstract interpretation:** A systematic way to build correct and terminating analyses
Galois connection and insertion

- $\alpha$ returns an abstraction from a set of concrete states
- $\gamma$ returns the set of concrete states corresponding to an abstraction

Following properties must hold:

1. $\alpha$ and $\gamma$ are monotonic
2. $\forall v^b \in L^b, \; v^b \sqsubseteq^b (\gamma \circ \alpha)(v^b)$
3. $\forall v^\# \in L^\#, \; (\alpha \circ \gamma)(v^\#) \sqsubseteq v^\#$

- Theorem [Cousot]: If $F^\# \sqsupseteq \alpha \circ F^b \circ \gamma$, abstraction is correct.
Overview of Static Analysis
Abstract Interpretation

- \( \alpha \) returns an abstraction from a set of concrete states
- \( \gamma \) returns the set of concrete states corresponding to an abstraction
- Following properties must hold:
  1. \( \alpha \) and \( \gamma \) are monotonic
  2. \( \forall v^b \in L^b, v^b \sqsubseteq^b (\gamma \circ \alpha)(v^b) \)
  3. \( \forall v^\# \in L^\#, (\alpha \circ \gamma)(v^\#) = v^\# \)
- **Theorem [Cousot]:** If \( F^\# \sqsubseteq \alpha \circ F^b \circ \gamma \), abstraction is correct.
Overview of Static Analysis
Abstract Interpretation

Relational and Non-relational Lattices

Non-relational domain

- Considers each variable independently
- ✔ Simpler and less costly
- ✗ lose properties over 2+ variables

Example: intervals

Relational domain

- Considers several variables at once
- ✔ More precise
- ✗ More complex and costly

Example: Polyhedra
Overview of Static Analysis
Abstract Interpretation

Widening

- for loop nodes, state grows slowly at each step
- convergence could require infinite time
- replace $\sqcup$ with widening operator $\triangledown$:
  - correctness $x \sqcup y \sqsubseteq x \triangledown y$
  - termination no infinitely growing sequence $x_0 \triangledown x_1 \triangledown \ldots \triangledown x_n \ldots$

$$
\begin{align*}
S_0 & \quad x = \text{arg} \quad x = 0 \\
S_1 & \quad x > 0? \\
S_2 & \quad y = y + 1 \\
S_3 & \quad x = x - 1 \\
S_4 & \quad x \leq 0? \\
S_5 & \quad S_1 \text{ (after)} \\
y \in [0; 0] \sqcup [1; 1] & = [0; 1] \\
\end{align*}
$$
Overview of Static Analysis
Abstract Interpretation

Widening

- for loop nodes, state grows slowly at each step
- convergence could require infinite time
- replace $\sqcup$ with widening operator $\triangledown$:
  - correctness $x \sqcup y \sqsubseteq x \triangledown y$
  - termination no infinitely growing sequence $x_0 \triangledown x_1 \triangledown \ldots \triangledown x_n \ldots$

$S_0$

$x = \text{arg} \quad x = 0$

$x > 0? \quad x \leq 0?$

$S_1$

$S_2$

$S_3$

$S_4$

$S_5$

$y = y + 1$

$x = x - 1$

$S_1 \text{ (before)} \quad S_4 \quad S_1 \text{ (after)}$

$y \in [0; 1] \sqcup [1; 2] = [0; 2]$
Overview of Static Analysis
Abstract Interpretation

Widening

- for loop nodes, state grows slowly at each step
- convergence could require infinite time
- replace $\sqcup$ with **widening operator** $\nabla$:

  - **correctness** $x \sqcup y \sqsubseteq x \nabla y$
  - **termination** no infinitely growing sequence
  - $x_0 \nabla x_1 \nabla \ldots \nabla x_n \ldots$

\[ x = \text{arg} \ x = 0 \]
\[ x > 0? \quad x \leq 0? \]
\[ y = y + 1 \]
\[ x = x - 1 \]

\[ S_0 \]
\[ S_1 \]
\[ S_2 \]
\[ S_3 \]
\[ S_4 \]
\[ S_5 \]

\[ S_1 \text{ (before)} \]
\[ S_4 \]
\[ S_1 \text{ (after)} \]

\[ y \in [0; 2] \sqcup [1; 3] = [0; 3] \]
Overview of Static Analysis
Abstract Interpretation

Widening

- for loop nodes, state grows slowly at each step
- convergence could require infinite time
- replace \( \sqcup \) with widening operator \( \nabla \):

  correctness
  \[ x \sqcup y \subseteq x \nabla y \]

  termination
  no infinitely growing sequence
  \[ x_0 \nabla x_1 \nabla \ldots \nabla x_n \ldots \]

\begin{align*}
S_1 \text{ (before)} & : y \in [0; 2] \\
S_4 & : x = x - 1 \\
S_1 \text{ (after)} & : y \in [1; 3]
\end{align*}
Overview of Static Analysis
Abstract Interpretation

Widening

- for loop nodes, state grows slowly at each step
- convergence could require infinite time
- replace $\sqcup$ with widening operator $\nabla$:

  correctness
  $x \sqcup y \subseteq x \nabla y$

  termination
  no infinitely growing sequence
  $x_0 \nabla x_1 \nabla \ldots \nabla x_n \ldots$

lower bound stable: don’t change

$S_1$ (before) $\nabla$ $S_4$ $\nabla$ $S_1$ (after)

$y \in [0, 2]$ $\nabla$ $[1, 3] = [0, +\infty]$
Overview of Static Analysis
Abstract Interpretation

