Structuring an Abstract Interpreter through Value and State Abstractions:

EVA, an Evolved Value Analysis for Frama–C

David Bühler

CEA LIST, Software Safety Lab

Frama-C & SPARK Day 2017 — May 30th, 2017

VALUE Analysis

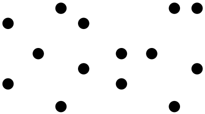
- An historic plugin of Frama-C.
- *Abstract interpretation* based analysis.
- Handles the subset of C99 used in embedded softwares.
- Emits alarms at potentially unsafe program points.

Abstract Interpretation [CC77]

- A general theory for the approximation of program semantics.
- A practical framework for automatic analyzes.
- Links a very precise but generally non computable semantics to relaxed *abstract* semantics through *abstract domains*.

Abstract Domains

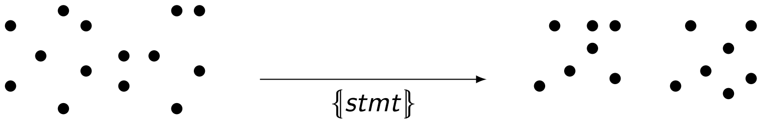
- Abstract domains $\mathbb{D}$ over-approximate the behaviors of a program.



- A set of discrete concrete states in $\mathcal{S}$ represents the possible behaviors at a program point.

Abstract Domains

- Abstract domains $\mathbb{D}$ over-approximate the behaviors of a program.

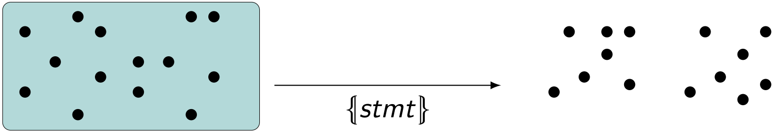


- A concrete semantics characterizes the effects of a statement as a function over states.

$$[\![\texttt{stmt}]\!] : \boldsymbol{\mathcal{S}} \rightarrow \boldsymbol{\mathcal{S}}$$

Abstract Domains

- Abstract domains $\mathbb{D}$ over-approximate the behaviors of a program.

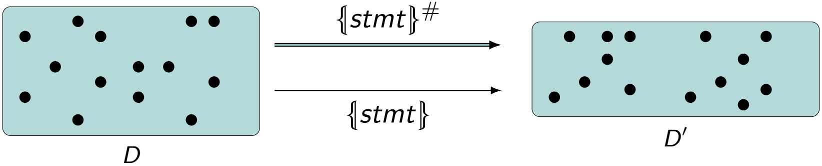


- An abstract domain $\mathbb{D}$ represents sets of concrete states.

$$\gamma : \mathbb{D} \to \mathcal{P}(\boldsymbol{\mathcal{S}})$$

Abstract Domains

- Abstract domains $\mathbb{D}$ over-approximate the behaviors of a program.

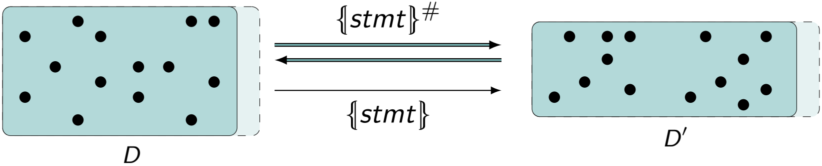


- Sound abstract semantics through transfer functions $\{\!| \cdot |\!\}^{\#} : \mathbb{D} \to \mathbb{D}$.

$$\{\!|\texttt{stmt}|\!\}(\gamma(D)) \subseteq \gamma(\{\!|\texttt{stmt}|\!\}^{\#}(D))$$

Abstract Domains

- Abstract domains $\mathbb{D}$ over-approximate the behaviors of a program.

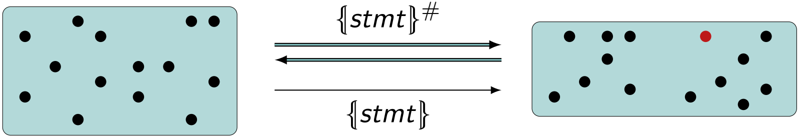


- Sound backward semantics $\overleftarrow{\llbracket \cdot \rrbracket}^{\#} : \mathbb{D} \rightarrow \mathbb{D}$.

$$\{S \mid \llbracket \texttt{stmt} \rrbracket(S) \in \gamma(D')\} \subseteq \gamma(\overleftarrow{\llbracket \texttt{stmt} \rrbracket}^{\#}(D'))$$

Abstract Domains

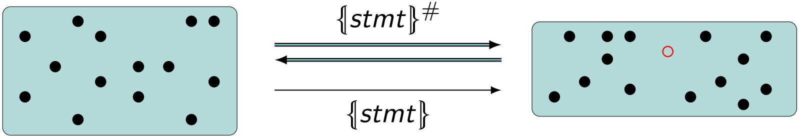
- Abstract domains $\mathbb{D}$ over-approximate the behaviors of a program.



