SSA-Form Register Allocation

Foundations

Sebastian Hack

Compiler Construction Course
Winter Term 2009/2010

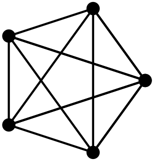
UNIVERSITÄT
DES
SAARLANDES

Overview

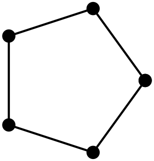
1. Graph Theory
   - Perfect Graphs
   - Chordal Graphs
2. SSA Form
   - Dominance
   - ϕ-functions
3. Interference Graphs
   - Non-SSA Interference Graphs
   - Perfect Elimination Orders
   - Chordal Graphs
4. Interference Graphs of SSA-form Programs
   - Dominance and Liveness
   - Dominance and Interference
   - Spilling
   - Implementing ϕ-functions
5. Intuition

Overview

Complete Graphs and Cycles

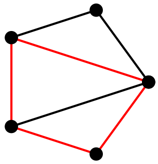


Complete Graph K^5

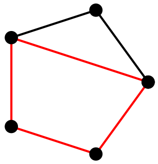


Cycle C^5

Induced Subgraphs

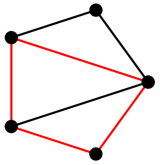


Graph with a C^4 subgraph

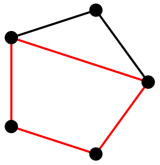


Graph with a C^4 induced subgraph

Induced Subgraphs



Graph with a C^4 subgraph



Graph with a C^4 induced subgraph

Note

Induced complete graphs are called cliques

Clique number and Chromatic number

Definition

$\omega(G)$ Size of the largest clique in G

$\chi(G)$ Number of colors in a minimum coloring of G

Clique number and Chromatic number

Definition

$\omega(G)$ Size of the largest clique in G

$\chi(G)$ Number of colors in a minimum coloring of G

Corollary

$\omega(G) \leq \chi(G)$ *holds for each graph* G

Clique number and Chromatic number

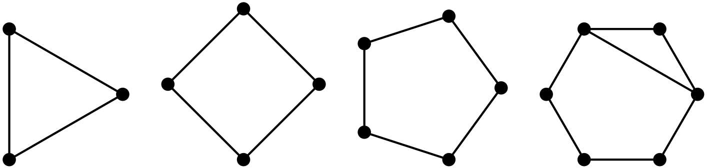
Definition

$\omega(G)$ Size of the largest clique in G

$\chi(G)$ Number of colors in a minimum coloring of G

Corollary

$\omega(G) \leq \chi(G)$ holds for each graph G



$\omega(G)$	3	2	2	3
$\chi(G)$	3	2	3	3

Perfect Graphs

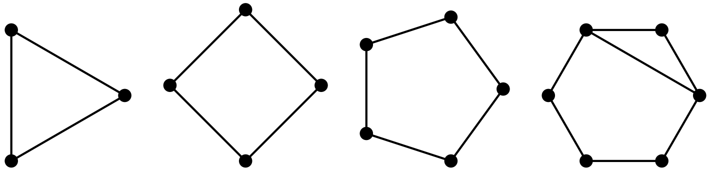
Definition

G is perfect $\iff \chi(H) = \omega(H)$ for each induced subgraph H of G

Perfect Graphs

Definition

G is perfect $\iff \chi(H) = \omega(H)$ for each induced subgraph H of G

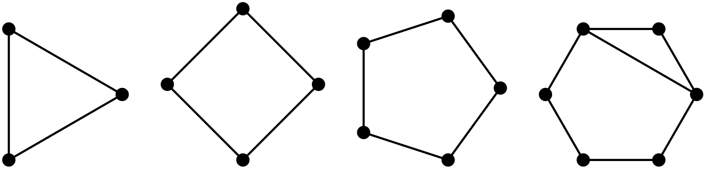


perfect?

Perfect Graphs

Definition

G is perfect $\iff$ $\chi(H) = \omega(H)$ for each induced subgraph H of G



perfect?	✓	✓		

Chordal Graphs

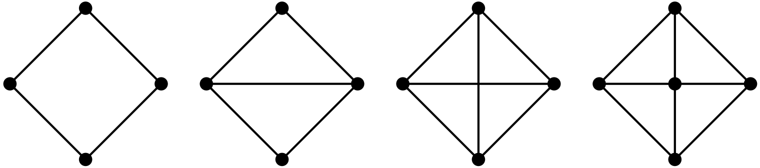
Definition

G is chordal $\iff$ G contains no induced cycles longer than 3

Chordal Graphs

Definition

G is chordal $\iff$ G contains no induced cycles longer than 3

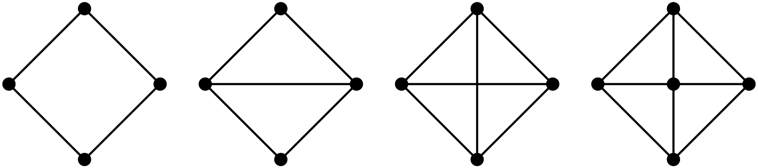


chordal?

