Compiler Optimisation

4 – Dataflow Analysis

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

Institute for Computing Systems Architecture
School of Informatics
University of Edinburgh

2019
This lecture:

- Data flow termination
- More data flow examples
- Dominance
- Static single-assignment form
Data flow termination

**Fixed Point**
A fixed point, \( x \), of function \( f : T \rightarrow T \) is when \( f(x) = x \)

**Partial ordering**
A binary relation, \( \leq \), among elements of a set, \( S \), that is:

- \( \forall a \in S, \ a \leq a \) \hspace{1cm} \text{reflexive}
- \( \forall a, b \in S, \ a \leq b \land b \leq a \implies a = b \) \hspace{1cm} \text{antisymmetric}
- \( \forall a, b, c \in S \ a \leq b \land b \leq c \implies a \leq c \) \hspace{1cm} \text{transitive}

**Partially ordered set (poset)**
A set with a partial order
Data flow termination

Join semi-lattice

Partially ordered set that has a join (a least upper bound) for any nonempty finite subset

\[ \forall x, y, z \in V \]
\[ x \land (y \land z) = (x \land y) \land z \]  \quad \text{Associativity}
\[ x \land y = y \land x \]  \quad \text{Commutativity}
\[ x \land x = x \]  \quad \text{Idempotency}
\[ \top \land x = x \]  \quad \text{Top element}

Note, *meet semi-lattice* has \( \bot \) and *complete lattice* has both

Often just use ‘semi-lattice’
Data flow termination
Lattices

Example lattice

Hasse diagram for the set of all subsets of $x, y, z$, ordered by inclusion
Is it a lattice?
Data flow termination

Lattices

Example lattice

Hasse diagram for the set of all subsets of $x, y, z$, ordered by inclusion

Yes, and is lattice for LiveOut, with variables, $x, y, z$
Example non-lattices
Data flow termination
Lattices

Example non-lattices

Diagram showing nodes and connections for non-lattice example.
Data flow termination

Lattices

Example non-lattices

![Diagram of non-lattice example](image)
Data flow termination

Each block or statement has a particular function, e.g. based on *Kill* and *Gen* sets
Let $F$ be the set of transfer functions

### Sufficient termination constraints

- $V$ and $\wedge$ for a semi-lattice
- $F$ has identity: $I(x) = x, \forall x \in V$
- $F$ is closed under composition: $\forall f, g \in F, h = f \circ g \in F$
- $F$ is monotonic: $\forall f \in F, f(x \wedge y) \leq f(x) \wedge f(y)$

See 10.11
Liveness

- A variable $v$ is **live-out** of statement $s$ if $v$ is used along some control path starting at $s$
- Otherwise, we say that $v$ is **dead**
- A variable is live if it holds a value that may be needed in the future

Information flows *backwards* from statement to predecessors
Liveness useful for optimisations (e.g. register allocation, store elimination, dead code...)
A variable $v$ is live-out of statement $s$ if $v$ is used along some control path starting at $s$. 
A variable $v$ is live-out of statement $s$ if $v$ is used along some control path starting at $s$. 

Liveness
A variable $v$ is live-out of statement $s$ if $v$ is used along some control path starting at $s$. 

Liveness
A variable v is live-out of statement s if v is used along some control path starting at s.
A variable $v$ is live-out of statement $s$ if $v$ is used along some control path starting at $s$. 

Liveness
A variable \( v \) is live-out of statement \( s \) if \( v \) is used along some control path starting at \( s \).
Liveness

A variable $v$ is live-out of statement $s$ if $v$ is used along some control path starting at $s$. 

For the given diagram, $a$ is live-out of $s_1$, $b$ is live-out of $s_4$, and $c$ is live-out of $s_5$. 

Note: $b, c$ defined but not used, so $s_4, s_5$ useless, if removed $s_1, s_3$ useless. 

A variable $v$ is live-out of statement $s$ if $v$ is used along some control path starting at $s$. 

({x, y})

({x, y})

({x, y, a})

({x, y, a})

{}
Liveness

- Live variables come up from their successors using them
  \[ Out(s) = \bigcup_{\forall n \in \text{Succ}(s)} \text{In}(n) \]
- Transfer back across the node
  \[ \text{In}(s) = Out(s) - \text{Kill}(s) \cup \text{Gen}(s) \]
- Used variables are live
  \[ \text{Gen}(s) = \{ u \text{ such that } u \text{ is used in } s \} \]
- Defined but not used variables are killed
  \[ \text{Kill}(s) = \{ d \text{ such that } d \text{ is defined in } s \text{ but not used in } s \} \]
- If we don’t know, start with empty
  \[ \text{Init}(s) = \emptyset \]
Constant propagation - show variable has same constant value at some point
  - Strictly speaking does not compute expressions except \( x := \text{const} \), or \( x := y \) and \( y \) is constant
  - Often combined with constant folding that computes expressions

Copy propagation - show variable is copy of other variable

Available expressions - set of expressions reaching by all paths

Very busy expressions - expressions evaluated on all paths leaving block - for code hoisting

Definite assignment - variable always assigned before use

Redundant expressions, and partial redundant expressions

Many more - read about them!
 CFG node $b_i$ dominates $b_j$, written $b_i \gg b_j$, iff every path from the start node to $b_j$ goes through $b_i$.