- Report illegal operations
  (divisions by zero, integer overflows, invalid accesses to memory. . .)
- All incorrect programs are detected.

Abstract Domains

- Abstract domains $\mathbb{D}$ over-approximate the behaviors of a program.

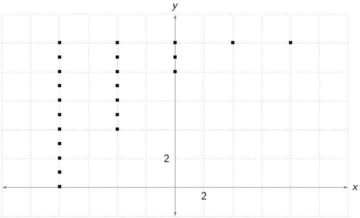


- But over-approximations may lead to false alarms.
- May fail to verify correct programs.

Known Abstract Domains

Numerous abstractions are already well-known in the literature:

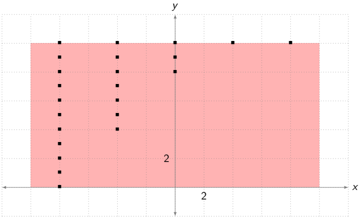
- Intervals: $x \in [-10..10]$ $y \in [0..10]$
- Congruences: $x \equiv 0[4]$
- Octagons: $x - y \leq 0$ $-8 \leq x \leq 8$ $0 \leq y \leq 10$
- And many more. . .



Known Abstract Domains

Numerous abstractions are already well-known in the literature:

- Intervals: $x \in [-10..10]$
  $y \in [0..10]$

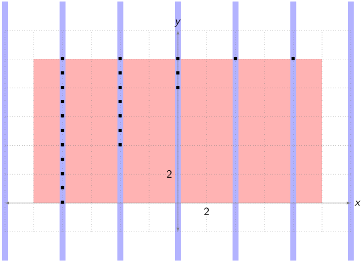


- Congruences: $x \equiv 0[4]$
- Octagons: $x - y \leq 0$
  $-8 \leq x \leq 8$
  $0 \leq y \leq 10$
- And many more...

Known Abstract Domains

Numerous abstractions are already well-known in the literature:

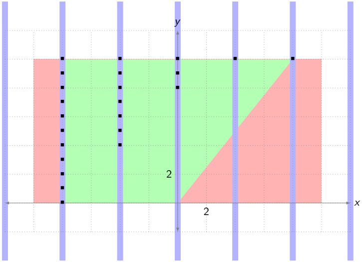
- Intervals: $x \in [-10..10]$ $y \in [0..10]$
- Congruences: $x \equiv 0[4]$
- Octagons: $x - y \leq 0$ $-8 \leq x \leq 8$ $0 \leq y \leq 10$
- And many more...



Known Abstract Domains

Numerous abstractions are already well-known in the literature:

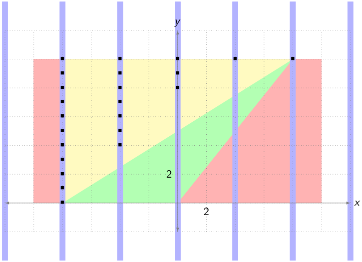
- Intervals: $x \in [-10..10]$ $y \in [0..10]$
- Congruences: $x \equiv 0[4]$
- Octagons: $x - y \leq 0$ $-8 \leq x \leq 8$ $0 \leq y \leq 10$
- And many more...



Known Abstract Domains

Numerous abstractions are already well-known in the literature:

- Intervals: $x \in [-10..10]$ $y \in [0..10]$
- Congruences: $x \equiv 0[4]$
- Octagons: $x - y \leq 0$ $-8 \leq x \leq 8$ $0 \leq y \leq 10$
- And many more...



In the VALUE Analysis plugin

- One rich abstract domain based on intervals and congruences.
- Strongly optimized to achieve scalability.
- However:
  - Not extensible.
  - Lack of relational properties.
- Limitations overcomed in EVA, the Evolved Value Analysis.

Combination of Abstract Domains

How to combine abstract domains?

How to let them interact?

Combination of Abstract Domains

How to combine abstract domains?

How to let them interact?

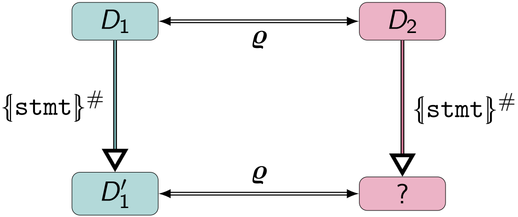
Combination of Abstract Domains

How to combine abstract domains?

