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

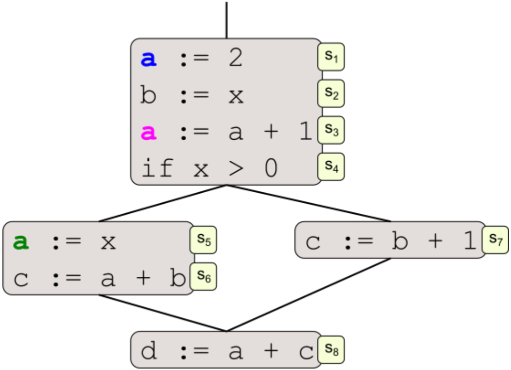
2019

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

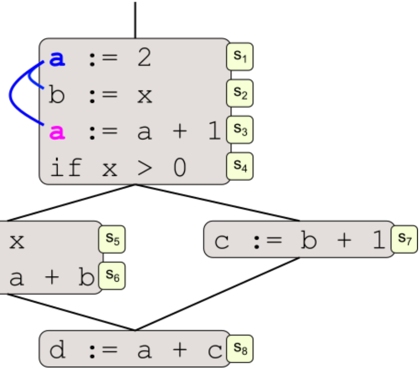
Definition of variable `x` at program point `d` **reaches** point `u` if
$\exists$ control-flow path `p` from `d` to `u` such that
no definition of `x` appears on that path



Where do definitions of `a` reach?

Reaching definitions

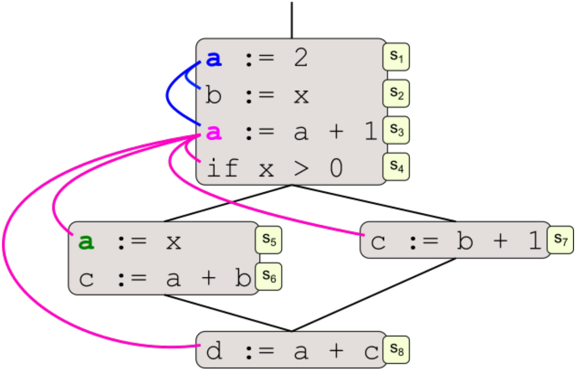
Definition of variable x at program point d **reaches** point u if
∃ control-flow path p from d to u such that
no definition of x appears on that path



Where do definitions of a reach?

Reaching definitions

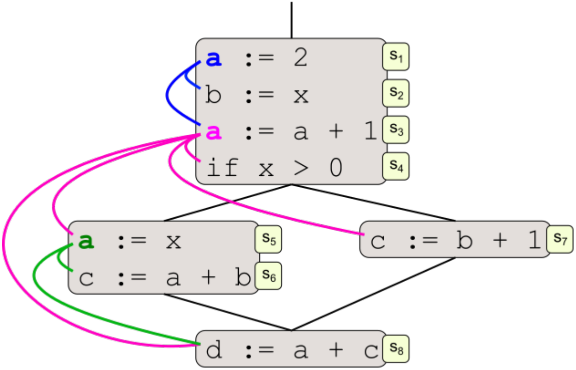
Definition of variable x at program point d **reaches** point u if
∃ control-flow path p from d to u such that
no definition of x appears on that path



Where do definitions of a reach?

Reaching definitions

Definition of variable x at program point d **reaches** point u if
∃ control-flow path p from d to u such that
no definition of x appears on that path



Where do definitions of a reach?

