Abstract Interpretation and Properties of C Programs

EJCP 2018

Virgile Prevosto
virgile.prevosto@cea.fr

June 26th, 2018

Introduction

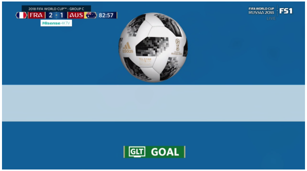
- Software is more and more pervasive in embedded systems...
- ...and keeps getting larger
- Tests and code review too costly beyond a certain size and coverage criterion
- Need for correct tools
  - ✔ Detect all potential issues
  - ✘ May issue spurious warnings
  - ✘ Impossible for an automated tool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

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)

Summary

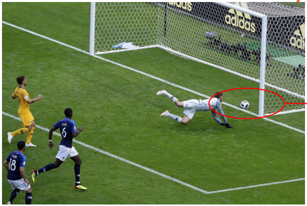
Abstract Interpretation in two pictures

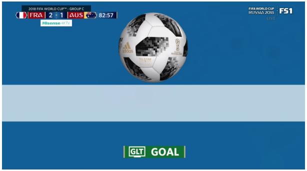


Abstract interpretation is about

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

Abstract Interpretation in two pictures

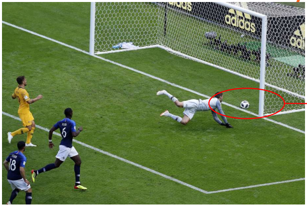


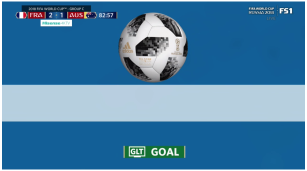


Abstract interpretation is about

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

Abstract Interpretation in two pictures





Abstract interpretation is about

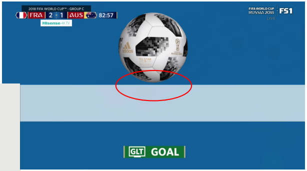
- abstracting away information
- ensuring answer in a reasonable time
- while retaining adequate precision
- and guaranteeing correct answers

Abstract Interpretation 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 in two pictures




Foot : après de nouveaux bugs, la Ligue suspend la "goal-line technology" jusqu'à nouvel ordre

Les derniers dysfonctionnements ont été constatés, mercredi soir, lors des quarts de finale de Coupe de la Ligue, entre Amiens-PSG et Angers-Montpellier.

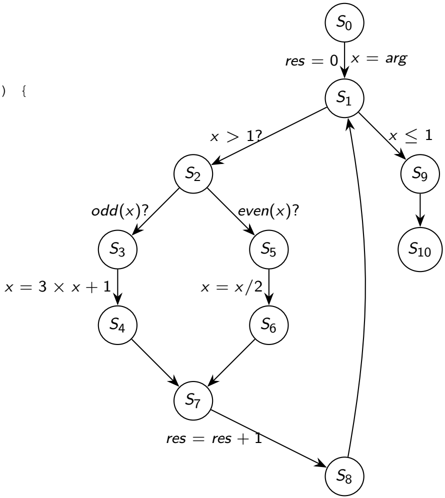

Surréaliste : un bug de la goal-line force l'arbitre à accorder puis annuler un but à Troyes

Abstract interpretation is about

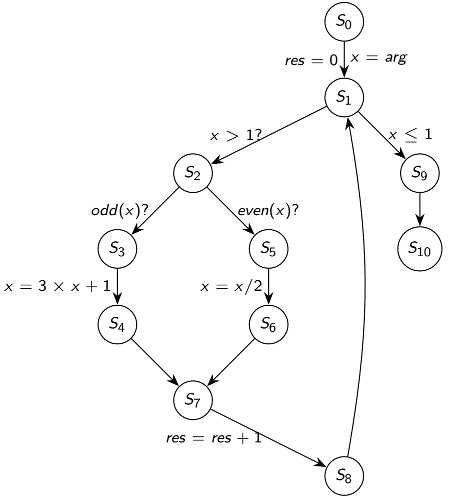
- abstracting away information
- ensuring answer in a reasonable time
- while retaining adequate precision
- and guaranteeing correct answers

Control-Flow Graph

```
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('\n');
    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;
}
```

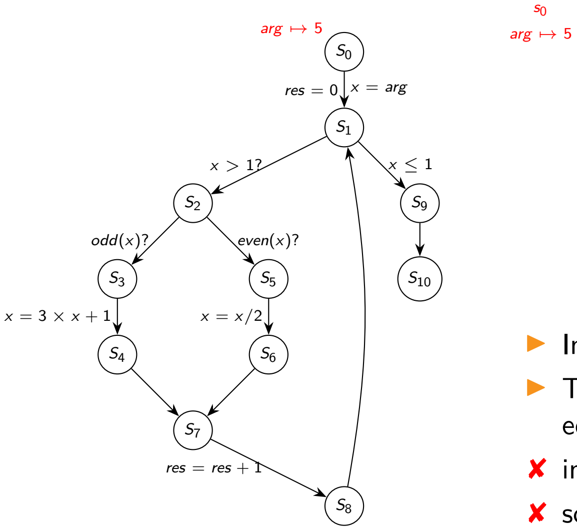


Trace Semantics



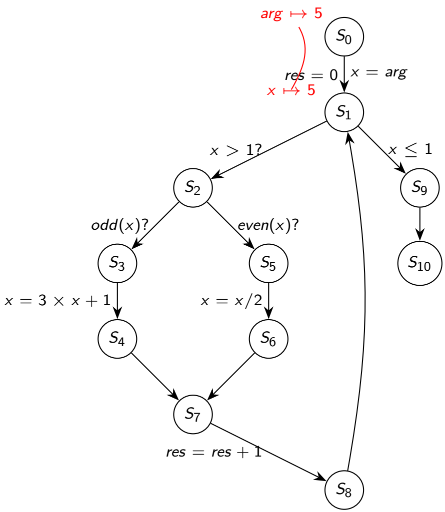
- ▶ Initial state on start node
- ▶ Transfer functions across edges
- ✘ infinite number of traces
- ✘ some traces might be infinite

Trace Semantics



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

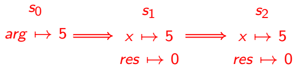
Trace Semantics

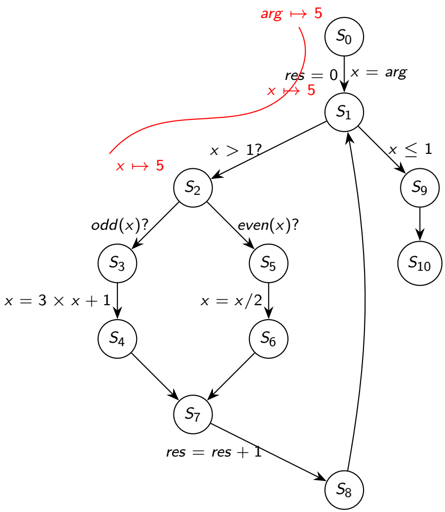


s_0: $arg \mapsto 5$ $\Longrightarrow$ s_1: $x \mapsto 5$, $res \mapsto 0$

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

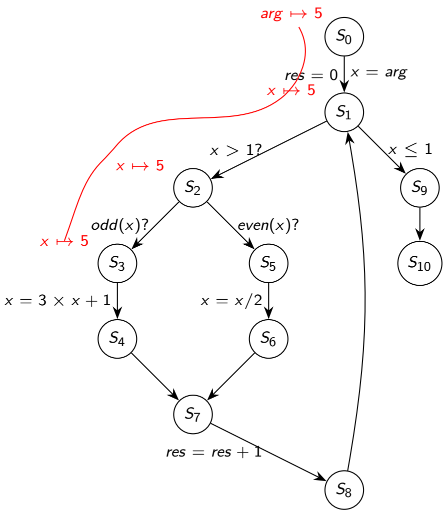
Trace Semantics

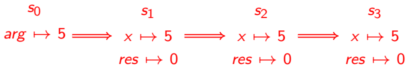




- Initial state on start node
- Transfer functions across edges
- ✘ infinite number of traces
- ✘ some traces might be infinite

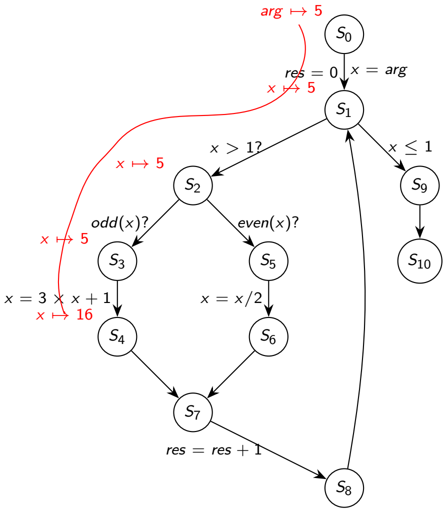
Trace Semantics

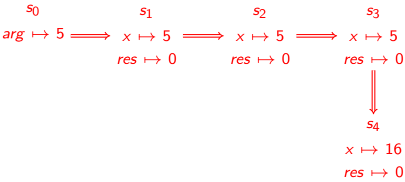




- Initial state on start node
- Transfer functions across edges
- ✘ infinite number of traces
- ✘ some traces might be infinite

