Compiler Optimisation
3 – Dataflow Analysis

Hugh Leather
IF 1.18a
hleather@inf.ed.ac.uk

Institute for Computing Systems Architecture
School of Informatics
University of Edinburgh

2019
Introduction

- Optimisations often split into
  - **Analysis**: Calculate some values at points in program
  - **Transformation**: Improve the program where analysis allows

- Data flow analyses are common class of analyses
- Data pushed around control flow graph simulating effect of statements
- This lecture introduces:
  - Reaching definitions analysis in detail
  - Algorithms to compute data flow
Reaching definitions

Definition of variable $x$ at program point $d$ reaches point $u$ if there exists a control-flow path $p$ from $d$ to $u$ such that no definition of $x$ appears on that path.

Where do definitions of $a$ reach?
Reaching definitions

Definition of variable $x$ at program point $d$ reaches point $u$ if there exists a control-flow path $p$ from $d$ to $u$ such that no definition of $x$ appears on that path.

Where do definitions of $a$ reach?
Reaching definitions

Definition of variable $x$ at program point $d$ reaches point $u$ if
\[ \exists \text{ control-flow path } p \text{ from } d \text{ to } u \text{ such that}
\] no definition of $x$ appears on that path

Where do definitions of $a$ reach?
Reaching definitions

Definition of variable \( x \) at program point \( d \) reaches point \( u \) if there exists a control-flow path \( p \) from \( d \) to \( u \) such that no definition of \( x \) appears on that path.

Where do definitions of \( a \) reach?
Local analysis works only on a single basic block. Computation by simulation or abstract interpretation\(^1\)

- Maintain a set of current reaching definitions
- At the start node, there are no definitions
- Go through all the statements from start to end
- If assignment statement \(x_i := \ldots\)
  - First, \(\forall j\) remove \(x_j\)
  - Then, add \(x_i\) to the set
- Otherwise set unchanged

\(^1\)Execute only bits we care about, namely where definitions reach
Reaching definitions
Local analysis

Reaching \( s_1 = {} \)

\[
\begin{align*}
\text{s}_1 & : \quad a_1 := 2 \\
\text{s}_2 & : \quad b := x + 1 \\
\text{s}_3 & : \quad c := a \times 3 \\
\text{s}_4 & : \quad a_4 := 4 \\
\text{s}_5 & : \quad d := a \\
\text{s}_6 & : \quad \text{return } d
\end{align*}
\]
Reaching definitions

Local analysis

s₁: \( a₁ := 2 \)

s₂: \( b := x + 1 \)

s₃: \( c := a * 3 \)

s₄: \( a₄ := 4 \)

s₅: \( d := a \)

s₆: return d

Reaching \( s₁ = \{ \} \)

\( s₁ \) defines \( a₁ \)

Reaching \( s₂ = \{ a₁ \} \)
Reaching definitions

Local analysis

\[\begin{align*}
S_1 & : a_1 := 2 \\
S_2 & : b := x + 1 \\
S_3 & : c := a \ast 3 \\
S_4 & : a_4 := 4 \\
S_5 & : d := a \\
S_6 & : \text{return } d
\end{align*}\]

Reaching \(S_1 = \emptyset\)

\(S_1\) defines \(a_1\)

Reaching \(S_2 = \{ a_1 \}\)

\(S_2\) defines \(b\)