Chordal Graphs

Definition

G is chordal $\iff$ G contains no induced cycles longer than 3



chordal?		$\checkmark$	$\checkmark$	

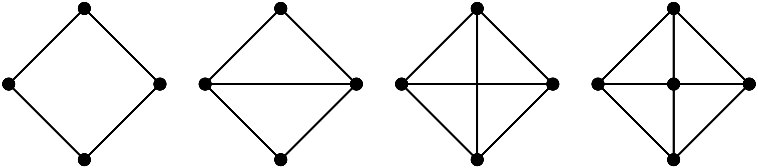
Theorem

Chordal graphs are perfect

Chordal Graphs

Definition

G is chordal $\iff$ G contains no induced cycles longer than 3



chordal?		✓	✓	

Theorem

Chordal graphs are perfect

Theorem

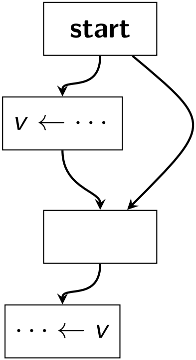
Chordal graphs can be colored optimally in $O(|V| \cdot \omega(G))$

Overview

Dominance

Definition

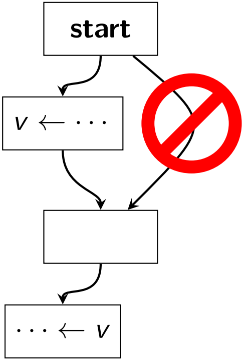
Every use of a variable is dominated by its definition



Dominance

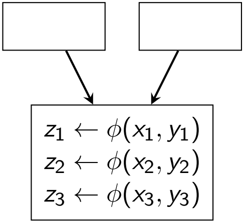
Definition

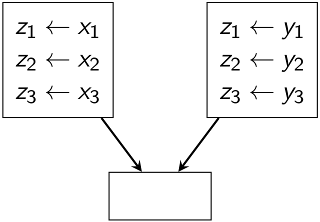
Every use of a variable is dominated by its definition



- You cannot reach the use without passing by the definition
- Else, you could use uninitialized variables
- Dominance induces a tree on the control flow graph
- Sometimes called strict SSA

What do ϕ-functions mean?

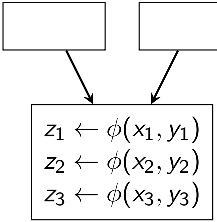


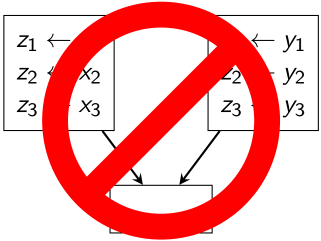


Frequent misconception

Put a sequence of copies in the predecessors

What do ϕ-functions mean?



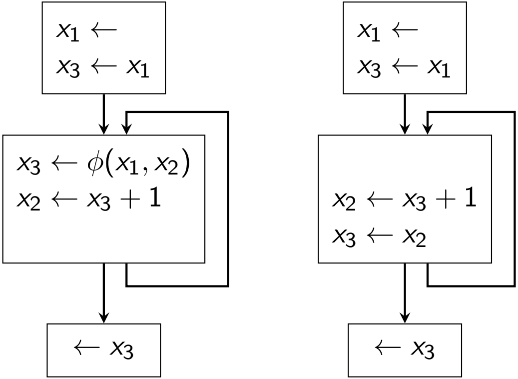


Frequent misconception

Put a sequence of copies in the predecessors

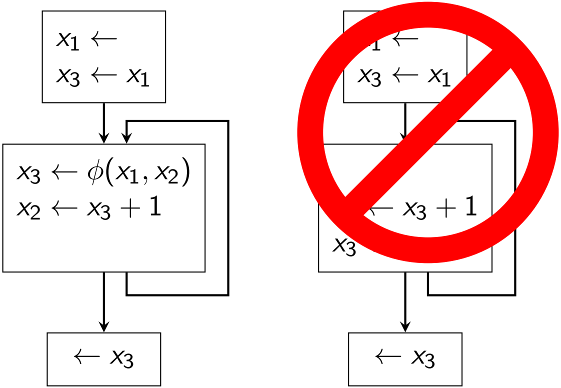
What do ϕ-functions mean?

Lost Copy Problem



What do ϕ-functions mean?