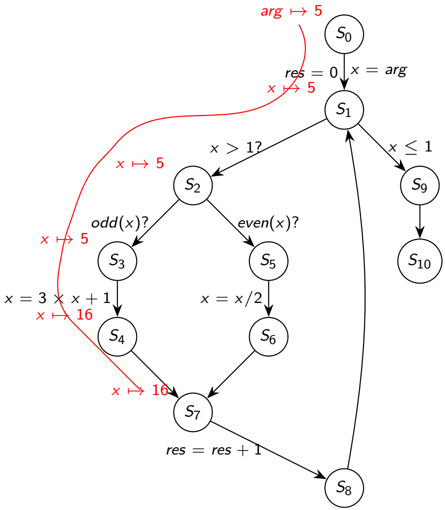
Trace Semantics

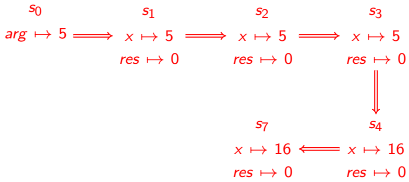




- ▶ Initial state on start node
- ▶ Transfer functions across edges
- ✘ infinite number of traces
- ✘ some traces might be infinite

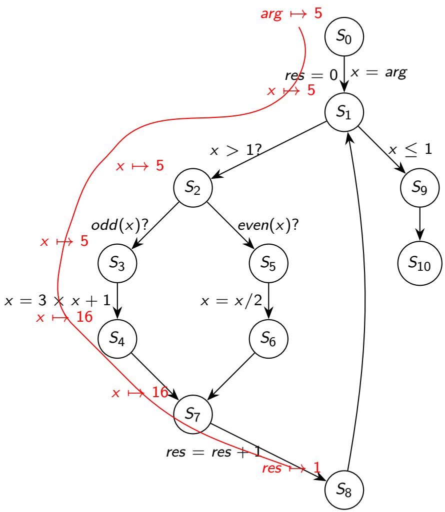
Trace Semantics

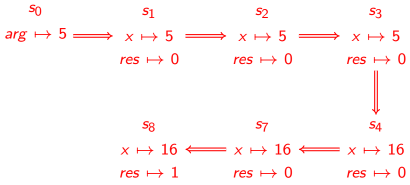




- ▶ Initial state on start node
- ▶ Transfer functions across edges
- ✘ infinite number of traces
- ✘ some traces might be infinite

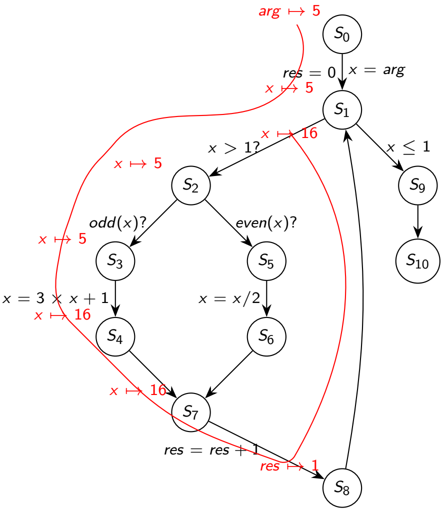
Trace Semantics

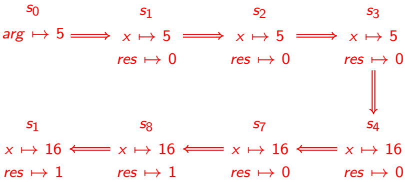




- Initial state on start node
- Transfer functions across edges
- ✘ infinite number of traces
- ✘ some traces might be infinite

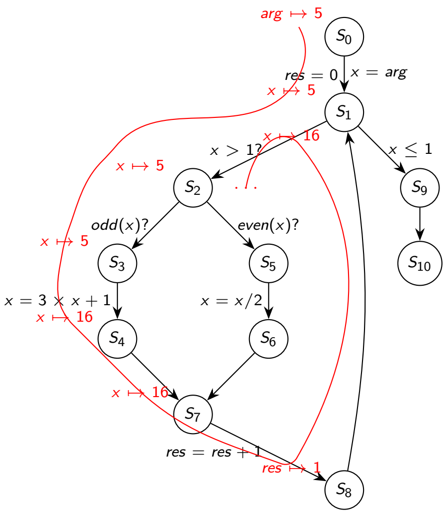
Trace Semantics

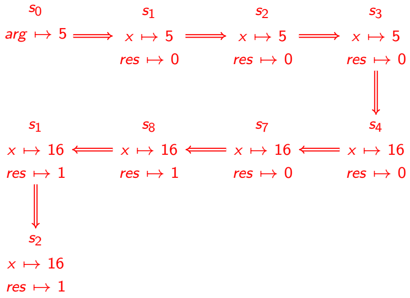




- Initial state on start node
- Transfer functions across edges
- ✘ infinite number of traces
- ✘ some traces might be infinite

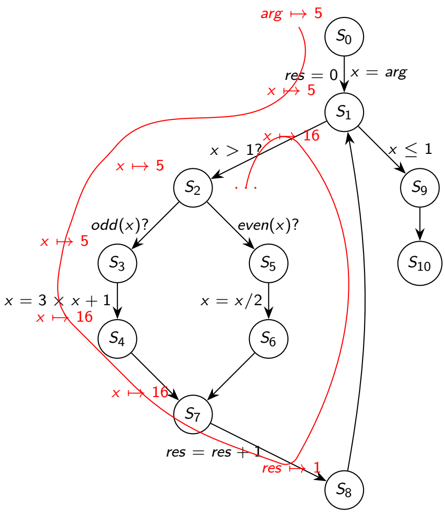
Trace Semantics

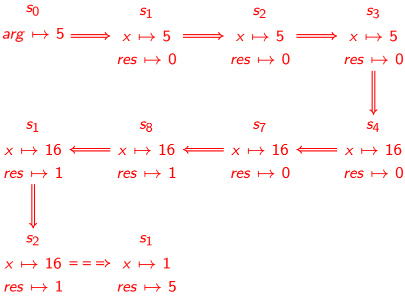




- Initial state on start node
- Transfer functions across edges
- ✘ infinite number of traces
- ✘ some traces might be infinite

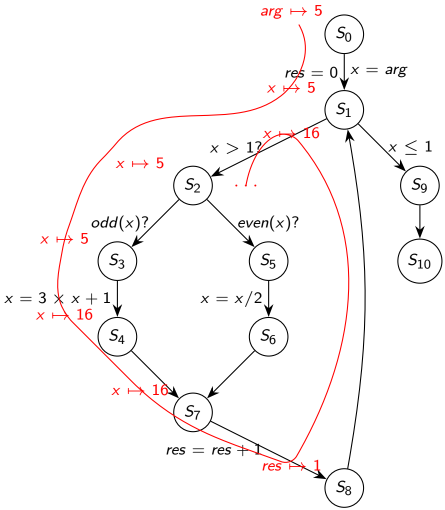
Trace Semantics

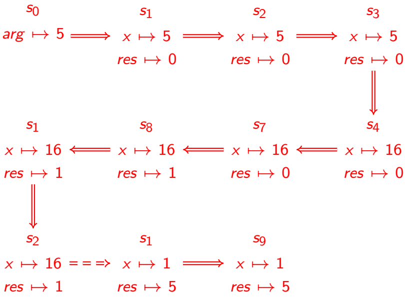




- Initial state on start node
- Transfer functions across edges
- ✘ infinite number of traces
- ✘ some traces might be infinite

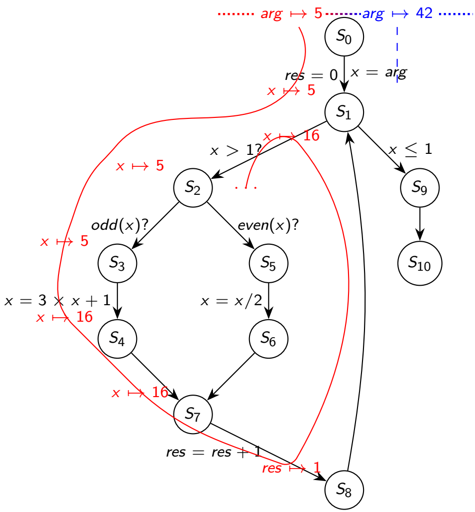
Trace Semantics

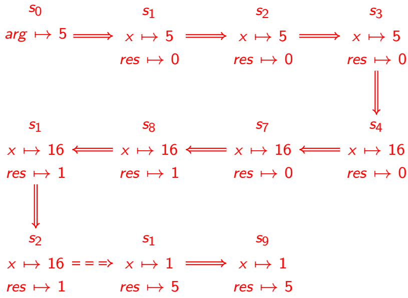




- ▶ Initial state on start node
- ▶ Transfer functions across edges
- ✘ infinite number of traces
- ✘ some traces might be infinite

