Design and Analysis of Real-Time Systems

Foundations of Abstract Interpretation and Numerical Abstractions

Jan Reineke

Advanced Lecture, Summer 2013
More domains are described at: http://bugseng.com/products/ppl/abstractions
Example: Interval Analysis

- Start
  - $x = 0$
  - $y = 2x$

1. $x = 0$
   - $x = x + 1$
   - $y = y + 1$
   - $y = y + 1$

2. $x = x + 1$
   - $y = y + 1$

3. $y = 2x$

4. $y = y + 1$

5. $y = y + 1$

Neg $(x < 3)$
Pos $(x < 3)$
Example: Interval Analysis

\[
x \rightarrow [0,0] \\
y \rightarrow \text{top}
\]

\[
x \rightarrow [0,0] \\
y \rightarrow \text{top}
\]

\[
x \rightarrow [1,1] \\
y \rightarrow \text{top}
\]

\[
x \rightarrow [1,1] \\
y \rightarrow [2,2]
\]

\[
x = 0
\]

\[
x = x + 1
\]

\[
y = y + 1
\]

\[
y = 2x
\]

\[
\text{start}
\]

\[
\text{Pos}(x < 3)
\]

\[
\text{Neg}(x < 3)
\]

\[
5
\]

\[
1
\]

\[
2
\]

\[
3
\]

\[
4
\]
Example: Interval Analysis

\[
\begin{align*}
x & \rightarrow [0,1] & x & \rightarrow [0,0] \\
y & \rightarrow [3,3] & y & \rightarrow \text{top} \\
\end{align*}
\]

\[
\begin{align*}
x & \rightarrow [0,1] & x & \rightarrow [0,0] \\
y & \rightarrow [3,3] & y & \rightarrow \text{top} \\
\end{align*}
\]

\[
\begin{align*}
x & \rightarrow [1,2] & x & \rightarrow [1,1] \\
y & \rightarrow [3,3] & y & \rightarrow \text{top} \\
\end{align*}
\]

\[
\begin{align*}
x & \rightarrow [1,2] & x & \rightarrow [1,1] \\
y & \rightarrow [2,4] & y & \rightarrow [2,2] \\
\end{align*}
\]
Example: Interval Analysis

\[
\begin{align*}
  x &\rightarrow [0,2] & x &\rightarrow [0,1] & x &\rightarrow [0,0] \\
  y &\rightarrow [3,5] & y &\rightarrow [3,3] & y &\rightarrow \text{top} \\

  x &\rightarrow [0,2] & x &\rightarrow [0,1] & x &\rightarrow [0,0] \\
  y &\rightarrow [3,5] & y &\rightarrow [3,3] & y &\rightarrow \text{top} \\

  x &\rightarrow [1,3] & x &\rightarrow [1,2] & x &\rightarrow [1,1] \\
  y &\rightarrow [3,5] & y &\rightarrow [3,3] & y &\rightarrow \text{top} \\

  x &\rightarrow [1,3] & x &\rightarrow [1,2] & x &\rightarrow [1,1] \\
\end{align*}
\]
Example: Interval Analysis

\[
\begin{align*}
x & \to [0,3] & x & \to [0,2] & x & \to [0,1] & x & \to [0,0] \\
y & \to [3,7] & y & \to [3,5] & y & \to [3,3] & y & \to \text{top} \\
\end{align*}
\]

\[
\begin{align*}
x & \to [0,2] & x & \to [0,1] & x & \to [0,0] \\
y & \to [3,5] & y & \to [3,3] & y & \to \text{top} \\
\end{align*}
\]

\[
\begin{align*}
x & \to [1,3] & x & \to [1,2] & x & \to [1,1] \\
y & \to [3,5] & y & \to [3,3] & y & \to \text{top} \\
\end{align*}
\]

\[
\begin{align*}
x & \to [1,3] & x & \to [1,2] & x & \to [1,1] \\
y & \to [2,6] & y & \to [2,4] & y & \to [2,2] \\
\end{align*}
\]

Diagram:

- Start at node 1 with conditions:
  - Pos(x < 3)
  - Neg(x < 3)