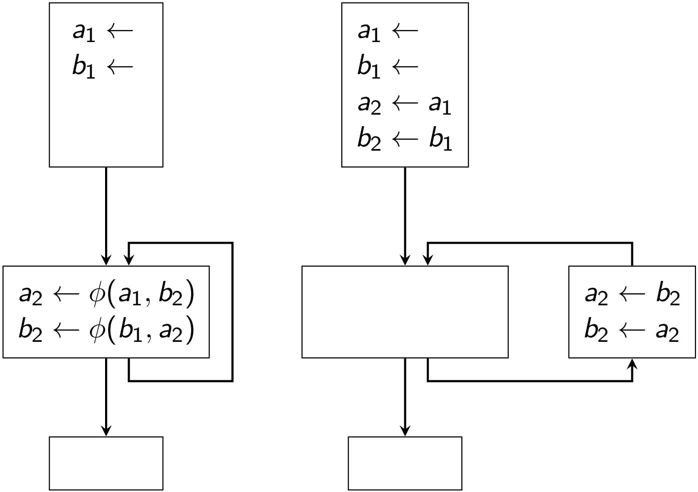
Lost Copy Problem



- Cannot simply push copies in predecessor
- Copies are also executed if we jump out of the loop
- Need to remove critical edges (loopback edge)

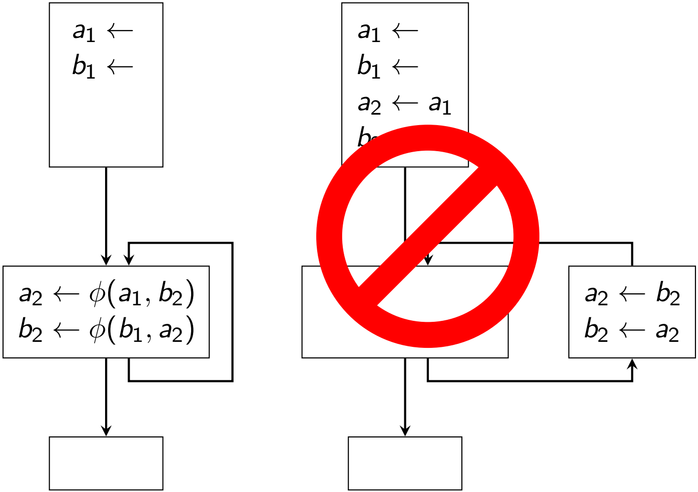
What do ϕ-functions mean?

Swap Problem



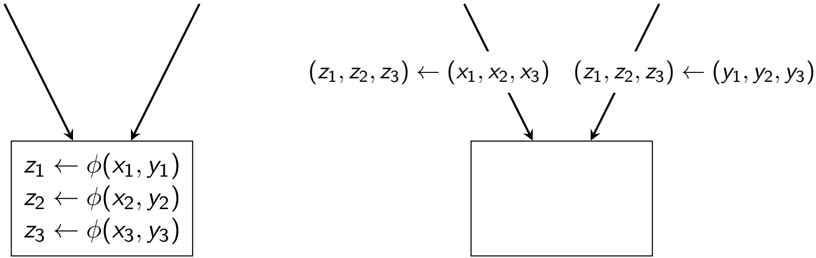
What do ϕ-functions mean?

Swap Problem



- a_2 overwritten before used
- All ϕs in a block need to be evaluated simultaneously

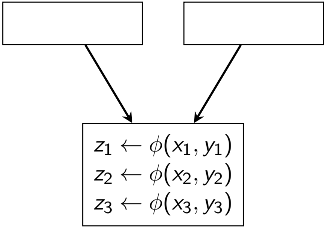
What do ϕ-functions mean?



The Reality

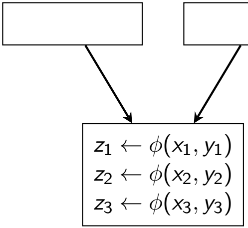
ϕ-functions correspond to parallel copies on the incoming edges

ϕ-functions and uses

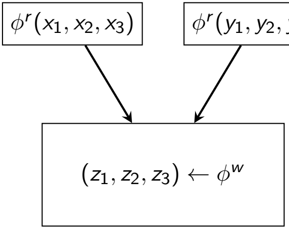


- Does not fulfill dominance property
- ϕs do not use their operands in the ϕ-block
- Uses happen in the predecessors

ϕ-functions and uses



- Does not fulfill dominance property
- ϕs do not use their operands in the ϕ-block
- Uses happen in the predecessors



Split ϕ-functions in two parts:

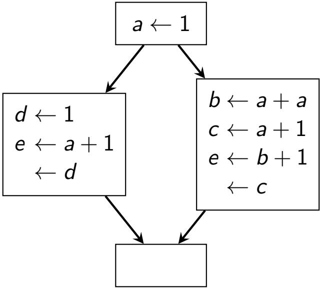
- Split critical edges
- Read part (ϕ^r) in the predecessors
- Write part (ϕ^w) in the block
- Correct modelling of liveness

Overview

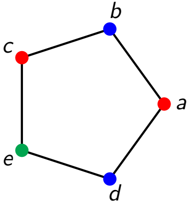
Non-SSA Interference Graphs

An inconvenient property

Program



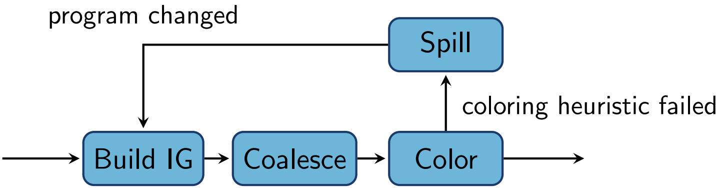
Interference Graph



- The number of live variables at each instruction (register pressure) is 2
- However, we need 3 registers for a correct register allocation
- In theory, this gap can be arbitrarily large (Mycielski Graphs)