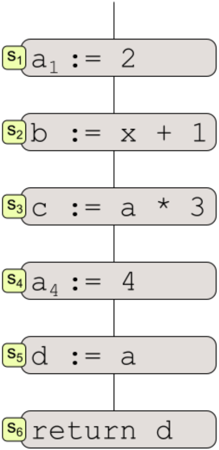
Reaching definitions
Local analysis

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 := ...
  - 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

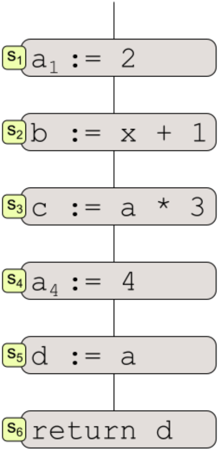
Reaching definitions
Local analysis



Reaching s_1 = {}

Reaching definitions

Local analysis

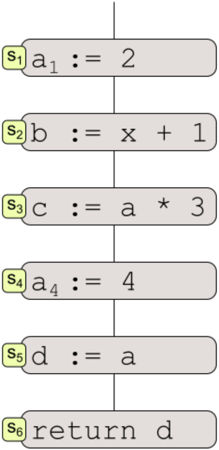


Reaching s_1 = {}

s_1 defines a_1

Reaching s_2 = { a_1 }

Reaching definitions
Local analysis



Reaching s_1 = {}

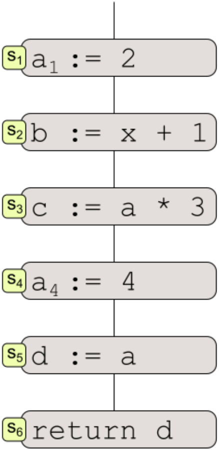
s_1 defines a_1

Reaching s_2 = { a_1 }

s_2 defines b

Reaching s_3 = { a_1, b }

Reaching definitions
Local analysis



Reaching s_1 = {}

s_1 defines a_1

Reaching s_2 = { a_1 }

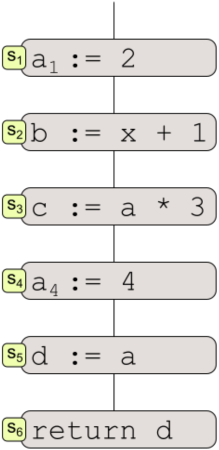
s_2 defines b

Reaching s_3 = { a_1, b }