- From node 1:
  - x = 0
  - x = x + 1
  - y = y + 1
  - y = 2 * x

- Transitions to nodes 2, 3, and 5:
  - Pos(x < 3) to 2
  - Neg(x < 3) to 5

- Node 5:
  - y = y + 1
Example: Interval Analysis

\[
\begin{align*}
x & \rightarrow [0,3] \quad x \rightarrow [0,2] \quad x \rightarrow [0,1] \quad x \rightarrow [0,0] \\
y & \rightarrow [3,7] \quad y \rightarrow [3,5] \quad y \rightarrow [3,3] \quad y \rightarrow \text{top}
\end{align*}
\]

\[
\begin{align*}
x & \rightarrow [0,2] \quad x \rightarrow [0,1] \quad x \rightarrow [0,0] \\
y & \rightarrow [3,5] \quad y \rightarrow [3,3] \quad y \rightarrow \text{top}
\end{align*}
\]

\[
\begin{align*}
x & \rightarrow [1,3] \quad x \rightarrow [1,2] \quad x \rightarrow [1,1] \\
y & \rightarrow [3,5] \quad y \rightarrow [3,3] \quad y \rightarrow \text{top}
\end{align*}
\]

\[
\begin{align*}
x & \rightarrow [1,3] \quad x \rightarrow [1,2] \quad x \rightarrow [1,1] \\
y & \rightarrow [2,6] \quad y \rightarrow [2,4] \quad y \rightarrow [2,2]
\end{align*}
\]
Example: Interval Analysis

\[
x \to [0,3] \quad x \to [0,2] \quad x \to [0,1] \quad x \to [0,0] \\
y \to [3,7] \quad y \to [3,5] \quad y \to [3,3] \quad y \to \text{top}
\]

\[
x \to [0,2] \quad x \to [0,1] \quad x \to [0,0] \\
y \to [3,5] \quad y \to [3,3] \quad y \to \text{top}
\]

\[
x \to [1,3] \quad x \to [1,2] \quad x \to [1,1] \\
y \to [3,5] \quad y \to [3,3] \quad y \to \text{top}
\]

\[
x \to [1,3] \quad x \to [1,2] \quad x \to [1,1] \\
y \to [2,6] \quad y \to [2,4] \quad y \to [2,2]
\]

Imprecise due to non-relational analysis
Example: Interval Analysis

Would Octagons determine that y must be 7 at program point 5?

Imprecise due to non-relational analysis
Example: Interval Analysis

Would Octagons determine that \( y \) must be 7 at program point 5?

Imprecise due to non-relational analysis
Intervals, Hasse diagram
Intervals, Hasse diagram

Ascending chain condition is not satisfied!

→ Kleene Iteration is not guaranteed to terminate!
Example: Interval Analysis

\[ x \mapsto \bot \]
\[ x \mapsto [0, 0] \]
\[ x \mapsto [0, 1] \]
\[ \ldots \]
\[ x \mapsto [0, 1000] \]
Solution: Widening
“Enforce Ascending Chain Condition”

- Widening enforces the ascending chain condition during analysis.
- Accelerates termination by moving up the lattice more quickly.
- May yield imprecise results…
A widening $\nabla$ is an operator $\nabla : D \times D \rightarrow D$ such that

1. **Safety:** $x \sqsubseteq (x \nabla y)$ and $y \sqsubseteq (x \nabla y)$

2. **Termination:**

   for all ascending chains $x_0 \sqsubseteq x_1 \sqsubseteq \ldots$ the chain
   
   $y_0 = x_0$
   $y_{i+1} = y_i \nabla x_{i+1}$

   is finite.
Widening Operator for Intervals

Simplest solution:

\[ \bot \triangledown x = x \triangledown \bot = x \]

\[ [l, u] \triangledown [l', u'] = \begin{cases} 
  l & : l' \geq l \\
  -\infty & : l' < l \\
  \infty & : u' > u \\
\end{cases} \]

Example:

\[ [3, 5] \triangledown [4, 5] = [3, 5] \]
\[ [3, 5] \triangledown [4, 6] = [3, \infty] \]
\[ [3, 5] \triangledown [2, 6] = [\infty, \infty] \]
Example Revisited: Interval Analysis with Simple Widening

