Synthesizing Predicates from Abstract Domain Losses

Bogdan Mihaila and Axel Simon

Technical University of Munich, Germany
May 1, 2014

$$10 < x \rightarrow y \leq 0$$

Abstract Interpretation

Static program analysis using abstract interpretation:

- use abstract domains to represent program states
- execute abstract semantics of program statements
- compute a fixpoint that over-approximates all possible program behaviors
- for each program point, the computed abstract state is an invariant

Example: Interval Analysis

The interval abstract domain is a common and cheap numeric abstract domain

Example program analyzed with intervals domain:

```
1 assume (-100 <= x && x <= 100)
2 if (x == 0)
3   return;
4 y = z / x;
```

state of x in line 1 is $[-100, 100]$

Example: Interval Analysis

The interval abstract domain is a common and cheap numeric abstract domain

Example program analyzed with intervals domain:

```
1 assume (-100 <= x && x <= 100)
2 if (x == 0)
3   return;
4 y = z / x;
```

state of x in line 1 is $[-100, 100]$
state of x in line 4 is $[-100, 1] \sqcup [1, 100]$

Example: Interval Analysis

The interval abstract domain is a common and cheap numeric abstract domain

Example program analyzed with intervals domain:

```
1 assume (-100 <= x && x <= 100)
2 if (x == 0)
3   return;
4 y = z / x;
```

state of x in line 1 is $[-100, 100]$
state of x in line 4 is $[-100, 1] \sqcup [1, 100]$
which is approximated by $x \in [-100, 100]$
$\rightsquigarrow$ cannot exclude division by zero

Convex approximation incurs precision loss

Convex approximation improves the scalability of numeric domains, but

- trades precision for scalability
- $\rightsquigarrow$ cannot express disjunctive invariants like $x \in [-100, -1] \vee x \in [1, 100]$

Convex approximation incurs precision loss

Convex approximation improves the scalability of numeric domains, but

- trades precision for scalability
- $\rightsquigarrow$ cannot express disjunctive invariants like $x \in [-100, -1] \vee x \in [1, 100]$

Joining two intervals

$$[-100, -1] \sqcup [1, 100] = \begin{array}{ll} \textit{Convex}: & x \in [-100, 100] \\ \textit{Precise}: & x \in [-100, -1] \cup [1, 100] \end{array}$$

Convex approximation incurs precision loss

Convex approximation improves the scalability of numeric domains, but

- trades precision for scalability
- $\rightsquigarrow$ cannot express disjunctive invariants like $x \in [-100, -1] \vee x \in [1, 100]$

Joining two intervals

$$[-100, -1] \sqcup [1, 100] = \begin{array}{ll} \textit{Convex}: & x \in [-100, 100] \\ \textit{Precise}: & x \in [-100, -1] \cup [1, 100] \end{array}$$

Applying a transfer function

$$[\![x \neq 0]\!]\{x \in [-100, 100]\} = \begin{array}{ll} \textit{Convex}: & x \in [-100, 100] \\ \textit{Precise}: & x \in [-100, -1] \cup [1, 100] \end{array}$$

Previous Approach: Predicate Abstraction

Abstract the C program to a program computing the truth values of predicates over C program variables.

$\rightsquigarrow$ state space can be described by Boolean formula.

Hence, disjunctions can be expressed naturally: $a \vee (b \wedge c)$

Previous Approach: Predicate Abstraction

Abstract the C program to a program computing the truth values of predicates over C program variables.
$\rightsquigarrow$ state space can be described by Boolean formula.
Hence, disjunctions can be expressed naturally: $a \vee (b \wedge c)$

Observations:

- usually the verification uses counterexample driven refinement, CEGAR
- finding good predicates is a hard task (**non-termination, scalability**)
- CEGAR requires abstraction, forward analysis on Boolean functions, theorem prover to do backward analysis (**complex infrastructure**)

