Outline

Background
- Iterative program analysis
- Abstract interpretation

Intraprocedural analysis
- Overview
- Path expressions
- Compositional Recurrence Analysis
- Proving soundness

Interprocedural analysis
- Functional approach
- Newtonian program analysis
- Newtonian program analysis via tensor product
- Newtonian program analysis and Gauss-Jordan elimination

PRINCETON UNIVERSITY

WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

Algebraic program analysis

Consists of:

① **Semantic algebra** $\mathcal{D} = \langle D, \otimes, \oplus, *, 0, 1 \rangle$

- D: Space of program properties
- $\otimes : D \times D \rightarrow D$: sequencing operator
- $\oplus : D \times D \rightarrow D$: choice operator
- $* : D \rightarrow D$: iteration operator
- $0, 1 \in D$: unit of $\oplus$, $\otimes$ respectively

② **Semantic function** $\mathcal{D}[\![\cdot]\!]$: *Edge* $\rightarrow D$

Algebraic program analysis

Consists of:

① **Semantic algebra** $\mathcal{D} = \langle D, \otimes, \oplus, *, 0, 1 \rangle$

- D: Space of program properties
- $\otimes : D \times D \to D$: sequencing operator
- $\oplus : D \times D \to D$: choice operator
- $* : D \to D$: iteration operator
- $0, 1 \in D$: unit of $\oplus$, $\otimes$ respectively

② **Semantic function** $\mathcal{D}[\![\cdot]\!]$: *Edge* $\to D$

L : Space of program properties

$\sqsubseteq \subseteq L \times L$: approximation order

$\sqcup : L \times L \to L$: join operator

$\nabla : L \times L \to L$: widening operator

$\bot \in L$: least element

$\mathcal{L}[\![\cdot]\!]$: *Edge* $\to (L \to L)$

Algebraic program analysis

Consists of:

① **Semantic algebra** $\mathcal{D} = \langle D, \otimes, \oplus, *, 0, 1 \rangle$

- D: Space of program properties
- $\otimes : D \times D \rightarrow D$: sequencing operator
- $\oplus : D \times D \rightarrow D$: choice operator
- $* : D \rightarrow D$: iteration operator
- $0, 1 \in D$: unit of $\oplus$, $\otimes$ respectively

② **Semantic function** $\mathcal{D}[\![\cdot]\!]$: *Edge* $\rightarrow D$

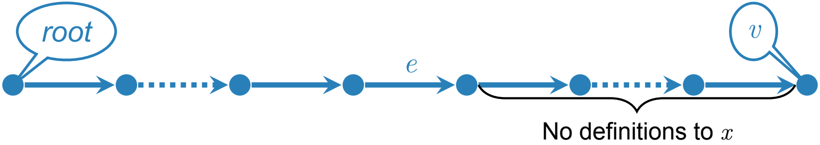
Effective denotational semantics: compute the “meaning” of a program by evaluating its syntax in a semantic algebra

$$\begin{aligned} \mathcal{D}[\![S_1; S_2]\!] &= \mathcal{D}[\![S_1]\!] \otimes \mathcal{D}[\![S_2]\!] \\ \mathcal{D}[\![\textbf{if}(*)\{S_1\}\textbf{else}\{S_2\}]\!] &= \mathcal{D}[\![S_1]\!] \oplus \mathcal{D}[\![S_2]\!] \\ \mathcal{D}[\![\textbf{while}(*)\{S\}]\!] &= (\mathcal{D}[\![P]\!])^* \end{aligned}$$

Reaching definitions analysis

If a control flow edge e is an assignment `x` *:=t, then we say that e is a* ***definition*** *that* ***defines*** *x.*

A definition e of a variable `x` reaches a vertex v if there exists a path from the root to v of the form:



No definitions to x

Iterative reaching definitions:

- $L \triangleq 2^{\textit{Def}}$
- $\mathcal{L}[\![e : x := t]\!](R) \triangleq (R \setminus \{e' : e' \text{ defines x}\}) \cup \{e\}$
- $R_1 \sqsubseteq R_2 \iff R_1 \subseteq R_2$
- $R_1 \sqcup R_2 \triangleq R_1 \cup R_2$
- $R_1 \triangledown R_2 \triangleq R_1 \cup R_2$
- $\bot \triangleq \emptyset$

Iterative reaching definitions:

- $L \triangleq 2^{\textit{Def}}$
- $\mathcal{L}[\![e : x := t]\!](R) \triangleq (R \setminus \{e' : e' \text{ defines x}\}) \cup \{e\}$
- $R_1 \sqsubseteq R_2 \iff R_1 \subseteq R_2$
- $R_1 \sqcup R_2 \triangleq R_1 \cup R_2$
- $R_1 \triangledown R_2 \triangleq R_1 \cup R_2$
- $\bot \triangleq \emptyset$

Algebraic reaching definitions :

- $D = (2^{\textit{Def}}) \times (2^{\textit{Def}})$
- $\mathcal{D}[\![e : x := t]\!] \triangleq (\{e\}, \{e' : e' \text{ defines x}\})$

Iterative reaching definitions:

- $L \triangleq 2^{\textit{Def}}$
- $\mathcal{L}[\![e : x := t]\!](R) \triangleq (R \setminus \{e' : e' \text{ defines x}\}) \cup \{e\}$
- $R_1 \sqsubseteq R_2 \iff R_1 \subseteq R_2$
- $R_1 \sqcup R_2 \triangleq R_1 \cup R_2$
- $R_1 \triangledown R_2 \triangleq R_1 \cup R_2$
- $\bot \triangleq \emptyset$

Algebraic reaching definitions :

- $D = (2^{\textit{Def}}) \times (2^{\textit{Def}})$
- $\mathcal{D}[\![e : x := t]\!] \triangleq (\{e\}, \{e' : e' \text{ defines x}\})$
- $(G_1, K_1) \otimes (G_2, K_2) \triangleq ((G_1 \setminus K_2) \cup G_2, (K_1 \setminus G_2) \cup K_2)$

Iterative reaching definitions:

- $L \triangleq 2^{\textit{Def}}$
- $\mathcal{L}[\![e : x := t]\!](R) \triangleq (R \setminus \{e' : e' \text{ defines x}\}) \cup \{e\}$
- $R_1 \sqsubseteq R_2 \iff R_1 \subseteq R_2$
- $R_1 \sqcup R_2 \triangleq R_1 \cup R_2$
- $R_1 \triangledown R_2 \triangleq R_1 \cup R_2$
- $\bot \triangleq \emptyset$

Algebraic reaching definitions :

- $D = (2^{\textit{Def}}) \times (2^{\textit{Def}})$
- $\mathcal{D}[\![e : x := t]\!] \triangleq (\{e\}, \{e' : e' \text{ defines x}\})$
- $(G_1, K_1) \otimes (G_2, K_2) \triangleq ((G_1 \setminus K_2) \cup G_2, (K_1 \setminus G_2) \cup K_2)$
- $(G_1, K_1) \oplus (G_2, K_2) \triangleq (G_1 \cup G_2, K_1 \cap K_2)$

Iterative reaching definitions:

- $L \triangleq 2^{\textit{Def}}$
- $\mathcal{L}[\![e : x := t]\!](R) \triangleq (R \setminus \{e' : e'$ defines x$\}) \cup \{e\}$
- $R_1 \sqsubseteq R_2 \iff R_1 \subseteq R_2$
- $R_1 \sqcup R_2 \triangleq R_1 \cup R_2$
- $R_1 \triangledown R_2 \triangleq R_1 \cup R_2$
- $\perp \triangleq \emptyset$

Algebraic reaching definitions :

- $D = (2^{\textit{Def}}) \times (2^{\textit{Def}})$
- $\mathcal{D}[\![e : x := t]\!] \triangleq (\{e\}, \{e' : e'$ defines x$\})$
- $(G_1, K_1) \otimes (G_2, K_2) \triangleq ((G_1 \setminus K_2) \cup G_2, (K_1 \setminus G_2) \cup K_2)$
- $(G_1, K_1) \oplus (G_2, K_2) \triangleq (G_1 \cup G_2, K_1 \cap K_2)$
- $(G, K)^* \triangleq (G, \emptyset)$

```
       while(*){
         if(*){
$x_1$ :      x := 1;
$y_1$ :      y := 1;
         } else {
$y_2$ :      y := 2;
         }
       }
$x_0$ : x := 0;
```

```
        while(*){
          if(*){
$x_1$ :       x := 1;  } $(\{x_1\}, \{x_1, x_0\})$
$y_1$ :       y := 1;  } $(\{y_1\}, \{y_1, y_2\})$
          } else {
$y_2$ :       y := 2;
          }
        }
$x_0$ : x := 0;
```

```
        while(*){
          if(*){
$x_1$ :       x := 1;  }  $(\{x_1, y_1\}, \{x_1, x_0, y_1, y_2\})$
$y_1$ :       y := 1;  }
          } else {
$y_2$ :       y := 2;
          }
        }
$x_0$ : x := 0;
```

```
        while(*){
          if(*){
$x_1$ :       x := 1;  }  $(\{x_1, y_1\}, \{x_1, x_0, y_1, y_2\})$
$y_1$ :       y := 1;  }
          } else {
$y_2$ :       y := 2;  }  $(\{y_2\}, \{y_1, y_2\})$
          }
        }
$x_0$ : x := 0;
```

```
        while(*){
          if(*){
$x_1$ :         x := 1;
$y_1$ :         y := 1;
          } else {
$y_2$ :         y := 2;
          }
        }
$x_0$ : x := 0;
```

$(\{x_1, y_1, y_2\}, \{y_1, y_2\})$

```
          while(*){
             if(*){
$x_1$ :         x := 1;
$y_1$ :         y := 1;
             } else {
$y_2$ :         y := 2;
             }
          }
$x_0$ :   x := 0;
```

$$\left.\vphantom{\begin{matrix}a\\a\\a\\a\\a\\a\\a\\a\end{matrix}}\right\}(\{x_1, y_1, y_2\}, \emptyset)$$

```
           while(*){
             if(*){
$x_1$ :          x := 1;
$y_1$ :          y := 1;
             } else {                 $(\{x_1, y_1, y_2\}, \emptyset)$
$y_2$ :          y := 2;
             }
           }
$x_0$ :    x := 0;                    $(\{x_0\}, \{x_0, x_1\})$
```

```
        while(*){
          if(*){
$x_1$ :       x := 1;
$y_1$ :       y := 1;
          } else {
$y_2$ :       y := 2;
          }
        }
$x_0$ : x := 0;
```

$$\left.\vphantom{\begin{matrix}a\\a\\a\\a\\a\\a\\a\\a\\a\end{matrix}}\right\}(\{x_0, y_1, y_2\}, \{x_0, x_1\})$$