How to let them interact ***in a modular way****?*

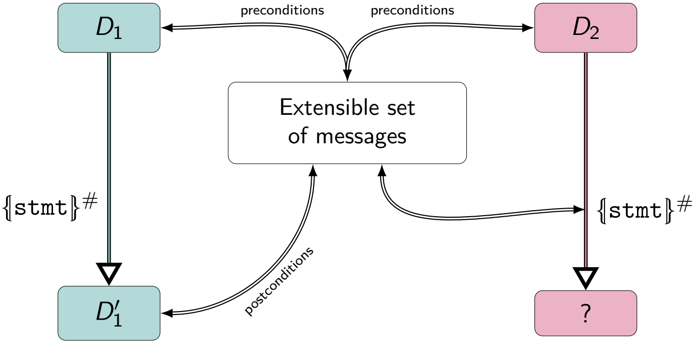
Reduced Product [CC79]

- Direct inter-reduction between two domains: $\varrho : \mathbb{D}_1 \times \mathbb{D}_2 \to \mathbb{D}_1 \times \mathbb{D}_2$



- Highly non-modular.
- No interaction during the interpretation of statements.

Communication by Messages [Cou+06]

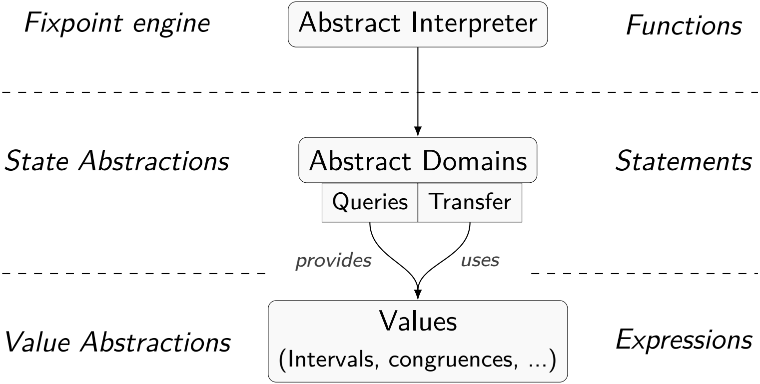


- Domains can send or request information.
- Communication system in parallel of the abstract semantics.

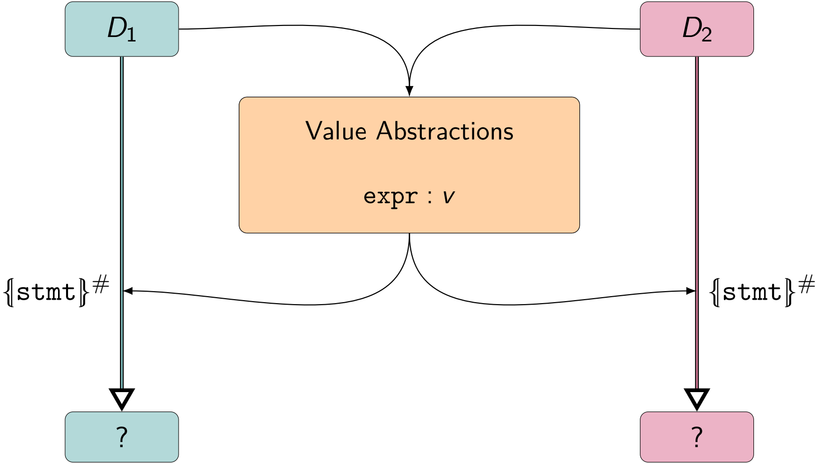
Our Proposal

Structuring the abstract semantics following the distinction between expressions and statements.