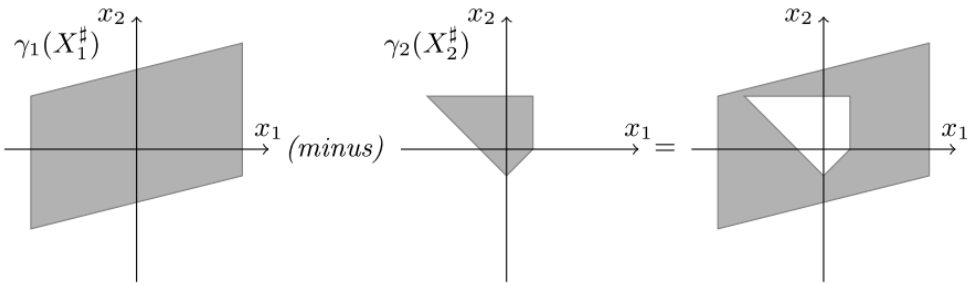
Previous Approach: Non-convex Numeric Domains

Disequality domains express the disequality between variables/constraints, e.g. $x \neq 0$

- Octagons with disequalities
- Difference Bound Matrices with disequalities

More general:

Donut Domain ($a \setminus b$)



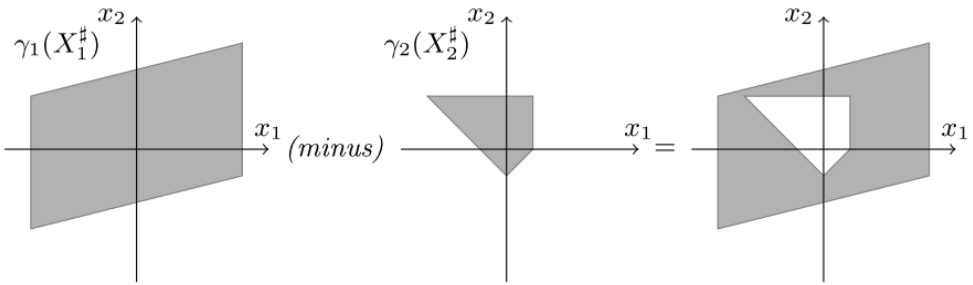
Previous Approach: Non-convex Numeric Domains

Disequality domains express the disequality between variables/constraints, e.g. $x \neq 0$

- Octagons with disequalities
- Difference Bound Matrices with disequalities

More general:

Donut Domain ($a \setminus b$)



These approaches are not general but tailored to specific numeric domains.

Previous Approach: Disjunctive Completion

Uses tree-like structures to separate tracked states, e.g. binary decision diagrams (BDD).

Previous Approach: Disjunctive Completion

Uses tree-like structures to separate tracked states, e.g. binary decision diagrams (BDD).

Challenge: scalability

- stores multiple states per program point, worst case exponential in number of predicates
- transfer functions are executed on every state in the tree
- transfer functions and widening are not trivial to implement; spills from one tree-branch to the other have to be taken into account

Predicate Domain to augment Numeric Domains

Idea: use implications $p \rightarrow q$ between predicates p, q to improve precision of numeric domains; here p, q express facts in the abstract domain.

Predicate Domain to augment Numeric Domains

Idea: use implications $p \to q$ between predicates p, q to improve precision of numeric domains; here p, q express facts in the abstract domain.

- prevent a precision loss in the abstract domain during join by synthesizing new implications
- no tracking of multiple numeric states
- $\neg p \to q$ expresses the disjunctive invariant $p \vee q$
- can express non-convex invariants, e.g. $x \in [-2, 2] \wedge x \neq 0$

Definition of the Predicate Domain $\mathcal{P}$

Analysis uses co-fibered domain with lattice $\langle \mathcal{P} \rhd \mathcal{C}, \sqsubseteq_{\mathcal{P}}, \sqcup_{\mathcal{P}}, \sqcap_{\mathcal{P}} \rangle$.
Here $\mathcal{P} : \wp(\mathit{Pred} \times \mathit{Pred})$ tracks implications $p \to q$ between some predicates p, q.

Definition of the Predicate Domain $\mathcal{P}$

Analysis uses co-fibered domain with lattice $\langle \mathcal{P} \rhd \mathcal{C}, \sqsubseteq_{\mathcal{P}}, \sqcup_{\mathcal{P}}, \sqcap_{\mathcal{P}} \rangle$.
Here $\mathcal{P} : \wp(\mathit{Pred} \times \mathit{Pred})$ tracks implications $p \to q$ between some predicates p, q.

In the co-fibered construction $\mathcal{P} \rhd \mathcal{C}$, $\mathcal{C}$ is a generic child domain.