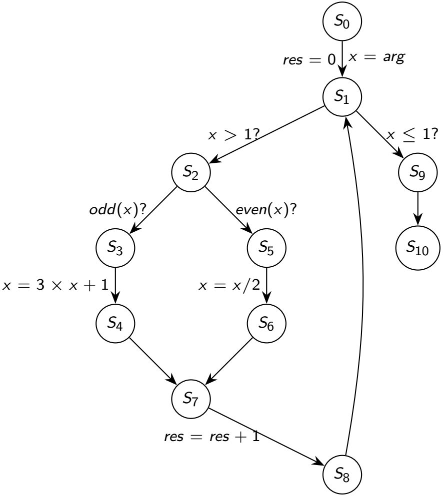
Trace Semantics





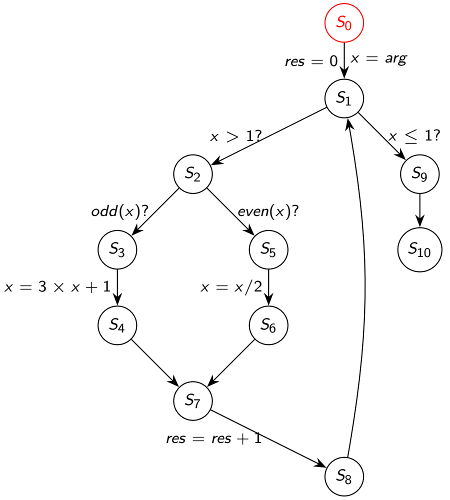
- ▶ Initial state on start node
- ▶ Transfer functions across edges
- ✘ infinite number of traces
- ✘ some traces might be infinite

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

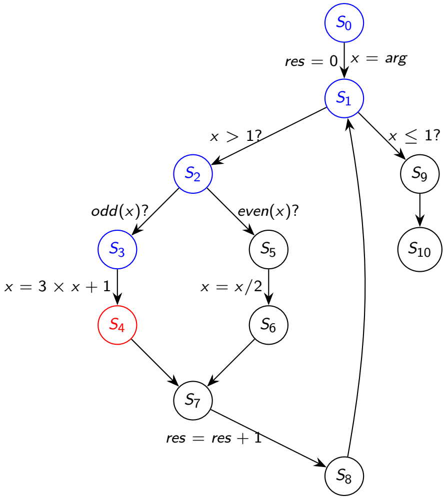
Collecting Semantics



$S_0 \mapsto arg \in \mathbb{Z}$

- 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



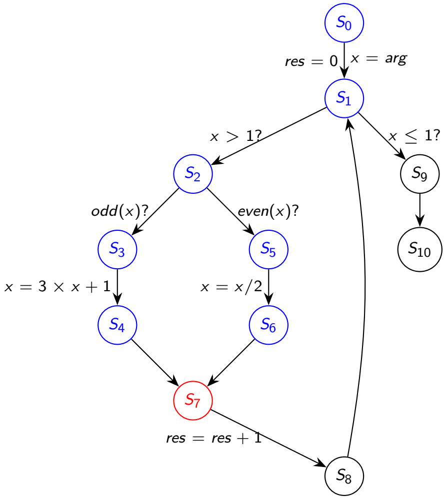
$S_3 \mapsto \{(x, res) | x = 2k + 1, res = 0\}$

$\Downarrow$

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

- 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



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

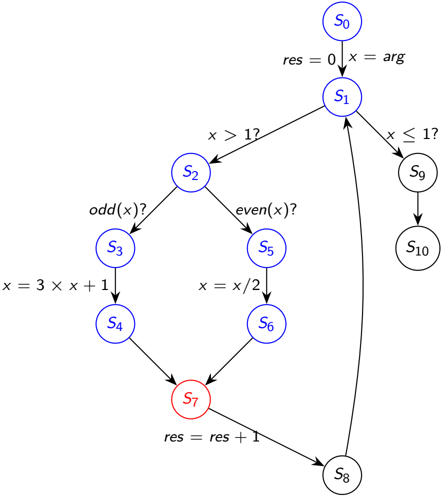
$S_6 \mapsto \{(x, res) | x = k, res = 0\}$

$\Downarrow$

$S_7 \mapsto \{(x, res) | x = k, res = 0\}$

- 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



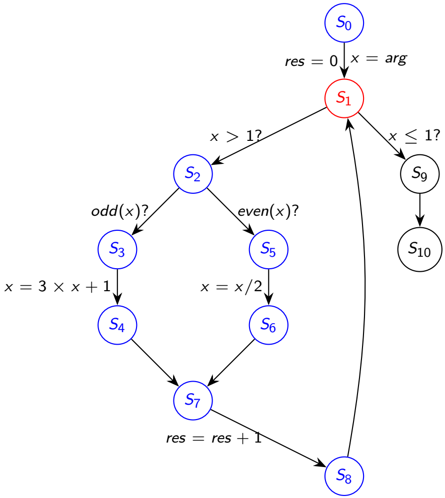
$$S_4 \mapsto \{(x, res) | x = 6k + 2, res = 0\}$$

$$S_6 \mapsto \{(x, res) | x = k, res = 0\}$$

$$S_7 \mapsto \{(x, res) | x = k, res = 0\}$$

- 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



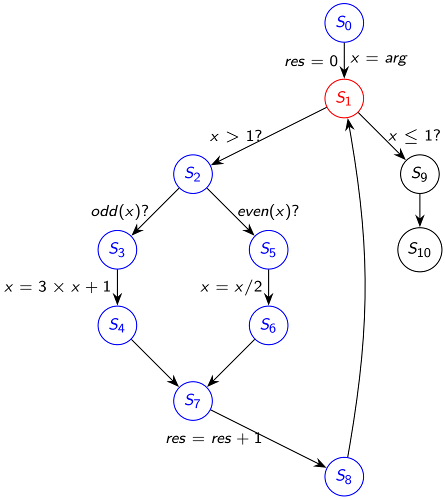
$S_1 \mapsto \{(x, res) | x = k, res = 0\}$

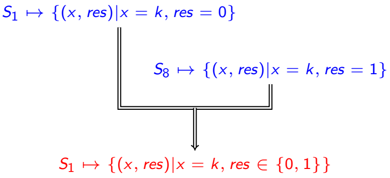
$S_8 \mapsto \{(x, res) | x = k, res = 1\}$

$S_1 \mapsto \{(x, res) | x = k, res \in \{0, 1\}\}$

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

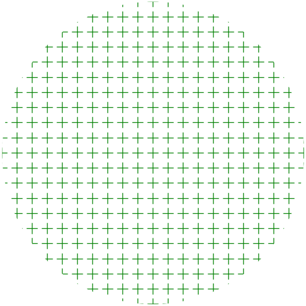
Collecting Semantics





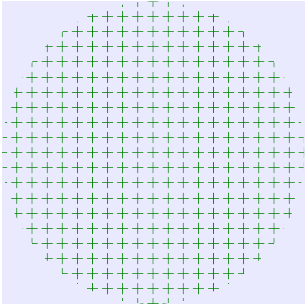
- From the set of all traces to set of all states
- multiple predecessors: take union
- lose "temporal" relations
- fixpoint computation
- may not terminate

Static Analysis framework



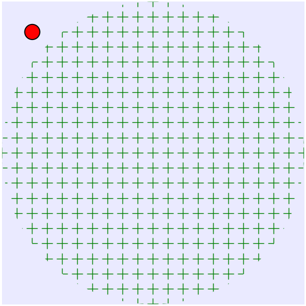
- Replace set of states ...
- ... by one element in an abstract lattice
- Over-approximation and false alarms
- Trade-off between precision and computation time

Static Analysis framework



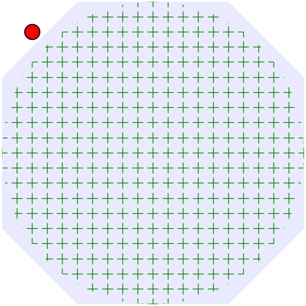
- Replace set of states ...
- ... by one element in an abstract lattice
- Over-approximation and false alarms
- Trade-off between precision and computation time

Static Analysis framework



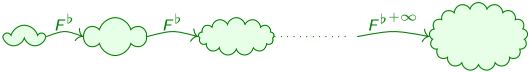
- Replace set of states …
- … by one element in an abstract lattice
- Over-approximation and false alarms
- Trade-off between precision and computation time

Static Analysis framework



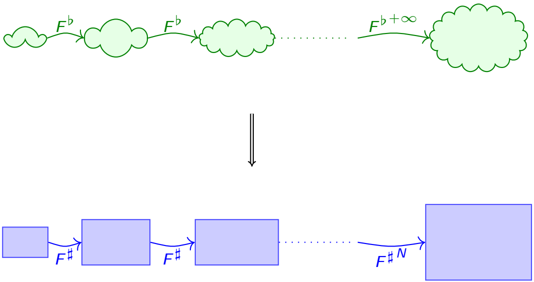
- Replace set of states ...
- ... by one element in an abstract lattice
- Over-approximation and false alarms
- Trade-off between precision and computation time