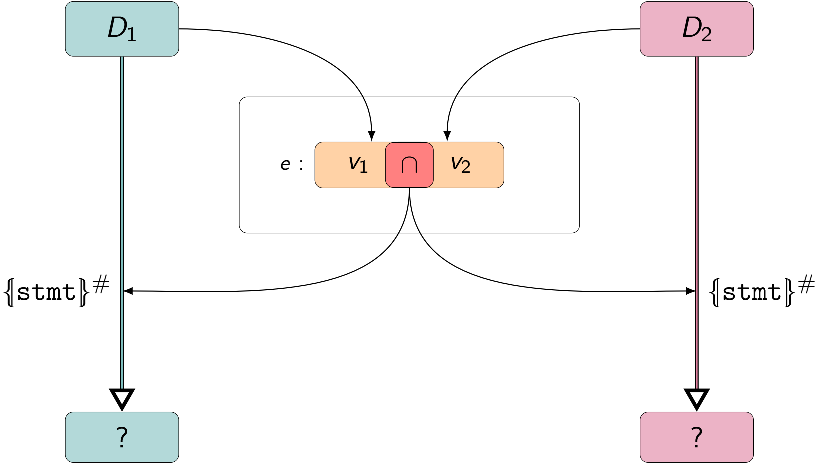
Core Concepts



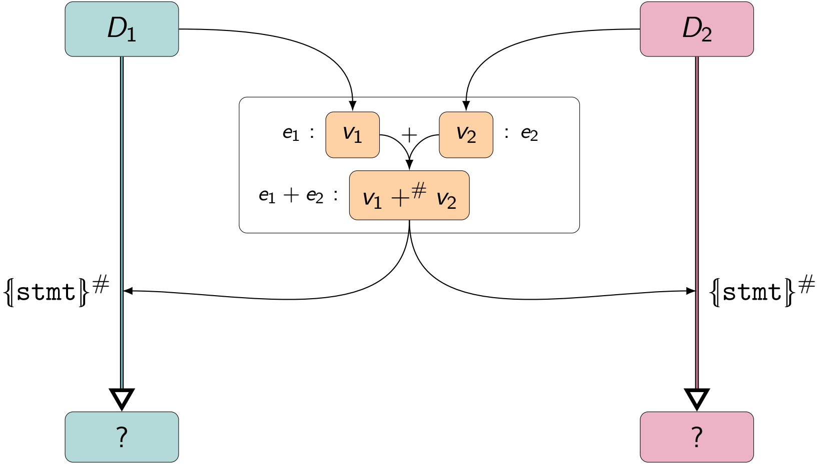
Interactions between Abstractions



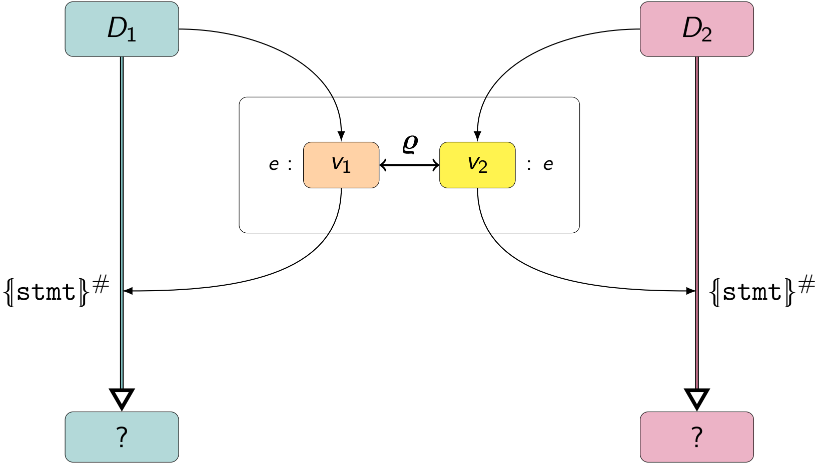
Interactions between Abstractions



Interactions between Abstractions



Interactions between Abstractions



Value Abstractions $\mathbb{V}^\#$

- Represent sets of scalar values:

$$\gamma_{\mathbb{V}} : \mathbb{V}^\# \to \mathcal{P}(\mathbb{V})$$

$\mathbb{V}$ is the set of the scalar values in C, including integer and pointers.

- Used in this talk:
  - For integers: intervals

$$[x..y]$$

  - For pointers: maps from variables to intervals

$$\{\{\&x \mapsto [i..j]\}\}$$

Value Abstractions $\mathbb{V}^\#$

- Represent sets of scalar values:

$$\gamma_{\mathbb{V}} : \mathbb{V}^\# \rightarrow \mathcal{P}(\mathbb{V})$$

  $\mathbb{V}$ is the set of the scalar values in C, including integer and pointers.

- Used in this talk:
  - For integers: intervals

$$[x..y]$$

  - For pointers: maps from variables to intervals

$$\{\{\&x \mapsto [i..j]\}\}$$

Operations of the Value Abstractions $\mathbb{V}^{\#}$

- Meet $\sqcap_{\mathbb{V}}$ operation, over-approximating the intersection of sets:

$$\gamma_{\mathbb{V}}(v_1) \cap \gamma_{\mathbb{V}}(v_2) \subseteq \gamma_{\mathbb{V}}(v_1 \sqcap_{\mathbb{V}} v_2)$$

- For each C operator $\diamondsuit$ on expressions, a *sound* abstract operator $\diamondsuit^{\#}$ on value abstractions:

$$\forall v_1 \in \gamma_{\mathbb{V}}(v_1^{\#}),\ \forall v_2 \in \gamma_{\mathbb{V}}(v_2^{\#}), \quad v_1 \diamondsuit v_2 \in \gamma_{\mathbb{V}}(v_1^{\#} \diamondsuit^{\#} v_2^{\#})$$

$$\diamondsuit \in \{+, -, \leq, \dots\}$$

- Injection of scalar values: $v \in \gamma_{\mathbb{V}}(v^{\#})$

State Abstractions $\mathbb{D}$

- Represent sets of concrete states:

$$\gamma_{\mathbb{D}} : \mathbb{D} \to \mathcal{P}(\boldsymbol{\mathcal{S}})$$

$\boldsymbol{\mathcal{S}}$ represents all possible behaviors when executing a program.