Definition of the Predicate Domain $\mathcal{P}$

Analysis uses co-fibered domain with lattice $\langle \mathcal{P} \rhd \mathcal{C}, \sqsubseteq_{\mathcal{P}}, \sqcup_{\mathcal{P}}, \sqcap_{\mathcal{P}} \rangle$. Here $\mathcal{P} : \wp(\mathit{Pred} \times \mathit{Pred})$ tracks implications $p \to q$ between some predicates p, q.

In the co-fibered construction $\mathcal{P} \rhd \mathcal{C}$, $\mathcal{C}$ is a generic child domain. Here, $\mathcal{C} = \mathcal{I}$ is the numeric domain of intervals and predicates are of the form $x \bowtie n$ where $n \in \mathbb{Z}$ and $\bowtie \in \{\leq, \nleq, <, \nless, =, \neq\}$.

Definition of the Predicate Domain $\mathcal{P}$

Analysis uses co-fibered domain with lattice $\langle \mathcal{P} \rhd \mathcal{C}, \sqsubseteq_{\mathcal{P}}, \sqcup_{\mathcal{P}}, \sqcap_{\mathcal{P}} \rangle$. Here $\mathcal{P} : \wp(\mathit{Pred} \times \mathit{Pred})$ tracks implications $p \to q$ between some predicates p, q.

In the co-fibered construction $\mathcal{P} \rhd \mathcal{C}$, $\mathcal{C}$ is a generic child domain. Here, $\mathcal{C} = \mathcal{I}$ is the numeric domain of intervals and predicates are of the form $x \bowtie n$ where $n \in \mathbb{Z}$ and $\bowtie \in \{\leq, \nleq, <, \nless, =, \neq\}$.

Example $\langle \bar{\iota}, c \rangle \in \mathcal{P} \rhd \mathcal{I}$

$\bar{\iota} = \{10 < x \to y \leq 0, f = 1 \to z \neq 0, f = 0 \to z = 0\}$
$c = \{x \in [0, 100], y \in [-10, 10], f \in [0, 1], z \in [-5, 5]\}$

Reduction Mechanism

Reduction of the domain is performed when a test is applied:

$$[\![x \bowtie y]\!]^{\mathcal{P}} \langle \bar{\iota}, c \rangle = \langle \bar{\iota}, \mathit{fixapply}(x \bowtie y, c) \rangle$$

where $\mathit{fixapply}(t, c)$ applies test t to the child and all predicates q in $p \rightarrow q$ where p holds in $c' = [\![t]\!]^{\mathcal{I}} c$ and recurses.

Reduction Mechanism

Reduction of the domain is performed when a test is applied:

$$[\![x \bowtie y]\!]^{\mathcal{P}} \langle \bar{\iota}, c \rangle = \langle \bar{\iota}, \mathit{fixapply}(x \bowtie y, c) \rangle$$

where $\mathit{fixapply}(t, c)$ applies test t to the child and all predicates q in $p \to q$ where p holds in $c' = [\![t]\!]^{\mathcal{I}} c$ and recurses.

Example $\langle \bar{\iota}, c \rangle \in \mathcal{P} \triangleright \mathcal{I}$

$\bar{\iota} = \{f = 1 \to z = 0, z \leq 1 \to x = 100\}$
$c = \{f \in [0, 1], z \in [-5, 5], x \in [0, 100]\}$

$[\![f = 1]\!] \langle \bar{\iota}, c \rangle = \langle \bar{\iota}, c' \rangle$

$c' = \{f \in [1, 1], z \in [0, 0], x \in [100, 100]\}$

Inference Mechanism from new Facts

Simple inference operation without employing a SMT solver for the predicates. Instead, we use syntactic predicate inference, semantic inference, modus ponens and the equivalence: $p \rightarrow q \equiv \neg q \rightarrow \neg p$

Inference Mechanism from new Facts

Simple inference operation without employing a SMT solver for the predicates. Instead, we use syntactic predicate inference, semantic inference, modus ponens and the equivalence: $p \rightarrow q \equiv \neg q \rightarrow \neg p$

Examples $\langle \bar{\iota}, c \rangle \in \mathcal{P} \rhd \mathcal{I}$

$\bar{\iota} = \{x \leq 10 \rightarrow y = 1\}, c = \{x \in [\ldots], y \in [\ldots]\}$