**Standard Kleene Iteration:**
\[ \perp \leq F(\perp) \leq F^2(\perp) \leq F^3(\perp) \leq \ldots \]

**Kleene Iteration with Widening:**
\[ F_{\nabla}(x) := x \nabla F(x) \]
\[ \perp \leq F_{\nabla}(\perp) \leq F_{\nabla}^2(\perp) \leq F_{\nabla}^3(\perp) \leq \ldots \]
Example Revisited: Interval Analysis with Simple Widening

**Standard Kleene Iteration:**

\[ \bot \leq F(\bot) \leq F^2(\bot) \leq F^3(\bot) \leq \ldots \]

**Kleene Iteration with Widening:**

\[ F_{\triangledown}(x) := x \triangledown F(x) \]

\[ \bot \leq F_{\triangledown}(\bot) \leq F_{\triangledown}^2(\bot) \leq F_{\triangledown}^3(\bot) \leq \ldots \]

\[ x \mapsto [0, 0] \]

\[ x \mapsto [0, \infty] \]

→ Quick termination but imprecise result!
More Sophisticated Widening for Intervals

Define set of jump points (barriers) based on constants appearing in program, e.g.:

\[ \mathcal{J} = \{-\infty, 0, 1, 1000, \infty\} \]

Intuition: “Don’t jump to –infty, +infty immediately but only to next jump point.”

\[
[l, u] \triangledown [l', u'] = \begin{cases} 
  l & : l' \geq l \\
  \max\{x \in \mathcal{J} \mid x \leq l'\} & : l' < l' \\
  u & : u' \leq u \\
  \min\{x \in \mathcal{J} \mid x \geq u'\} & : u' > u 
\end{cases}
\]
Example Revisited: Interval Analysis with Sophisticated Widening

\[ l, u \]
\[ l_0, u_0 = \left( \min \{ x^2 \| x \leq l \}, \max \{ x^2 \| x \geq u \} \right) \]

\[ x \in [0, 0] \]
\[ x \in [0, 1] \]
\[ x \in [0, 1000] \]

2. REFERENCES
Example Revisited: Interval Analysis with Sophisticated Widening

\[
\begin{align*}
x &\mapsto [0, 0] \\
x &\mapsto [0, 1] \\
x &\mapsto [0, 1000]
\end{align*}
\]

\[x = 0\]

\[\neg(x < 1000)\]

\[\text{Pos}(x < 1000)\]

\[x = x+1\]

\[\text{More precise, potentially terminates more slowly.}\]
Example Revisited:
Interval Analysis with Sophisticated Widening

\[
\begin{align*}
x & \mapsto [0, 0] \\
x & \mapsto [0, 1] \\
x & \mapsto [0, 1000]
\end{align*}
\]

\[ \rightarrow \text{More precise, potentially terminates more slowly.} \]

Do we need to apply widening “everywhere”? Do we need to apply widening “immediately”?
Example Revisited: Interval Analysis with Sophisticated Widening

\[
\begin{align*}
    x &\mapsto [0, 0] \\
    x &\mapsto [0, 1] \\
    x &\mapsto [0, 1000]
\end{align*}
\]

→ More precise, potentially terminates more slowly.

Do we need to apply widening “everywhere”? Do we need to apply widening “immediately”?
Selective Application of Widening

- To ensure convergence it is sufficient to apply widening at cut points.
  
  Cut points = set of locations that cut each loop (in the control-flow graph)

- Delayed widening: apply a fixed number of rounds of standard Kleene iteration before starting to apply widening operator.
Another Example: Interval Analysis with Sophisticated Widening

\[ x \mapsto [0, 0] \]
\[ x \mapsto [0, 1] \]
\[ x \mapsto [0, 1000] \]
Another Example: Interval Analysis with Sophisticated Widening