Reaching \(S_3 = \{ a_1, b \}\)
Reaching definitions
Local analysis

Reaching $s_1 = \emptyset$
$s_1$ defines $a_1$

Reaching $s_2 = \{ a_1 \}$
$s_2$ defines $b$

Reaching $s_3 = \{ a_1, b \}$
$s_3$ defines $c$

Reaching $s_4 = \{ a_1, b, c \}$

$s_1: a_1 := 2$

$s_2: b := x + 1$

$s_3: c := a \times 3$

$s_4: a_4 := 4$

$s_5: d := a$

$s_6: \text{return } d$
Reaching definitions

Local analysis

\begin{align*}
    &s_1 \quad \text{Reaching } s_1 = \emptyset \\
    &\quad a_1 := 2 \\
    &s_2 \quad \text{s}_1 \text{ defines } a_1 \\
    &\quad b := x + 1 \\
    &s_3 \quad \text{Reaching } s_3 = \{ a_1, b \} \\
    &\quad c := a \times 3 \\
    &s_4 \quad \text{s}_3 \text{ defines } c \\
    &\quad a_4 := 4 \\
    &s_5 \quad \text{s}_4 \text{ defines } a_4, \text{kills } a_1 \\
    &\quad d := a \\
    &s_6 \quad \text{Reaching } s_5 = \{ b, c, a_4 \} \\
    &\quad \text{return } d
\end{align*}
Reaching definitions
Local analysis

Reaching $s_1 = \{\}$
$s_1$ defines $a_1$

Reaching $s_2 = \{ a_1 \}$
$s_2$ defines $b$

Reaching $s_3 = \{ a_1, b \}$
$s_3$ defines $c$

Reaching $s_4 = \{ a_1, b, c \}$
$s_4$ defines $a_4$, kills $a_1$

Reaching $s_5 = \{ b, c, a_4 \}$
$s_5$ defines $d$

Reaching $s_6 = \{ b, c, a_4, d \}$
Reaching definitions
Global analysis

- Control flow complicates matters
- Consider reaching definitions:
  - Entering a statement - the \textit{In} program point for the statement
  - Leaving a statement - the \textit{Out} program point for the statement
- Root is a special start node
- We will try the previous approach on this and see where it fails
Reaching definitions

Global analysis

Control flow example; try the previous approach

\[ S_1: a_1 := 2 \]

\[ S_2: \text{if } x > 0 \]

\[ S_3: a_3 := x + 1 \]

\[ S_4: b := 0 \]

\[ S_5: c := a \times 2 \]

\[ S_6: \text{if } y < x \]
$s_4$ has 2 predecessors; and don’t know $Out(s_6)$
But, we know at least that $a_1$ reaches $s_4$
Reaching definitions

Global analysis

$s_5$ has 2 predecessors
All incoming definitions reach; do union
Inconsistency now we know more about \( Out(s_6) \)
Reaching definitions
Global analysis

All incoming definitions reach; do union; inconsistency
Reaching definitions

Global analysis

Inconsistency
Reaching definitions
Global analysis

Consistent state

$\begin{align*}
&s_1 \quad a_1 := 2 \\
&s_2 \quad \text{if } x > 0 \\
&s_3 \quad a_3 := x + 1 \\
&s_4 \quad b := 0 \\
&s_5 \quad c := a \times 2 \\
&s_6 \quad \text{if } y < x
\end{align*}$

$\begin{align*}
&\{ a_1 \} \\
&\{ a_1 \} \\
&\{ a_1, a_3, b, c \} \\
&\{ a_1, a_3, b, c \} \\
&\{ a_1, a_3, b, c \} \\
&\{ a_1, a_3, b, c \}
\end{align*}$
Let us formalise our intuition
Let us formalise our intuition

- To simulate a statement, $s$, compute $Out(s)$ from $In(s)$
  - If assignment to $x$, delete all definitions of $x$, add new definition
    
    $$Out(s : d_i := ...) = (In(s) \setminus \{d_j; \forall j\}) \cup \{d_i\}$$
Let us formalise our intuition

- To simulate a statement, $s$, compute $Out(s)$ from $In(s)$
  If assignment to $x$, delete all definitions of $x$, add new definition
  $$Out(s : d_i := ...) = (In(s) - \{d_j; \forall j\}) \cup \{d_i\}$$

- Multiple edges must merge to compute $In(s)$ from $Pred(s)$
  All incoming definitions reach
  $$In(s) = \bigcup_{\forall p \in Pred(s)} Out(p)$$
Let us formalise our intuition