s_3 defines c

Reaching s_4 = { a_1, b, c }

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

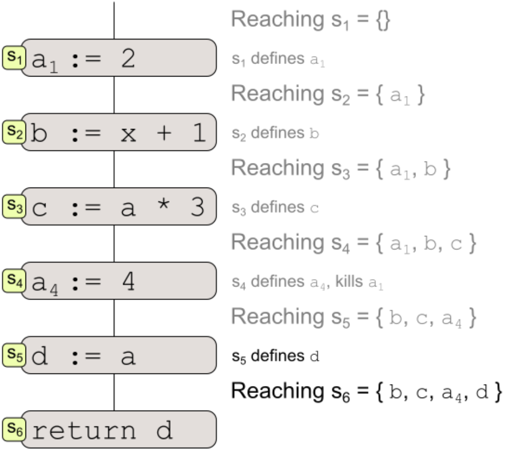
Reaching s_4 = { a_1, b, c }

s_4 defines a_4, kills a_1

Reaching s_5 = { b, c, a_4 }

Reaching definitions

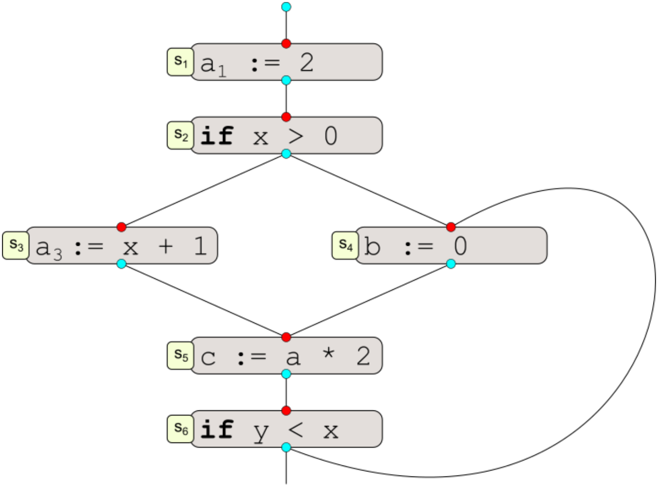
Local analysis



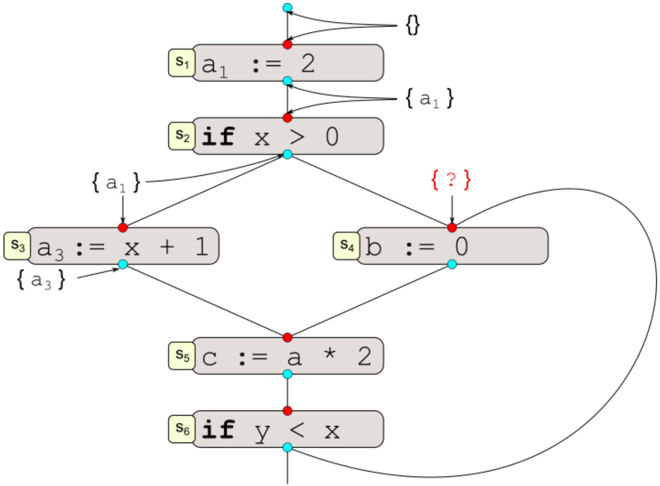
Reaching definitions
Global analysis

- Control flow complicates matters
- Consider reaching definitions:
  - Entering a statement - the *In* program point for the statement
  - Leaving a statement - the *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

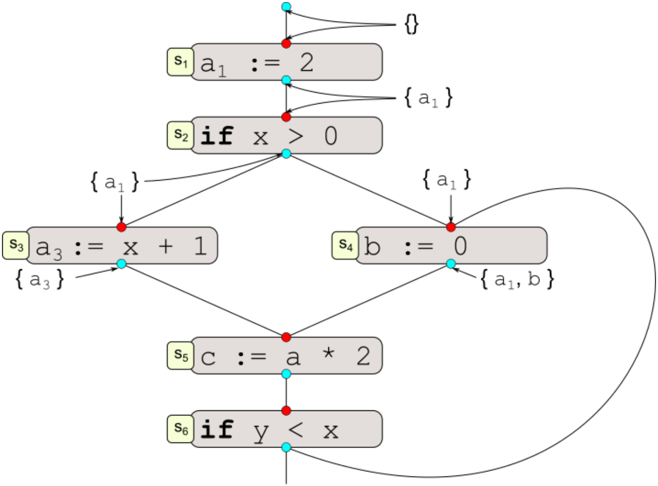
Control flow example; try the previous approach



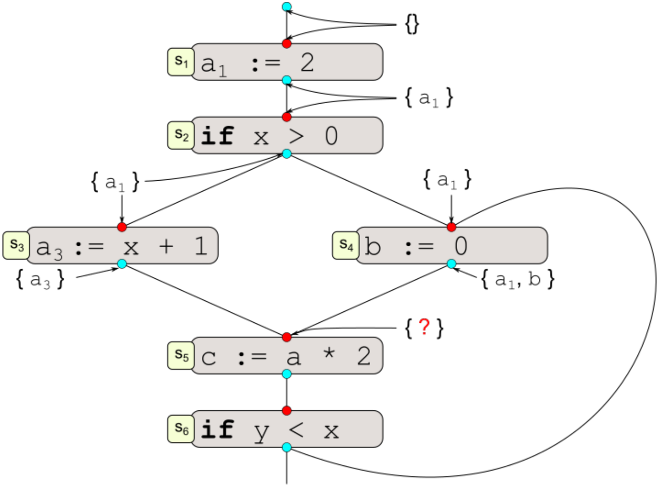
s_4 has 2 predecessors; and don't know $Out(s_6)$