- Provide a *sound* abstract semantics $*^{\#}$ of dereferences:

$$*^{\#} : \mathbb{D} \to \mathbb{V}^{\#} \to \mathbb{V}^{\#}$$

- Product of semantics:

$$*^{\#}(D_1 \times D_2, v) = *^{\#}(D_1, v) \sqcap_{\mathbb{V}} *^{\#}(D_2, v)$$

Cooperative Evaluation: First Example

```
int t[5] = {1,2,3,4,5};
if (0 <= i && i < 3)
  x = t[i+1];
```

Array domain D_A

$t : \{1, 2, 3, 4, 5\}$

Interval domain D_I

$\begin{cases} i \in [0..2] \\ x \in \top \end{cases}$

- Evaluation of $t[i+1]$:

$$
\begin{aligned}
i &\rightharpoonup *^{\#}(D_A \times D_I, \{\{\&i \mapsto [0]\}\}) = \top_{\mathbb{V}} \sqcap_{\mathbb{V}} [0..2] = [0..2] \\
i+1 &\rightharpoonup [0..2] +^{\#} [1] = [1..3] \\
\&t[i+1] &\rightharpoonup \{\{\&t \mapsto [0]\}\} +^{\#} [1..3] = \{\{\&t \mapsto [1..3]\}\} \\
t[i+1] &\rightharpoonup *^{\#}(D_A \times D_I, \{\{\&t \mapsto [1..3]\}\}) = [2..4] \sqcap_{\mathbb{V}} \top_{\mathbb{V}} = [2..4]
\end{aligned}
$$

Cooperative Evaluation: First Example

```
int t[5] = {1,2,3,4,5};
if (0 <= i && i < 3)
  x = t[i+1];
```

Array domain D_A

$t : \{1, 2, 3, 4, 5\}$

Interval domain D_I

$\begin{cases} i \in [0..2] \\ x \in \top \end{cases}$

- Evaluation of $t[i + 1]$:

$$i \rightharpoonup *^{\#}(D_A \times D_I, \{\{\&i \mapsto [0]\}\}) = \top_{\mathbb{V}} \sqcap_{\mathbb{V}} [0..2] = [0..2]$$

$$i + 1 \rightharpoonup [0..2] +^{\#} [1] = [1..3]$$

$$\&t[i + 1] \rightharpoonup \{\{\&t \mapsto [0]\}\} +^{\#} [1..3] = \{\{\&t \mapsto [1..3]\}\}$$

$$t[i + 1] \rightharpoonup *^{\#}(D_A \times D_I, \{\{\&t \mapsto [1..3]\}\}) = [2..4] \sqcap_{\mathbb{V}} \top_{\mathbb{V}} = [2..4]$$

Cooperative Evaluation: First Example

```
int t[5] = {1,2,3,4,5};
if (0 <= i && i < 3)
  x = t[i+1];
```

Array domain D_A

$t : \{1, 2, 3, 4, 5\}$

Interval domain D_I

$\begin{cases} i \in [0..2] \\ x \in \top \end{cases}$

- Evaluation of $t[i + 1]$:

$$
\begin{aligned}
i &\rightharpoonup *^{\#}(D_A \times D_I, \{\{\&i \mapsto [0]\}\}) = \top_{\mathbb{V}} \sqcap_{\mathbb{V}} [0..2] = [0..2] \\
i + 1 &\rightharpoonup [0..2] +^{\#} [1] = [1..3] \\
\&t[i + 1] &\rightharpoonup \{\{\&t \mapsto [0]\}\} +^{\#} [1..3] = \{\{\&t \mapsto [1..3]\}\} \\
t[i + 1] &\rightharpoonup *^{\#}(D_A \times D_I, \{\{\&t \mapsto [1..3]\}\}) = [2..4] \sqcap_{\mathbb{V}} \top_{\mathbb{V}} = [2..4]
\end{aligned}
$$

Cooperative Evaluation: First Example

```
int t[5] = {1,2,3,4,5};
if (0 <= i && i < 3)
  x = t[i+1];
```