$x \mapsto [0, 0]$
$x \mapsto [0, 1]$
$x \mapsto [0, 1000]$

$y \mapsto [2, 2]$
$y \mapsto [2, 1000]$
$y \mapsto [2, \infty]$
Another Example: Interval Analysis with Sophisticated Widening

\[ \begin{align*}
    x &\mapsto [0, 0] \\
    x &\mapsto [0, 1] \\
    x &\mapsto [0, 1000] \\
    y &\mapsto [2, 2] \\
    y &\mapsto [2, 1000] \\
    y &\mapsto [2, \infty] \\
\end{align*} \]

Would be \([2, 2000]\) in least fixed point, but 2000 does not appear in the program…
Narrowing: Recovering Precision

- Widening may yield imprecise results by overshooting the least fixed point.
- Narrowing is used to approach the least fixed point from above.

\[ \{ x \mid x \supseteq lfp \ F \} \]
Narrowing: Recovering Precision

- Widening may yield imprecise results by overshooting the least fixed point.
- Narrowing is used to approach the least fixed point from above.

\[
\{ x \mid x \subseteq \text{lfp } F \}
\]

How can we safely move down the lattice?
Narrowing: Recovering Precision

- Widening may yield imprecise results by overshooting the least fixed point.
- Narrowing is used to approach the least fixed point from above.

Possible problem: infinite descending chains
Is it really a problem?
Narrowing:
Recovering Precision

Widening terminates at a point $x \supseteq \text{lfp } F$.
We can iterate:

$$
\begin{align*}
  x_0 &= x \\
  x_{i+1} &= F(x_i) \cap x_i
\end{align*}
$$

Safety:
By monotonicity we know $F(x) \supseteq F(\text{lfp } F) = \text{lfp } F$.
By induction we can easily show that $x_i \supseteq \text{lfp } F$ for all $i$.

Termination:
Depends on existence of infinite descending chains.
Narrowing: Formal Requirement

A narrowing $\Delta$ is an operator $\Delta : D \times D \rightarrow D$ such that

1. **Safety**: $l \sqsubseteq x$ and $l \sqsubseteq y \Rightarrow l \sqsubseteq (x \Delta y) \sqsubseteq x$

2. **Termination**: for all descending chains $x_0 \sqsupseteq x_1 \sqsupseteq \ldots$ the chain

$$y_0 = x_0$$
$$y_{i+1} = y_i \Delta x_{i+1}$$

is finite.
Narrowing: Formal Requirement

A narrowing $\Delta$ is an operator $\Delta : D \times D \to D$ such that

1. **Safety:** $l \sqsubseteq x$ and $l \sqsubseteq y \Rightarrow l \sqsubseteq (x \Delta y) \sqsubseteq x$

2. **Termination:**
   
   for all descending chains $x_0 \sqsupseteq x_1 \sqsupseteq \ldots$ the chain
   
   $y_0 = x_0$
   
   $y_{i+1} = y_i \Delta x_{i+1}$

   is finite.

*Is $\sqcap$ ("meet") a narrowing operator on intervals?*
Another Example Revisited: Interval Analysis with Widening and Narrowing

Result after Widening:

\[ x \mapsto [0, 0] \]
\[ x \mapsto [0, 1] \]
\[ x \mapsto [0, 1000] \]

Result after Narrowing:
Another Example Revisited: Interval Analysis with Widening and Narrowing

Result after Widening:

\[ x \mapsto [0, 0] \]
\[ x \mapsto [0, 1] \]
\[ x \mapsto [0, 1000] \]
\[ y \mapsto [2, 2] \]
\[ y \mapsto [2, 1000] \]
\[ y \mapsto [2, \infty] \]

Result after Narrowing:
Another Example Revisited: Interval Analysis with Widening and Narrowing

Result after Widening:

- $x \mapsto [0, 0]$
- $x \mapsto [0, 1]$
- $x \mapsto [0, 1000]$
- $y \mapsto [2, 2]$
- $y \mapsto [2, 1000]$
- $y \mapsto [2, \infty]$

Result after Narrowing:

$\rightarrow$ Precisely the least fixed point!
Another Example Revisited: Interval Analysis with Widening and Narrowing

Result after Widening:

\[
\begin{align*}
  x &\mapsto [0, 0] \\
  x &\mapsto [0, 1] \\
  x &\mapsto [0, 1000] \\
  y &\mapsto [2, 2] \\
  y &\mapsto [2, 1000] \\
  y &\mapsto [2, \infty]
\end{align*}
\]