Inference Mechanism from new Facts

Simple inference operation without employing a SMT solver for the predicates. Instead, we use syntactic predicate inference, semantic inference, modus ponens and the equivalence: $p \rightarrow q \equiv \neg q \rightarrow \neg p$

Examples $\langle \bar{\iota}, c \rangle \in \mathcal{P} \rhd \mathcal{I}$

$\bar{\iota} = \{x \leq 10 \rightarrow y = 1\}, c = \{x \in [\ldots], y \in [\ldots]\}$

Syntactic inference $a \vdash b$

given the tests $[\![x < 5]\!]$ or $[\![x = 1]\!]$ we can infer $y = 1$
given the tests $[\![y \neq 1]\!]$ or $[\![y = 3]\!]$ we can infer $x \not\leq 10$

Inference Mechanism from new Facts

Simple inference operation without employing a SMT solver for the predicates. Instead, we use syntactic predicate inference, semantic inference, modus ponens and the equivalence: $p \rightarrow q \equiv \neg q \rightarrow \neg p$

Examples $\langle \bar{\iota}, c \rangle \in \mathcal{P} \rhd \mathcal{I}$

$\bar{\iota} = \{x \leq 10 \rightarrow y = 1\}, c = \{x \in [\ldots], y \in [\ldots]\}$

Syntactic inference $a \vdash b$

given the tests $[\![x < 5]\!]$ or $[\![x = 1]\!]$ we can infer $y = 1$
given the tests $[\![y \neq 1]\!]$ or $[\![y = 3]\!]$ we can infer $x \not\leq 10$

Semantic inference $a \Vdash b$

given $c = \{x \in [0, 8], y \in [\ldots]\}$ we can infer $y = 1$
given $c = \{x \in [\ldots], y \in [2, 3]\}$ we can infer $x \not\leq 10$

Synthesis Mechanism

On joining two states in the numeric domain we observe the precision loss and synthesize predicates to maintain the precision:

$$\langle \bar{\iota}_1, c_1 \rangle \sqcup_{\mathcal{P}} \langle \bar{\iota}_2, c_2 \rangle = \langle \mathit{join}(\bar{\iota}_1, \bar{\iota}_2) \cup \mathit{synth}^{\mathcal{C}}(c_1, c_2), c_1 \sqcup_{\mathcal{C}} c_2 \rangle$$

Synthesis Mechanism

On joining two states in the numeric domain we observe the precision loss and synthesize predicates to maintain the precision:

$$\langle \bar{\iota}_1, c_1 \rangle \sqcup_{\mathcal{P}} \langle \bar{\iota}_2, c_2 \rangle = \langle \mathit{join}(\bar{\iota}_1, \bar{\iota}_2) \cup \mathit{synth}^{\mathcal{C}}(c_1, c_2), c_1 \sqcup_{\mathcal{C}} c_2 \rangle$$

Example

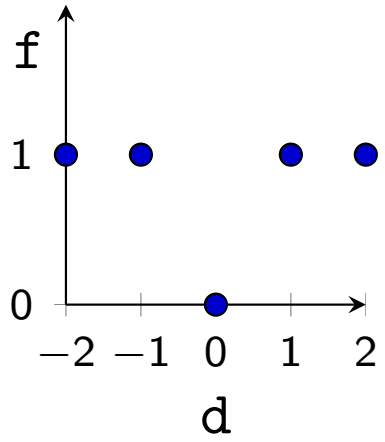
	c_1	c_2	$c_1 \sqcup_{\mathcal{I}} c_2$
$x \in$	$[0, 5]$	$[10, 15]$	$[0, 15]$
$y \in$	$[-5, -1]$	$[2, 3]$	$[-5, 3]$