- To simulate a statement, $s$, compute $Out(s)$ from $In(s)$
- If assignment to $x$, delete all definitions of $x$, add new definition
  \[ Out(s : d_i := ...) = (In(s) - \{d_j; \forall j\}) \cup \{d_i\} \]
- Multiple edges must merge to compute $In(s)$ from $Pred(s)$
  All incoming definitions reach
  \[ In(s) = \bigcup_{\forall p \in Pred(s)} Out(p) \]
- If we don’t know, start with empty
  \[ Init(s) = \emptyset \]
Let us formalise our intuition

- To simulate a statement, \( s \), compute \( \text{Out}(s) \) from \( \text{In}(s) \)
  If assignment to \( x \), delete all definitions of \( x \), add new definition
  \[
  \text{Out}(s : d_i := ...) = (\text{In}(s) - \{d_j; \forall j\}) \cup \{d_i\}
  \]

- Multiple edges must merge to compute \( \text{In}(s) \) from \( \text{Pred}(s) \)
  All incoming definitions reach
  \[
  \text{In}(s) = \bigcup_{\forall p \in \text{Pred}(s)} \text{Out}(p)
  \]

- If we don’t know, start with empty
  \[
  \text{Init}(s) = \emptyset
  \]

- Note that often \( \text{Out}(s) \) is written
  \[
  \text{Out}(s : d_i := ...) = (\text{In}(s) - \text{Kill}(s)) \cup \text{Gen}(s)
  \]
  The \( \text{Gen} \) and \( \text{Kill} \) sets can often be precomputed
  Also, \( \EaC \) combines \( \text{In} \) and \( \text{Out} \) to use only one equation
Analysis defines properties at points with *recurrence relations*.
Assumes a control flow graph.
Start with a conservative approximation.
Refine the approximations.
Stop when consistent (no further change).
Information flows *forward* from a statement to its successors.
Ingredients of dataflow analysis

- **Direction** - forward or backward

2 In a later lecture
Ingredients of dataflow analysis

- **Direction** - forward or backward
- **Transfer function** - computes statement effect
  - e.g. \( Out(s) = Gen(s) \cup (In(s) - Kill(s)) \)

\(^2\text{In a later lecture}\)
Ingredients of dataflow analysis

- **Direction** - forward or backward
- **Transfer function** - computes statement effect
  - e.g. $Out(s) = Gen(s) \cup (In(s) - Kill(s))$
- **Meet operator** - merges values from multiple incoming edges
  - e.g. $In(s) = \bigcup_{\forall p \in Pred(s)} Out(p)$

---

2 In a later lecture
Ingredients of dataflow analysis

- **Direction** - forward or backward
- **Transfer function** - computes statement effect
  - e.g. $Out(s) = Gen(s) \cup (In(s) - Kill(s))$
- **Meet operator** - merges values from multiple incoming edges
  - e.g. $In(s) = \bigcup_{p \in Pred(s)} Out(p)$
- **Value set** - the bits information being passed around
  - e.g. Sets of definitions

\(^2\text{In a later lecture}\)
Ingredients of dataflow analysis

- **Direction** - forward or backward
- **Transfer function** - computes statement effect
  - e.g. $Out(s) = Gen(s) \cup (In(s) - Kill(s))$
- **Meet operator** - merges values from multiple incoming edges
  - e.g. $In(s) = \bigcup_{\forall p \in Pred(s)} Out(p)$
- **Value set** - the bits information being passed around
  - e.g. Sets of definitions
- **Initial values**
  - Should be most conservative value
  - Start node often a special case; e.g. encoding function parameters