Result after Narrowing:

\[
\begin{align*}
  x &\mapsto [0, 999]
\end{align*}
\]

→ Precisely the least fixed point!
Another Example Revisited: Interval Analysis with Widening and Narrowing

Result after Widening:

- $x \mapsto [0, 0]$
- $x \mapsto [0, 1]$
- $x \mapsto [0, 1000]$

- $y \mapsto [2, 2]$
- $y \mapsto [2, 1000]$
- $y \mapsto [2, \infty]$

Result after Narrowing:

- $x \mapsto [0, 999]$
- $x \mapsto [1, 1000]$

→ Precisely the least fixed point!
Another Example Revisited: Interval Analysis with Widening and Narrowing

Result after Widening:

\[
\begin{align*}
x &\mapsto [0, 0] \\
x &\mapsto [0, 1] \\
x &\mapsto [0, 1000] \\
y &\mapsto [2, 2] \\
y &\mapsto [2, 1000] \\
y &\mapsto [2, \infty] \\
\end{align*}
\]

Result after Narrowing:

\[
\begin{align*}
x &\mapsto [0, 999] \\
x &\mapsto [1, 1000] \\
y &\mapsto [2, 2000] \\
\end{align*}
\]

→ Precisely the least fixed point!
Another Example Revisited: Interval Analysis with Widening and Narrowing

Result after Widening:
- $x \mapsto [0, 0]$
- $x \mapsto [0, 1]$
- $x \mapsto [0, 1000]$
- $y \mapsto [2, 2]$
- $y \mapsto [2, 1000]$
- $y \mapsto [2, \infty]$

Result after Narrowing:
- $x \mapsto [1000, 1000]$
- $y \mapsto [3, 2001]$
- $x \mapsto [0, 999]$
- $x \mapsto [1, 1000]$
- $y \mapsto [2, 2000]$

$\rightarrow$ Precisely the least fixed point!
Applications of Numerical Domains

As input to other analyses:
- Cache Analysis
- To detect dependencies between memory accesses in pipeline
- Loop Bound Analysis:
  - Instrument program with loop iteration counters
  - Determine maximal value of counter
  - Requires relational analysis
Multiple approaches of varying sophistication
- Pattern-based approach
- Data-flow based approach
- Slicing + Value Analysis + Invariant Analysis
- Reduction to Value Analysis
Loop Bound Analysis: Pattern-based Approach

Identify common loop patterns; derive loop bounds for pattern once manually

```plaintext
for (x < 6)
{
  ...
  x++;
}
```
Loop Bound Analysis: Pattern-based Approach

Identify common loop patterns; derive loop bounds for patterns once manually

\[ \text{for } (x < 6) \]
\[ \{ \]
\[ \text{... } \]
\[ x++; \]
\[ \} \]
Loop Bound Analysis: Pattern-based Approach

Identify common loop patterns; derive loop bounds for pattern once manually

```c
for (x < 6)
{
    ...
    x++; 
}
```

→ Loop bound: 6-minimal value of x
Loop Bound Analysis: Data-flow-based Approach
[Cullmann and Martin, 2007]

Combination of multiple analyses:
1. Identify possible loop counters
2. “Invariant analysis”: determine how loop counters may change in one loop iteration
3. Bound calculation: combine results from step 2 with branch conditions
Loop Bound Analysis:
Data-flow-based Approach
[Cullmann and Martin, WCET 2007]

Example:

```c
for (x < 6) {
    y++;  
    if (y % 2 == 0) 
        x++; 
    else 
        x = x+2; 
    z++; 
}
```
Loop Bound Analysis: Data-flow-based Approach
[Cullmann and Martin, WCET 2007]

Example:

```c
for (x < 6) {
    y++;
    if (y % 2 == 0)
        x++;
    else
        x = x+2;
    z++;
}
```

1. x, y, and z are potential loop counters
Loop Bound Analysis:
Data-flow-based Approach
[Cullmann and Martin, WCET 2007]