Correctness and Termination



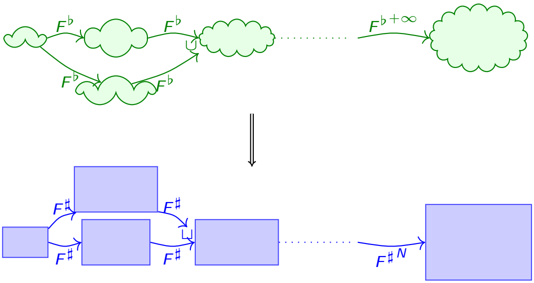
- Abstract transfer functions $F^\sharp$
- Merge abstract states for nodes with multiple predecessors
- Correction: Do we include all concrete states in the end?
- Termination: Converge in a finite number of steps
- Abstract interpretation: A systematic way to build correct and terminating analyses

Correctness and Termination



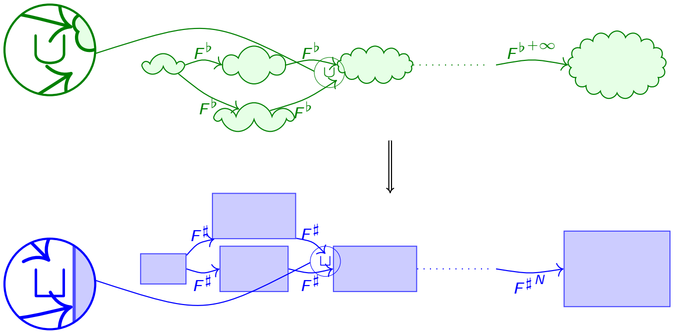
- Abstract transfer functions $F^\sharp$
- Merge abstract states for nodes with multiple predecessors
- Correction: Do we include all concrete states in the end?
- Termination: Converge in a finite number of steps
- Abstract interpretation: A systematic way to build correct and terminating analyses

Correctness and Termination



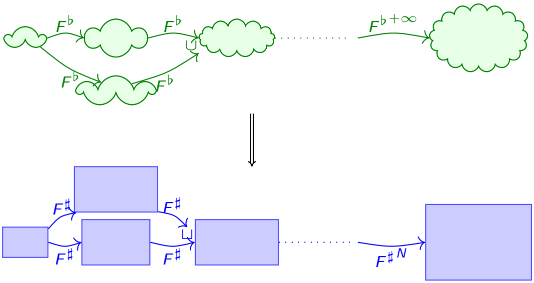
- Abstract transfer functions $F^\sharp$
- Merge abstract states for nodes with multiple predecessors
- Correction: Do we include all concrete states in the end?
- Termination: Converge in a finite number of steps
- Abstract interpretation: A systematic way to build correct and terminating analyses

Correctness and Termination



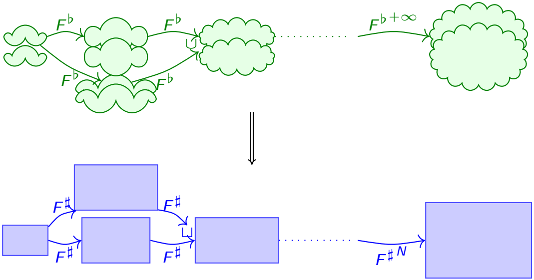
- Abstract transfer functions $F^\sharp$
- Merge abstract states for nodes with multiple predecessors
- Correction: Do we include all concrete states in the end?
- Termination: Converge in a finite number of steps
- Abstract interpretation: A systematic way to build correct and terminating analyses

Correctness and Termination



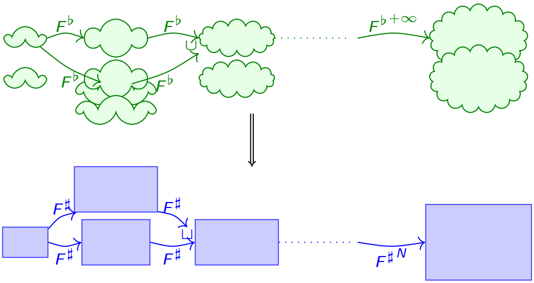
- Abstract transfer functions $F^\sharp$
- Merge abstract states for nodes with multiple predecessors
- Correction: Do we include all concrete states in the end?
- Termination: Converge in a finite number of steps
- Abstract interpretation: A systematic way to build correct and terminating analyses

Correctness and Termination



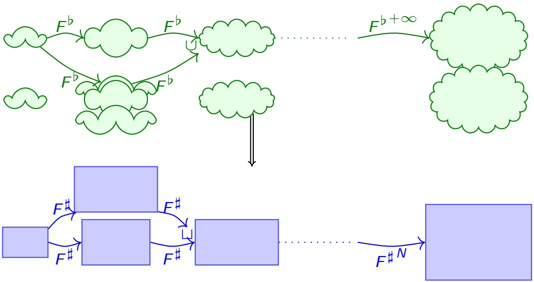
- Abstract transfer functions $F^\sharp$
- Merge abstract states for nodes with multiple predecessors
- Correction: Do we include all concrete states in the end?
- Termination: Converge in a finite number of steps
- Abstract interpretation: A systematic way to build correct and terminating analyses

Correctness and Termination



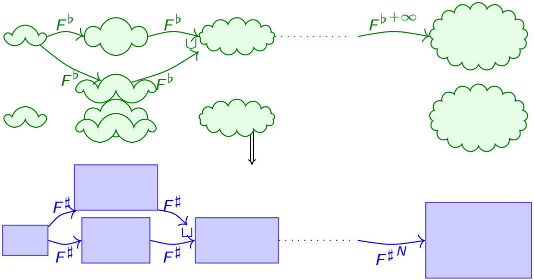
- Abstract transfer functions $F^\sharp$
- Merge abstract states for nodes with multiple predecessors
- Correction: Do we include all concrete states in the end?
- Termination: Converge in a finite number of steps
- Abstract interpretation: A systematic way to build correct and terminating analyses

Correctness and Termination



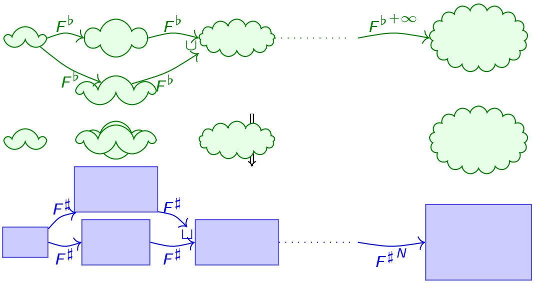
- Abstract transfer functions $F^\sharp$
- Merge abstract states for nodes with multiple predecessors
- Correction: Do we include all concrete states in the end?
- Termination: Converge in a finite number of steps
- Abstract interpretation: A systematic way to build correct and terminating analyses

Correctness and Termination



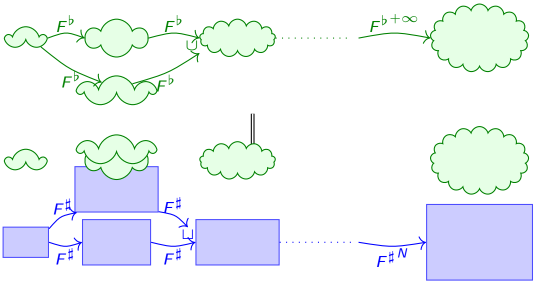
- Abstract transfer functions $F^\sharp$
- Merge abstract states for nodes with multiple predecessors
- Correction: Do we include all concrete states in the end?
- Termination: Converge in a finite number of steps
- Abstract interpretation: A systematic way to build correct and terminating analyses

Correctness and Termination



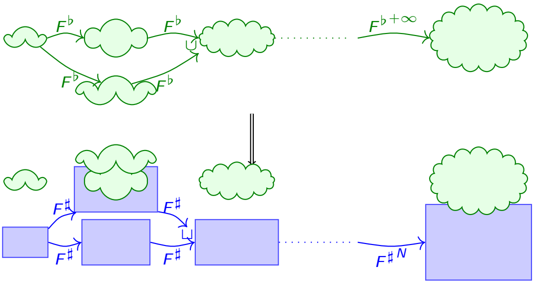
- Abstract transfer functions $F^\sharp$
- Merge abstract states for nodes with multiple predecessors
- Correction: Do we include all concrete states in the end?
- Termination: Converge in a finite number of steps
- Abstract interpretation: A systematic way to build correct and terminating analyses

Correctness and Termination



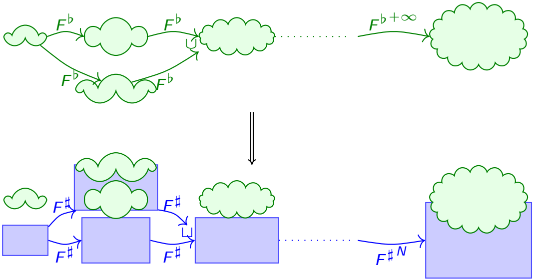
- Abstract transfer functions $F^\sharp$
- Merge abstract states for nodes with multiple predecessors
- Correction: Do we include all concrete states in the end?
- Termination: Converge in a finite number of steps
- Abstract interpretation: A systematic way to build correct and terminating analyses