But, we know at least that a_1 reaches s_4

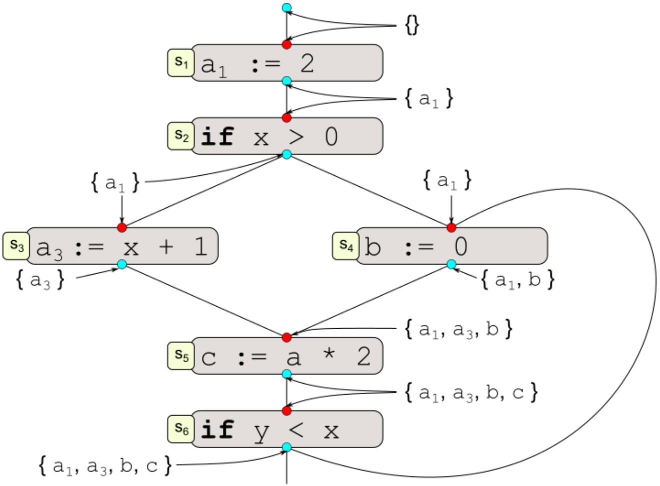


s_5 has 2 predecessors



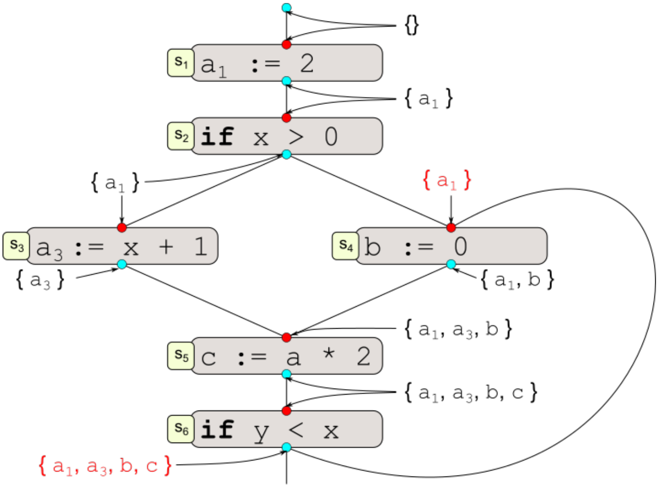
Reaching definitions
Global analysis

All incoming definitions reach; do union



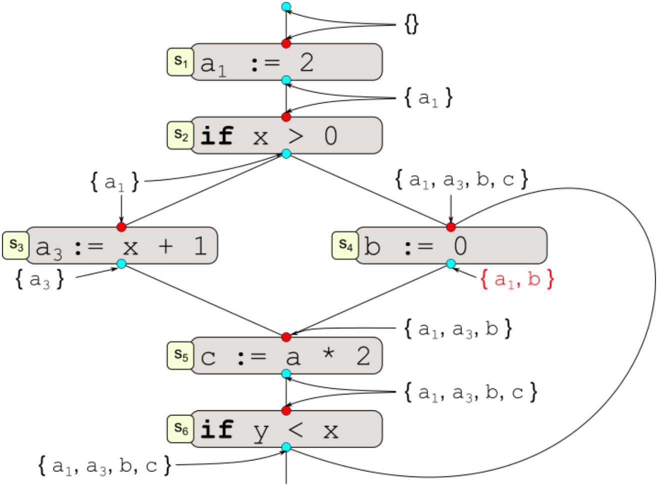
Reaching definitions
Global analysis

Inconsistency now we know more about $Out(s_6)$

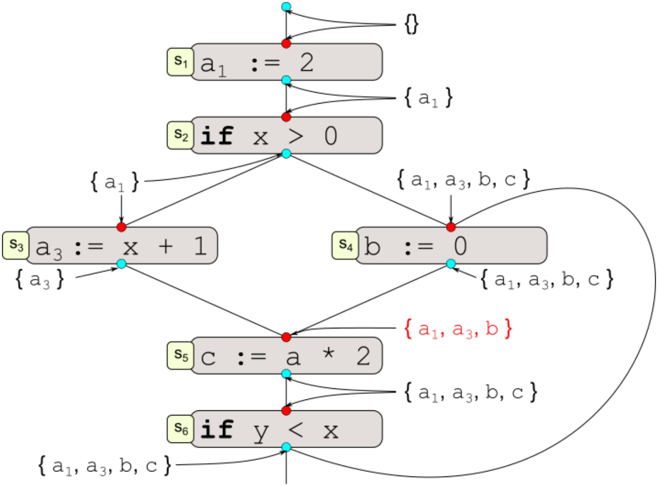


Reaching definitions
Global analysis

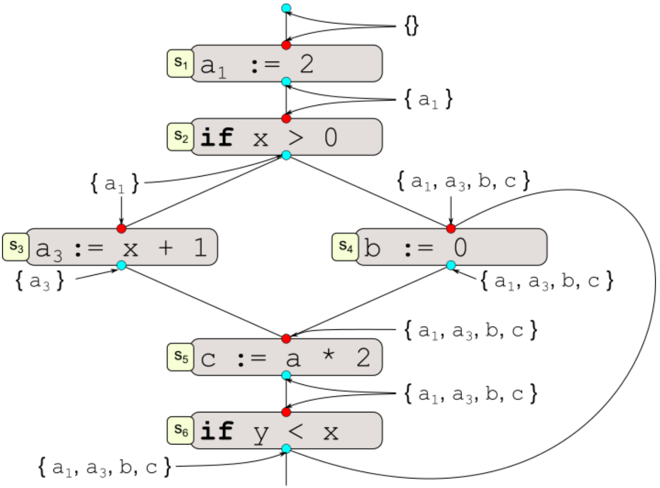
All incoming definitions reach; do union; inconsistency



Inconsistency



Consistent state



Reaching definitions
Dataflow equations

Let us formalise our intuition

Reaching definitions
Dataflow equations

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

Reaching definitions

Dataflow equations

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
  $$Out(s : d_i := \ldots) = (In(s) - \{d_j; \forall j\}) \cup \{d_i\}$$
- Multiple edges must merge to compute $In(s)$ from $Pred(s)$
  All incoming definitions reach
  $$In(s) = \bigcup_{\forall p \in Pred(s)} Out(p)$$

Reaching definitions

Dataflow equations

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
  $$Init(s) = \varnothing$$

Reaching definitions

Dataflow equations