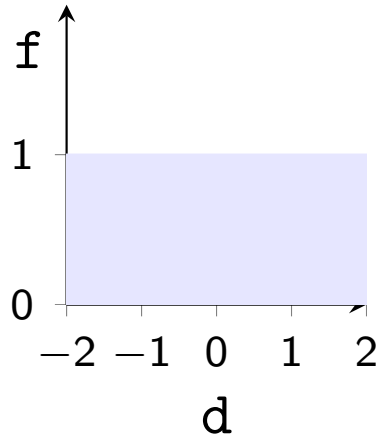
$$\mathit{synth}^{\mathcal{C}}(c_1, c_2) = \{5 < x \rightarrow 2 \leq y, -1 < y \rightarrow 10 \leq x\}$$

```
if (0 < y)          // c' = c1 ⊔_I c2 = {x ∈ [0, 15], y ∈ [−5, 3]}
   then ...         // c'' = c2 = {x ∈ [10, 15], y ∈ [2, 3]}
   else ...         // c'' = c1 = {x ∈ [0, 5], y ∈ [−5, −1]}
```

Division by Zero

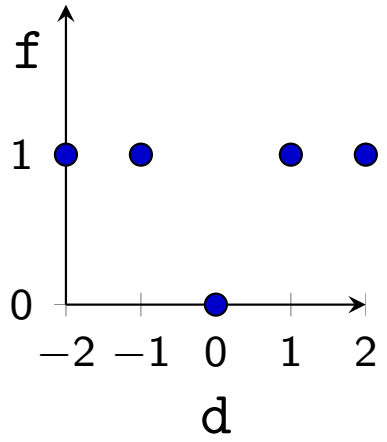
```
1 f = d!=0;
2 ...
3 if (f) {
4    assert(d!=0);
5    y = x / d;
6 }
```

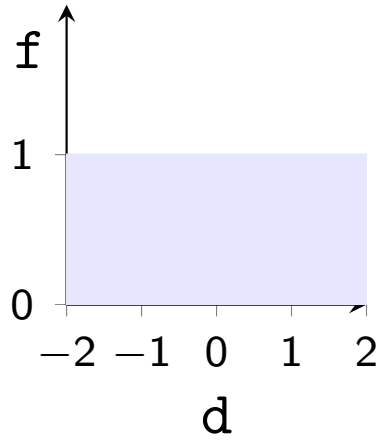




Division by Zero

```
1 f = d!=0;
2 ...
3 if (f) {
4    assert(d!=0);
5    y = x / d;
6 }
```

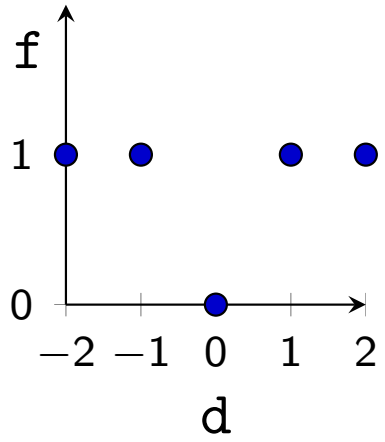


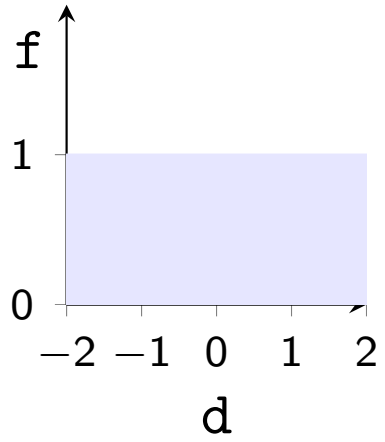


Proving the assertion in line 4 is not possible with intervals because of the convex approximation (right figure).

Division by Zero

```
1 f = d!=0;
2 ...
3 if (f) {
4   assert(d!=0);
5   y = x / d;
6 }
```





Proving the assertion in line 4 is not possible with intervals because of the convex approximation (right figure).
But using the implication $f = 1 \rightarrow d \neq 0$ generated in line 1 the assertion can be proved when applying the reduction to the state: $\{f = [1, 1], d = [0, 0]\}$

Path-Sensitive Invariants

Accessing a file only if it was already opened

```
1  FILE *out;
2  out->is_open = 1;
3  assert(out->is_open == 1);
4  out->is_open = 0;
5  ...
6  if (flag)
7    out->is_open = 1;
8  ...
9  if (flag)
10   assert(out->is_open == 1);
```

Path-Sensitive Invariants

Accessing a file only if it was already opened

```
1  FILE *out;
2  out->is_open = 1;
3  assert(out->is_open == 1);
4  out->is_open = 0;
5  ...
6  if (flag)
7    out->is_open = 1;
8  ...
9  if (flag)
10   assert(out->is_open == 1);
```