---

2 In a later lecture


Ingredients of dataflow analysis

- **Direction** - forward or backward
- **Transfer function** - computes statement effect
  - e.g. $Out(s) = Gen(s) \cup (In(s) - Kill(s))$
- **Meet operator** - merges values from multiple incoming edges
  - e.g. $In(s) = \bigcup_{p \in Pred(s)} Out(p)$
- **Value set** - the bits information being passed around
  - e.g. Sets of definitions
- **Initial values**
  - Should be most conservative value
  - Start node often a special case; e.g. encoding function parameters
- Some properties of the above to ensure termination\(^2\)

\(^2\)In a later lecture
There are many, many data flow algorithms that fit
Algorithms
Round-robin iterative algorithm

Reaching definitions control flow example - Calculate RD sets?

\[
\begin{align*}
\text{In}(s) &= \bigcup_{\forall p \in \text{Pred}(s)} \text{Out}(p) \\
\text{Out}(s : d_i := \ldots) &= (\text{In}(s) - \{d_j; \forall j\}) \cup d_i \\
\Downarrow \\
\text{RD}(s) &= \bigcup_{\forall p : d_i = \ldots \in \text{Pred}(s)} (\text{RD}(p) - \{d_j; \forall j\}) \cup \{d_i\}
\end{align*}
\]

<table>
<thead>
<tr>
<th>Node</th>
<th>(s_1)</th>
<th>(s_2)</th>
<th>(s_3)</th>
<th>(s_4)</th>
<th>(s_5)</th>
<th>(s_6)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RD(^4)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
</tr>
</tbody>
</table>

\(^4\text{For brevity, In and Out are combined}\)
Reaching definitions control flow example - Calculate RD sets?

\[ In(s) = \bigcup_{p \in \text{Pred}(s)} Out(p) \]

\[ Out(s : d_i := \ldots) = (In(s) - \{d_j; \forall j\}) \cup d_i \]

\[ RD(s) = \bigcup_{p : d_i = \ldots \in \text{Pred}(s)} (RD(p) - \{d_j; \forall j\}) \cup \{d_i\} \]

<table>
<thead>
<tr>
<th>Node</th>
<th>( s_1 )</th>
<th>( s_2 )</th>
<th>( s_3 )</th>
<th>( s_4 )</th>
<th>( s_5 )</th>
<th>( s_6 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>RD(^4)</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>RD(^4)</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
</tr>
</tbody>
</table>

\(^4\)For brevity, \( In \) and \( Out \) are combined
Reaching definitions control flow example - Calculate RD sets?

\[
\begin{align*}
\text{In}(s) &= \bigcup_{p \in \text{Pred}(s)} \text{Out}(p) \\
\text{Out}(s : d_i := \ldots) &= (\text{In}(s) - \{d_j; \forall j\}) \cup d_i \\
\downarrow & \\
\text{RD}(s) &= \bigcup_{p : d_i = \ldots \in \text{Pred}(s)} (\text{RD}(p) - \{d_j; \forall j\}) \cup \{d_i\}
\end{align*}
\]

<table>
<thead>
<tr>
<th>Node</th>
<th>s₁</th>
<th>s₂</th>
<th>s₃</th>
<th>s₄</th>
<th>s₅</th>
<th>s₆</th>
</tr>
</thead>
<tbody>
<tr>
<td>RD⁴</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td></td>
<td>∅</td>
<td>a₁</td>
<td></td>
<td></td>
<td>∅</td>
<td>∅</td>
</tr>
</tbody>
</table>

⁴For brevity, In and Out are combined
Reaching definitions control flow example - Calculate RD sets?

\[ \text{In}(s) = \bigcup_{p \in \text{Pred}(s)} \text{Out}(p) \]

\[ \text{Out}(s : d_i := ...) = (\text{In}(s) - \{d_j; \forall j\}) \cup d_i \]

\[ \Downarrow \]

\[ \text{RD}(s) = \bigcup_{p : d_i = \ldots \in \text{Pred}(s)} (\text{RD}(p) - \{d_j; \forall j\}) \cup \{d_i\} \]