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
  $$Init(s) = \varnothing$$
- Note that often $Out(s)$ is written
  $$Out(s : d_i := ...) = (In(s) - Kill(s)) \cup Gen(s)$$
  The *Gen* and *Kill* sets can often be precomputed
  Also, 🕮EaC combines *In* and *Out* to use only one equation

Reaching definitions
Observations

- 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] 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. $In(s) = \bigcup_{\forall p \in Pred(s)} Out(p)$
- **Value set** - the bits information being passed around
  - e.g. Sets of definitions

[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. $In(s) = \bigcup_{\forall p \in Pred(s)} Out(p)$
- **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. $In(s) = \bigcup_{\forall p \in Pred(s)} Out(p)$
- **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
- Some properties of the above to ensure termination[2]

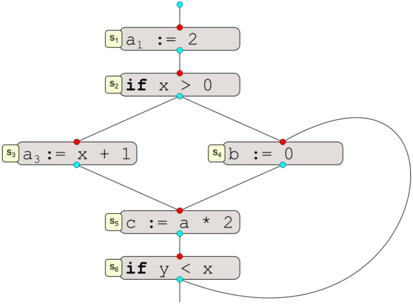
[2] In a later lecture

```
for each node[3], n, do
    Initialise n
while values changing do
    for each node do
        Apply meet and transfer function
```

There are many, many data flow algorithms that fit

[3] Note, node not statement. Include special start node

Reaching definitions control flow example - Calculate RD sets?



