Using Widenings/Narrowings in Data Flow Analyses

An introduction

Vesal Maynard Vojdani
vesal@ut.ee

Tartu University
Overview

• Introduction ← you are here!
• Galois Connections
• From MOP to MFP
• Insufficiency of Galois connections
• Widenings/Narrowings
• Examples
Interval Analysis

- The goal is to do an interval analysis on the following simple program.

```plaintext
a := 0;
while a < 10 do
  a := a + 1;
```

```
1
   |   
   v   
 a := 0
  2
  /  
/    
   g   a<10
     3
     |    
   v    
 a := a+1
  4
   skip

h    f

b
```
Standard Semantics

- The standard semantics of a programming language defines how expression modify the state.
- The semantics is embodied in the transfer functions of the graph.
- The analysis must take into account:
  - all possible input states
  - all possible executions
All possible inputs
Sets of States Semantics

- To analyse a program for all possible input states, we have to use the Set of States semantics.
- It is naturally derived from the Standard semantics.
- Most precise domain, but impossible to use.
- Approximation is necessary.
The Lattice ordering

- Interesting properties are undecidable.
- Analyses must err on the safe side.
- The Lattice ordering: $x \sqsubseteq y$ means:
  - The set of programs satisfying $x$ is included in the set of programs satisfying $y$.
  - The state $y$ is a correct approximation of $x$.
  - The state $x$ is more precise than $y$.
**Galois connections**

- Galois connection (more precisely adjunctions):

\[
\alpha: X \to Y \\
\gamma: Y \to X
\]

total functions that satisfy

\[
\alpha(x) \sqsubseteq y \iff x \sqsubseteq \gamma(y)
\]
Meaning of Galois connections

\[ \alpha(x) \sqsubseteq y \iff x \sqsubseteq \gamma(y) \]

- \( \alpha(x) \) is the most precise approximation of \( x \).
- \( \gamma(y) \) is the most general element, which can be soundly approximated by \( y \).
Moving to the interval domain

- We can now abstract
  - Sets of State Semantics (visualize as points in space)
  - Collecting Semantics (abstract with projection, concretize as grid)
  - Interval domain (project to interval, concretizes to rectangle)
- The transfer functions are induced by our abstraction:

\[ f' = \alpha \circ f \circ \gamma \]
All possible executions
Notation

- The Control Flow Graph $G = (N, E, s)$, where
  - $N$ is the set of nodes.
  - $E = N \times N$
  - $s$ is the initial node.
- The analysis is an annotation $N \rightarrow D$.
- The transfer functions $tf : E \rightarrow (D \rightarrow D)$
- $\iota$ is the initial state.
**Merge Over all Paths**

**Definition 1.** The path semantics $[\pi]_{tf}$ is simply the composition of the transfer functions along that path

$$
[\varepsilon]_{tf} = id_{D \rightarrow D}
$$

$$
[e_1, \ldots, e_n]_{tf} = [e_2, \ldots, e_n]_{tf} \circ tf(e_1)
$$

**Definition 2.** The MOP solution to the data flow problem is defined at each node by

$$
MOP(n) = \bigcup \left\{ [\pi]_{tf}(\ell) \mid \pi \text{ is a path from } s \text{ to } n \right\}
$$
Least Fixed-point

MOP is not calculable.

Definition 3. The least (minimal) fixed point $\text{MFP}(n)$ is the least solution to the following system:

$$
\text{MFP}(n) = \begin{cases} 
\top & \text{if } n = s \\
\bigcup \left\{ tf(e)(\text{MFP}(n')) \mid e = (n', n) \in E \right\} & \text{otherwise}
\end{cases}
$$

This is correct with respect to MOP, i.e.

$$
\forall n : \text{MOP}(n) \sqsubseteq \text{MFP}(n).
$$
The analysis is implemented in PAG – a Program Analysis Generator. See [www.absint.com](http://www.absint.com) for further information.

The output represents functions.

\[ f(x) = \begin{cases} 
(0, 10) & \text{if } x = a \\
(-\infty, \infty) & \text{otherwise}
\end{cases} \]

is written as

\[ D \rightarrow (\perp, \top) \]

\[ 0 \rightarrow (0, 10) \]
Testing the Analysis
Testing the Analysis

```
0: program loop

3: a := (a+1)

1: a := 0

2: while (a<10)

4: end (loop)

D -> (⊥,T)
```

true_edge ⊥ false_edge ⊥ normal ⊥ normal
Testing the Analysis

Diagram:

0: program loop

1: a := 0

2: while (a<10)

3: a := (a+1)

4: end (loop)

Decision points:

D -> (⊥, T)
0 -> (0, 0)

True edge: true_edge
False edge: false_edge
Testing the Analysis

0: program loop

3: a := (a+1)

1: a := 0

2: while (a<10)

4: end (loop)

normal

D → (⊥,T)
0 → (0,0)

true_edge

D → (⊥,T)

false_edge

normal

⊥
Testing the Analysis
Testing the Analysis

3: \( a := (a+1) \)

0: program loop

\( D \rightarrow (\bot, T) \) normal \( D \rightarrow (\bot, T) \)