<table>
<thead>
<tr>
<th>Node</th>
<th>( s_1 )</th>
<th>( s_2 )</th>
<th>( s_3 )</th>
<th>( s_4 )</th>
<th>( s_5 )</th>
<th>( s_6 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>RD(^4)</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td></td>
<td>( \emptyset )</td>
<td>( a_1 )</td>
<td>( a_1 )</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
</tr>
</tbody>
</table>

\(^4\)For brevity, \( \text{In} \) and \( \text{Out} \) are combined
Reaching definitions control flow example - Calculate RD sets?

\[
\begin{align*}
\text{In}(s) &= \bigcup_{p \in \text{Pred}(s)} \text{Out}(p) \\
\text{Out}(s : d_i := \ldots) &= (\text{In}(s) - \{d_j; \forall j\}) \cup d_i \\
\downarrow \\
\text{RD}(s) &= \bigcup_{p : d_i = \ldots \in \text{Pred}(s)} (\text{RD}(p) - \{d_j; \forall j\}) \cup \{d_i\}
\end{align*}
\]

<table>
<thead>
<tr>
<th>Node</th>
<th>(s_1)</th>
<th>(s_2)</th>
<th>(s_3)</th>
<th>(s_4)</th>
<th>(s_5)</th>
<th>(s_6)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RD(^4)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
</tr>
<tr>
<td></td>
<td>(\emptyset)</td>
<td>(a_1)</td>
<td>(a_1)</td>
<td>(a_1)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\(^4\)For brevity, \text{In} and \text{Out} are combined
Reaching definitions control flow example - Calculate RD sets?

\[
\begin{align*}
\text{In}(s) &= \bigcup_{p \in \text{Pred}(s)} \text{Out}(p) \\
\text{Out}(s : d_i := \ldots) &= (\text{In}(s) - \{d_j; \forall j\}) \cup d_i \\
\text{RD}(s) &= \bigcup_{p : d_i = \ldots \in \text{Pred}(s)} (\text{RD}(p) - \{d_j; \forall j\}) \cup \{d_i\}
\end{align*}
\]

<table>
<thead>
<tr>
<th>Node</th>
<th>(s_1)</th>
<th>(s_2)</th>
<th>(s_3)</th>
<th>(s_4)</th>
<th>(s_5)</th>
<th>(s_6)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RD(^4)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
</tr>
<tr>
<td></td>
<td>(\emptyset)</td>
<td>(a_1)</td>
<td>(a_1)</td>
<td>(a_1)</td>
<td>(a_1, a_3, b)</td>
<td>(\emptyset)</td>
</tr>
</tbody>
</table>

\(^4\)For brevity, In and Out are combined
Reaching definitions control flow example - Calculate RD sets?

\[ \text{In}(s) = \bigcup_{p \in \text{Pred}(s)} \text{Out}(p) \]
\[ \text{Out}(s : d_i := \ldots) = (\text{In}(s) - \{d_j; \forall j\}) \cup d_i \]
\[ \text{RD}(s) = \bigcup_{p : d_i := \ldots \in \text{Pred}(s)} (\text{RD}(p) - \{d_j; \forall j\}) \cup \{d_i\} \]

<table>
<thead>
<tr>
<th>Node</th>
<th>(s_1)</th>
<th>(s_2)</th>
<th>(s_3)</th>
<th>(s_4)</th>
<th>(s_5)</th>
<th>(s_6)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RD(^4)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
</tr>
<tr>
<td></td>
<td>(\emptyset)</td>
<td>(a_1)</td>
<td>(a_1)</td>
<td>(a_1)</td>
<td>(a_1, a_3, b)</td>
<td>(a_1, a_3, b, c)</td>
</tr>
</tbody>
</table>

\(^4\)For brevity, \(\text{In}\) and \(\text{Out}\) are combined
Reaching definitions control flow example - Calculate RD sets?