Outline

Background
- Iterative program analysis
- Abstract interpretation

Intraprocedural analysis
- Overview
- Path expressions
- Compositional Recurrence Analysis
- Proving soundness

Interprocedural analysis
- Functional approach
- Newtonian program analysis
- Newtonian program analysis via tensor product
- Newtonian program analysis and Gauss-Jordan elimination

Path expressions [Tarjan '81]

Let $G = \langle \textit{Loc}, \textit{Edge}, \textit{root} \rangle$ be a control flow graph.
A *path expression* of G is a regular expression E over the alphabet *Edge* such that each word recognized by E corresponds to a path in G.

$$E, F \in \mathsf{RegExp}(G) ::= e \in \textit{Edge} \mid E + F \mid EF \mid E^* \mid 0 \mid 1$$

Path expressions [Tarjan ’81]

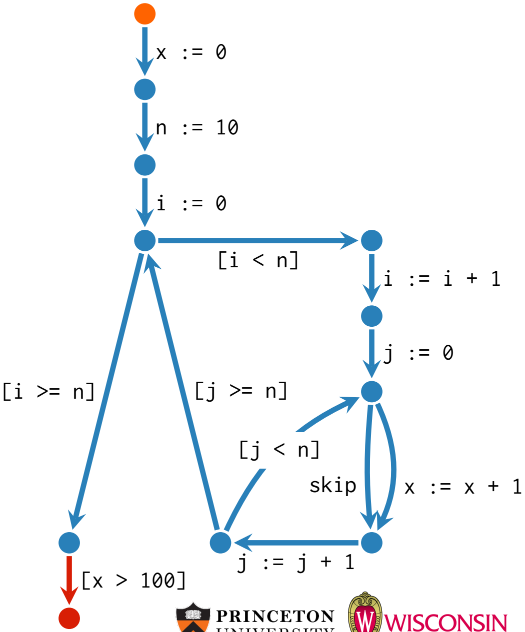
Let $G = \langle \mathit{Loc}, \mathit{Edge}, \mathit{root} \rangle$ be a control flow graph.
A *path expression* of G is a regular expression E over the alphabet *Edge* such that each word recognized by E corresponds to a path in G.

$$E, F \in \text{RegExp}(G) ::= e \in \mathit{Edge} \mid E + F \mid EF \mid E^* \mid 0 \mid 1$$

If $u, v \in \mathit{Loc}$ are control locations, a *path expression from* u *to* v is a path expression that recognizes the set of all paths from u to v in G.

```
        x := 0
        n := 10
        i := 0
outer:  if(i >= n):
           goto end
        i := i + 1
inner:  j := 0
        if(*):
           x := x + 1
        j := j + 1
        if(j < n):
           goto inner
        goto outer
end:    assert(x <= 100)
```

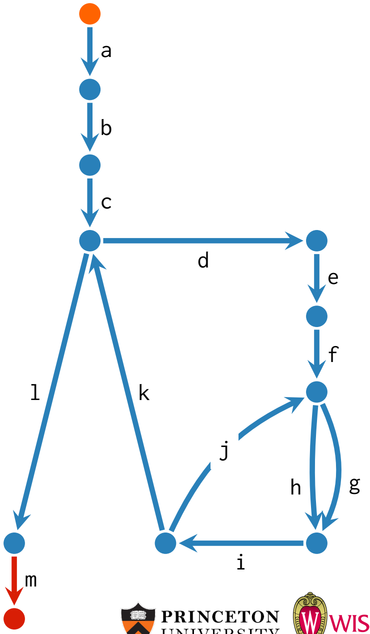
```
        x := 0
        n := 10
        i := 0
outer:  if(i >= n):
           goto end
        i := i + 1
inner:  j := 0
        if(*):
           x := x + 1
        j := j + 1
        if(j < n):
           goto inner
        goto outer
end:    assert(x <= 100)
```



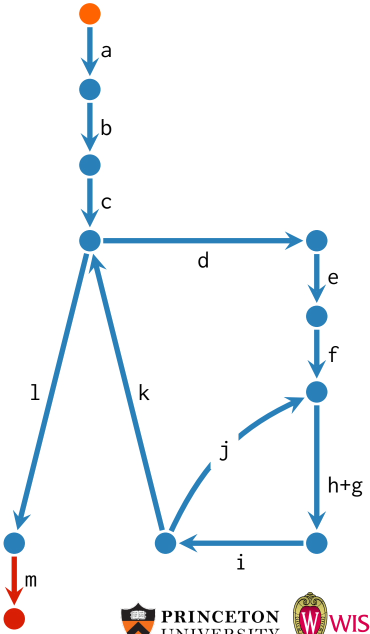
PRINCETON UNIVERSITY

WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

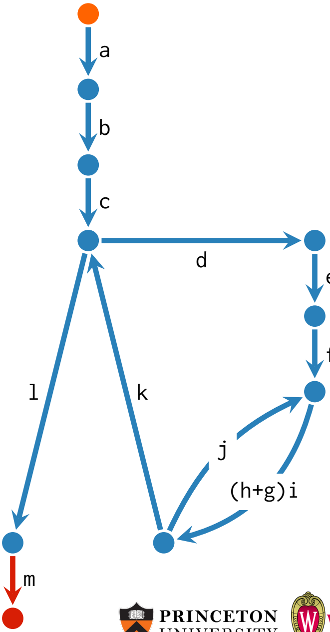
```
        x := 0
        n := 10
        i := 0
outer:  if(i >= n):
           goto end
        i := i + 1
inner:  j := 0
        if(*):
           x := x + 1
        j := j + 1
        if(j < n):
           goto inner
        goto outer
end:    assert(x <= 100)
```