Widening

- for loop nodes, state grows slowly at each step
- convergence could require infinite time
- replace $\sqcup$ with widening operator $\triangledown$:
  - correctness
  - termination: no infinitely growing sequence

$x = \arg x = 0$

$x > 0?\ ▽\ x \leq 0?$

$y = y + 1$

$x = x - 1$

$y \in [0; 2] \triangledown [1; 3] = [0; +\infty]$

$S_1$ (before) $\triangledown$ $S_4$ $\triangledown$ $S_1$ (after)

upper bound grows: widen interval
Overview of Static Analysis
Abstract Interpretation

Widening

- for loop nodes, state grows slowly at each step
- convergence could require infinite time
- replace $\sqcup$ with widening operator $\nabla$:

  correctness
  $x \sqcup y \subseteq x \nabla y$

  termination
  no infinitely growing sequence
  $x_0 \nabla x_1 \nabla \ldots \nabla x_n \ldots$

$S_1$ (before) $S_4$ $S_1$ (after)
$y \in [0; +\infty] \nabla [1; +\infty] = [0; +\infty]$
Overview of Static Analysis
Abstract Interpretation

Narrowing

Recall some precision
- Widening can be very coarse
- Use narrowing after reaching fixpoint:
  - correctness $y \sqsubseteq (x \triangle y) \sqsubseteq x$
  - termination no infinitely decreasing sequence
- In practice, very often better to directly improve widening

widen and propagate new bound
Narrowing

Recover some precision

- Widening can be very coarse
- Use narrowing after reaching fixpoint:
  - Correctness: \( y \subseteq (x \triangle y) \subseteq x \)
  - Termination: no infinitely decreasing sequence
- In practice, very often better to directly improve widening

\(\infty\) may be too much try to narrow it down
Overview of Static Analysis
Abstract Interpretation

Narrowing

Recover some precision

- Widening can be very coarse
- Use narrowing after reaching fixpoint:
  - Correctness: \( y \sqsubseteq (x \bigtriangleup y) \sqsubseteq x \)
  - Termination: no infinitely decreasing sequence

- In practice, very often better to directly improve widening

Candidate bound to be propagated

\[
\begin{align*}
S_2 \text{ (before)} & \quad S_4 \quad S_2 \text{ (after)} \\
y \in [0; +\infty] \bigtriangleup [1; 301] & = [0; 301]
\end{align*}
\]
Overview of Static Analysis
Abstract Interpretation

Narrowing

Recover some precision

- Widening can be very coarse
- Use narrowing after reaching fixpoint:
  correctness $y \subseteq (x \triangle y) \subseteq x$
  termination no infinitely decreasing sequence
- In practice, very often better to directly improve widening

\[ y \in [0; 301] \triangle [1; 301] = [0; 301] \]
Overview of Static Analysis
Abstract Interpretation

Narrowing

Recover some precision

- Widening can be very coarse
- Use narrowing after reaching fixpoint:
  - correctness: \( y \subseteq (x \triangle y) \subseteq x \)
  - termination: no infinitely decreasing sequence
- In practice, very often better to directly improve widening

\[
\begin{align*}
S_0 & \quad \text{arg} \leq 300? & S_5 & \quad \text{arg} > 300? \\
S_1 & \quad x = \text{arg} & S_2 & \quad y = 0 \\
S_3 & \quad y \leq x? & S_4 & \quad y > x? \\
S_2 & \quad y = y + 1 & S_6 & \quad \\
S_2 \text{ (before)} & \quad y \in [0,0] & S_4 & \quad [1,1] = [0,301] \\
S_2 \text{ (after)} & \quad \text{widen and propagate new bound}
\end{align*}
\]
Overview of Static Analysis
Abstract Interpretation

Narrowing

Recover some precision
- Widening can be very coarse
- Use narrowing after reaching fixpoint:

\[ y \subseteq (x \triangle y) \subseteq x \]

termination no infinitely decreasing sequence

- In practice, very often better to directly improve widening
Question
We have information from two domains:
Intervals:
- \( x \in [0; 20] \)
- \( y \in [5; 10] \)
Octagons:
\( 0 \leq x - y \leq 20 \)
What can be said about \( x \) and \( y \)?

Answers

- \( x \in [0; 20], y \in [5; 10]; 0 \leq x - y \leq 20 \)
- \( x \in [5; 20], y \in [5; 10], 0 \leq x - y \leq 15 \)
- \( x \in [5; 20], y \in [5; 10], 0 \leq x - y \leq 10 \)
- \( x \in [5; 20], y \in [0; 20], 0 \leq x - y \leq 20 \)
Reduced product

- Combining abstract domains
- **reduce** abstract value from one domain using information from the other
- ✗ In practice, not as simple and generic as it looks
- ✗ Combining transfer function is complex
Overview of Static Analysis

Abstract Interpretation

- Combining abstract domains
- reduce abstract value from one domain using information from the other

- In practice, not as simple and generic as it looks
- Combining transfer function is complex
Reduced product

- Combining abstract domains
- **reduce** abstract value from one domain using information from the other
- ✗ In practice, not as simple and generic as it looks
- ✗ Combining transfer function is complex
Reduced product

- Combining abstract domains
- **reduce** abstract value from one domain using information from the other
- ✗ In practice, not as simple and generic as it looks
- ✗ Combining transfer function is complex
Overview of Static Analysis
Abstract Interpretation

- Combining abstract domains
- reduce abstract value from one domain using information from the other
- In practice, not as simple and generic as it looks
- Combining transfer function is complex
Overview of Static Analysis
Abstract Interpretation

Reduced product

- Combining abstract domains
- reduce abstract value from one domain using information from the other
- In practice, not as simple and generic as it looks
- Combining transfer function is complex
Overview of Static Analysis