\[ \text{In}(s) = \bigcup_{\forall p \in \text{Pred}(s)} \text{Out}(p) \]
\[ \text{Out}(s : d_i := \ldots) = (\text{In}(s) - \{d_j; \forall j\}) \cup d_i \]
\[ \text{RD}(s) = \bigcup_{\forall p : d_i := \ldots \in \text{Pred}(s)} (\text{RD}(p) - \{d_j; \forall j\}) \cup \{d_i\} \]

<table>
<thead>
<tr>
<th>Node</th>
<th>(s_1)</th>
<th>(s_2)</th>
<th>(s_3)</th>
<th>(s_4)</th>
<th>(s_5)</th>
<th>(s_6)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RD(^4)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
</tr>
<tr>
<td></td>
<td>(\emptyset)</td>
<td>(a_1)</td>
<td>(a_1)</td>
<td>(a_1)</td>
<td>(a_1, a_3, b)</td>
<td>(a_1, a_3, b, c)</td>
</tr>
</tbody>
</table>

\(^4\)For brevity, In and Out are combined
Reaching definitions control flow example - Calculate RD sets?

\[ \text{In}(s) = \bigcup_{p \in \text{Pred}(s)} \text{Out}(p) \]
\[ \text{Out}(s : d_i := ...) = (\text{In}(s) - \{d_j; \forall j\}) \cup d_i \]
\[ \text{RD}(s) = \bigcup_{p : d_i = \ldots \in \text{Pred}(s)} (\text{RD}(p) - \{d_j; \forall j\}) \cup \{d_i\} \]

<table>
<thead>
<tr>
<th>Node</th>
<th>( s_1 )</th>
<th>( s_2 )</th>
<th>( s_3 )</th>
<th>( s_4 )</th>
<th>( s_5 )</th>
<th>( s_6 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>RD(^4)</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td></td>
<td>∅</td>
<td>( a_1 )</td>
<td>( a_1 )</td>
<td>( a_1 )</td>
<td>( a_1, a_3, b )</td>
<td>( a_1, a_3, b, c )</td>
</tr>
</tbody>
</table>

\(^4\)For brevity, \( \text{In} \) and \( \text{Out} \) are combined
Reaching definitions control flow example - Calculate RD sets?

\[
\begin{align*}
In(s) &= \bigcup_{p \in \text{Pred}(s)} \text{Out}(p) \\
\text{Out}(s : d_i := \ldots) &= (In(s) - \{d_j; \forall j\}) \cup d_i \\
RD(s) &= \bigcup_{p : d_i := \ldots \in \text{Pred}(s)} (RD(p) - \{d_j; \forall j\}) \cup \{d_i\}
\end{align*}
\]

<table>
<thead>
<tr>
<th>Node</th>
<th>(s_1)</th>
<th>(s_2)</th>
<th>(s_3)</th>
<th>(s_4)</th>
<th>(s_5)</th>
<th>(s_6)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RD(^4)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
</tr>
<tr>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>(a_1)</td>
<td>(a_1)</td>
<td>(a_1)</td>
<td>(a_1, a_3, b)</td>
<td>(a_1, a_3, b, c)</td>
</tr>
</tbody>
</table>

\(^4\text{For brevity, } In \text{ and } Out \text{ are combined}\)
Reaching definitions control flow example - Calculate RD sets?

\[ In(s) = \bigcup_{p \in \text{Pred}(s)} \text{Out}(p) \]
\[ \text{Out}(s : d_i := ...) = (In(s) - \{d_j : \forall j\}) \cup d_i \]
\[ RD(s) = \bigcup_{p : d_i := ... \in \text{Pred}(s)} (RD(p) - \{d_j : \forall j\}) \cup \{d_i\} \]