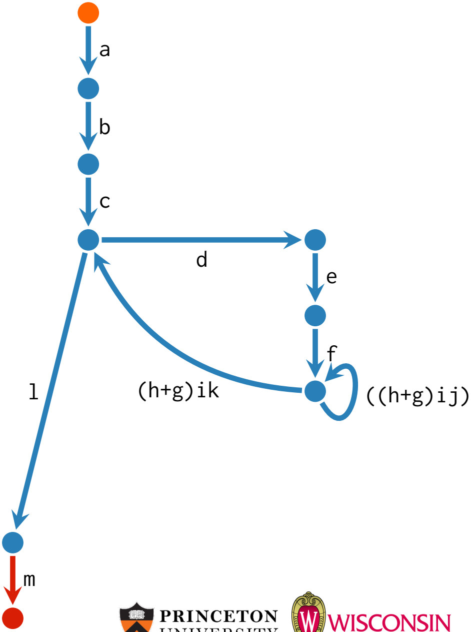
```
        x := 0
        n := 10
        i := 0
outer:  if(i >= n):
           goto end
        i := i + 1
inner:  j := 0
        if(*):
           x := x + 1
        j := j + 1
        if(j < n):
           goto inner
        goto outer
end:    assert(x <= 100)
```



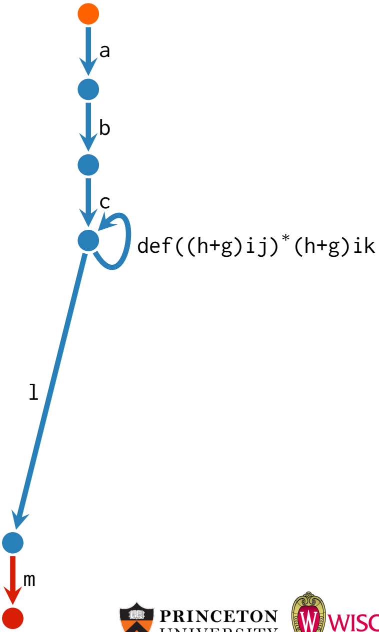
```
        x := 0
        n := 10
        i := 0
outer:  if(i >= n):
           goto end
        i := i + 1
inner:  j := 0
        if(*):
           x := x + 1
        j := j + 1
        if(j < n):
           goto inner
        goto outer
end:    assert(x <= 100)
```



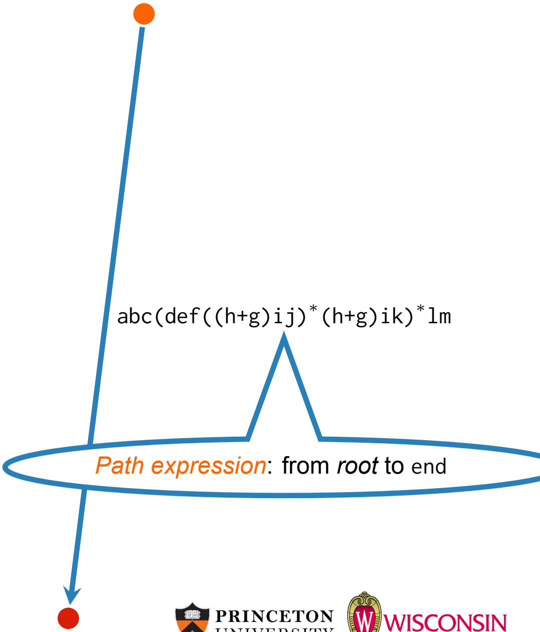
```
       x := 0
       n := 10
       i := 0
outer: if(i >= n):
          goto end
       i := i + 1
inner: j := 0
       if(*):
          x := x + 1
       j := j + 1
       if(j < n):
          goto inner
       goto outer
end:   assert(x <= 100)
```



```
        x := 0
        n := 10
        i := 0
outer:  if(i >= n):
           goto end
        i := i + 1
inner:  j := 0
        if(*):
           x := x + 1
        j := j + 1
        if(j < n):
           goto inner
        goto outer
end:    assert(x <= 100)
```



```
        x := 0
        n := 10
        i := 0
outer:  if(i >= n):
           goto end
        i := i + 1
inner:  j := 0
        if(*):
           x := x + 1
        j := j + 1
        if(j < n):
           goto inner
        goto outer
end:    assert(x <= 100)
```



$abc(def((h+g)ij)^*(h+g)ik)^*lm$

Path expression: from *root* to end

Running an algebraic program analysis

① Compute a *path expression* from the program entry to each vertex

② Evaluate the path expressions in the *semantic algebra* defining the analysis.