Example:

```c
for (x < 6) {
    y++;
    if (y % 2 == 0)
        x++;  
    else
        x = x+2;
    z++;
}
```

1. x, y, and z are potential loop counters

2. Invariants:
x' - x in [1,2]
y' - y in [1,1]
z' - z in [1,1]
Loop Bound Analysis: Data-flow-based Approach
[Cullmann and Martin, WCET 2007]

Example:

```c
for (x < 6) {
    y++;
    if (y % 2 == 0)
        x++;
    else
        x = x+2;
    z++;
}
```

1. x, y, and z are potential loop counters
2. Invariants:
   - $x' - x \in [1,2]$
   - $y' - y \in [1,1]$
   - $z' - z \in [1,1]$
3. Loop bound:
   - 6 assuming $x \geq 0$ initially
Combination of multiple analyses:
1. **Slicing**: eliminate code that is irrelevant for loop termination
2. **Value analysis**: determine possible values of all variables in slice
3. **Invariant analysis**: determine variables that do not change during loop execution
4. Loop bound = set of possible valuations of non-invariant variables

*Program slicing* is the computation of the set of programs statements, the program slice, that may affect the values at some point of interest, referred to as a *slicing criterion.*
Step 1: Slicing with slicing criterion \((i <= INPUT)\)

```c
int OUTPUT = 0;
int i = 1;
while (i <= INPUT) {
    OUTPUT += 2;
    i += 2;
}
```

```c
int i = 1;
while (i <= INPUT) {
    i += 2;
}
```
Slicing + Value Analysis + Invariant Analysis [Ermedahl et al., WCET 2007]

Step 2: Value Analysis

Observation:
If the loop terminates, the program can only be in any particular state once.
$\rightarrow$ Determine number of states the program can be in at the loop header.

```c
int i = 1;
while (i <= INPUT) {
    i += 2;
}
```

Value Analysis:
- INPUT in $[10, 20]$ (assumption)
- $i$ in $[1, 20]$, $i \% 2 = 1$
Slicing + Value Analysis + Invariant Analysis
[Ermedahl et al., WCET 2007]

**Step 2: Value Analysis**

**Observation:**
If the loop terminates, the program can only be in any particular state once.
→ Determine number of states the program can be in at the loop header.

```c
int i = 1;
while (i <= INPUT) {
    i += 2;
}
```

**Value Analysis:**
*INPUT in [10, 20] (assumption)*
*i in [1, 20], i % 2 = 1*
→ 11 * 10 states
→ Loop bound 110!
Step 3: Invariant Analysis

Observation:
Value of INPUT is not completely known, but INPUT does not change during loop.
→ Determine variables that are invariant during loop.

```c
int i = 1;
while (i <= INPUT) {
    i += 2;
}
```

Value Analysis:
INPUT in [10, 20] (assumption)
i in [1, 20], i % 2 = 1
Slicing + Value Analysis + Invariant Analysis
[Ermedahl et al., WCET 2007]

Step 3: Invariant Analysis

Observation:
Value of INPUT is not completely known, but INPUT does not change during loop.
→ Determine variables that are invariant during loop.

```cpp
int i = 1;
while (i <= INPUT) {
    i += 2;
}
```

Value Analysis:
- INPUT in [10, 20] (assumption)
- i in [1, 20], i % 2 = 1
→ INPUT is invariant!
→ Loop bound 10!
Reduction: Loop Bound Analysis to Value Analysis

Instrument program with counters of loop iterations and other interesting events
Reduction: Loop Bound Analysis to Value Analysis

Instrument program with counters of loop iterations and other interesting events

Upper bound for loopc is loop bound!
Reduction: Loop Bound Analysis to Value Analysis

Instrument program with counters of loop iterations and other interesting events

Upper bound for loopc is loop bound!
Requires very powerful relational analysis…
Summary

- Interval Analysis:
  A non-relational value analysis
- Widenings for termination in the presence of Infinite Ascending Chains
- Narrowings to recover precision
- Basic Approach to Loop Bound Analysis based on Value Analysis
Outlook

- Cache Abstractions
- Schedulability Analysis
- Cache-Related Preemption Delay
- Predictable Microarchitectures