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

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

$$\Downarrow$$

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

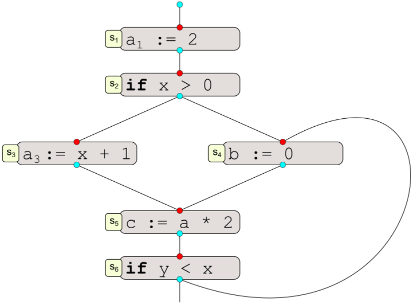
Node	s_1	s_2	s_3	s_4	s_5	s_6
RD[4]	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

[4] For brevity, *In* and *Out* are combined

Algorithms

Round-robin iterative algorithm

Reaching definitions control flow example - Calculate RD sets?



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

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

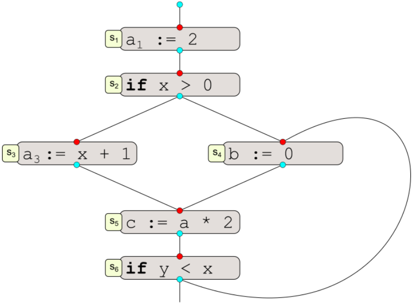
$$\Downarrow$$

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

Node	s_1	s_2	s_3	s_4	s_5	s_6
RD[4]	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$
	$\varnothing$					

[4] For brevity, *In* and *Out* are combined

Reaching definitions control flow example - Calculate RD sets?



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

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

$$\Downarrow$$

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

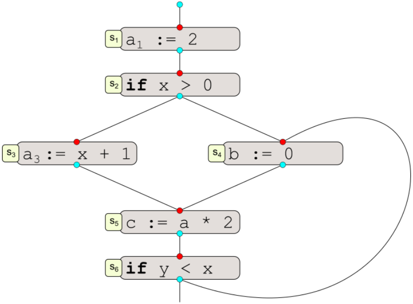
Node	s_1	s_2	s_3	s_4	s_5	s_6
RD[4]	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$
	$\varnothing$	a_1				

[4] For brevity, *In* and *Out* are combined

Algorithms

Round-robin iterative algorithm

Reaching definitions control flow example - Calculate RD sets?



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

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

$$\Downarrow$$

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

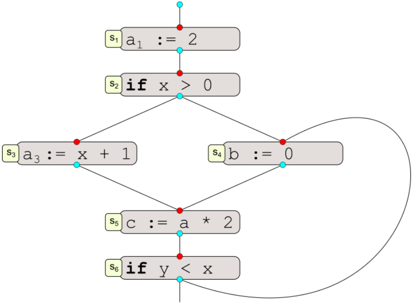
Node	s_1	s_2	s_3	s_4	s_5	s_6
RD[4]	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
	$\emptyset$	a_1	a_1			

[4] For brevity, *In* and *Out* are combined

Algorithms

Round-robin iterative algorithm

Reaching definitions control flow example - Calculate RD sets?



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

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

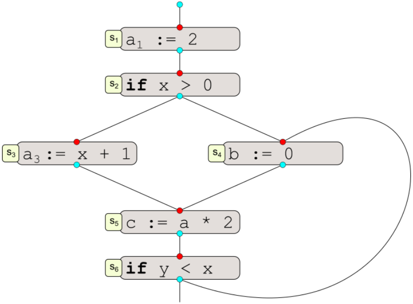
$$\Downarrow$$

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

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		

[4] For brevity, *In* and *Out* are combined

Reaching definitions control flow example - Calculate RD sets?



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

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

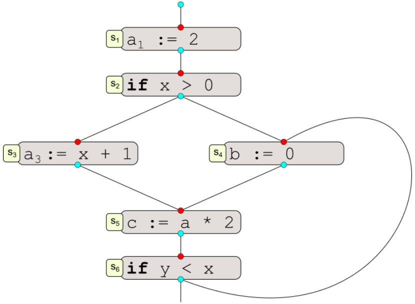
$$\Downarrow$$

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

Node	s_1	s_2	s_3	s_4	s_5	s_6
RD[4]	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$
	$\varnothing$	a_1	a_1	a_1	a_1, a_3, b	

[4] For brevity, *In* and *Out* are combined

Reaching definitions control flow example - Calculate RD sets?