Array domain D_A

$t : \{1, 2, 3, 4, 5\}$

Interval domain D_I

$$\begin{cases} i \in [0..2] \\ x \in \top \end{cases}$$

- Evaluation of $t[i+1]$:

$$i \rightarrow *^{\#}(D_A \times D_I, \{\{\&i \mapsto [0]\}\}) = \top_{\mathbb{V}} \sqcap_{\mathbb{V}} [0..2] = [0..2]$$

$$i+1 \rightarrow [0..2] +^{\#} [1] = [1..3]$$

$$\&t[i+1] \rightarrow \{\{\&t \mapsto [0]\}\} +^{\#} [1..3] = \{\{\&t \mapsto [1..3]\}\}$$

$$t[i+1] \rightarrow *^{\#}(D_A \times D_I, \{\{\&t \mapsto [1..3]\}\}) = [2..4] \sqcap_{\mathbb{V}} \top_{\mathbb{V}} = [2..4]$$

Cooperative Evaluation: First Example

```
int t[5] = {1,2,3,4,5};
if (0 <= i && i < 3)
  x = t[i+1];
```

Array domain D_A

$t : \{1, 2, 3, 4, 5\}$

Interval domain D_I

$\begin{cases} i \in [0..2] \\ x \in \top \end{cases}$

- Evaluation of $t[i+1]$:

$$
\begin{aligned}
i &\rightharpoondown *^{\#}(D_A \times D_I, \{\{\&i \mapsto [0]\}\}) = \top_{\mathbb{V}} \sqcap_{\mathbb{V}} [0..2] = [0..2] \\
i+1 &\rightharpoondown [0..2] +^{\#} [1] = [1..3] \\
\&t[i+1] &\rightharpoondown \{\{\&t \mapsto [0]\}\} +^{\#} [1..3] = \{\{\&t \mapsto [1..3]\}\} \\
t[i+1] &\rightharpoondown *^{\#}(D_A \times D_I, \{\{\&t \mapsto [1..3]\}\}) = [2..4] \sqcap_{\mathbb{V}} \top_{\mathbb{V}} = [2..4]
\end{aligned}
$$

Requesting Evaluations

- A domain may need further information to compute a precise value.
- A domain $\mathbb{D}$ can request the evaluation of an expression
  - The evaluation involves all available domains.
  - The result is then returned to $\mathbb{D}$.
- Information flows between domains through value abstractions.
- No direct interaction between domains.

Cooperative Evaluation: Second Example

```
int t[5] = {1,2,3,4,5};
int tmp = t[i+1];
if (0 <= i && i < 3)
  r = 2 * tmp;
```

Equalities D_E	Intervals D_I	Array D_A
$tmp = t[i+1]$	$\begin{cases} i \in [0..2] \\ tmp \in [1..5] \end{cases}$	$t : \{1,2,3,4,5\}$

$$\llbracket tmp \rrbracket^{\#}_{D_I} = [1..5]$$

$$\llbracket tmp \rrbracket^{\#}_{D_E} = \llbracket t[i+1] \rrbracket^{\#}_{D_E \times D_I \times D_A} = [2..4]$$

$$\llbracket tmp \rrbracket^{\#}_{D_E \times D_I \times D_A} = [2..4] \sqcap_{\mathbb{V}} [1..5] \sqcap_{\mathbb{V}} \top_{\mathbb{V}} = [2..4]$$

$$\llbracket 2 * tmp \rrbracket^{\#}_{D_E \times D_I \times D_A} = [4..8]$$

Cooperative Evaluation: Second Example

```
int t[5] = {1,2,3,4,5};
int tmp = t[i+1];
if (0 <= i && i < 3)
  r = 2 * tmp;
```

Equalities D_E	*Intervals D_I*	*Array D_A*
$tmp = t[i+1]$	$\begin{cases} i \in [0..2] \\ tmp \in [1..5] \end{cases}$	$t : \{1,2,3,4,5\}$

$$[\![tmp]\!]^{\#}_{D_I} = [1..5]$$

$$[\![tmp]\!]^{\#}_{D_E} = [\![t[i+1]]\!]^{\#}_{D_E \times D_I \times D_A} = [2..4]$$

$$[\![tmp]\!]^{\#}_{D_E \times D_I \times D_A} = [2..4] \sqcap_{\mathbb{V}} [1..5] \sqcap_{\mathbb{V}} \top_{\mathbb{V}} = [2..4]$$

$$[\![2 * tmp]\!]^{\#}_{D_E \times D_I \times D_A} = [4..8]$$

Cooperative Evaluation: Second Example

```
int t[5] = {1,2,3,4,5};
int tmp = t[i+1];
if (0 <= i && i < 3)
  r = 2 * tmp;
```