Graph-Coloring Register Allocation

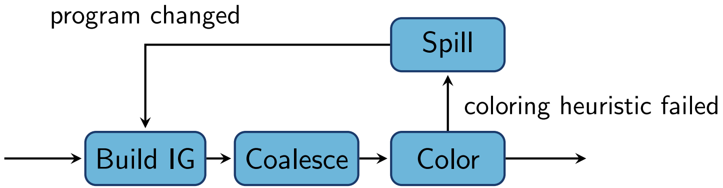
[Chaitin '80, Briggs '92, Appel & George '96, Park & Moon '04]



- Every undirected graph can occur as an interference graph
  $\Longrightarrow$ Finding a k-coloring is NP-complete
- Color using heuristic
  $\Longrightarrow$ Iteration necessary
- Might introduce spills although IG is k-colorable
- Rebuilding the IG each iteration is costly

Graph-Coloring Register Allocation

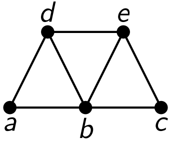
[Chaitin '80, Briggs '92, Appel & George '96, Park & Moon '04]



- Spill-code insertion is crucial for the program's performance
- Hence, it should be very sensitive to the structure of the program
  - Place load and stores carefully
  - Avoid spilling in loops!
- Here, it is merely a fail-safe for coloring

Coloring

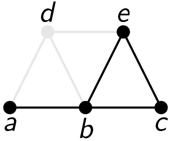
- Subsequently remove the nodes from the graph



elimination order

Coloring

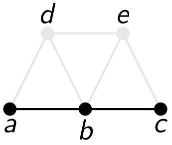
- Subsequently remove the nodes from the graph



elimination order
d,

Coloring

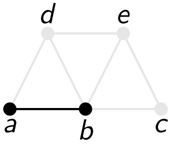
- Subsequently remove the nodes from the graph



elimination order
d, e,

Coloring

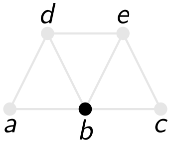
- Subsequently remove the nodes from the graph



elimination order
d, e, c,

Coloring

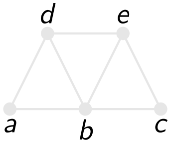
- Subsequently remove the nodes from the graph



elimination order
d, e, c, a,

Coloring

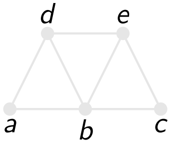
- Subsequently remove the nodes from the graph



elimination order
d, e, c, a, b

Coloring

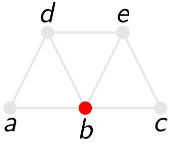
- Subsequently remove the nodes from the graph
- Re-insert the nodes in reverse order
- Assign each node the next possible color



elimination order
d, e, c, a, b

Coloring

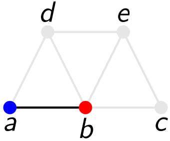
- Subsequently remove the nodes from the graph
- Re-insert the nodes in reverse order
- Assign each node the next possible color



elimination order
d, e, c, a,

Coloring

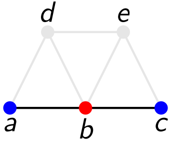
- Subsequently remove the nodes from the graph
- Re-insert the nodes in reverse order
- Assign each node the next possible color



elimination order
d, e, c,

Coloring

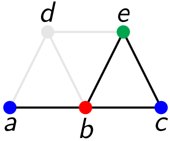
- Subsequently remove the nodes from the graph
- Re-insert the nodes in reverse order
- Assign each node the next possible color



elimination order
d, e,

Coloring

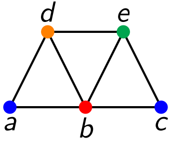
- Subsequently remove the nodes from the graph
- Re-insert the nodes in reverse order
- Assign each node the next possible color



elimination order
d,

Coloring

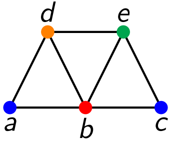
- Subsequently remove the nodes from the graph
- Re-insert the nodes in reverse order
- Assign each node the next possible color



elimination order

Coloring

- Subsequently remove the nodes from the graph
- Re-insert the nodes in reverse order
- Assign each node the next possible color



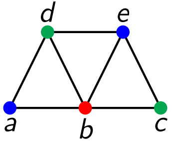
elimination order

But...

this graph is 3-colorable. We obviously picked a wrong order.

Coloring

- Subsequently remove the nodes from the graph
- Re-insert the nodes in reverse order
- Assign each node the next possible color



elimination order

But...