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

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

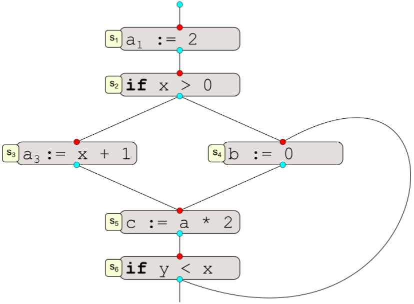
$$\Downarrow$$

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

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

[4] For brevity, *In* and *Out* are combined

Reaching definitions control flow example - Calculate RD sets?



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

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

$$\Downarrow$$

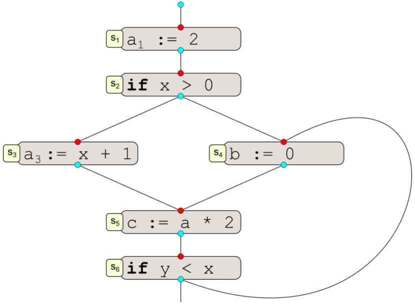
$$RD(s) = \bigcup_{\forall p : d_i = \ldots \in Pred(s)} (RD(p) - \{d_j; \forall j\}) \cup \{d_i\}$$

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
	$\emptyset$					

[4] For brevity, *In* and *Out* are combined

Algorithms
Round-robin iterative algorithm

Reaching definitions control flow example - Calculate RD sets?



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

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

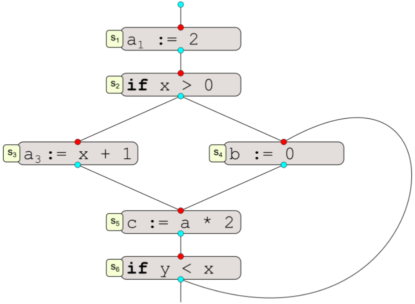
$$\Downarrow$$

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

Node	s_1	s_2	s_3	s_4	s_5	s_6
RD[4]	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$
	$\varnothing$	a_1	a_1	a_1	a_1, a_3, b	a_1, a_3, b, c
	$\varnothing$	a_1				

[4] For brevity, *In* and *Out* are combined

Reaching definitions control flow example - Calculate RD sets?



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

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

$$\Downarrow$$

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

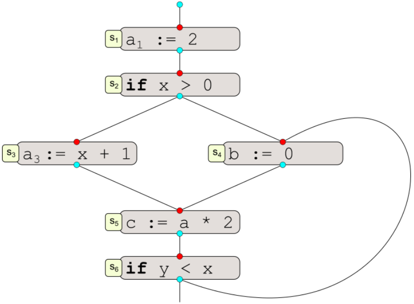
Node	s_1	s_2	s_3	s_4	s_5	s_6
RD[4]	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$
	$\varnothing$	a_1	a_1	a_1	a_1, a_3, b	a_1, a_3, b, c
	$\varnothing$	a_1	a_1			

[4] For brevity, *In* and *Out* are combined

Algorithms

Round-robin iterative algorithm

Reaching definitions control flow example - Calculate RD sets?



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

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

$$\Downarrow$$

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

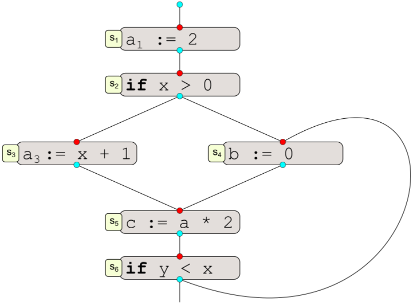
Node	s_1	s_2	s_3	s_4	s_5	s_6
RD[4]	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$
	$\varnothing$	a_1	a_1	a_1	a_1, a_3, b	a_1, a_3, b, c
	$\varnothing$	a_1	a_1	a_1, a_3, b, c		

[4] For brevity, *In* and *Out* are combined

Algorithms

Round-robin iterative algorithm

Reaching definitions control flow example - Calculate RD sets?