<table>
<thead>
<tr>
<th>Node</th>
<th>(s_1)</th>
<th>(s_2)</th>
<th>(s_3)</th>
<th>(s_4)</th>
<th>(s_5)</th>
<th>(s_6)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RD(^4)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
</tr>
<tr>
<td></td>
<td>(\emptyset)</td>
<td>(a_1)</td>
<td>(a_1)</td>
<td>(a_1)</td>
<td>(a_1, a_3, b)</td>
<td>(a_1, a_3, b, c)</td>
</tr>
<tr>
<td></td>
<td>(\emptyset)</td>
<td>(a_1)</td>
<td>(a_1)</td>
<td>(a_1, a_3, b, c)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\(^4\)For brevity, \(In\) and \(Out\) are combined
Algorithms
Round-robin iterative algorithm

Reaching definitions control flow example - Calculate RD sets?

\[
\begin{align*}
    \text{In}(s) &= \bigcup_{\forall p \in \text{Pred}(s)} \text{Out}(p) \\
    \text{Out}(s : d_i := \ldots) &= (\text{In}(s) - \{d_j; \forall j\}) \cup d_i \\
    \Downarrow \\
    \text{RD}(s) &= \bigcup_{\forall p : d_i = \ldots \in \text{Pred}(s)} (\text{RD}(p) - \{d_j; \forall j\}) \cup \{d_i\}
\end{align*}
\]

<table>
<thead>
<tr>
<th>Node</th>
<th>s₁</th>
<th>s₂</th>
<th>s₃</th>
<th>s₄</th>
<th>s₅</th>
<th>s₆</th>
</tr>
</thead>
<tbody>
<tr>
<td>RD⁴</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td></td>
<td>∅</td>
<td>a₁</td>
<td>a₁</td>
<td>a₁</td>
<td>a₁, a₃, b</td>
<td>a₁, a₃, b, c</td>
</tr>
<tr>
<td></td>
<td>∅</td>
<td>a₁</td>
<td>a₁</td>
<td>a₁, a₃, b, c</td>
<td>a₁, a₃, b, c</td>
<td>a₁, a₃, b, c</td>
</tr>
</tbody>
</table>

⁴For brevity, In and Out are combined
Reaching definitions control flow example - Calculate RD sets?

Node | $s_1$ | $s_2$ | $s_3$ | $s_4$ | $s_5$ | $s_6$
--- | --- | --- | --- | --- | --- | ---
RD$^4$ | $\emptyset$ | $\emptyset$ | $\emptyset$ | $\emptyset$ | $\emptyset$ | $\emptyset$
| $\emptyset$ | $a_1$ | $a_1$ | $a_1$ | $a_1, a_3, b$ | $a_1, a_3, b, c$
| $\emptyset$ | $a_1$ | $a_1$ | $a_1, a_3, b, c$ | $a_1, a_3, b, c$ | $a_1, a_3, b, c$

$In(s) = \bigcup_{\forall p \in Pred(s)} Out(p)$

$Out(s : d_i := ...) = (In(s) - \{d_j; \forall j\}) \cup d_i$

$RD(s) = \bigcup_{\forall p : d_i = \ldots \in Pred(s)} (RD(p) - \{d_j; \forall j\}) \cup \{d_i\}$

$^4$For brevity, $In$ and $Out$ are combined
Reaching definitions control flow example - Calculate RD sets?

\[ \text{In}(s) = \bigcup_{\forall p \in \text{Pred}(s)} \text{Out}(p) \]

\[ \text{Out}(s : d_i := ...) = (\text{In}(s) - \{d_j; \forall j\}) \cup d_i \]

\[ \text{RD}(s) = \bigcup_{\forall p : d_i = ... \in \text{Pred}(s)} (\text{RD}(p) - \{d_j; \forall j\}) \cup \{d_i\} \]