Correctness and Termination



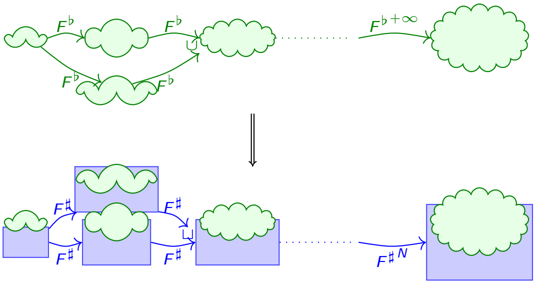
- Abstract transfer functions $F^\sharp$
- Merge abstract states for nodes with multiple predecessors
- Correction: Do we include all concrete states in the end?
- Termination: Converge in a finite number of steps
- Abstract interpretation: A systematic way to build correct and terminating analyses

Correctness and Termination



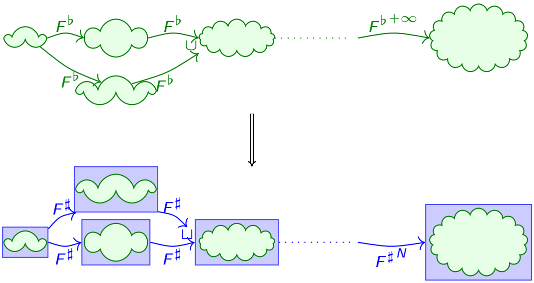
- Abstract transfer functions $F^\sharp$
- Merge abstract states for nodes with multiple predecessors
- Correction: Do we include all concrete states in the end?
- Termination: Converge in a finite number of steps
- Abstract interpretation: A systematic way to build correct and terminating analyses

Correctness and Termination



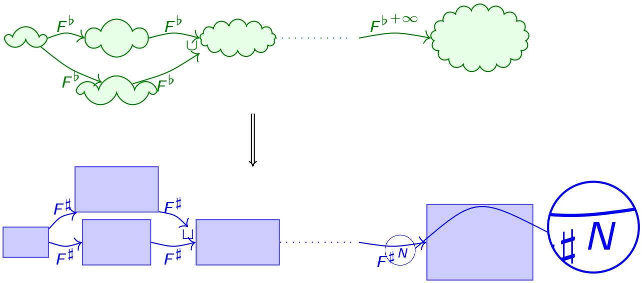
- Abstract transfer functions $F^\sharp$
- Merge abstract states for nodes with multiple predecessors
- Correction: Do we include all concrete states in the end?
- Termination: Converge in a finite number of steps
- Abstract interpretation: A systematic way to build correct and terminating analyses

Correctness and Termination



- Abstract transfer functions $F^\sharp$
- 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

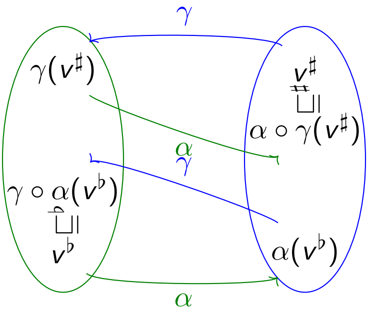
Correctness and Termination



- Abstract transfer functions $F^\sharp$
- 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

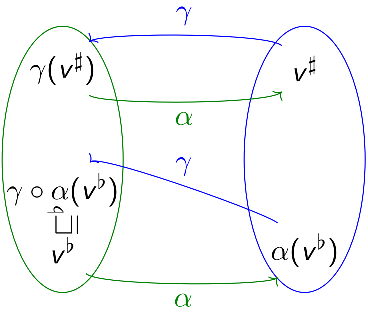
Galois connection and insertion

- α returns an abstraction from a set of concrete states
- γ returns the set of concrete states corresponding to an abstraction
- Following properties must hold:
  1. α and γ are monotonic
  2. $\forall v^\flat \in L^\flat$, $v^\flat \sqsubseteq^\flat (\gamma \circ \alpha)(v^\flat)$
  3. $\forall v^\sharp \in L^\sharp$, $(\alpha \circ \gamma)(v^\sharp) \sqsubseteq v^\sharp$
- Theorem [Cousot]: If $F^\sharp \sqsupseteq \alpha \circ F^\flat \circ \gamma$, abstraction is correct.



Galois connection and insertion

- α returns an abstraction from a set of concrete states
- γ returns the set of concrete states corresponding to an abstraction
- Following properties must hold:
  1. α and γ are monotonic
  2. $\forall v^\flat \in L^\flat,\ v^\flat \sqsubseteq^\flat (\gamma \circ \alpha)(v^\flat)$
  3. $\forall v^\sharp \in L^\sharp,\ (\alpha \circ \gamma)(v^\sharp) = v^\sharp$
- Theorem [Cousot]: If $F^\sharp \sqsupseteq \alpha \circ F^\flat \circ \gamma$, abstraction is correct.



Relational and Non-relational Lattices

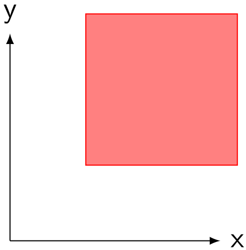
Non-relational domain

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

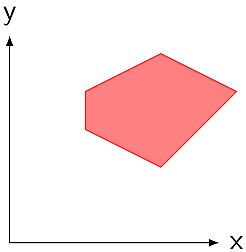
Relational domain

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

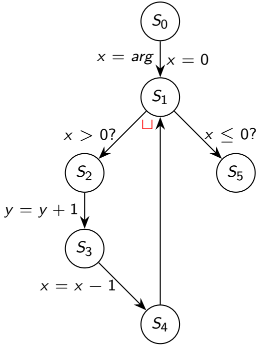
Example: intervals



Example: Polyhedra



Widening



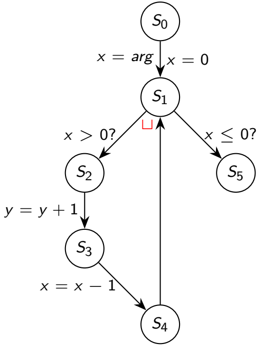
	S_1 (before)		S_4		S_1 (after)
$y \in$	$[0;0]$	$\sqcup$	$[1;1]$	$=$	$[0;1]$

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

correctness $x \sqcup y \sqsubseteq x \nabla y$

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

Widening



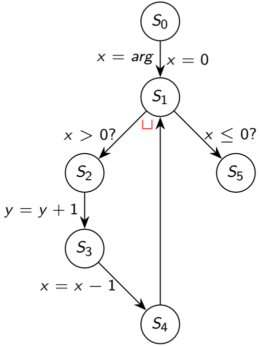
	S_1 (before)		S_4		S_1 (after)
$y \in$	$[0;1]$	$\sqcup$	$[1;2]$	$=$	$[0;2]$

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

correctness $x \sqcup y \sqsubseteq x \nabla y$

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

Widening



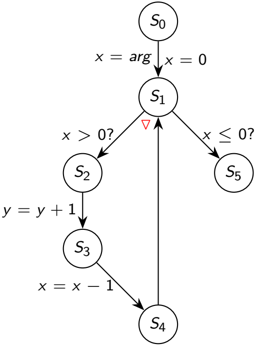
	S_1 (before)		S_4		S_1 (after)
$y \in$	$[0; 2]$	$\sqcup$	$[1; 3]$	$=$	$[0; 3]$

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

correctness $x \sqcup y \sqsubseteq x \nabla y$

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

Widening



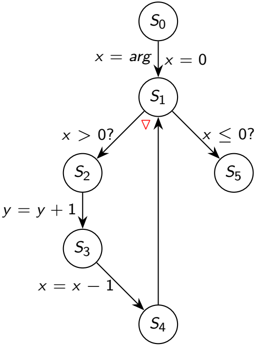
	S_1 (before)		S_4	S_1 (after)
$y \in$	$[0;2]$	$\triangledown$	$[1;3]$	$=[0;+\infty]$

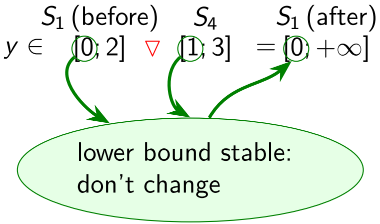
- for loop nodes, state grows slowly at each step
- convergence could require infinite time
- replace $\sqcup$ with widening operator $\triangledown$:

correctness $x \sqcup y \sqsubseteq x \triangledown y$

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

Widening

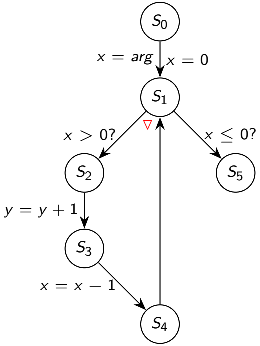




- for loop nodes, state grows slowly at each step
- convergence could require infinite time
- replace $\sqcup$ with widening operator $\bigtriangledown$:

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

Widening



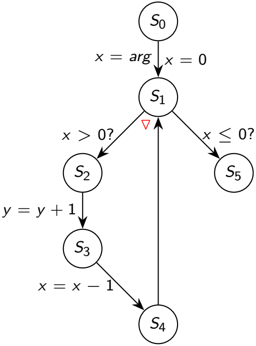
	S_1 (before)		S_4		S_1 (after)
$y \in$	$[0; 2]$	∇	$[1; 3]$	$=$	$[0; +\infty]$

upper bound grows:
widen interval

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

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

Widening



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

$y \in [0; +\infty] \nabla [1; +\infty] = [0; +\infty]$

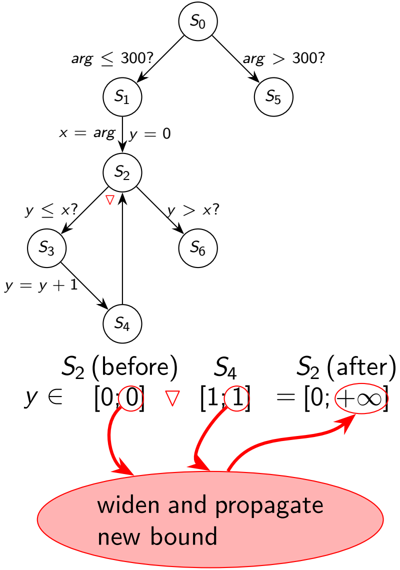
check fixpoint is reached

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

correctness $x \sqcup y \sqsubseteq x \nabla y$

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

Narrowing



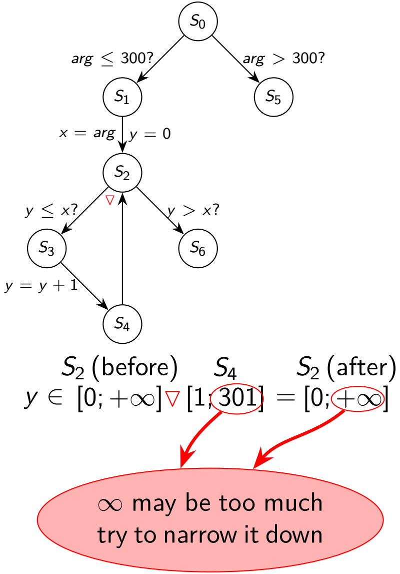
Recover some precision

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

correctness	$y \sqsubseteq (x \vartriangle y) \sqsubseteq x$
termination	no infinitely decreasing sequence

- In practice, very often better to directly improve widening

Narrowing



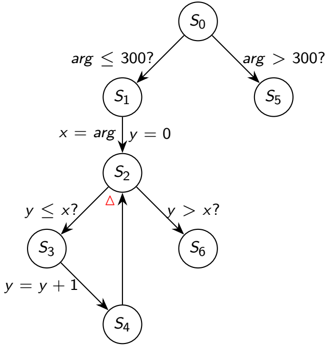
Recover some precision

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

correctness	$y \sqsubseteq (x \triangle y) \sqsubseteq x$
termination	no infinitely decreasing sequence

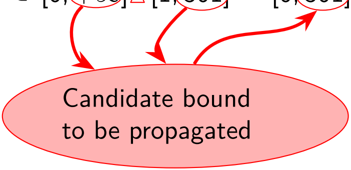
- In practice, very often better to directly improve widening

Narrowing



S_2 (before) S_4 S_2 (after)

$$y \in [0; +\infty] \triangle [1; 301] = [0; 301]$$



Recover some precision

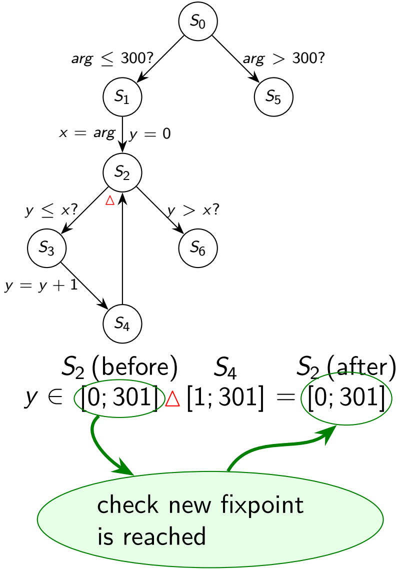
- Widening can be very coarse
- Use narrowing after reaching fixpoint:

correctness $y \sqsubseteq (x \triangle y) \sqsubseteq x$

termination no infinitely decreasing sequence

- In practice, very often better to directly improve widening

Narrowing



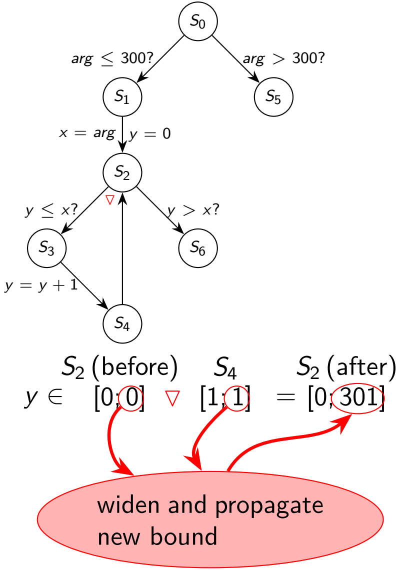
Recover some precision

- 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

Narrowing



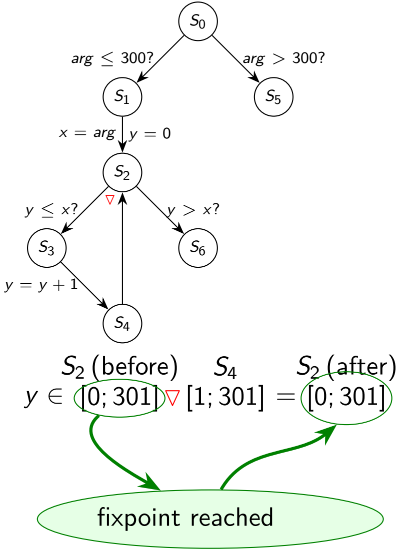
Recover some precision

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

correctness	$y \sqsubseteq (x \vartriangle y) \sqsubseteq x$
termination	no infinitely decreasing sequence

- In practice, very often better to directly improve widening

Narrowing



Recover some precision

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

correctness $y \sqsubseteq (x \vartriangle y) \sqsubseteq x$

termination no infinitely decreasing sequence

- In practice, very often better to directly improve widening

Reduced product

Question

We have information from two domains:
Intervals:

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

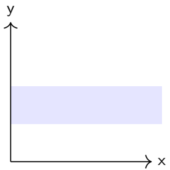
Octagons:
$0 \leq x - y \leq 20$

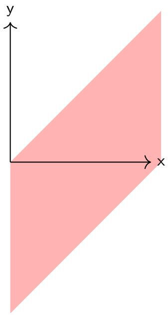
What can be said about x and y?

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

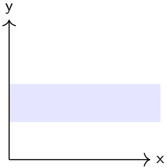
Reduced product

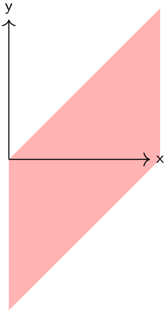




- Combining abstract domains
- reduce abstract value from one domain using information from the other
- ✘ In practice, not as simple and generic as it looks
- ✘ Combining transfer function is complex

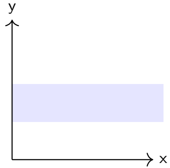
Reduced product

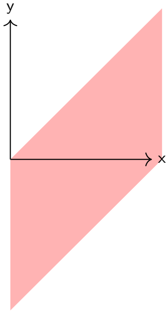




- Combining abstract domains
- reduce abstract value from one domain using information from the other
- ✘ In practice, not as simple and generic as it looks
- ✘ Combining transfer function is complex

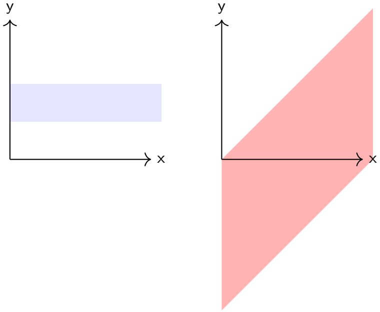
Reduced product





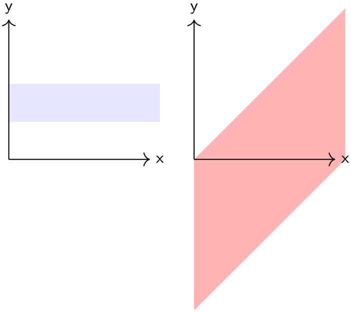
- Combining abstract domains
- reduce abstract value from one domain using information from the other
- ✘ In practice, not as simple and generic as it looks
- ✘ Combining transfer function is complex