Equalities D_E	Intervals D_I	Array D_A
$tmp = t[i+1]$	$\begin{cases} i \in [0..2] \\ tmp \in [1..5] \end{cases}$	$t : \{1, 2, 3, 4, 5\}$

$$\llbracket tmp \rrbracket^{\#}_{D_I} = [1..5]$$

$$\llbracket tmp \rrbracket^{\#}_{D_E} = \llbracket t[i+1] \rrbracket^{\#}_{D_E \times D_I \times D_A} = [2..4]$$

$$\llbracket tmp \rrbracket^{\#}_{D_E \times D_I \times D_A} = [2..4] \sqcap_{\mathbb{V}} [1..5] \sqcap_{\mathbb{V}} \top_{\mathbb{V}} = [2..4]$$

$$\llbracket 2 * tmp \rrbracket^{\#}_{D_E \times D_I \times D_A} = [4..8]$$

Cooperative Evaluation: Second Example

```
int t[5] = {1,2,3,4,5};
int tmp = t[i+1];
if (0 <= i && i < 3)
  r = 2 * tmp;
```

Equalities D_E	*Intervals D_I*	*Array D_A*
$tmp = t[i+1]$	$\begin{cases} i \in [0..2] \\ tmp \in [1..5] \end{cases}$	$t : \{1,2,3,4,5\}$

$$
\begin{aligned}
[\![tmp]\!]^{\#}_{D_I} &= [1..5] \\
[\![tmp]\!]^{\#}_{D_E} &= [\![t[i+1]]\!]^{\#}_{D_E \times D_I \times D_A} = [2..4] \\
[\![tmp]\!]^{\#}_{D_E \times D_I \times D_A} &= [2..4] \sqcap_{\mathbb{V}} [1..5] \sqcap_{\mathbb{V}} \top_{\mathbb{V}} = [2..4] \\
[\![2 * tmp]\!]^{\#}_{D_E \times D_I \times D_A} &= [4..8]
\end{aligned}
$$

Cooperative Evaluation: Second Example

```
int t[5] = {1,2,3,4,5};
int tmp = t[i+1];
if (0 <= i && i < 3)
  r = 2 * tmp;
```

Equalities D_E	*Intervals D_I*	*Array D_A*
$tmp = t[i+1]$	$\begin{cases} i \in [0..2] \\ tmp \in [1..5] \end{cases}$	$t : \{1, 2, 3, 4, 5\}$

$$\begin{aligned}
[\![tmp]\!]^{\#}_{D_I} &= [1..5] \\
[\![tmp]\!]^{\#}_{D_E} &= [\![t[i+1]]\!]^{\#}_{D_E \times D_I \times D_A} = [2..4] \\
[\![tmp]\!]^{\#}_{D_E \times D_I \times D_A} &= [2..4] \sqcap_{\mathbb{V}} [1..5] \sqcap_{\mathbb{V}} \top_{\mathbb{V}} = [2..4] \\
[\![2 * tmp]\!]^{\#}_{D_E \times D_I \times D_A} &= [4..8]
\end{aligned}$$

Backward Propagation

- Value and state abstractions also implement backward counterparts to the forward abstract operators.
- Domains can trigger new backward propagations.

Cooperative Evaluation: Third Example

```
int t[5] = {1,2,3,4,5};
int tmp = t[i+1];
if (tmp < 3) ...
```

Equality domain D_E	*Interval domain D_I*	*Array domain D_A*
$tmp = t[i+1]$	$\begin{cases} i \in [-1..3] \\ tmp \in [1..5] \end{cases}$	$t : \{1, 2, 3, 4, 5\}$

- Backward propagation on $tmp < 3$:

tmp	is reduced to	$[1..2]$	by the value semantics
$t[i+1]$	is reduced to	$[1..2]$	by the equality domain
$i+1$	is reduced to	$[0..1]$	by the array domain
i	is reduced to	$[-1..0]$	by the value semantics

Cooperative Evaluation: Third Example

```
int t[5] = {1,2,3,4,5};
int tmp = t[i+1];
if (tmp < 3) ...
```

Equality domain D_E	*Interval domain* D_I	*Array domain* D_A
$tmp = t[i+1]$	$\begin{cases} i \in [-1..3] \\ tmp \in [1..5] \end{cases}$	$t : \{1,2,3,4,5\}$

- Backward propagation on $tmp < 3$:

tmp	is reduced to	$[1..2]$	by the value semantics
$t[i+1]$	is reduced to	$[1..2]$	by the equality domain
$i+1$	is reduced to	$[0..1]$	by the array domain
i	is reduced to	$[-1..0]$	by the value semantics

Cooperative Evaluation: Third Example

```
int t[5] = {1,2,3,4,5};
int tmp = t[i+1];
if (tmp < 3) ...
```

Equality domain D_E	*Interval domain* D_I	*Array domain* D_A
$tmp = t[i+1]$	$\begin{cases} i \in [-1..3] \\ tmp \in [1..5] \end{cases}$	$t : \{1, 2, 3, 4, 5\}$