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

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

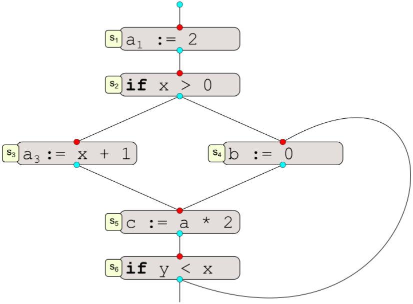
$$\Downarrow$$

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

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	

[4] For brevity, *In* and *Out* are combined

Reaching definitions control flow example - Calculate RD sets?



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

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

$$\Downarrow$$

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

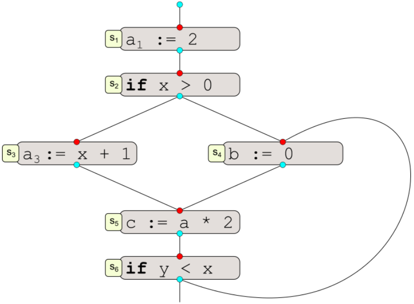
Node	s_1	s_2	s_3	s_4	s_5	s_6
RD[4]	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$
	$\varnothing$	a_1	a_1	a_1	a_1, a_3, b	a_1, a_3, b, c
	$\varnothing$	a_1	a_1	a_1, a_3, b, c	a_1, a_3, b, c	a_1, a_3, b, c

[4] For brevity, *In* and *Out* are combined

Algorithms

Round-robin iterative algorithm

Reaching definitions control flow example - Calculate RD sets?



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

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

$$\Downarrow$$

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

Node	s_1	s_2	s_3	s_4	s_5	s_6
RD[4]	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$
	$\varnothing$	a_1	a_1	a_1	a_1, a_3, b	a_1, a_3, b, c
	$\varnothing$	a_1	a_1	a_1, a_3, b, c	a_1, a_3, b, c	a_1, a_3, b, c
	$\varnothing$	a_1	a_1	a_1, a_3, b, c	a_1, a_3, b, c	a_1, a_3, b, c

[4] For brevity, *In* and *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

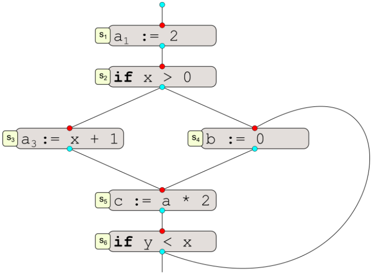
- Round-robin algorithm is slow, may require many passes through nodes
- Can speed up by considering basic blocks (e.g. compute `Gen` and `Kill` for whole block)
- Only nodes which have inputs changed need to be processed - use work list
- Reducible graphs can be handled more efficiently (see 🕮`EaC`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

May reduce number of iterations by changing evaluation order[5]

- Backward analysis - evaluate node after successors
  Use **postorder**
- Forward analysis - evaluate node before successors
  Use **reverse postorder**

Orders for reaching definitions example

Post(1)	$s_4 s_6 s_5 s_3 s_2 s_1$
Post(2)	$s_6 s_5 s_4 s_3 s_2 s_1$
Rev(1)	$s_1 s_2 s_3 s_5 s_6 s_4$
Rev(2)	$s_1 s_2 s_3 s_4 s_5 s_6$



[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

PPar CDT Advert

EPSRC Centre for Doctoral Training in

Pervasive Parallelism

- 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

The biggest revolution in the technological landscape for fifty years

Now accepting applications!
Find out more and apply at:
pervasiveparallelism.inf.ed.ac.uk

THE UNIVERSITY of EDINBURGH
informatics

EPSRC
Engineering and Physical Sciences Research Council