Without predicates the state in line 8 is: $\{out\_is\_open \in [0, 1]\}$

Path-Sensitive Invariants

Accessing a file only if it was already opened

```
1  FILE *out;
2  out->is_open = 1;
3  assert(out->is_open == 1);
4  out->is_open = 0;
5  ...
6  if (flag)
7    out->is_open = 1;
8  ...
9  if (flag)
10   assert(out->is_open == 1);
```

Without predicates the state in line 8 is: $\{out\_is\_open \in [0, 1]\}$
With predicates the state in line 8 is:
$\langle\{0 < f \rightarrow 1 \leq out\_is\_open\}, \{out\_is\_open \in [0, 1]\}\rangle$
$\rightsquigarrow$ using the implication at line 10 the numeric state is reduced to:
$\{out\_is\_open \in [1, 1]\}$

Separation of Loop Iterations

```
1  p = &some_var;
2  n = 5;
3  while (n >= 0) {
4    assert(p != 0);
5    // dereference p
6    ...
7    if (n == 0)
8      p = 0; // free p
9    n--;
10 }
```

Separation of Loop Iterations

```
1  p = &some_var;
2  n = 5;
3  while (n >= 0) {
4    assert(p != 0);
5    // dereference p
6    ...
7    if (n == 0)
8      p = 0; // free p
9    n--;
10 }
```

Problem here is that in the last loop iteration `p` is set to 0 and we have to analyze the loop again with this new and greater state.

Separation of Loop Iterations

```
1  p = &some_var;
2  n = 5;
3  while (n >= 0) {
4    assert(p != 0);
5    // dereference p
6    ...
7    if (n == 0)
8      p = 0; // free p
9    n--;
10 }
```

Problem here is that in the last loop iteration `p` is set to 0 and we have to analyze the loop again with this new and greater state.
$\rightsquigarrow$ the assertion in line 4 will then be analyzed with the joined values: $p \neq 0 \sqcup p = 0$

Separation of Loop Iterations (continued)

```
1  p = &some_var;
2  n = 5;
3  while (n >= 0) {
4    assert(p != 0);
5    // dereference p
6    ...
7    if (n == 0)
8      p = 0; // free p
9    n--;
10 }
```

step	line		intervals		implications
			p	n	
1	2		$[99, 99]$		
2	3		$[99, 99]$	$[5, 5]$	
...			...		...
8	3	$\sqcup$	$[99, 99]$	$[4, 5]$	
8'	3'	∇	$[99, 99]$	$[-1, 5]$	
9	4		$[99, 99]$	$[\mathbf{0}, \mathbf{5}]$	
...			...		...
12	**8**		$[99, 99]$	$[0, 0]$	
13	9	$\sqcup$	$[\mathbf{0}, \mathbf{99}]$	$[0, 5]$	$\{\mathbf{0} < \mathbf{n} \rightarrow \mathbf{99} \leq \mathbf{p}, \mathbf{0} < \mathbf{p} \rightarrow \mathbf{0} \leq \mathbf{n}\}$
14	10		$[0, 99]$	$[-1, 4]$	$\{-1 < n \rightarrow 99 \leq p, 0 < p \rightarrow -1 \leq n\}$
15	3	$\sqcup$	$[0, 99]$	$[-1, 5]$	$\{-1 < n \rightarrow 99 \leq p, 0 < p \rightarrow -1 \leq n\}$
16	4		$[\mathbf{99}, \mathbf{99}]$	$[0, 5]$	$\{\mathbf{-1} < \mathbf{n} \rightarrow \mathbf{99} \leq \mathbf{p}, 0 < p \rightarrow -1 \leq n\}$
...			...		...
22	3	$\sqsubseteq$	$[0, 99]$	$[-1, 5]$	$\{-1 < n \rightarrow 99 \leq p, 0 < p \rightarrow -1 \leq n\}$