Abstract Interpretation

- Combining abstract domains
- \textbf{reduce} abstract value from one domain using information from the other
- In practice, not as simple and generic as it looks
- Combining transfer function is complex
Overview of Static Analysis

Abstract Interpretation

- Combining abstract domains
- reduce abstract value from one domain using information from the other
- In practice, not as simple and generic as it looks
- Combining transfer function is complex
Overview of Static Analysis
Abstract Interpretation

▶ Combining abstract domains
▶ reduce abstract value from one domain using information from the other
× In practice, not as simple and generic as it looks
× Combining transfer function is complex
Combining abstract domains
- reduce abstract value from one domain using information from the other
- In practice, not as simple and generic as it looks
- Combining transfer function is complex
Reduced product

- Combining abstract domains
- reduce abstract value from one domain using information from the other
- In practice, not as simple and generic as it looks
- Combining transfer function is complex
Overview of Static Analysis
Abstract Interpretation

Reduced product

- Combining abstract domains
- reduce abstract value from one domain using information from the other
- In practice, not as simple and generic as it looks
- Combining transfer function is complex
Overview of Static Analysis
Abstract Interpretation

- Reducing abstract value from one domain using information from the other

- Combining abstract domains

- In practice, not as simple and generic as it looks

- Combining transfer function is complex
Combining abstract domains

- reduce abstract value from one domain using information from the other

- In practice, not as simple and generic as it looks

- Combining transfer function is complex
Overview of Static Analysis
Abstract Interpretation

Consider several abstract traces separately...

...At least for some time

More precise than collecting semantics

Finding appropriate partition is difficult
Consider several abstract traces separately...

...At least for some time

More precise than collecting semantics

Finding appropriate partition is difficult
Consider several abstract traces separately...

...At least for some time

✔ More precise than collecting semantics

✗ Finding appropriate partition is difficult
Context

Overview of Static Analysis

Analyzing C code with Frama-C
The Frama-C platform
ACSL
Frama-C for Software Assessment

EVA Plugin
Analyzing C code with Frama-C

Abstract interpretation in practice

A few tools

► **Polyspace Verifier**: check absence of runtime errors (C/C++/Ada)

► **ASTRÉE**: absence of runtime errors *without false alarm* in SCADE-generated code
  [https://www.absint.com/astree/index.htm](https://www.absint.com/astree/index.htm)

► **Verasco**: certified (in Coq) analyzer
  [http://compcert.inria.fr/verasco/](http://compcert.inria.fr/verasco/)

► **aiT/StackAnalyzer**: WCET and stack size (assembly code)
  [https://www.absint.com/ait/](https://www.absint.com/ait/)

► **FLUCTUAT**: accuracy of floating-point computations and origin of rounding errors

► **Frama-C**: platform for analyzing C code, including through abstract interpretation
  [https://frama-c.com](https://frama-c.com)
Frama-C at a glance

- A Framework for modular analysis of C code.
- Developed at CEA Tech List and Inria
- Released under LGPL license (v17.0 Chlorine in June 2018)
- Kernel based on CIL (Necula et al. – Berkeley).
- ACSL annotation language.
- Extensible platform
  - Collaboration of analyses over same code
  - Inter plug-in communication through ACSL formulas.
  - Adding specialized plug-in is easy
Analyzing C code with Frama-C
The Frama-C platform

Some plugins

- EVA
- Jessie
- WP
- Araï
- CaFE
- Mthread

- Executable-ACSL
- PathCrawler
- SANTE

- Dynamic Analysis
- Concurrency

- Code Transformation
- Abstract Interpretation

- Deductive Verification

- Temporal Properties

- Formal Methods

- Semantic constant folding
- Spare code
- Slicing

- Browsing of unfamiliar code
- Metrics computation
- Impact Analysis
- Variable occurrences
- Scope & Data-flow browsing

- From Analysis

- Included in main distribution
- Distributed externally
Analyzing C code with Frama-C
The Frama-C platform

Frama-C Kernel

Main role

- Parsing and pretty-printing C code
- Manage internal state of plugins
- Manage properties status
- Orchestrate inter-plugins collaboration
- Save and load internal state

Example

frama-c examples/code.c \ 
   -val -main f \ 
   -then -wp \ 
   -then -save code.sav
frama-c-gui -load code.sav
frama-c -load code.sav -report
Analyzing C code with Frama-C

ANSI/ISO C Specification Language

Presentation

▶ Based on the notion of contract, like in Eiffel
▶ Allows users to specify functional properties of their code
▶ Allows communication between various plugins
▶ Independent from a particular analysis
▶ ACSL manual at 
  https://github.com/acsl-language/acsl/releases

Basic Components

▶ First-order logic
▶ Pure C expressions
▶ C types + \( \mathbb{Z} \) (integer) and \( \mathbb{R} \) (real)
▶ Built-ins predicates and logic functions, particularly over pointers.
Analyzing C code with Frama-C

ACSL

Integer Arithmetic in ACSL

- All operations are done over $\mathbb{Z}$: no overflow

- ACSL predicate $\text{INT}_\text{MIN} \leq x + y \leq \text{INT}_\text{MAX}$
  \[ \iff \]
  C operation $x+y$ does not overflow (undefined behavior)

- $(\text{int})z \equiv z \mod 2^{8\times\text{sizeof}(\text{int})}$

- and $\text{INT}_\text{MIN} \leq (\text{int})z \leq \text{INT}_\text{MAX}$
Operations over $\mathbb{R}$: infinite precision

- \texttt{\round\_double(r, NearestEven)} to explicitly choose rounding mode

- Predicates $\texttt{is\_finite(d)}$, $\texttt{is\_plus\_infinity(d)}$, $\texttt{is\_NaN(d)}$, ...