Design data flow equations to compute which nodes dominate each node:

What direction?
What value set?
What transfer?
What Meet?
Initial values?
CFG node $b_i$ dominates $b_j$, written $b_i \gg b_j$, iff every path from the start node to $b_j$ goes through $b_i$.

Design data flow equations to compute which nodes dominate each node.

What direction?
What value set?
What transfer?
What Meet?
Initial values?
Dominators

CFG node $b_i$ dominates $b_j$, written $b_i \gg b_j$, iff every path from the start node to $b_j$ goes through $b_i$

Design data flow equations to compute which nodes dominate each node

**Direction**: Forward
What value set?  
What transfer?  
What Meet?  
Initial values?
Dominators

CFG node $b_i$ dominates $b_j$, written $b_i \gg b_j$, iff every path from the start node to $b_j$ goes through $b_i$

Design data flow equations to compute which nodes dominate each node

**Direction:** Forward

**What value set?**

What transfer?

What Meet?

Initial values?
CFG node $b_i$ dominates $b_j$, written $b_i \gg b_j$, iff every path from the start node to $b_j$ goes through $b_i$.

Design data flow equations to compute which nodes dominate each node.

**Direction:** Forward  
**Values:** Sets of nodes  
What transfer?  
What Meet?  
Initial values?
Dominators

CFG node $b_i$ dominates $b_j$, written $b_i \gg b_j$, iff every path from the start node to $b_j$ goes through $b_i$

Design data flow equations to compute which nodes dominate each node

*Direction:* Forward

*Values:* Sets of nodes

**What transfer?**

**What Meet?**

Initial values?
CFG node $b_i$ dominates $b_j$, written $b_i \gg b_j$, iff every path from the start node to $b_j$ goes through $b_i$.

Design data flow equations to compute which nodes dominate each node

**Direction:** Forward

**Values:** Sets of nodes

**Transfer:** $Out(n) = In(n) \cup \{n\}$

What Meet?

Initial values?
Dominators

CFG node $b_i$ dominates $b_j$, written $b_i \gg b_j$, iff every path from the start node to $b_j$ goes through $b_i$.

Design data flow equations to compute which nodes dominate each node.

**Direction:** Forward

**Values:** Sets of nodes

**Transfer:** $Out(n) = In(n) \cup \{n\}$

**What Meet?**

Initial values?
Dominators

CFG node $b_i$ dominates $b_j$, written $b_i \gg b_j$, iff every path from the start node to $b_j$ goes through $b_i$.

Design data flow equations to compute which nodes dominate each node.

**Direction:** Forward

**Values:** Sets of nodes

**Transfer:** $Out(n) = In(n) \cup \{n\}$

**Meet:** $In(n) = \bigcap_{\forall n \in Pred(s)} Out(s)$

Initial values?
Dominators

CFG node $b_i$ dominates $b_j$, written $b_i \gg b_j$, iff every path from the start node to $b_j$ goes through $b_i$.

Design data flow equations to compute which nodes dominate each node.

**Direction:** Forward

**Values:** Sets of nodes

**Transfer:** $Out(n) = In(n) \cup \{n\}$

**Meet:** $In(n) = \bigcap_{\forall n \in Pred(s)} Out(s)$

Initial values?
CFG node $b_i$ dominates $b_j$, written $b_i \gg b_j$, iff every path from the start node to $b_j$ goes through $b_i$.

Design data flow equations to compute which nodes dominate each node

**Direction:** Forward

**Values:** Sets of nodes

**Transfer:** $\text{Out}(n) = \text{In}(n) \cup \{n\}$

**Meet:** $\text{In}(n) = \bigcap_{\forall n \in \text{Pred}(s)} \text{Out}(s)$

**Initial:** $\text{Init}(n_0) = \{n_0\}; \text{Init}(n) = \text{all}$
## Dominators

<table>
<thead>
<tr>
<th><strong>Post-dominator</strong></th>
<th>Node ( z ) is said to post-dominate a node ( n ) if all paths to the exit node of the graph starting at ( n ) must go through ( z )</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Strict dominance</strong></td>
<td>Node ( a ) strictly dominates ( b ) iff ( a \gg b \land a \neq b )</td>
</tr>
<tr>
<td><strong>Immediate dominator</strong></td>
<td>( idom(n) ) strictly dominates ( n ) but not any other node that strictly dominates ( n )</td>
</tr>
<tr>
<td><strong>Dominator tree</strong></td>
<td>Tree where node’s children are those it immediately dominates</td>
</tr>
<tr>
<td><strong>Dominance frontier</strong></td>
<td>( DF(n) ) is set of nodes, ( d ) s.t. ( n ) dominates an immediate predecessor of ( d ), but ( n ) does not strictly dominate ( d )</td>
</tr>
</tbody>
</table>
Dominators