<table>
<thead>
<tr>
<th>Node</th>
<th>( s_1 )</th>
<th>( s_2 )</th>
<th>( s_3 )</th>
<th>( s_4 )</th>
<th>( s_5 )</th>
<th>( s_6 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>RD(^4)</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>( a_1 )</td>
<td>( a_1 )</td>
<td>( a_1 )</td>
<td>( a_1, a_3, b )</td>
<td>( a_1, a_3, b, c )</td>
</tr>
<tr>
<td>( \emptyset )</td>
<td>( a_1 )</td>
<td>( a_1 )</td>
<td>( a_1, a_3, b, c )</td>
<td>( a_1, a_3, b, c )</td>
<td>( a_1, a_3, b, c )</td>
<td></td>
</tr>
<tr>
<td>( a_1 )</td>
<td>( a_1 )</td>
<td>( a_1, a_3, b, c )</td>
<td>( a_1, a_3, b, c )</td>
<td>( a_1, a_3, b, c )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\(^4\)For brevity, \text{In} and \text{Out} are combined
Does round robin for reaching definitions always terminate?
Does round robin for reaching definitions always terminate?

Yes

- Each step of the iteration can only grow a set or leave unchanged
- Finite number of elements in each set, so finite number of times can change
- Each iteration either has a change or stops
- Must terminate
Algorithms
Speeding up

- Round-robin algorithm is slow, may require many passes through nodes
- Can speed up by considering basic blocks (e.g. compute Gen and Kill for whole block)
- Only nodes which have inputs changed need to be processed - use work list
- Reducible graphs can be handled more efficiently (see \textit{EaC} p.527)
Algorithms
Order matters

May reduce number of iterations by changing evaluation order\(^5\)

- Backward analysis - evaluate node after successors
  Use \textit{postorder}
- Forward analysis - evaluate node before successors
  Use \textit{reverse postorder}

Orders for reaching definitions example

<table>
<thead>
<tr>
<th>Post(1)</th>
<th>$s_4 s_6 s_5 s_3 s_2 s_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Post(2)</td>
<td>$s_6 s_5 s_4 s_3 s_2 s_1$</td>
</tr>
<tr>
<td>Rev(1)</td>
<td>$s_1 s_2 s_3 s_5 s_6 s_4$</td>
</tr>
<tr>
<td>Rev(2)</td>
<td>$s_1 s_2 s_3 s_4 s_5 s_6$</td>
</tr>
</tbody>
</table>

\(^5\)A lot of theory about this. Given certain conditions then a round-robin postorder alg will finish in $d(G) + 3$ passes where $d(G)$ is the loop connectedness. Muchnick for more details
Data flow analyses have some limitations:

- **Static analysis may be very conservative**
- **True CFG generally undecidable**
  - (e.g. condition may be constant but unprovable)
- **Pointers introduce aliases**
  - E.g. \(*x = 10\); Does \(x\) point to another variable, \(y\) or \(z\)? That would give a definition of \(y\) or \(z\). May not know at compile time which
  - Precise alias analysis not solved
- **Array access**
  - Generally cannot tell which indices are used
- **Function calls may not be reasoned across**
  - If inter-procedural, virtual calls and function pointer expand sets of functions
Some IRs/analyses force different information along edges
- Range analysis: compute possible ranges of integers; must know which edge out of if
- Java exception: change the stack contents

Each edge has a label - (e.g. THEN, ELSE, EXCEPTION)

Transfer function includes label as argument
Summary

- Reaching definitions
- Data flow algorithms
The biggest revolution in the technological landscape for fifty years
Now accepting applications!
Find out more and apply at: pervasiveparallelism.inf.ed.ac.uk

• 4-year programme: MSc by Research + PhD

• Research-focused: Work on your thesis topic from the start

• Collaboration between:
  ▶ University of Edinburgh’s School of Informatics
    ✴ Ranked top in the UK by 2014 REF
  ▶ Edinburgh Parallel Computing Centre
    ✴ UK’s largest supercomputing centre

• Research topics in software, hardware, theory and application of:
  ▶ Parallelism
  ▶ Concurrency
  ▶ Distribution

• Full funding available

• Industrial engagement programme includes internships at leading companies

Now accepting applications!