\( O \rightarrow (1, 1) \)

1: \( a := 0 \)

\( D \rightarrow (\bot, T) \)

\( O \rightarrow (0, 0) \) normal

2: while \((a < 10)\) true_edge

\( D \rightarrow (\bot, T) \)

\( O \rightarrow (0, 1) \)

false_edge 1

4: end (loop)
Testing the Analysis
Testing the Analysis
Testing the Analysis

Data Flow Analysis – p. 16/31
Testing the Analysis

3: a := (a+1)  
0: program loop

D -> (⊥,T)  
0 -> (1,4)  

1: a := 0

D -> (⊥,T)  
0 -> (0,0)

2: while (a<10)

D -> (⊥,T)  
0 -> (0,3)  

true_edge

false_edge  

4: end (loop)
Testing the Analysis
Testing the Analysis

3: $a := (a+1)$

0: program loop

D $\rightarrow (⊥, T)$
0 $\rightarrow (1, 5)$

1: $a := 0$

D $\rightarrow (⊥, T)$
0 $\rightarrow (0, 0)$

2: while $(a < 10)$

D $\rightarrow (⊥, T)$
0 $\rightarrow (0, 4)$

true_edge

false_edge

4: end (loop)
Testing the Analysis

Data Flow Analysis – p. 16/31
Testing the Analysis

3: a := (a+1)

0: program loop

D -> (⊥,T)
0 -> (1,6)

1: a := 0

D -> (⊥,T)
0 -> (0,0)

2: while (a<10)

true_edge

D -> (⊥,T)
0 -> (0,5)

false_edge

4: end (loop)
Testing the Analysis

3: a := (a+1)

0: program loop

D -> (1,T)
0 -> (1,6)

normal

1: a := 0

normal

2: while (a<10)

d -> (1,T)
0 -> (0,0)

true_edge

D -> (1,T)
0 -> (0,6)

false_edge

4: end (loop)
Testing the Analysis

Data Flow Analysis – p. 16/31
Testing the Analysis
Testing the Analysis
Testing the Analysis
Testing the Analysis

3: a := (a+1)

0: program loop

D -> (⊥,T)
0 -> (1,9)

normal

1: a := 0

D -> (⊥,T)
0 -> (0,0)

normal

2: while (a<10)

D -> (⊥,T)
0 -> (0,8)

false_edge

4: end (loop)
Testing the Analysis
Testing the Analysis

3: a := (a+1)

0: program loop

D -> (1,T)
0 -> (1,10)

normal

1: a := 0

D -> (1,T)
0 -> (0,0)

normal

2: while (a<10)

true_edge

D -> (1,T)
0 -> (0,9)

false_edge

4: end (loop)
Testing the Analysis
Testing the Analysis

3: a := (a+1)

0: program loop

D -> (⊥,T)
0 -> (1,10)

1: a := 0

D -> (⊥,T)
0 -> (0,0)

2: while (a<10)

true_edge

D -> (⊥,T)
0 -> (0,9)

false_edge

D -> (⊥,T)
0 -> (10,10)

4: end (loop)
This will not do

- You might as well run the program. We were lucky the analysis terminated.
- We need further approximation. . . But that’s not possible!
  - For each program there is a finite domain.
  - But no single finite domain will do for all programs.
  - Can we find the suitable domain by some form of textual analysis?
Textual analysis?

- Cousot brings the following examples
  - McCarty’s 91-function (Bourdoncle)
  - Rational congruence analysis (Granger)
  - Linear inequality analysis (Cousot/Halbwachs)
- Here’s the 91-function

\[
f(n) = \begin{cases} 
  n - 10 & \text{if } n > 100 \\
  f(f(n + 11)) & \text{otherwise}
\end{cases}
\]
How did they do it?
Widenings

- A widening is a lattice operator $\nabla : L \times L \rightarrow L$ such that
  - It’s an upper bound operator
  - For all increasing chains $x_0 \sqsubseteq x_1 \sqsubseteq \cdots$, the increasing chain defined by
    \[
    \begin{align*}
    y_0 &= x_0 \\
    y_{i+1} &= y_i \nabla x_{i+1}
    \end{align*}
    \]
is not strictly increasing.
Iterating with widenings

- The upward iteration sequence with widening

\[ X_0 = \bot \]

\[ X_{i+1} = \begin{cases} 
X_i & \text{if } F(X_i) \subseteq X_i \\
X_i \triangledown F(X_i) & \text{otherwise}
\end{cases} \]

stabilizes to a safe approximation of the fix-point of \( F \).

- \( F \) is reductive at that point.
Why is that?

- If we reach a point in $\text{Red}(F)$, then we’re ok.
  
  $$X_0 = \perp$$

  $$X_{i+1} = \begin{cases} 
  X_i & \text{if } F(X_i) \subseteq X_i \\
  X_i \triangledown F(X_i) & \text{otherwise}
  \end{cases}$$

- The sequence is clearly ascending.

- If $\exists i : F(X_i) \subseteq X_i$, then
  
  - The sequence will immediately stabilize.
  - Since $X_i \in \text{Red}(F)$, we have $\text{lfp}(F) \subseteq X_i$. 