$$\begin{aligned} \mathcal{D}[\![S_1 S_2]\!] &= \mathcal{D}[\![S_1]\!] \otimes \mathcal{D}[\![S_2]\!] \\ \mathcal{D}[\![S_1 + S_2]\!] &= \mathcal{D}[\![S_1]\!] \oplus \mathcal{D}[\![S_2]\!] \\ \mathcal{D}[\![S^*]\!] &= (\mathcal{D}[\![P]\!])^* \end{aligned}$$

Running an algebraic program analysis

① Compute a *path expression* from the program entry to each vertex

② Evaluate the path expressions in the *semantic algebra* defining the analysis.

$$\begin{aligned} \mathcal{D}\llbracket S_1 S_2 \rrbracket &= \mathcal{D}\llbracket S_1 \rrbracket \otimes \mathcal{D}\llbracket S_2 \rrbracket \\ \mathcal{D}\llbracket S_1 + S_2 \rrbracket &= \mathcal{D}\llbracket S_1 \rrbracket \oplus \mathcal{D}\llbracket S_2 \rrbracket \\ \mathcal{D}\llbracket S^* \rrbracket &= (\mathcal{D}\llbracket P \rrbracket)^* \end{aligned}$$

Tarjan's algorithm [Tarjan '81]: do both steps & avoid repeated work

Running an algebraic program analysis

① Compute a *path expression* from the program entry to each vertex

② Evaluate the path expressions in the *semantic algebra* defining the analysis.

$$
\begin{aligned}
\mathcal{D}[\![S_1 S_2]\!] &= \mathcal{D}[\![S_1]\!] \otimes \mathcal{D}[\![S_2]\!] \\
\mathcal{D}[\![S_1 + S_2]\!] &= \mathcal{D}[\![S_1]\!] \oplus \mathcal{D}[\![S_2]\!] \\
\mathcal{D}[\![S^*]\!] &= (\mathcal{D}[\![P]\!])^*
\end{aligned}
$$

Tarjan's algorithm [Tarjan '81]: do both steps & avoid repeated work

More path-expression/elimination algorithms: [Sreedhar, Gao, Lee '98], [Scholz, Blieberger '07], ...

Outline

Background

- Iterative program analysis
- Abstract interpretation

Intraprocedural analysis

- Overview
- Path expressions
- Compositional Recurrence Analysis
- Proving soundness

Interprocedural analysis

- Functional approach
- Newtonian program analysis
- Newtonian program analysis via tensor product
- Newtonian program analysis and Gauss-Jordan elimination

Compositional Recurrence Analysis (CRA)

[Farzan & Kincaid '15]

- D: set of arithmetic *transition formulas*

$$\mathcal{D}[\![\texttt{x := x + 1}]\!] \triangleq \texttt{x}' = \texttt{x} + 1 \land \texttt{y}' = \texttt{y} \land \texttt{i}' = \texttt{i} \land \texttt{j}' = \texttt{j} \land \texttt{n}' = \texttt{n}$$

Compositional Recurrence Analysis (CRA)

[Farzan & Kincaid '15]

- D: set of arithmetic *transition formulas*

  $\mathcal{D}[\![\texttt{x := x + 1}]\!] \triangleq \mathtt{x}' = \mathtt{x} + 1 \land \mathtt{y}' = \mathtt{y} \land \mathtt{i}' = \mathtt{i} \land \mathtt{j}' = \mathtt{j} \land \mathtt{n}' = \mathtt{n}$

- $\varphi \otimes \psi \triangleq \exists \mathbf{x}''. \varphi[\mathbf{x}' \mapsto \mathbf{x}''] \land \psi[\mathbf{x} \mapsto \mathbf{x}'']$

Compositional Recurrence Analysis (CRA)

[Farzan & Kincaid '15]

- D: set of arithmetic *transition formulas*

  $\mathcal{D}[\![\texttt{x := x + 1}]\!] \triangleq \texttt{x}' = \texttt{x} + 1 \land \texttt{y}' = \texttt{y} \land \texttt{i}' = \texttt{i} \land \texttt{j}' = \texttt{j} \land \texttt{n}' = \texttt{n}$

- $\varphi \otimes \psi \triangleq \exists \mathbf{x}''. \varphi[\mathbf{x}' \mapsto \mathbf{x}''] \land \psi[\mathbf{x} \mapsto \mathbf{x}'']$
- $\varphi \oplus \psi \triangleq \varphi \lor \psi$

Compositional Recurrence Analysis (CRA)

[Farzan & Kincaid '15]

- D: set of arithmetic *transition formulas*

$$\mathcal{D}[\![\texttt{x := x + 1}]\!] \triangleq \texttt{x}' = \texttt{x} + 1 \land \texttt{y}' = \texttt{y} \land \texttt{i}' = \texttt{i} \land \texttt{j}' = \texttt{j} \land \texttt{n}' = \texttt{n}$$

- $\varphi \otimes \psi \triangleq \exists \mathbf{x}''. \varphi[\mathbf{x}' \mapsto \mathbf{x}''] \land \psi[\mathbf{x} \mapsto \mathbf{x}'']$
- $\varphi \oplus \psi \triangleq \varphi \lor \psi$
- $\varphi^* \triangleq \ldots$