Reduced product



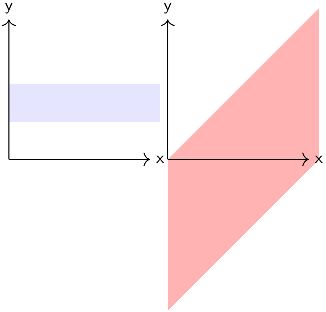
- Combining abstract domains
- reduce abstract value from one domain using information from the other
- ✘ In practice, not as simple and generic as it looks
- ✘ Combining transfer function is complex

Reduced product



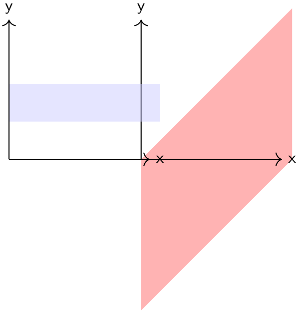
- Combining abstract domains
- reduce abstract value from one domain using information from the other
- ✘ In practice, not as simple and generic as it looks
- ✘ Combining transfer function is complex

Reduced product



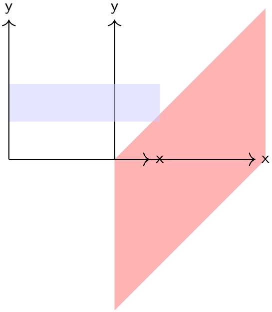
- Combining abstract domains
- reduce abstract value from one domain using information from the other
- ✘ In practice, not as simple and generic as it looks
- ✘ Combining transfer function is complex

Reduced product



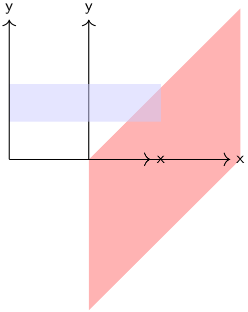
- Combining abstract domains
- reduce abstract value from one domain using information from the other
- ✘ In practice, not as simple and generic as it looks
- ✘ Combining transfer function is complex

Reduced product



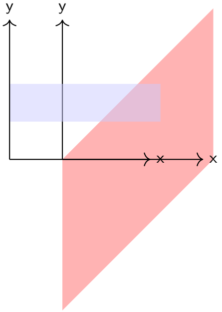
- Combining abstract domains
- reduce abstract value from one domain using information from the other
- ✘ In practice, not as simple and generic as it looks
- ✘ Combining transfer function is complex

Reduced product



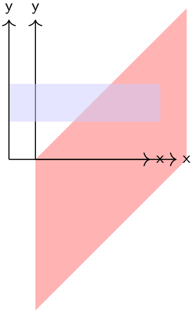
- Combining abstract domains
- reduce abstract value from one domain using information from the other
- ✘ In practice, not as simple and generic as it looks
- ✘ Combining transfer function is complex

Reduced product



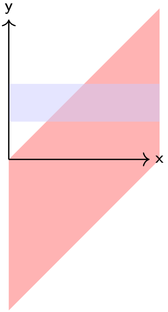
- Combining abstract domains
- reduce abstract value from one domain using information from the other
- ✘ In practice, not as simple and generic as it looks
- ✘ Combining transfer function is complex

Reduced product



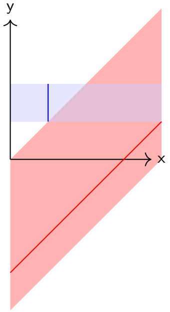
- Combining abstract domains
- reduce abstract value from one domain using information from the other
- ✘ In practice, not as simple and generic as it looks
- ✘ Combining transfer function is complex

Reduced product



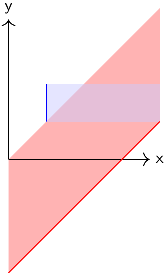
- Combining abstract domains
- reduce abstract value from one domain using information from the other
- ✘ In practice, not as simple and generic as it looks
- ✘ Combining transfer function is complex

Reduced product



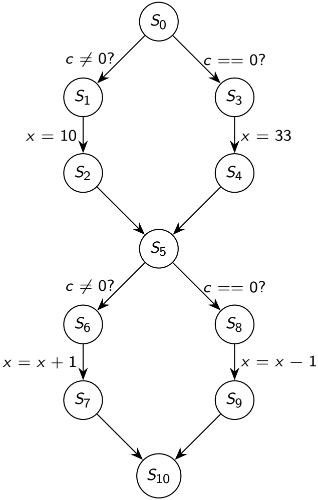
- Combining abstract domains
- reduce abstract value from one domain using information from the other
- ✘ In practice, not as simple and generic as it looks
- ✘ Combining transfer function is complex

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

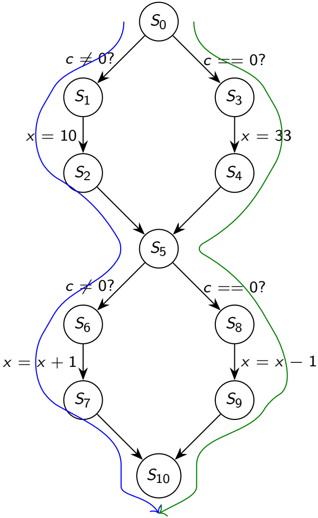
Trace Partitioning



	S_5	S_6	S_8	S_{10}
c	$\mathbb{Z}$	$\mathbb{Z}$	0	$\mathbb{Z}$
x	$[10; 33]$	$[10; 33]$	$[10; 33]$	$[9; 34]$

- Consider several abstract traces separately...
- ...At least for some time
- ✔ More precise than collecting semantics
- ✘ Finding appropriate partition is difficult

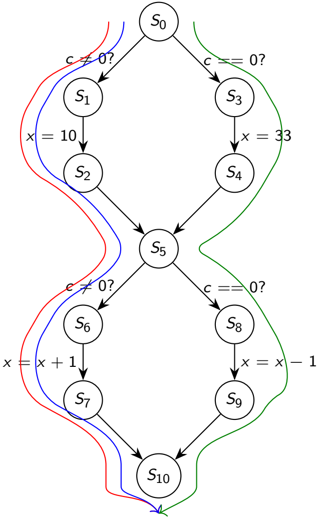
Trace Partitioning



	S_5	S_6	S_8	S_{10}
c	0	$\bot$	0	0
x	33	$\bot$	33	32
c	$\mathbb{Z}$	$\mathbb{Z}$	0	$\mathbb{Z}$
x	10	10	10	$[9; 11]$

- Consider several abstract traces separately...
- ...At least for some time
- ✔ More precise than collecting semantics
- ✘ Finding appropriate partition is difficult

Trace Partitioning



	S_5	S_6	S_8	S_{10}
c	0	$\bot$	0	0
x	33	$\bot$	33	32
c	$[1; +\infty]$	$[1; +\infty]$	$\bot$	$[1; +\infty]$
x	10	10	$\bot$	11
c	$[-\infty; -1]$	$[-\infty; -1]$	$\bot$	$[-\infty; -1]$
x	10	10	$\bot$	11

- Consider several abstract traces separately...
- ...At least for some time
- ✔ More precise than collecting semantics
- ✘ Finding appropriate partition is difficult

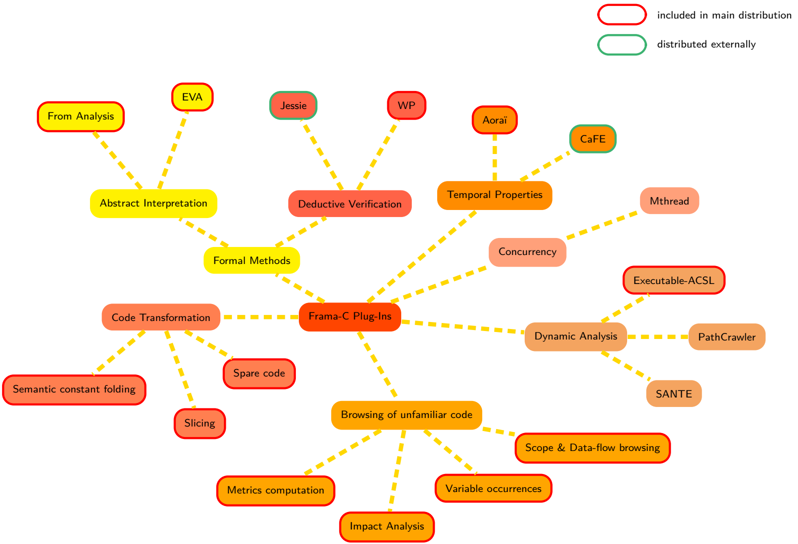
A few tools

- Polyspace Verifier: check absence of runtime errors (C/C++/Ada) https://fr.mathworks.com/products/polyspace.html
- ASTRÉE: absence of runtime errors without false alarm in SCADE-generated code https://www.absint.com/astree/index.htm
- Verasco: certified (in Coq) analyzer http://compcert.inria.fr/verasco/
- aiT/StackAnalyzer: WCET and stack size (assembly code) https://www.absint.com/ait/
- FLUCTUAT: accuracy of floating-point computations and origin of rounding errors http://www.lix.polytechnique.fr/~putot/fluctuat.html
- Frama-C: platform for analyzing C code, including through abstract interpretation https://frama-c.com

Frama-C at a glance

- A Framework for modular analysis of C code.
- `http://frama-c.com/`
- 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