Separation of Loop Iterations (continued)

```
1  p = &some_var;
2  n = 5;
3  while (n >= 0) {
4    assert(p != 0);
5    // dereference p
6    ...
7    if (n == 0)
8      p = 0; // free p
9    n--;
10 }
```

step 13: $\{p \in [99,99], n \in [1,5]\} \sqcup \{p \in [0,0], n \in [0,0]\}$ synthesizes implications

step	line		intervals		implications
			p	n	
1	2		$[99,99]$		
2	3		$[99,99]$	$[5,5]$	
...			...		...
8	3	$\sqcup$	$[99,99]$	$[4,5]$	
8'	3'	∇	$[99,99]$	$[-1,5]$	
9	4		$[99,99]$	$[\mathbf{0},\mathbf{5}]$	
...			...		...
12	**8**		$[99,99]$	$[0,0]$	
13	9	$\sqcup$	$[\mathbf{0},\mathbf{99}]$	$[0,5]$	$\{\mathbf{0} < \mathbf{n} \rightarrow \mathbf{99} \leq \mathbf{p}, \mathbf{0} < \mathbf{p} \rightarrow \mathbf{0} \leq \mathbf{n}\}$
14	10		$[0,99]$	$[-1,4]$	$\{-1 < n \rightarrow 99 \leq p, 0 < p \rightarrow -1 \leq n\}$
15	3	$\sqcup$	$[0,99]$	$[-1,5]$	$\{-1 < n \rightarrow 99 \leq p, 0 < p \rightarrow -1 \leq n\}$
16	4		$[\mathbf{99},\mathbf{99}]$	$[0,5]$	$\{\mathbf{-1} < \mathbf{n} \rightarrow \mathbf{99} \leq \mathbf{p}, 0 < p \rightarrow -1 \leq n\}$
...			...		...
22	3	$\sqsubseteq$	$[0,99]$	$[-1,5]$	$\{-1 < n \rightarrow 99 \leq p, 0 < p \rightarrow -1 \leq n\}$

Separation of Loop Iterations (continued)

```
1  p = &some_var;
2  n = 5;
3  while (n >= 0) {
4    assert(p != 0);
5    // dereference p
6    ...
7    if (n == 0)
8      p = 0; // free p
9    n--;
10 }
```

step 13: $\{p \in [99,99], n \in [1,5]\} \sqcup \{p \in [0,0], n \in [0,0]\}$ synthesizes implications

step 14: transforms the implications

step	line		intervals p	intervals n	implications
1	2		$[99,99]$		
2	3		$[99,99]$	$[5,5]$	
...			...		...
8	3	$\sqcup$	$[99,99]$	$[4,5]$	
8'	3'	∇	$[99,99]$	$[-1,5]$	
9	4		$[99,99]$	$[\mathbf{0},\mathbf{5}]$	
...			...		...
12	**8**		$[99,99]$	$[0,0]$	
13	9	$\sqcup$	$[\mathbf{0},\mathbf{99}]$	$[0,5]$	$\{\mathbf{0} < \mathbf{n} \rightarrow \mathbf{99} \leq \mathbf{p}, \mathbf{0} < \mathbf{p} \rightarrow \mathbf{0} \leq \mathbf{n}\}$
14	10		$[0,99]$	$[-1,4]$	$\{-1 < n \rightarrow 99 \leq p, 0 < p \rightarrow -1 \leq n\}$
15	3	$\sqcup$	$[0,99]$	$[-1,5]$	$\{-1 < n \rightarrow 99 \leq p, 0 < p \rightarrow -1 \leq n\}$
16	4		$[\mathbf{99},\mathbf{99}]$	$[0,5]$	$\{\mathbf{-1} < \mathbf{n} \rightarrow \mathbf{99} \leq \mathbf{p}, 0 < p \rightarrow -1 \leq n\}$
...			...		...
22	3	$\sqsubseteq$	$[0,99]$	$[-1,5]$	$\{-1 < n \rightarrow 99 \leq p, 0 < p \rightarrow -1 \leq n\}$