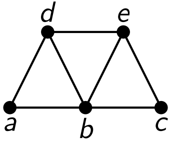
this graph is 3-colorable. We obviously picked a wrong order.

Coloring

PEOs

Perfect Elimination Order (PEO)

All not yet eliminated neighbors of a node are mutually connected



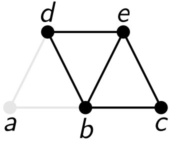
elimination order

Coloring

PEOs

Perfect Elimination Order (PEO)

All not yet eliminated neighbors of a node are mutually connected



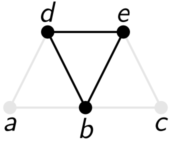
elimination order
a,

Coloring

PEOs

Perfect Elimination Order (PEO)

All not yet eliminated neighbors of a node are mutually connected



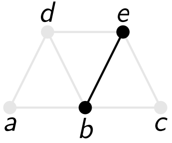
elimination order
a, c,

Coloring

PEOs

Perfect Elimination Order (PEO)

All not yet eliminated neighbors of a node are mutually connected



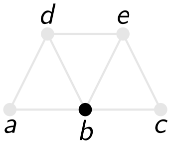
elimination order
a, c, d,

Coloring

PEOs

Perfect Elimination Order (PEO)

All not yet eliminated neighbors of a node are mutually connected



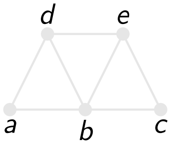
elimination order
a, c, d, e,

Coloring

PEOs

Perfect Elimination Order (PEO)

All not yet eliminated neighbors of a node are mutually connected



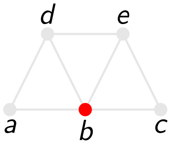
elimination order
a, c, d, e, b

Coloring

PEOs

Perfect Elimination Order (PEO)

All not yet eliminated neighbors of a node are mutually connected

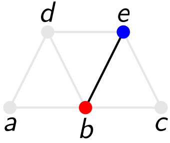


elimination order
a, c, d, e,

Coloring
PEOs

Perfect Elimination Order (PEO)

All not yet eliminated neighbors of a node are mutually connected



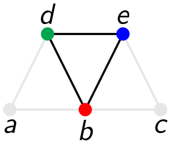
elimination order
a, c, d,

Coloring

PEOs

Perfect Elimination Order (PEO)

All not yet eliminated neighbors of a node are mutually connected



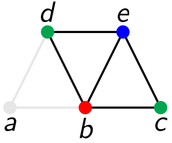
elimination order
a, c,

Coloring

PEOs

Perfect Elimination Order (PEO)

All not yet eliminated neighbors of a node are mutually connected



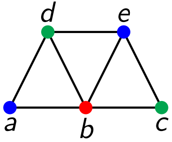
elimination order
a,

Coloring

PEOs

Perfect Elimination Order (PEO)

All not yet eliminated neighbors of a node are mutually connected



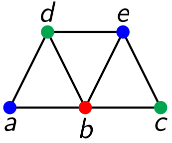
elimination order

Coloring

PEOs

Perfect Elimination Order (PEO)

All not yet eliminated neighbors of a node are mutually connected



elimination order

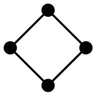
From Graph Theory [Berge '60, Fulkerson/Gross '65, Gavril '72]

- A PEO allows for an optimal coloring in $k \times |V|$
- The number of colors is bound by the size of the largest clique

Coloring

PEOs

- Graphs with holes larger than 3 have no PEO, e.g.

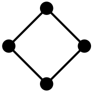


- G has a PEO $\iff$ G is chordal

Coloring

PEOs

- Graphs with holes larger than 3 have no PEO, e.g.



- G has a PEO $\iff$ G is chordal

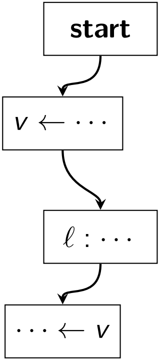
Core Theorem of SSA Register Allocation

- The dominance relation in SSA programs induces a PEO in the IG
- Thus, SSA IGs are chordal

Overview

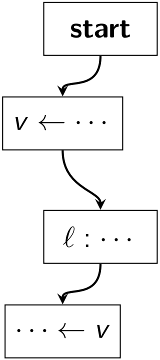
Liveness and Dominance

- Each instruction ℓ where a variable v is live, is dominated by v



Liveness and Dominance

- Each instruction ℓ where a variable v is live, is dominated by v

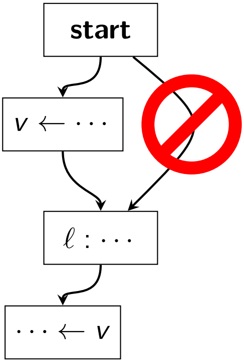


Why?

- Assume ℓ is not dominated by v
- Then there's a path from **start** to some usage of v not containing the definition of v
- This cannot be since each value must have been defined before it is used

Liveness and Dominance

- Each instruction ℓ where a variable v is live, is dominated by v

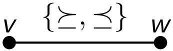


Why?