Will we get there?

- Assume $\forall i : X_i \sqsubseteq F(X_i)$, then
  
  \[
  X_0 = \perp \\
  X_{i+1} = X_i \triangledown F(X_i)
  \]

- This is just the widening of the following ascending chain

  \[
  Y_0 = \perp \\
  Y_{i+1} = F(X_i)
  \]

QED! Really!
Widening for Upper bounds

• The upper bound of an interval, can be widened by

\[ x \triangledown y = \begin{cases} \infty & \text{if } x < y \\ x & \text{otherwise} \end{cases} \]

• It’s a widening! Any chain is stabilized immediately.
Approximating the Fixed-Point

1: a := 0

2: while (a < 10)
   true edge
   normal
   \[\top\]
   false edge
   normal
   \[\top\]

3: a := (a + 1)

0: program loop

\[\top\]
Approximating the Fixed-Point

0: program loop

1: a := 0

2: while (a<10)

3: a := (a+1)

4: end (loop)

Data Flow Analysis – p. 26/31
Approximating the Fixed-Point

0: program loop

1: a := 0

2: while (a<10)

3: a := (a+1)

4: end (loop)

Data Flow Analysis – p. 26/31
Approximating the Fixed-Point
Approximating the Fixed-Point

3: a := (a+1)  

2: while (a<10)
   true_edge
   D -> (⊥,T)
   0 -> (1,1)

1: a := 0
   normal
   D -> (⊥,T)
   0 -> (0,0)

0: program loop
   D -> (⊥,T)
   0 -> (1,1)

4: end (loop)
   false_edge
   D -> (⊥,T)
   0 -> (0,0)

widening about to happen!
Approximating the Fixed-Point

3: a := (a+1)

0: program loop

D -> (⊥,T)
0 -> (1,1)

1: a := 0

D -> (⊥,T)
0 -> (0,0)

2: while (a<10)

true_edge

D -> (⊥,T)
0 -> (0,T)

false_edge

4: end (loop)
Approximating the Fixed-Point
Approximating the Fixed-Point
Approximating the Fixed-Point

3: a := (a+1)
0: program loop

D -> (⊥,T)
0 -> (1,T)
normal

1: a := 0

D -> (⊥,T)
0 -> (0,0)
normal

2: while (a<10)

D -> (⊥,T)
0 -> (0,T)
true_edge

D -> (⊥,T)
0 -> (10,T)
false_edge

4: end (loop)
Narrowings

- Since $F$ is reductive at the point reached through the upward iteration, we can fine-tune the result.
- Downward sequence may also not reach the fix-point, so we might need a similar procedure, called narrowings, to ensure termination.
- The only difference:

$$\forall x, y \in L : (y \sqsubseteq x) \implies (y \sqsubseteq (x \triangle y) \sqsubseteq x)$$
Not the dual of widening

- The difference:

\[ \forall x, y \in L : (y \sqsubseteq x) \implies (y \sqsubseteq (x \triangle y) \sqsubseteq x) \]

means it is not a lower bound operator.

- While narrowings keep the sequence in \text{Red}, the dual of widenings would step out of \text{Red} and eventually into \text{Ext}.
Fine-tuning the result
Fine-tuning the result

```
3: a := (a+1)
   \__________
    | normal
    v
   2: while (a<10)
       \__________
        | true_edge
        v
      1: a := 0
          \__________
           | normal
           v
          0: program loop
              \__________
               | normal
               v
              D -> (⊥,T)
               v
             0 -> (1,T)
```

```
D -> (⊥,T)
0 -> (0,T)
```

```
D -> (⊥,T)
0 -> (0,0)
```

```
D -> (⊥,T)
0 -> (10,T)
```

```
D -> (⊥,T)
0 -> (1,T)
```

```
D -> (⊥,T)
0 -> (0,0)
```

```
D -> (⊥,T)
0 -> (10,T)
```
Fine-tuning the result

Data Flow Analysis – p. 29/31
Fine-tuning the result
Fine-tuning the result

Data Flow Analysis – p. 29/31
Fine-tuning the result

3: \( a \leftarrow (a+1) \)

0: program loop

D \rightarrow (\bot, T)
0 \rightarrow (1, 10)

1: \( a \leftarrow 0 \)

D \rightarrow (\bot, T)
0 \rightarrow (0, 0)

2: while \((a < 10)\)

D \rightarrow (\bot, T)
0 \rightarrow (0, 9)

true_edge

false_edge

4: end (loop)

D \rightarrow (\bot, T)
0 \rightarrow (10, 10)
Fine-tuning the result

3: a := (a+1)

0: program loop

D \rightarrow (\perp,T)
0 \rightarrow (1,10)

1: a := 0

D \rightarrow (\perp,T)
0 \rightarrow (0,0)

2: while (a<10)

true_edge

D \rightarrow (\perp,T)
0 \rightarrow (0,9)

false_edge

D \rightarrow (\perp,T)
0 \rightarrow (10,10)

4: end (loop)
Examples
Questions