- Function $\texttt{exact(x)}$: the value that C variable $x$ would have if all computations had been done using $\mathbb{R}$. $\texttt{round\_error}$ is the distance between $x$ and $\texttt{exact(x)}$

- Typical specification:
  \texttt{round\_error(result)} $\leq$ acceptable\_limit
Analyzing C code with Frama-C

ACSL

Memory description in ACSL

\begin{verbatim}
struct S {
  short x;
  int y;
} s[2];
\end{verbatim}

\valid(&s[0]+(0 .. 1))
\valid((char*)&s[0] + (0 .. 15))
!\initialized(*((char*)&s[0].x+2))
\block_length(&s[0]) == 16
\base_addr(&s[0].y) == s
\offset(&s[1].y) == 12
\separated(&s[0],&s[1])
Question
If we have `\text{valid}(p+(0 \ldots 2))`, with `p` a pointer to `int`, and `\text{sizeof}(\text{int})==4`, what can we say about `block\_length(p)`?

Answers

- **a** `block\_length(p) == 2`
- **b** `block\_length(p) == 3`
- **c** `block\_length(p) == 8`
- **d** `block\_length(p) == 12`
- **e** `block\_length(p) >= 12`
Function Contract

/*@
requires R(x);
ensures E(result, x);
behavior extra:
  assumes A(x);
  ensures more_result(result, x);
*/
int f(int x);
Analyzing C code with Frama-C

ACSL

Function Contract

What is required from caller

```c
/*@ requires R(x);

ensures E(result,x);

behavior extra:
    assumes A(x);
    ensures more_res
*/

int f(int x);
```

What the function guarantees when returning successfully
Analyzing C code with Frama-C

ACSL

@requires R(x);
@ensures E(@result, x);
@behavior extra:
  @assumes A(x);
  @ensures more_result(@result, x);

int f(int x);
Question
Assuming an ACSL function `acsl_strlen` that returns the offset of the first ‘\0’ char if it exists and -1 otherwise, what would be an appropriate requires for the standard library function `size_t strlen(const char* s)`?

Answers

- a  `acsl_strlen(s) >= 0`
- b  `acsl_strlen(s) >=0 && valid(s+ (0 .. acsl_strlen(s)))`
- c  `valid(s + (0 .. acsl_strlen(s)))`
- d  `acsl_strlen(s) >= 0 && valid(s)`
/*@ assert p == NULL || !valid(p); */
if (p) { *p = 42; }
if (0) { /*@ assert false; */ exit (1); }
Assess a property at given point

/*@ assert p == NULL || \valid(p); */
if (p) { *p = 42; }

if (0) { /*@ assert \false; */ exit (1); }
Assess a property at given point

```c
/*@ assert p == NULL || valid(p); */
if (p) { *p = 42; }

if (0) {/*@ assert false; */ exit (1); }
```

Indicates dead code
What is verified by Frama-C?

**Code Properties**
- Functional properties (contract)
- Absence of run-time error
- Dependencies
- Termination
- Non-interference
- Temporal properties

**Perimeter of the verification**
- Which part of the code is under analysis?
- Which initial context?

**Trusted Code Base**
- ACSL Axioms
- Hypotheses made by analyzers
- Stub Functions
- Frama-C itself
Context

Overview of Static Analysis

Analyzing C code with Frama-C

EVA Plugin
  Basics
  Refining Analysis
  Setting Analysis Context
Credits

- Pascal Cuoq
- Boris Yakobowski
- André Maroneze
- David Buhler
- Valentin Perrelle
- Matthieu Lemerre
- A few other developers...

More information

Main Objective

Find the domains of the variables of a program

- based on abstract interpretation
- alarms on operations that may be invalid
- alarms on the specifications that may be invalid
- Correct: if no alarm is raised, no runtime error can occur
Some specificities

- Precise handling of pointers
- Several representation for dynamic allocation (precision vs. time)
- Time and memory efficient (as much as achievable)
- Precise enough
  - for proving absence of runtime errors on some critical code
  - to serve as a back-end for other semantical analyzes through its API
**Integer and Floating Point Arithmetic**

**Corresponding Abstract Domain**

- small set of integers (by default, cardinal \( \leq 8 \))
- integer interval \( \times \) modulo information
- finite floating-point interval

**Examples**

- \( \{0; 40;\} = 0 \) or 40
- \([0..40] = \) an integer between 0 and 40 (inclusive)
- \([-..-] = \) any integer (within the bound of the corresponding integral type)
- \([3..39], 3\%4 = 3, 7, 11, 15, 19, 23, 27, 31, 35 \) or 39
- \([0.25..3.125] = \) floating-point between 0.25 and 3.125 (inclusive)
```c
int x, y, t, m; double d;
extern char z; char z1;

void f(int c) {
    if (c) x = 40;
    for (int i = 0; i<=40; i++) {
        Frama_C_show_each_loop_1(i);
        if (c == i) y = i;
    }
    z1 = z;
    t = z;
    m=3;
    for (int i = 3; i<=40; i+=4) {
        if (c == i) m = i;
    }
    if (c) { d = 0.25; } else { d = 3.125; }
}
```
frama-c -val -main f integer.c

[value] Called Frama_C_show_each_loop_1({0; 1})
[value] Called Frama_C_show_each_loop_1({0; 1; 2})
[value] Called Frama_C_show_each_loop_1([0..16])
[value] Called Frama_C_show_each_loop_1([0..40])
[value] ====== VALUES COMPUTED ======
x IN {0; 40}
y IN [0..40]
z1 IN [--..--]
t IN [-128..127]
m IN [3..39], 3%4
d IN [0.25 .. 3.125]
Question

if \( x \) is in the interval \([-10 \ .. \ 10]\) before the execution of statement

\[
\text{if} \ (x==0) \ { \ y = 14; } \\
\text{else} \ { \ y = x<0 \ ? \ 13 \ : \ x + 2; } 
\]

What is the value associated to \( y \) after the statement?