- Assume ℓ is not dominated by v
- Then there's a path from **start** to some usage of v not containing the definition of v
- This cannot be since each value must have been defined before it is used

Interference and Dominance

- Assume v, w interfere, i.e. they are live at some instruction ℓ
- Then, $v \succeq \ell$ and $w \succeq \ell$
- Since dominance is a tree, either $v \succeq w$ or $w \succeq v$



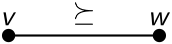
Interference and Dominance

- Assume v, w interfere, i.e. they are live at some instruction ℓ
- Then, $v \succeq \ell$ and $w \succeq \ell$
- Since dominance is a tree, either $v \succeq w$ or $w \succeq v$

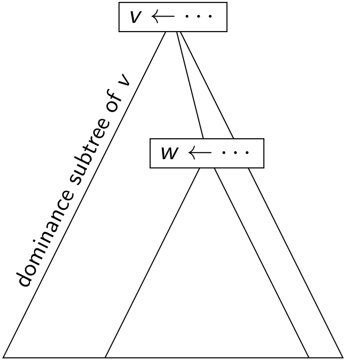
Consequences

- Each edge in the IG is directed by dominance
- The interference graph is an “excerpt” of the dominance relation

Interference and Dominance

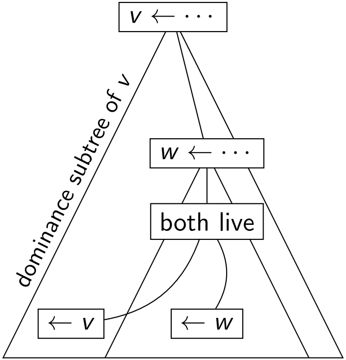
- Assume 

- Then, v is live at w



Interference and Dominance

- Assume $v \overset{\succeq}{\bullet\!\!-\!\!-\!\!\bullet} w$
- Then, v is live at w

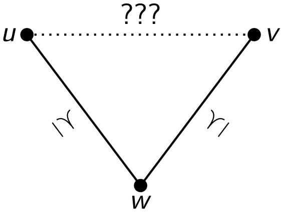


Why?

- If v and w interfere then there is a place where both are live
- w dominates all places where w is live
- Hence, v is live inside w's dominance tree
- Thus, v is live at w

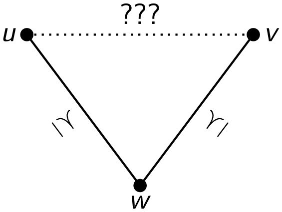
Interference and Dominance

Consider three nodes u, v, w in the IG:



Interference and Dominance

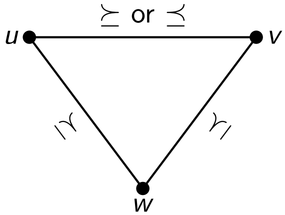
Consider three nodes u, v, w in the IG:



- u is live at w
- v is live at w

Interference and Dominance

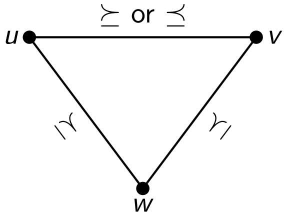
Consider three nodes u, v, w in the IG:



- u is live at w
- v is live at w
- Thus, they interfere

Interference and Dominance

Consider three nodes u, v, w in the IG:



- u is live at w
- v is live at w
- Thus, they interfere

Conclusion

All variables that ...

- interfere with w
- dominate w

... are mutually connected in the IG

Dominance and PEOs

- Before a value v is added to a PEO,
  add all values whose definitions are dominated by v
- A post order walk of the dominance tree defines a PEO
- A pre order walk of the dominance tree yields a coloring sequence
- IGs of SSA-form programs can be colored optimally in $O(\omega(G) \cdot |V|)$
- Without constructing the interference graph itself

Spilling

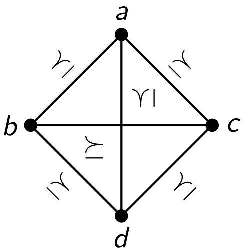
Theorem

For each clique in the IG there is a program point where all nodes in the clique are live.

Spilling

Theorem

For each clique in the IG there is a program point where all nodes in the clique are live.



- Dominance induces a total order inside the clique
  $\Rightarrow$ There is a "smallest" value d

- All others are live at the definition of d

Spilling

Consequences

- The chromatic number of the IG is exactly determined by the number of live variables at the labels
- Lowering the number of values live at each label to k makes the IG k-colorable
- We know in advance where values must be spilled
  $\implies$ All labels where the pressure is larger than k
- Spilling can be done before coloring and
- coloring will always succeed afterwards

Spilling

Consequences

- The chromatic number of the IG is exactly determined by the number of live variables at the labels
- Lowering the number of values live at each label to k makes the IG k-colorable
- We know in advance where values must be spilled
  $\Longrightarrow$ All labels where the pressure is larger than k
- Spilling can be done before coloring and
- coloring will always succeed afterwards

Conclusion

- No iteration as in Chaitin/Briggs-allocators
- No interference graph necessary

Getting out of SSA

- We now have a k-coloring of the SSA interference graph
- Can we turn that program into a non-SSA program
  and maintain the coloring?