- Backward propagation on $tmp < 3$:

tmp	is reduced to	$[1..2]$	by the value semantics
$t[i+1]$	is reduced to	$[1..2]$	by the equality domain
$i+1$	is reduced to	$[0..1]$	by the array domain
i	is reduced to	$[-1..0]$	by the value semantics

Cooperative Evaluation: Third Example

```
int t[5] = {1,2,3,4,5};
int tmp = t[i+1];
if (tmp < 3) ...
```

Equality domain D_E	*Interval domain* D_I	*Array domain* D_A
$tmp = t[i+1]$	$\begin{cases} i \in [-1..3] \\ tmp \in [1..5] \end{cases}$	$t : \{1, 2, 3, 4, 5\}$

- Backward propagation on $tmp < 3$:

tmp	is reduced to	$[1..2]$	by the value semantics
$t[i+1]$	is reduced to	$[1..2]$	by the equality domain
$i+1$	is reduced to	$[0..1]$	by the array domain
i	is reduced to	$[-1..0]$	by the value semantics

Cooperative Evaluation: Third Example

```
int t[5] = {1,2,3,4,5};
int tmp = t[i+1];
if (tmp < 3) ...
```

Equality domain D_E	*Interval domain D_I*	*Array domain D_A*
$tmp = t[i+1]$	$\begin{cases} i \in [-1..3] \\ tmp \in [1..5] \end{cases}$	$t : \{1, 2, 3, 4, 5\}$

- Backward propagation on $tmp < 3$:

tmp	is reduced to	$[1..2]$	by the value semantics
$t[i+1]$	is reduced to	$[1..2]$	by the equality domain
$i+1$	is reduced to	$[0..1]$	by the array domain
i	is reduced to	$[-1..0]$	by the value semantics

Alarms

- In the abstractions, alarms in $\mathbb{A}$ report illegal operations.
- Each forward abstract semantics also produces alarms to over-approximate the error cases.
- The alarms produced by different abstractions are intersected.

$$\sqcap_{\mathbb{A}} : \mathbb{A} \rightarrow \mathbb{A} \rightarrow \mathbb{A}$$

- Collaboration for the emission of the alarms.

Abstract Semantics of Statements

- To interpret a statement (an assignment $*p = e$ or a test $\text{if}(c)$):

  1. All expressions are cooperatively evaluated;
  2. The resulting alarms are emitted by the analyzer;
  3. The transfer functions approximate the effect of the statement.
     They can use all value abstractions produced by the evaluations.

Frama-C/EVA

- Implemented within EVA, a major evolution of the former Value Analysis plugin.
- Same scope and functionalities for the end-user.
- Similar analysis time but slightly more precise results.

Abstract Values in EVA

- Numerical values:
  - small set of integers: `{0; 3; 8}`
  - integer intervals with congruence information: `[15..51], 3%4`
  - floating point intervals: `[3.59999990463 .. 5.60000038147]`
- Pointer values:
  - map from memory bases to numerical offsets (in bytes):
    `{{ NULL+{0} ; &t+[2..159] ; &u+{6} }}`
- Extensible.

Abstract Domains in EVA

- Main domain: low-level memory model for C [Kir+15]
  - map from (variables $\times$ bits-expressed intervals) to values.
- Symbolic equalities.
- Symbolic locations domain.
- Simple binding to the APRON domains. [JM09]
- Gauges domain. [Ven12]
- Bitwise domain.
- Also extensible.

Future Domains?

- New abstract domains dedicated to:
  - arrays
  - strings
  - dynamically allocated memories
- Make the plugins derived from the Value Analysis (inout, from) abstract domains of EVA.

Conclusion

Structuring an abstract interpreter through value and state abstractions:

- Communication embedded in the abstract semantics.
- Allows new interactions with a domain without modifying it.
- Cooperative emission of the alarms.
- Implemented within EVA, the abstract interpreter of Frama-C.

Bibliography

[CC77] P. Cousot and R. Cousot. "Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints". In: *Principles of Programming Languages*. 1977.

[CC79] P. Cousot and R. Cousot. "Systematic Design of Program Analysis Frameworks". In: *Principles of Programming Languages*. 1979.

[Cou+06] P. Cousot et al. "Combination of Abstractions in the ASTRÉE Static Analyzer". In: *Asian Computing Science Conference*. 2006.

[JM09] B. Jeannet and A. Miné. "Apron: A Library of Numerical Abstract Domains for Static Analysis". In: *Computer Aided Verification*. 2009.

[Kir+15] F. Kirchner et al. "Frama-C: A software analysis perspective". In: *Formal Asp. Comput.* (2015).

[Ven12] A. Venet. "The Gauge Domain: Scalable Analysis of Linear Inequality Invariants". In: *Computer Aided Verification*. 2012.