Example: Dominator tree

Where are dominance frontiers?
Dominators

Example: Dominator tree

\[ \text{DF}(b_5) = \{ b_3 \} \]
Dominators

Example: Dominator tree

```
DF(b_1) = \{b_1\}
```
Static single-assignment form (SSA)

- Often allowing variable redefinition complicates analysis
- In SSA:
  - One variable per definition
  - Each use refers to one definition
  - Definitions merge with $\phi$ functions
  - $\Phi$ functions execute instantaneously in parallel
- Used by or simplifies many analyses
Static single-assignment form (SSA)

Example: Intuitive conversion to SSA

Original CFG
Example: Intuitive conversion to SSA

Rename multiple definitions of same variable
Static single-assignment form (SSA)

Example: Intuitive conversion to SSA

Repeatedly merge definitions with $\phi$
Static single-assignment form (SSA)

Example: Intuitive conversion to SSA

Now in SSA form
Static single-assignment form (SSA)

Types of SSA

- **Maximal SSA** - Places $\phi$ node for variable $x$ at every *join* block if block uses or defines $x$
- **Minimal SSA** - Places $\phi$ node for variable $x$ at every *join* block with 2+ reaching definitions of $x$
- **Semipruned SSA** - Eliminates $\phi$s not live across block boundaries
- **Pruned SSA** - Adds liveness test to avoid $\phi$s of dead definitions
Static single-assignment form (SSA)
Conversion to SSA sketch

For each definition\(^1\) of \(x\) in block \(b\), add \(\phi\) for \(x\) in each block in \(DF(b)\)
- This introduces more definitions, so repeat
- Rename variables
- Can be done in \(T(n) = O(n)\), if liveness cheap

---

\(^1\)Different liveness tests (including none) here change SSA type
\(^2\)See EAC 9.3.1-9.3.4
Static single-assignment form (SSA)
Conversion from SSA sketch\textsuperscript{3}

- Cannot just remove $\phi$ nodes; optimisations make this unsafe
- Place copy operations on incoming edges
- Split edges if necessary
- Delete $\phi$s
- Remove redundant copies afterwards

\textsuperscript{3}See EaC 9.3.5
Static single-assignment form (SSA)
Conversion from SSA

Example: Intuitive conversion from SSA

Original SSA CFG
Static single-assignment form (SSA)

Conversion from SSA

Example: Intuitive conversion from SSA

```
1. a₁ := 2
2. a₃ := a₁
3. if x > 0
   4. a₂ := x + 1
   5. a₄ := a₂
5. a₄ := μ(a₂, a₃)
6. a₃ := μ(a₁, a₄)
7. b := 0
8. a₄ := a₃
8. a₃ := a₄
9. c := a₄ * 2
9. if y < x
```

Place copies

Cannot insert

Actually fine here, just for demo
Static single-assignment form (SSA)
Conversion from SSA

Example: Intuitive conversion from SSA

Split where necessary
Static single-assignment form (SSA)

Conversion from SSA

Example: Intuitive conversion from SSA

Remove $\phi$s
Static single-assignment form (SSA)

Extensions

- Dataflow assumes that all paths in the CFG are taken hence conservative approximations
  - Guarded SSA attempts to overcome this by having additional meet nodes $\gamma$, $\eta$ and $\mu$ to carry conditional information around
  - Array based SSA models access patterns\(^4\)
- Inter-procedural challenging. Pointers destroy analysis! Large research effort in points-to analysis.

\(^4\)Can be generalised using Presburger formula
Three possible values per variable

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\top$</td>
<td>Not a constant</td>
</tr>
<tr>
<td>$k$</td>
<td>Constant value $k$</td>
</tr>
<tr>
<td>$\bot$</td>
<td>Not computed (maybe never)</td>
</tr>
</tbody>
</table>

Meet is $\top \land x = x$, $\bot \land x = \bot$, $c \land c = c$, $c \land d = \bot$ if $c \neq d$

Transfer functions compute value if all inputs are constant
Summary

- Data flow termination
- More data flow examples
- Dominance
- Static single-assignment form
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

The biggest revolution in the technological landscape for fifty years

Now accepting applications!