Answers

- a \([-8 \ .. \ 14]\)
- b \[2 \ .. \ 13]\)
- c \[2 \ .. \ 14]\)
- d \[3 \ .. \ 14]\)
Memory Address

Base Address

Global variable
⊕ Formal parameter of main function
⊕ literal string constant
⊕ NULL
⊕ ...

Addresses

▷ Base address + Offset (integer)
▷ Each base has a maximal valid offset
▷ Abstract Values are sets of addresses
Examples of Addresses

Precise Base

- $\{\& p + \{4; 8\}\}$ = address of $p$ shifted from 4 or 8 octets
- $\{\&" foobar" ; \}$ = Address of literal string "foobar" (shifted from 0)
- $\{\& NULL + \{1024 ; \}\}$ = Absolute location 1024

Imprecision

- garbled mix of $\&\{x_1 ; \ldots ; x_n\}$ = unknown address built upon arithmetic operations over integers and addresses $x_1 ; \ldots ; x_n$.
- ANYTHING = top of the lattice. Should not occur in practice
```c
int* x,*z,*t; const char* y; int p[3];
const char* string = "foobar";

void f(int c) {
    if (c) { x = &p[1]; }
    else    { x = &p[2]; }
    y = string;
    z = (int*)1024;
    t = (int*) ((int)x | 4096);
}
```
[value] ====== VALUES COMPUTED ======

[value] Values at end of function f:
   x IN {{ &p{[1], [2]} }}
   y IN {{ "foobar" }}
   z IN {1024}
   t IN
   {{ garbled mix of &{p}
     (origin: Arithmetic
     {examples/value/address.c:16}) }}
Abstract Domain
written address = valid left value

- address
- × initialized?
- × not dangling pointer?

Example

```c
int x, y;
if (e) x = 2;
L: if (e) y = x + 1;
```

- At L, we know that $x$ equals 2 iff it has been initialized
- Depending on the complexity of $e$, we know that $y$ equals 3 if $x$ equals 2
```c
int X, Y, *p;
void f(int c) {
    int x, y;
    if (c<=0) x = 2;
    L: if (c<=0) y = x + 1; else y = 4;
    X = x;
    Y = y;
    p = c ? &X : &x;
}

int main(int c) {
    f(c);
    if (Y==4) *p = 3;
    return 0;
}
```
examples/value/address_written.c:8:
[kernel] warning:
  accessing uninitialized left-value:
  assert \initialized(&x);
examples/value/address_written.c:16:
[kernel] warning:
  accessing left-value that contains escaping addresses:
  assert !\dangling(&p);
[value] Values at end of function main:
  X IN {2; 3} or UNINITIALIZED
  Y IN {3; 4}
  p IN {{ &X }} or ESCAPINGADDR
  __retres IN {0}
Question

if \( a \) is an array of size 3, initialized to 0, and \( c \in [0 \ldots 2] \), what would be the content of \( a \) after executing the following statement:

\[
\text{if } (c) \{ \ a[c] = c; \ } \text{ else } a[1] = 3;
\]

Answers

- **a** \( a[0] \) IN \{0\}, \( a[1] \) IN \{0,1,3\}, \( a[2] \) IN \{0,2\}
- **b** \( a[i] \) IN \{0,1,2,3\} for all indices
- **c**
  \( a[0] \) IN \{0\}, \( a[1] \) IN \{0,1,2,3\} \( a[2] \) IN \{0,1,2\}
- **d** \( a[0] \) IN \{0\}, \( a[1] \) IN \{1,3\}, \( a[2] \) IN \{2\}

See solution
Adding other domains

- New domains can provide additional information:
  - equalities between values
  - values of symbolic locations
  - gauges, affine relation wrt number of loop steps

- Possible to add new domains

- Inter-domain communication done through queries:

  **val** extract_expr :
  (exp -> value evaluated) ->
  state -> exp -> (value * origin) evaluated

  **val** extract_lval :
  (exp -> value evaluated) ->
  state -> lval -> typ -> location -> (value * o
Example

```c
#include "__fc_builtin.h"

int main () {
    int x = Frama_C_interval(0,10);
    int y = x;
    if ( y <= 5) {
        return x;
    } else {
        return 10 - x;
    }
}
```
Loops and Branching

Main options

- **option** \(-\text{slevel}\): allows EVA to explore \(n\) separated paths before joining them
- **option** \(-\text{slevel-function}\): same as previous, but for a particular function
- **annotation** loop pragma \texttt{UNROLL}: syntactic loop unrolling
- **annotation** loop pragma \texttt{WIDEN HINTS}: give bounds for widening

For specialists only

- **option** \(-\text{iLevel}\): maximum number of elements in the set before conversion into intervals
- **option** \(-\text{pLevel}\): maximum number of distinct array cells
ACSL assertions can be used to restrict propagated domains
but only if Value can interpret it

/*@ assert x % 2 == 0; */
// potentially useful
/*@ assert \exists integer y; x == 2 * y; */
// useless