Getting out of SSA

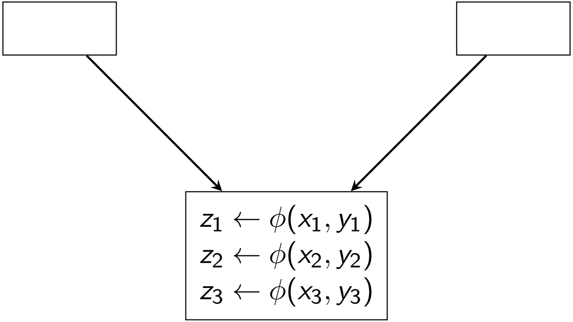
- We now have a k-coloring of the SSA interference graph
- Can we turn that program into a non-SSA program
  and maintain the coloring?

Central question

What to do about ϕ-functions?

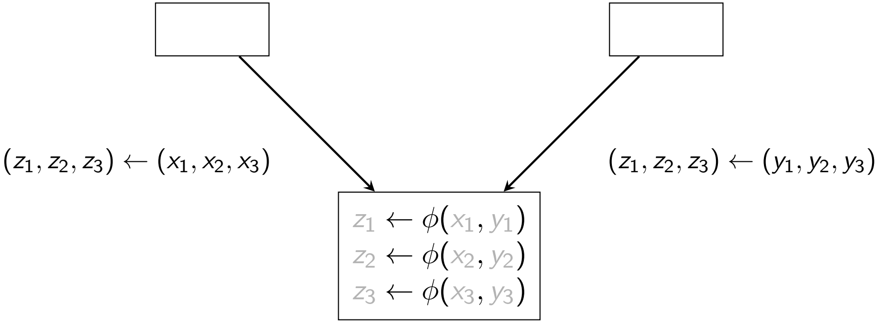
Φ-Functions

- Consider following example



Φ-Functions

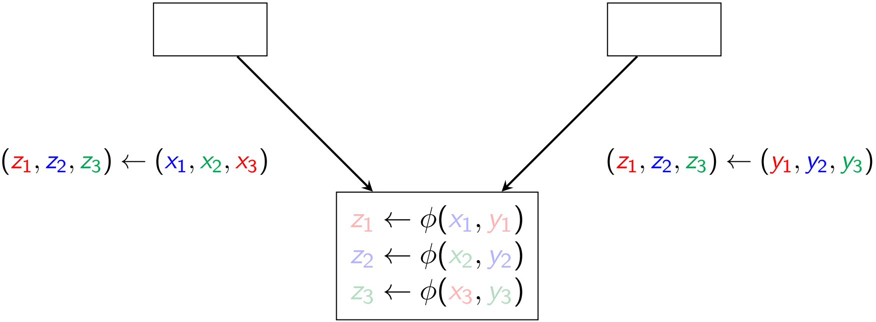
- Consider following example



- Φ-functions are parallel copies on control flow edges

Φ-Functions

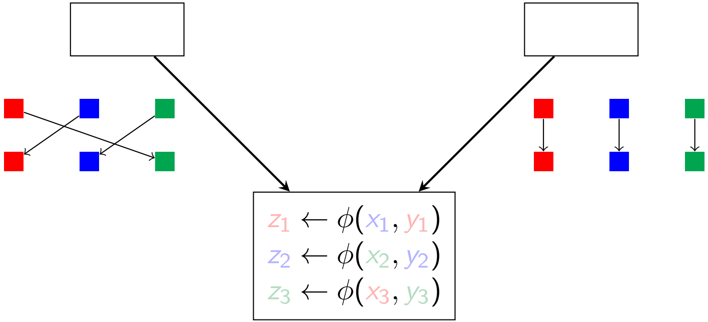
- Consider following example



- Φ-functions are parallel copies on control flow edges
- Considering assigned registers ...

Φ-Functions

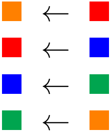
- Consider following example



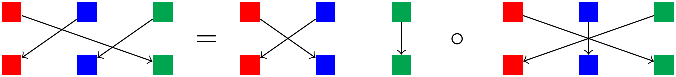
- Φ-functions are parallel copies on control flow edges
- Considering assigned registers ...
- ... Φs represent register permutations

Permutations

- A permutation can be implemented with copies if one auxiliary register  is available



- Permutations can be implemented by a series of transpositions (i.e. swaps)



- A transposition can be implemented by three `xors` *without* a third register

Intuition: Why do SSA IGs do not have cycles?

Why are SSA IGs chordal?