Some plugins



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
```

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.

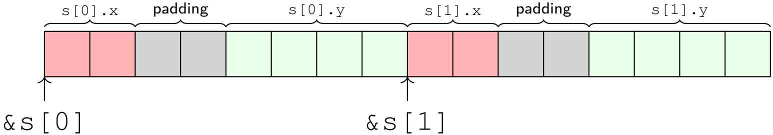
Integer Arithmetic in ACSL

- All operations are done over $\mathbb{Z}$: no overflow
- ACSL predicate `INT_MIN <= x + y <= INT_MAX`
  $\Leftrightarrow$
  C operation `x+y` does not overflow (undefined behavior)
- `(int)z` $\equiv$ `z` mod $2^{8*\texttt{sizeof(int)}}$
- and `INT_MIN <= (int)z <= INT_MAX`

Floating-point and Real Arithmetic

- Operations over $\mathbb{R}$: infinite precision
- **\round_double**(r,**\NearestEven**) to explicitly choose rounding mode
- predicates **\is_finite**(d), **\is_plus_infinity**(d), **\is_NaN**(d), ...
- function **\exact**(x): the value that C variable x would have if all computations had been done using $\mathbb{R}$. **\round_error** is the distance between x and **\exact**(x)
- typical specification:
  \round_error(**\result**) <= acceptable_limit

Memory description in ACSL




```
struct S {
short x;
int y;
} s[2];
```

```
\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])
```

ACSL Quiz

Question

If we have `\valid(p+(0 .. 2))`, with `p` a pointer to `int`, and `sizeof(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`

Back to presentation | See solution | Skip quiz

Function Contract

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

    ensures E(\result,x);

    behavior extra:
      assumes A(x);
      ensures more_result(\result,x);
*/
int f(int x);
```

Function Contract

What is required from caller

```
/*@ 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

Function Contract

```
/*@ requi
    ensures E(\result,
    behavior extra:
      assumes A(x);
      ensures more_result(\result,x);
*/
int f(int x);
```

Possible to distinguish various cases

Function contract quiz

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)`

ACSL Assertions

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

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

ACSL Assertions

Assess a property at given point

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

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

ACSL Assertions

Assess a property at given point

```
/*@ 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

EVA plugin

Credits

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

More information

- `http://frama-c.com/download/frama-c-value-analysis.pdf`

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 ≤ 8)
$\uplus$ integer interval $\times$ modulo information
$\uplus$ 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)

next

Code sample

```
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; }
}
```

EVA Result

```
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]
```

Integers in EVA Quiz

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]
- c [2 .. 14]
- d [3 .. 14]

Memory Address

Base Address

$$\begin{aligned} & \text{Global variable} \\ \uplus\ & \text{Formal parameter of main function} \\ \uplus\ & \text{literal string constant} \\ \uplus\ & \text{NULL} \\ \uplus\ & \ldots \end{aligned}$$

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

Code Sample

```
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);
}
```

EVA Result

```
[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}) }}
```

Write to an Address

Abstract Domain

written address = valid left value

$$\begin{aligned} & \text{adress} \\ \times\ & \text{initialized?} \\ \times\ & \textit{not dangling pointer}\text{?} \end{aligned}$$

Example

```
   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

next

Code Sample

```
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;
}
```

EVA Result

```
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}
```

Memory in EVA Quiz

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:

```
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}`

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

```
#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 `-slevel`: allows EVA to explore *n* separated paths before joining them
- option `-slevel-function`: same as previous, but for a particular function
- annotation `loop pragma UNROLL`: syntactic loop unrolling
- annotation `loop pragma WIDEN HINTS`: give bounds for widening

For specialists only

- option `-ilevel`: maximum number of elements in the set before conversion into intervals
- option `-plevel`: maximum number of distinct array cells

Driving Value through Annotations

- 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

Loop example

```c
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;
}
```

Code Sample

```
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--; }
}
```

EVA Result

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

```
/*@ assert c<=0 || c > 0; */

[value] Assertion got status valid.
  x IN {9; 34}
  y IN {37; 41}
```

Entry Point

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

```
void f_wrapper() {
  setup_analysis_context();
  f(arg_1, arg_2);
}
```

Example

```
int search(char* a, char key) {
  char* orig = a;
  while (*a) {
    if (*a == key) return a - orig;
    a++;
  }
  return -1;
}
```

Results without context

```
frama-c -val -context-width 3 -main search context.c

[...]

context.c:3:[kernel] warning: out of bounds read. ass
context.c:4:[kernel] warning: out of bounds read. ass
context.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}
```

With Context

```
#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 [--..--]
```

External Library Functions

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-builtins-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

next

CERT ARR30-C bad code sample

```
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;
}
```

Analyzing real code

- D. Delmas and J. Souyris: ASTRÉE: from Research to Industry, SAS 2007
- TrustInSoft startup (created 2013): `https://trust-in-soft.com/`
- A. Ourghanlian: Evaluation of static analysis tools used to assess software important to nuclear power plant safety. In Nuclear Engineering and Technology, vol 47 issue 2, 2015.
- INGOPCS project: `https://www.ingopcs.net`
- Open-Source Case Studies:

  `https://github.com/Frama-C/open-source-case-studies`
- A. Maroneze: Analysis of the Chrony NTP server.

  `http://blog.frama-c.com/index.php?post/2018/06/19/Analyzing-Chrony-with-Frama-C/Eva`

General

- Correnson &al. Frama-C User Manual (v17 - Chlorine). May 2018
- Kirchner &al. Frama-C, a Software Analysis Perspective, vol 37 of Formal Aspects of Computing, March 2015.

ACSL

- Baudin &al. ACSL: ANSI/ISO C Specification Language. Preliminary Design (v 1.13). May 2018
- 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

- Hanne Nielson, Flemming Nielson, and Chris Hankin. *Principles of Program Analysis*. Springer 1999
- Neil Jones and Flemming Nielson, *Abstract Interpretation: a Semantics-Based Tool for Program Analysis*. In *Handbook of Logic in Computer Science, vol. 4*, Oxford University Press 1994

Static Analysis (cont'd)

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
- `http://www.di.ens.fr/~cousot/COUSOTpapers.shtml`

Solutions to Quizzes

Reduced product

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

Back to presentation

See solution

Reduced product

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

Back to presentation
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 $x \in [5; 20]$, $y \in [0; 20]$, $0 \leq x - y \leq 20$

Back to presentation
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 $x \in [5;20]$, $y \in [0;20]$, $0 \leq x - y \leq 20$ ✘

Back to presentation

See solution

ACSL Quiz

Question

If we have `\valid(p+(0 .. 2))`, with `p` a pointer to `int`, and `sizeof(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`

Back to presentation
See solution

ACSL Quiz

Question

If we have `\valid(p+(0 .. 2))`, with `p` a pointer to `int`, and `sizeof(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`

Back to presentation

See solution

ACSL Quiz

Question

If we have `\valid(p+(0 .. 2))`, with `p` a pointer to `int`, and `sizeof(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`

Back to presentation

See solution

ACSL Quiz

Question

If we have `\valid(p+(0 .. 2))`, with `p` a pointer to `int`, and `sizeof(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`

Back to presentation See solution

ACSL Quiz

Question

If we have `\valid(p+(0 .. 2))`, with `p` a pointer to `int`, and `sizeof(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` ✔

Back to presentation

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 && \valid(s+ (0 .. acsl_strlen(s)))`
- c `\valid(s + (0 .. acsl_strlen(s)))`
- d `acsl_strlen(s) >= 0 && \valid(s)`

Back to presentation
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 && \valid(s+ (0 .. acsl_strlen(s)))` ✔
- c `\valid(s + (0 .. acsl_strlen(s)))`
- d `acsl_strlen(s) >= 0 && \valid(s)`

Back to presentation

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 && \valid(s+ (0 .. acsl_strlen(s)))`
- c `\valid(s + (0 .. acsl_strlen(s)))` ✘
- d `acsl_strlen(s) >= 0 && \valid(s)`

Back to presentation

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 && \valid(s+ (0 .. acsl_strlen(s)))`
- c `\valid(s + (0 .. acsl_strlen(s)))`
- d `acsl_strlen(s) >= 0 && \valid(s)` ✗

Back to presentation
See solution

Integers in EVA Quiz

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]`
- c `[2 .. 14]`
- d `[3 .. 14]`

Back to presentation
See solution

Integers in EVA Quiz

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] ✘
- c [2 .. 14]
- d [3 .. 14]

Back to presentation   See solution

Integers in EVA Quiz

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]`
- c `[2 .. 14]` ✔
- d `[3 .. 14]`

Back to presentation

See solution

Integers in EVA Quiz

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]`
- c `[2 .. 14]`
- d `[3 .. 14]` ✗

Back to presentation

See solution

Memory in EVA Quiz

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:

```
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}`

Memory in EVA Quiz

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:

```
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}`

Back to presentation
See solution

Memory in EVA Quiz

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:

```
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}`

Memory in EVA Quiz

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:

```
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}` ✗

Back to presentation

See solution