Case analysis using disjunctions
int S = 0;

int T[5];

int main(void) {
    int i;
    int *p = &T[0];
    for (i = 0; i < 5; i++) {
        S = S + i; *p++ = S;
    }
    return S;
}
int x, y;

void main (int c) {
    if (c) { x = 10; } else { x = 33; }
    if (!c) { x++; } else { x--; }

    if (c<=0) { y = 42; } else { y = 36; }
    if (c>0) { y++; } else { y--; }
}
without slevel

\[ x \in \{9; 11; 32; 34\} \]
\[ y \in \{35; 37; 41; 43\} \]

with slevel, no assertion

\[ x \in \{9; 11; 34\} \]
\[ y \in \{37; 41\} \]

with slevel and assertion

\[ \texttt{/*@ assert } c \leq 0 || c > 0\texttt{; */} \]

[value] Assertion got status valid.
\[
\begin{align*}
x & \in \{9; 34\} \\
y & \in \{37; 41\}
\end{align*}
\]
Entry Point

- Which part of the code should be analyzed?
- `-main f` starts the analysis at function `f`
- `-lib-entry` indicates that the initial global context is not 0-initialized
- `-context-width`, `-context-depth`
- Use of a driver function with some builtins to provide non-determinism:

```c
void f_wrapper() {
    setup_analysis_context();
    f(arg_1, arg_2);
}
```
```c
int search(char* a, char key) {
    char* orig = a;
    while (*a) {
        if (*a == key) return a - orig;
        a++;
    }
    return -1;
}
```
frama-c -val -context-width 3 -main search context.c

[...]

collection.c:3:[kernel] warning: out of bounds read. assert \valid_read(a);
collection.c:4:[kernel] warning: out of bounds read. assert \valid_read(a);
collection.c:4:[kernel] warning: pointer subtraction:
  assert \base_addr(a) == \base_addr(orig);
[value] Recording results for search
[value] done for function search
[value] ====== VALUES COMPUTED ======
[value] Values at end of function search:
  a IN {{ \&S_a[[0], [1], [2]] }}
  orig IN {{ NULL ; \&S_a[0] }}
  __retres IN {{-1; 0; 1; 2}}
```c
#include "__fc_builtin.h"
#include "limits.h"

int search(char* a, char key);

char buffer[1024];

int driver() {
    buffer[1023] = 0;
    char key = Frama_C_interval(CHAR_MIN, CHAR_MAX);
    return search(buffer, key);
}
```
frama-c -val -context-width 3 -main driver \context.c context_driver.c -lib-entry \ -slevel 1024

[ ... No alarm ... ]

[value] Values at end of function search:
a IN {{ &buffer + [0..1023] }}
orig IN {{ &buffer[0] }}
__retres IN [-1..1022]

[value] Values at end of function driver:
Frama_C_entropy_source IN [---.--]
buffer[0..1022] IN [---.--]
[1023] IN {0}
key IN [---.--]
Provide an “implementation” for EVA

▶ Assumed to match the real implementation
▶ Write stub directly in C (aimed at ease of analysis, not performance)
▶ Provide an ACSL specification
▶ -val-use-spec f
▶ Use an EVA built-in (-val-builtin)
▶ -val-builttins-list
Assumptions made by EVA

Command-line Options

- `--val-ignore-recursive-calls` assumes recursive calls have no effect
- `--all-rounding-modes` do not assume floating-point computations use same rounding as host machine

ACSL Properties

- Alarms emitted by Value
- Annotations with Unknown status
static int *table = NULL;
static size_t size = 0;

int insert_in_table(size_t pos, int value) {
    if (size < pos) {
        int *tmp;
        size = pos + 1;
        tmp = (int *)realloc(table, sizeof(*table) * size);
        if (tmp == NULL) {
            return -1; /* Failure */
        }
        table = tmp;
    }
    table[pos] = value;
    return 0;
}
EVA Plugin
Setting Analysis Context

Analyzing real code

▶ D. Delmas and J. Souyris: ASTRÉE: from Research to Industry, SAS 2007
▶ INGOPCS project: https://www.ingopcs.net
▶ Open-Source Case Studies: https://github.com/Frama-C/open-source-case-studies
Context

Overview of Static Analysis

Analyzing C code with Frama-C

EVA Plugin
General


ACSL

► Burghardt & al. ACSL by Example (v16.1). December 2017.
https://github.com/fraunhoferfokus/acsl-by-example

EVA

► Cuoq & al. Frama-C’s value analysis plug-in. May 2018
► Blazy & al. Structuring Abstract Interpreters through State and
Course

- Patrick Cousot, MIT 2005
  http://web.mit.edu/afs/athena.mit.edu/course/16/16.399/www/

Books

Founding Articles

- Patrick and Radhia Cousot, *Abstract Interpretation: a Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints*. PoPL’77
- Patrick Cousot and Nicolas Halbwachs, *Automatic Discovery of Linear Restraints Among Variables of a Program*. PoPL’78
- Patrick and Radhia Cousot, *Systematic Design of Program Analysis Frameworks*. PoPL’79
Solutions to Quizzes
Question
We have information from two domains: Intervals:

- $x \in [0; 20]$
- $y \in [5; 10]$

Octagons:

- $0 \leq x - y \leq 20$

What can be said about $x$ and $y$?

Answers

- **a** $x \in [0; 20], y \in [5; 10]; 0 \leq x - y \leq 20$  ✗
- **b** $x \in [5; 20], y \in [5; 10], 0 \leq x - y \leq 15$
- **c** $x \in [5; 20], y \in [5; 10], 0 \leq x - y \leq 10$
- **d** $x \in [5; 20], y \in [0; 20], 0 \leq x - y \leq 20$
Question

We have information from two domains:

Intervals:

- $x \in [0; 20]$
- $y \in [5; 10]$

Octagons:

- $0 \leq x - y \leq 20$

What can be said about $x$ and $y$?