Program

$a \leftarrow \cdots$

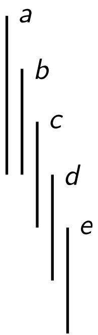
$b \leftarrow \cdots$

$c \leftarrow \cdots$

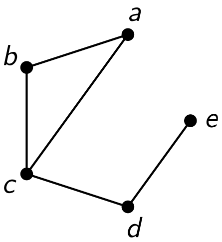
$d \leftarrow a + b$

$e \leftarrow c + 1$

Live Ranges



Interference Graph



- How can we create a 4-cycle $\{a, c, d, e\}$?

Intuition: Why do SSA IGs do not have cycles?

Why are SSA IGs chordal?

Program

$a \leftarrow \cdots$

$b \leftarrow \cdots$

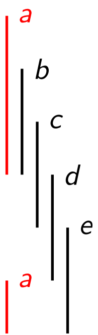
$c \leftarrow \cdots$

$d \leftarrow a + b$

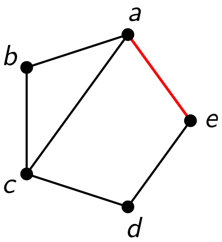
$e \leftarrow c + 1$

$a \leftarrow \cdots$

Live Ranges



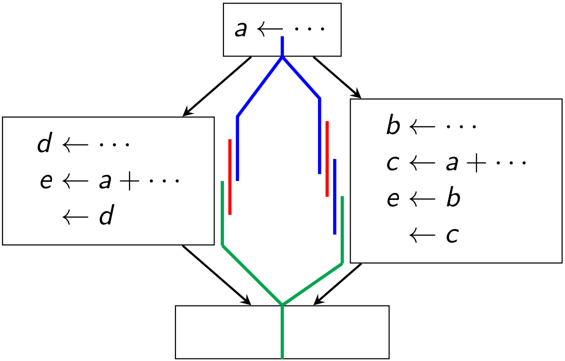
Interference Graph



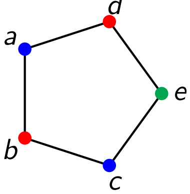
- How can we create a 4-cycle $\{a, c, d, e\}$?
- Redefine $a \implies$ SSA violated!

Intuition: ϕ-functions break cycles in the IG

Program and live ranges

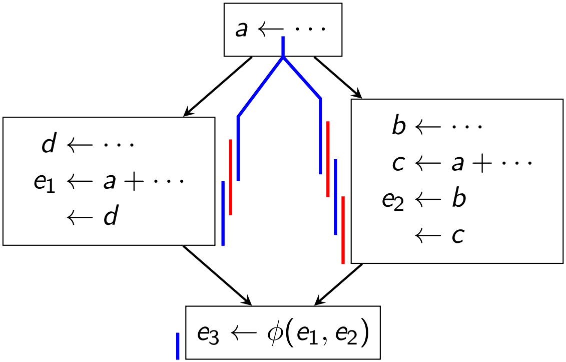


Interference Graph

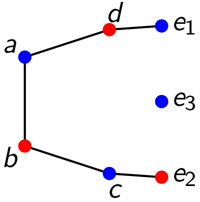


Intuition: ϕ-functions break cycles in the IG

Program and live ranges



Interference Graph



Intuition: Why destroying SSA before RA is bad

Parallel copies

$$(a', b', c', d') \leftarrow (a, b, c, d)$$

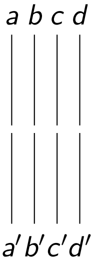
Sequential copies

$$\begin{aligned} d' &\leftarrow d \\ c' &\leftarrow c \\ b' &\leftarrow b \\ a' &\leftarrow a \end{aligned}$$

Intuition: Why destroying SSA before RA is bad

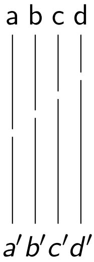
Parallel copies

$$(a', b', c', d') \leftarrow (a, b, c, d)$$



Sequential copies

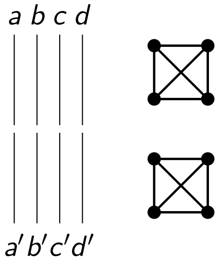
$$\begin{aligned} d' &\leftarrow d \\ c' &\leftarrow c \\ b' &\leftarrow b \\ a' &\leftarrow a \end{aligned}$$



Intuition: Why destroying SSA before RA is bad

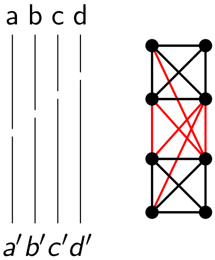
Parallel copies

$$(a', b', c', d') \leftarrow (a, b, c, d)$$



Sequential copies

$$d' \leftarrow d$$
$$c' \leftarrow c$$
$$b' \leftarrow b$$
$$a' \leftarrow a$$



Summary

- IGs of SSA-form programs are chordal
- The dominance relation induces a PEO
- No further spills after pressure is lowered
- Register assignment optimal in linear time
- Do not need to construct interference graph
- Allocator without iteration