CRA's iteration operator

```
while(i < n):
  if (*):
    x := x + i
  else
    y := y + i
  i := i + 1
```

loop body

$$
\begin{aligned}
& i < n \\
\wedge & \left(\begin{array}{ll} & (x' = x + i \wedge y' = y) \\ \vee & (y' = y + i \wedge x' = x) \end{array} \right) \\
\wedge & i' = i + 1 \\
\wedge & n' = n
\end{aligned}
$$

loop abstraction

$$\exists k. k \geq 0 \wedge i' = i + k \wedge x' + y' = x + y + k(k+1)/2 + k i_0 \wedge x' \geq x \wedge y' \geq y$$

CRA's iteration operator

```
while(i < n):
    if (*):
        x := x + i
    else
        y := y + i
    i := i + 1
```

loop body

$$\begin{array}{l} \mathtt{i} < \mathtt{n} \\ \wedge \left(\begin{array}{ll} & (\mathtt{x}' = \mathtt{x} + \mathtt{i} \wedge \mathtt{y}' = \mathtt{y}) \\ \vee & (\mathtt{y}' = \mathtt{y} + \mathtt{i} \wedge \mathtt{x}' = \mathtt{x}) \end{array} \right) \\ \wedge \mathtt{i}' = \mathtt{i} + 1 \\ \wedge \mathtt{n}' = \mathtt{n} \end{array}$$

recurrences

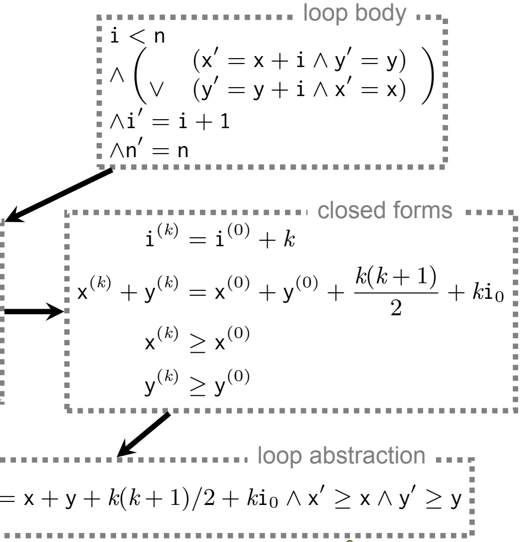
$$\begin{aligned} \mathtt{i}^{(k)} &= \mathtt{i}^{(k-1)} + 1 \\ \mathtt{x}^{(k)} + \mathtt{y}^{(k)} &= \mathtt{x}^{(k-1)} + \mathtt{y}^{(k-1)} + \mathtt{i} \\ \mathtt{x}^{(k)} &\geq \mathtt{x}^{(k-1)} \\ \mathtt{y}^{(k)} &\geq \mathtt{y}^{(k-1)} \end{aligned}$$

loop abstraction

$$\exists k. k \geq 0 \wedge \mathtt{i}' = \mathtt{i} + k \wedge \mathtt{x}' + \mathtt{y}' = \mathtt{x} + \mathtt{y} + k(k+1)/2 + k\mathtt{i}_0 \wedge \mathtt{x}' \geq \mathtt{x} \wedge \mathtt{y}' \geq \mathtt{y}$$

CRA's iteration operator

```
while(i < n):
    if (*):
        x := x + i
    else
        y := y + i
    i := i + 1
```



loop body

$$\begin{aligned} & \mathtt{i} < \mathtt{n} \\ & \wedge \left(\begin{array}{ll} & (\mathtt{x}' = \mathtt{x} + \mathtt{i} \wedge \mathtt{y}' = \mathtt{y}) \\ \vee & (\mathtt{y}' = \mathtt{y} + \mathtt{i} \wedge \mathtt{x}' = \mathtt{x}) \end{array} \right) \\ & \wedge \mathtt{i}' = \mathtt{i} + 1 \\ & \wedge \mathtt{n}' = \mathtt{n} \end{aligned}$$

recurrences

$$\begin{aligned} \mathtt{i}^{(k)} &= \mathtt{i}^{(k-1)} + 1 \\ \mathtt{x}^{(k)} + \mathtt{y}^{(k)} &= \mathtt{x}^{(k-1)} + \mathtt{y}^{(k-1)} + \mathtt{i} \\ \mathtt{x}^{(k)} &\geq \mathtt{x}^{(k-1)} \\ \mathtt{y}^{(k)} &\geq \mathtt{y}^{(k-1)} \end{aligned}$$

closed forms

$$\begin{aligned} \mathtt{i}^{(k)} &= \mathtt{i}^{(0)} + k \\ \mathtt{x}^{(k)} + \mathtt{y}^{(k)} &= \mathtt{x}^{(0)} + \mathtt{y}^{(0)} + \frac{k(k+1)}{2} + k\mathtt{i}_0 \\ \mathtt{x}^{(k)} &\geq \mathtt{x}^{(0)} \\ \mathtt{y}^{(k)} &\geq \mathtt{y}^{(0)} \end{aligned}$$

loop abstraction

$$\exists k. k \geq 0 \wedge \mathtt{i}' = \mathtt{i} + k \wedge \mathtt{x}' + \mathtt{y}' = \mathtt{x} + \mathtt{y} + k(k+1)/2 + k\mathtt{i}_0 \wedge \mathtt{x}' \geq \mathtt{x} \wedge \mathtt{y}' \geq \mathtt{y}$$

Non-Linear Reasoning For Invariant Synthesis

with Jason Breck, John Cyphert, and Thomas Reps

January 12, 2018 @ 15:50, Program Analysis II session.