Answers

- a $x \in [0; 20]$, $y \in [5; 10]$; $0 \leq x - y \leq 20$
- b $x \in [5; 20]$, $y \in [5; 10]$, $0 \leq x - y \leq 15$
- c $x \in [5; 20]$, $y \in [5; 10]$, $0 \leq x - y \leq 10$
- d $x \in [5; 20]$, $y \in [0; 20]$, $0 \leq x - y \leq 20$
Question
We have information from two domains:
Intervals:

▶ \( x \in [0; 20] \)
▶ \( y \in [5; 10] \)

Octagons:
\( 0 \leq x - y \leq 20 \)

What can be said about \( x \) and \( y \)?

Answers

- **a** \( x \in [0; 20], \ y \in [5; 10]; \ 0 \leq x - y \leq 20 \)
- **b** \( x \in [5; 20], \ y \in [5; 10], \ 0 \leq x - y \leq 15 \)
- **c** \( x \in [5; 20], \ y \in [5; 10], \ 0 \leq x - y \leq 10 \) ✗
- **d** \( x \in [5; 20], \ y \in [0; 20], \ 0 \leq x - y \leq 20 \)

See solution
Question
We have information from two domains:
Intervals:
- \( x \in [0; 20] \)
- \( y \in [5; 10] \)
Octagons:
- \( 0 \leq x - y \leq 20 \)
What can be said about \( x \) and \( y \)?

Answers
- a \( x \in [0; 20], y \in [5; 10]; 0 \leq x - y \leq 20 \)
- b \( x \in [5; 20], y \in [5; 10], 0 \leq x - y \leq 15 \)
- c \( x \in [5; 20], y \in [5; 10], 0 \leq x - y \leq 10 \)
- d \( x \in [5; 20], y \in [0; 20], 0 \leq x - y \leq 20 \) ✗
Question
If we have `\text{valid}(p+(0 .. 2))`, with `p` a pointer to `int`, and `\text{sizeof}(\text{int})==4`, what can we say about `\text{block\_length}(p)`?

Answers

- a \hspace{1em} `\text{block\_length}(p) == 2` \hspace{1em} ✗
- b \hspace{1em} `\text{block\_length}(p) == 3`
- c \hspace{1em} `\text{block\_length}(p) == 8`
- d \hspace{1em} `\text{block\_length}(p) == 12`
- e \hspace{1em} `\text{block\_length}(p) >= 12`
Question

If we have $\text{valid}(p+(0 .. 2))$, with $p$ a pointer to $\text{int}$, and sizeof(int)==4, what can we say about $\text{block_length}(p)$?

Answers

- $a$ $\text{block_length}(p) == 2$
- $b$ $\text{block_length}(p) == 3$ ✗
- $c$ $\text{block_length}(p) == 8$
- $d$ $\text{block_length}(p) == 12$
- $e$ $\text{block_length}(p) >= 12$
Question
If we have \( \text{valid}(p+(0 .. 2)) \), with \( p \) a pointer to int, and sizeof(int)==4, what can we say about \( \text{block_length}(p) \)?

Answers

- **a** \( \text{block_length}(p) == 2 \)
- **b** \( \text{block_length}(p) == 3 \)
- **c** \( \text{block_length}(p) == 8 \)  
  - ✗
- **d** \( \text{block_length}(p) == 12 \)
- **e** \( \text{block_length}(p) >= 12 \)
Question

If we have $\text{valid}(p + (0 .. 2))$, with $p$ a pointer to int, and $\text{sizeof}(\text{int}) == 4$, what can we say about $\text{block\_length}(p)$?

Answers

- **a** $\text{block\_length}(p) == 2$
- **b** $\text{block\_length}(p) == 3$
- **c** $\text{block\_length}(p) == 8$
- **d** $\text{block\_length}(p) == 12$ ✗
- **e** $\text{block\_length}(p) >= 12$
Question

If we have `\texttt{valid(p+(0 .. 2))}`, with `p` a pointer to `int`, and `\texttt{sizeof(int)==4}`, what can we say about `\texttt{block_length(p)}`?

Answers

- **a** `\texttt{block_length(p) == 2}`
- **b** `\texttt{block_length(p) == 3}`
- **c** `\texttt{block_length(p) == 8}`
- **d** `\texttt{block_length(p) == 12}`
- **e** `\texttt{block_length(p) >= 12}`  ✔️
Question
Assuming an ACSL function `acsl_strlen` that returns the offset of the first `\0` char if it exists and -1 otherwise, what would be an appropriate `requires` for the standard library function `size_t strlen(const char* s)`?

Answers

- **a** `acsl_strlen(s) >= 0` ❌
- **b** `acsl_strlen(s) >=0 && valid(s+ (0 .. acsl_strlen(s)))`
- **c** `valid(s + (0 .. acsl_strlen(s)))`
- **d** `acsl_strlen(s) >= 0 && valid(s)`

See solution
Question

Assuming an ACSL function `acsl_strlen` that returns the offset of the first ‘\0’ char if it exists and -1 otherwise, what would be an appropriate requires for the standard library function `size_t strlen(const char* s)`?