Separation of Loop Iterations (continued)

```
1  p = &some_var;
2  n = 5;
3  while (n >= 0) {
4    assert(p != 0);
5    // dereference p
6    ...
7    if (n == 0)
8      p = 0; // free p
9    n--;
10 }
```

step 13: $\{p \in [99,99], n \in [1,5]\} \sqcup \{p \in [0,0], n \in [0,0]\}$ synthesizes implications

step 14: transforms the implications

step 16: reduce value of p using implication

step	line		intervals		implications
			p	n	
1	2		$[99, 99]$		
2	3		$[99, 99]$	$[5, 5]$	
...			...		...
8	3	$\sqcup$	$[99, 99]$	$[4, 5]$	
8'	3'	∇	$[99, 99]$	$[-1, 5]$	
9	4		$[99, 99]$	$[\mathbf{0}, \mathbf{5}]$	
...			...		...
12	**8**		$[99, 99]$	$[0, 0]$	
13	9	$\sqcup$	$[\mathbf{0}, \mathbf{99}]$	$[0, 5]$	$\{\mathbf{0} < \mathbf{n} \rightarrow \mathbf{99} \leq \mathbf{p}, \mathbf{0} < \mathbf{p} \rightarrow \mathbf{0} \leq \mathbf{n}\}$
14	10		$[0, 99]$	$[-1, 4]$	$\{-1 < n \rightarrow 99 \leq p, 0 < p \rightarrow -1 \leq n\}$
15	3	$\sqcup$	$[0, 99]$	$[-1, 5]$	$\{-1 < n \rightarrow 99 \leq p, 0 < p \rightarrow -1 \leq n\}$
16	4		$[\mathbf{99}, \mathbf{99}]$	$[0, 5]$	$\{-\mathbf{1} < \mathbf{n} \rightarrow \mathbf{99} \leq \mathbf{p}, 0 < p \rightarrow -1 \leq n\}$
...			...		...
22	3	$\sqsubseteq$	$[0, 99]$	$[-1, 5]$	$\{-1 < n \rightarrow 99 \leq p, 0 < p \rightarrow -1 \leq n\}$

Separation of Loop Iterations (continued)

```
1  p = &some_var;
2  n = 5;
3  while (n >= 0) {
4    assert(p != 0);
5    // dereference p
6    ...
7    if (n == 0)
8      p = 0; // free p
9    n--;
10 }
```

step 13: $\{p \in [99,99], n \in [1,5]\} \sqcup \{p \in [0,0], n \in [0,0]\}$ synthesizes implications

step 14: transforms the implications

step 16: reduce value of p using implication $\rightsquigarrow$ assertion holds!

step	line		intervals		implications
			p	n	
1	2		$[99, 99]$		
2	3		$[99, 99]$	$[5, 5]$	
...			...		...
8	3	$\sqcup$	$[99, 99]$	$[4, 5]$	
8'	3'	∇	$[99, 99]$	$[-1, 5]$	
9	4		$[99, 99]$	$[\mathbf{0}, \mathbf{5}]$	
...			...		...
12	**8**		$[99, 99]$	$[0, 0]$	
13	9	$\sqcup$	$[\mathbf{0}, \mathbf{99}]$	$[0, 5]$	$\{\mathbf{0} < \mathbf{n} \rightarrow \mathbf{99} \leq \mathbf{p}, \mathbf{0} < \mathbf{p} \rightarrow \mathbf{0} \leq \mathbf{n}\}$
14	10		$[0, 99]$	$[-1, 4]$	$\{-1 < n \rightarrow 99 \leq p, 0 < p \rightarrow -1 \leq n\}$
15	3	$\sqcup$	$[0, 99]$	$[-1, 5]$	$\{-1 < n \rightarrow 99 \leq p, 0 < p \rightarrow -1 \leq n\}$
16	4		$[\mathbf{99}, \mathbf{99}]$	$[0, 5]$	$\{-\mathbf{1} < \mathbf{n} \rightarrow \mathbf{99} \leq \mathbf{p}, 0 < p \rightarrow -1 \leq n\}$
...			...		...
22	3	$\sqsubseteq$	$[0, 99]$	$[-1, 5]$	$\{-1 < n \rightarrow 99 \leq p, 0 < p \rightarrow -1 \leq n\}$

Benefits of the Predicate Domain

- no exponential state storage like disjunctive completion, only one numeric state and one implication-tracking domain
- no iterative counterexample driven refinement but least fixpoint computation with improved precision
- uses simple inference rules for implications as the reduction mechanism (no SMT solver)
- observes precision loss in numeric domains and repairs it by synthesizing implications that are not syntactically present in the program
- synthesizes relational information that cannot be expressed in the numeric domain
- generalizes to arbitrary predicates in other domains