PRINCETON UNIVERSITY

WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

Outline

Background

- Iterative program analysis
- Abstract interpretation

Intraprocedural analysis

- Overview
- Path expressions
- Compositional Recurrence Analysis
- Proving soundness

Interprocedural analysis

- Functional approach
- Newtonian program analysis
- Newtonian program analysis via tensor product
- Newtonian program analysis and Gauss-Jordan elimination

Relational interpretation

- $D^\natural \triangleq 2^{\textit{Store}\times\textit{Store}}$: set of transition relations
- $R \otimes S \triangleq \{(s, s'') : \exists s'.(s, s') \in R \wedge (s', s'') \in S\}$ is relational composition
- $R \oplus S \triangleq R \cup S$
- $R^* \triangleq$ reflexive, transitive closure of R
- $0 \triangleq \emptyset$
- $1 \triangleq \{\langle s, s\rangle : s \in \textit{Store}\}$
- $\mathcal{D}^\natural[\![e]\!] \triangleq \{(s, s') : s \xrightarrow{e} s'\}$

Soundness relations

Given concrete & abstract semantic algebras:

$$\mathcal{D}^\natural = \langle D^\natural, \otimes^\natural, \oplus^\natural, *^\natural, 0^\natural, 1^\natural \rangle$$
$$\mathcal{D}^\sharp = \langle D^\sharp, \otimes^\sharp, \oplus^\sharp, *^\sharp, 0^\sharp, 1^\sharp \rangle$$

A *soundness relation* is a relation $\Vdash \subseteq D^\natural \times D^\sharp$ such that $0^\natural \Vdash 0^\sharp$, $1^\natural \Vdash 1^\sharp$, and

For all

$c_1, c_2 \in D^\natural$

$a_1, a_2 \in D^\sharp$

such that

$c_1 \Vdash a_1$

$c_2 \Vdash a_2$

Then:

- $c_1 \otimes^\natural c_2 \Vdash a_1 \otimes^\sharp a_2$
- $c_1 \oplus^\natural c_2 \Vdash a_1 \oplus^\sharp a_2$
- $c_1^{*^\natural} \Vdash a_1^{*^\sharp}$

(i.e., $\Vdash$ is a sub-algebra of the direct product $D^\natural \times D^\sharp$).

Soundness relations

Given concrete & abstract semantic algebras:

$$\mathcal{D}^\natural = \langle D^\natural, \otimes^\natural, \oplus^\natural, *^\natural, 0^\natural, 1^\natural \rangle$$
$$\mathcal{D}^\sharp = \langle D^\sharp, \otimes^\sharp, \oplus^\sharp, *^\sharp, 0^\sharp, 1^\sharp \rangle$$

A *soundness relation* is a relation $\Vdash \subseteq D^\natural \times D^\sharp$ such that $0^\natural \Vdash 0^\sharp$, $1^\natural \Vdash 1^\sharp$, and

For all

$c_1, c_2 \in D^\natural$

$a_1, a_2 \in D^\sharp$

Then:

- $c_1 \otimes^\natural c_2 \Vdash a_1 \otimes^\sharp a_2$

If $\forall e \in$ *Edge*, $\mathcal{D}^\natural[\![e]\!] \Vdash \mathcal{D}^\sharp[\![e]\!]$, then
$\forall$ path expressions E: $\mathcal{D}^\natural[\![E]\!] \Vdash \mathcal{D}^\sharp[\![E]\!]$.

(i.e., $\Vdash$ is a sub-algebra of the direct product $D^\natural \times D^\sharp$).

CRA simulates relational interpretation

$R \Vdash \varphi(\mathbf{x}, \mathbf{x}')$ iff $\forall (s, s') \in R.\ \varphi[\mathbf{x} \mapsto s(\mathbf{x}), \mathbf{x}' \mapsto s'(\mathbf{x})]$ holds

CRA simulates relational interpretation

$R \Vdash \varphi(\mathbf{x}, \mathbf{x}')$ iff $\forall (s, s') \in R.\ \varphi[\mathbf{x} \mapsto s(\mathbf{x}), \mathbf{x}' \mapsto s'(\mathbf{x})]$ holds

For all

R, S transition relations

φ, ψ transition formulas

such that $R \Vdash \varphi \qquad S \Vdash \psi$

Then:

- $\{(s, s'') : \exists s'.(s, s') \in R \land (s', s'') \in S\} \Vdash \exists \mathbf{x}''.\varphi[\mathbf{x}' \mapsto \mathbf{x}''] \land \psi[\mathbf{x} \mapsto \mathbf{x}'']$
- $R \cup S \Vdash \varphi \lor \psi$

CRA simulates relational interpretation

$R \Vdash \varphi(\mathbf{x}, \mathbf{x}')$ iff $\forall(s, s') \in R.\ \varphi[\mathbf{x} \mapsto s(\mathbf{x}), \mathbf{x}' \mapsto s'(\mathbf{x})]$ holds

For all

R, S transition relations

φ, ψ transition formulas

such that $R \Vdash \varphi$ $S \Vdash \psi$

Then:

- $\{(s, s'') : \exists s'.(s, s') \in R \land (s', s'') \in S\} \Vdash \exists \mathbf{x}''.\varphi[\mathbf{x}' \mapsto \mathbf{x}''] \land \psi[\mathbf{x} \mapsto \mathbf{x}'']$
- $R \cup S \Vdash \varphi \lor \psi$
- $R^{*^\natural} \Vdash \varphi^{*^\sharp}$

Algebraic laws

$\langle D, \oplus, \otimes, 0, 1 \rangle$ is a *idempotent semiring*:

- $\oplus$ is associative, commutative, and idempotent, and has identity 0

$a \oplus (b \oplus c) = (a \oplus b) \oplus c$	Associative
$a \oplus b = b \oplus a$	Commutative
$a \oplus a = a$	Idempotent
$a \oplus 0 = a$	Identity

- $\otimes$ is associative and has 1 as identity and 0 as annihilator

$a \otimes (b \otimes c) = (a \otimes b) \otimes c$	Associative
$a \otimes 1 = 1 \otimes a = a$	Identity
$0 \otimes a = a \otimes 0 = 0$	Annihilation

- $\otimes$ distributes over $\oplus$: $a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$

Iteration axioms

Write $a \leq b$ iff $a \oplus b = b$.

Iteration axioms

Write $a \leq b$ iff $a \oplus b = b$.

$\langle D, \oplus, \otimes, *, 0, 1 \rangle$ is a *Kleene algebra*: idempotent semiring +

① $1 \leq a^*$

② $a \otimes (a^*) \leq a^*$

③ $(a^*) \otimes a \leq a^*$

④ for all x, $a \otimes x \leq x \Rightarrow (a^*) \otimes x \leq x$

⑤ for all x, $x \otimes a \leq x \Rightarrow x \otimes (a^*) \leq x$

(i.e., a^* is least fixed point of $X = 1 + aX$ and $X = 1 + Xa$)

Iteration axioms

Write $a \leq b$ iff $a \oplus b = b$.

$\langle D, \oplus, \otimes, *, 0, 1 \rangle$ is a *Kleene algebra*: idempotent semiring +

① $1 \leq a^*$

② $a \otimes (a^*) \leq a^*$

③ $(a^*) \otimes a \leq a^*$

④ for all x, $a \otimes x \leq x \Rightarrow (a^*) \otimes x \leq x$

⑤ for all x, $x \otimes a \leq x \Rightarrow x \otimes (a^*) \leq x$

(i.e., a^* is least fixed point of $X = 1 + aX$ and $X = 1 + Xa$)

$\langle D, \equiv, \oplus, \otimes, *, 0, 1 \rangle$ is a *quasi weight domain*:

- replace $=$ with some equivalence relation $\equiv$
- relax requirement that a^* is a least fixed point (axioms 4 & 5)

Consequences of algebraic laws

- $\mathcal{D}$ is a Kleene algebra: choice of path expression algorithm is irrelevant.

Consequences of algebraic laws

- $\mathcal{D}$ is a Kleene algebra: choice of path expression algorithm is irrelevant.
- $\mathcal{D}$ is a quasi-weight domain: For any path expression E, for any path w recognized by E we have $\mathcal{D}^\sharp[\![w]\!] \leq \mathcal{D}^\sharp[\![E]\!]$

Consequences of algebraic laws

- $\mathcal{D}$ is a Kleene algebra: choice of path expression algorithm is irrelevant.
- $\mathcal{D}$ is a quasi-weight domain: For any path expression E, for any path w recognized by E we have $\mathcal{D}^\sharp[\![w]\!] \leq \mathcal{D}^\sharp[\![E]\!]$
- If $\mathcal{D}^\natural$ is a Kleene algebra and $\mathcal{D}^\sharp$ is a quasi-weight domain, then $c_1^{*^\natural} \Vdash a_1^{*^\sharp}$ follows from the rest of the conditions on a soundness relation.

Designing an algebraic analysis

1 **Define**:
  - Semantic algebra $\mathcal{D} = \langle D, \otimes, \oplus, *, 0, 1 \rangle$
  - Semantic function $\mathcal{D}^\sharp[\![\cdot]\!]$: *Edge* $\rightarrow D$

2 **Apply**: Tarjan's path expression algorithm

Proving soundness

① **Define**:

- Concrete semantics
- Relation $\Vdash$

② **Prove**:

- $\Vdash$ is a soundness relation
- soundness of atomic interpretations: $\forall e, \mathcal{D}^{\natural}[\![e]\!] \Vdash \mathcal{D}^{\sharp}[\![e]\!]$

③ **Apply** theorem: If $\Vdash$ is a soundness relation and $\mathcal{D}^{\natural}[\![e]\!] \Vdash \mathcal{D}^{\sharp}[\![e]\!]$ for all edges e, then path expression algorithm computes properties that are sound w.r.t. concrete semantics.

Iterative vs. algebraic program analysis

Iterative Framework	Algebraic Framework
Join semi-lattice	Semantic Algebra
Abstract transformers	Semantic function
Chaotic iteration algorithm	Path-expression algorithm
Concretization function	Soundness relation

Iterative vs. algebraic program analysis

Iterative Framework	Algebraic Framework
Join semi-lattice	Semantic Algebra
Abstract transformers	Semantic function
Chaotic iteration algorithm	Path-expression algorithm
Concretization function	Soundness relation

Key point: loop analysis is *internal* to an algebraic program analysis.