Answers

- a) `acsl_strlen(s) >= 0`
- b) `acsl_strlen(s) >= 0 && 
  \textcolor{red}{\texttt{valid}}(s + (0 .. acsl_strlen(s)))`
- c) `\textcolor{red}{\texttt{valid}}(s + (0 .. acsl_strlen(s)))`
- d) `acsl_strlen(s) >= 0 && \textcolor{red}{\texttt{valid}}(s)`
Question
Assuming an ACSL function `acsl_strlen` that returns the offset of the first ‘\0’ char if it exists and -1 otherwise, what would be an appropriate requires for the standard library function `size_t strlen(const char* s)`?

Answers

- **a** `acsl_strlen(s) >= 0`
- **b** `acsl_strlen(s) >= 0 && \valid(s + (0 .. acsl_strlen(s)))`
- **c** `\valid(s + (0 .. acsl_strlen(s))) ×`
- **d** `acsl_strlen(s) >= 0 && \valid(s)`
Question

Assuming an ACSL function `acsl_strlen` that returns the offset of the first ‘\0’ char if it exists and -1 otherwise, what would be an appropriate requires for the standard library function `size_t strlen(const char* s)`?

Answers

- a) `acsl_strlen(s) >= 0`
- b) `acsl_strlen(s) >=0 && \valid(s+ (0 .. acsl_strlen(s)))`
- c) `\valid(s + (0 .. acsl_strlen(s)))`
- d) `acsl_strlen(s) >= 0 && \valid(s) ×`
**Question**

if \( x \) is in the interval \([-10 .. 10]\) before the execution of statement

```c
if (x==0) { y = 14; }
else { y = x<0 ? 13 : x + 2; }
```

What is the value associated to \( y \) after the statement?

**Answers**

- a \([-8 .. 14]\)  \(\times\)
- b \([2 .. 13]\)
- c \([2 .. 14]\)
- d \([3 .. 14]\)
Question

if x is in the interval [-10 .. 10] before the execution of statement

if (x==0) { y = 14; }
else { y = x<0 ? 13 : x + 2; }

What is the value associated to y after the statement?

Answers

a [−8 .. 14]
b [2 .. 13]  x
c [2 .. 14]
d [3 .. 14]
Question

if \( x \) is in the interval \([-10 \ldots 10]\) before the execution of statement

\[
\text{if } (x==0) \{ \ y = 14; \ \}
\]
\[
\text{else } \{ \ y = x<0 \ ? \ 13 : x + 2; \ \}
\]

What is the value associated to \( y \) after the statement?

Answers

- \( a \) \([-8 \ldots 14]\)
- \( b \) \([2 \ldots 13]\)
- \( c \) \([2 \ldots 14]\)
- \( d \) \([3 \ldots 14]\)
Question

if \( x \) is in the interval \([-10 \ldots 10]\) before the execution of statement

\[
\text{if } (x==0) \{ y = 14; \} \\
\text{else} \{ y = x<0 ? 13 : x + 2; \}
\]

What is the value associated to \( y \) after the statement?

Answers

- **a** \([-8 \ldots 14]\)
- **b** \([2 \ldots 13]\)
- **c** \([2 \ldots 14]\)
- **d** \([3 \ldots 14]\) ✗
Question
if $a$ is an array of size 3, initialized to 0, and $c$ in $[0 .. 2]$ what would be the content of $a$ after executing the following statement:

```c
if (c) { a[c] = c; } else a[1] =3;
```

Answers

- **a** $a[0]$ IN {0}, $a[1]$ IN {0,1,3}, $a[2]$ IN {0,2}
- **b** $a[i]$ IN {0,1,2,3} for all indices
- **c**
  - $a[0]$ IN {0}, $a[1]$ IN {0,1,2,3} $a[2]$ IN {0,1,2}
- **d** $a[0]$ IN {0}, $a[1]$ IN {1,3}, $a[2]$ IN {2}
Question

if \( a \) is an array of size 3, initialized to 0, and \( c \) in \([0 .. 2]\) what would be the content of \( a \) after executing the following statement:

\[
\text{if } (c) \{ \ a[c] = c; \ \} \ \text{else} \ a[1] = 3;
\]

Answers

- **a** \( a[0] \) IN \{0\}, \( a[1] \) IN \{0,1,3\}, \( a[2] \) IN \{0,2\}
- **b** \( a[i] \) IN \{0,1,2,3\} for all indices ✗
- **c**
  - \( a[0] \) IN \{0\}, \( a[1] \) IN \{0,1,2,3\} \( a[2] \) IN \{0,1,2\}
- **d** \( a[0] \) IN \{0\}, \( a[1] \) IN \{1,3\}, \( a[2] \) IN \{2\}
Question

if a is an array of size 3, initialized to 0, and c in [0 .. 2] what would be the content of a after executing the following statement:

```c
if (c) { a[c] = c; } else a[1] =3;
```

Answers

- **a** a[0] IN {0}, a[1] IN {0,1,3}, a[2] IN {0,2}
- **b** a[i] IN {0,1,2,3} for all indices
- **c**
  
  a[0] IN {0}, a[1] IN {0,1,2,3} a[2] IN {0,1,2} ✔
- **d** a[0] IN {0}, a[1] IN {1,3}, a[2] IN {2}
Question

if a is an array of size 3, initialized to 0, and c in [0 .. 2] what would be the content of a after executing the following statement:

```c
if (c) { a[c] = c; } else a[1] =3;
```

Answers

- **a** a[0] IN {0}, a[1] IN {0,1,3}, a[2] IN {0,2}
- **b** a[i] IN {0,1,2,3} for all indices
- **c** a[0] IN {0}, a[1] IN {0,1,2,3} a[2] IN {0,1,2}
- **d** a[0] IN {0}, a[1] IN {1,3}, a[2] IN {2} ✗