Featuring
Florent Bouchez Tichadou

florent.bouchez-tichadou@imag.fr

Université Joseph Fourier GRENOBLE

EJCP Rennes, June 20th, 2014

What's in a name?

Cytron & Ferrante, 1987

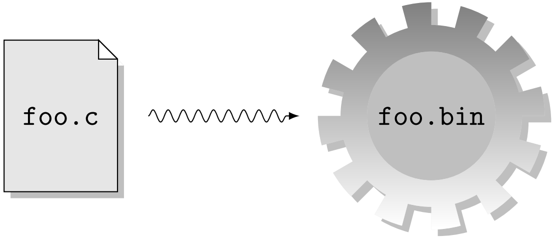
What's in a name?

Or,

the value of renaming for parallelism detection and storage allocation

Cytron & Ferrante, 1987

What's in a compiler?



What's in a compiler?

Good stuff

Bad stuff

What's in a compiler?

Good stuff

Bad stuff

Goals for this lecture

- Understand the importance of "names" ☞ SSA Form
- Introduce an interesting optimization problem ☞ register allocation

Outline

Outline

Static Single Assignment (SSA)

¿SSA?

- *Assignment*: variable's definition (e.g., `x` in `‘‘x=y+1’’`)
- *Single*: only one definition per variable
- *Static*: in the program text

Referential transparency

Example (y and z are not equal)

opaque (context dependent)	referentially transparent SSA form
$x = 1;$	$x_1 = 1;$
$y = x + 1;$	$y = x_1 + 1;$
$x = 2;$	$x_2 = 2;$
$z = x + 1;$	$z = x_2 + 1;$

Referential transparency

Example (y and z are not equal)

opaque (context dependent)	referentially transparent SSA form
$x = 1;$	$x_1 = 1;$
$y = x + 1;$	$y = x_1 + 1;$
$x = 2;$	$x_2 = 2;$
$z = x + 1;$	$z = x_2 + 1;$

Referential transparency

- value of variable independent of its position
- may refine our knowledge (e.g., ‘‘`if (x==0)`’’) but underlying value of x does not change

Informal Semantics

Each variable `v` is:

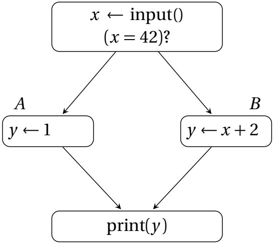
- used only once as `v = ...` (target/definition/left-hand-side)
- can be many times as `... = v` (source/use/right-hand side)

Each variable `v` is:

- used only once as `v = ...` (target/definition/left-hand-side)
- can be many times as `... = v` (source/use/right-hand side)

```
x = input();
if (x == 42) {
  y = 1;
} else {
  y = x + 2;
}

print(y);
```



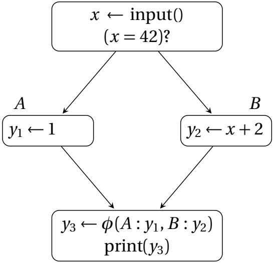
▸ CFG

Informal Semantics

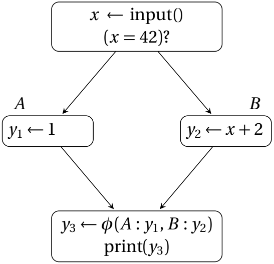
Each variable `v` is:

- used only once as `v = ...` (target/definition/left-hand-side)
- can be many times as `... = v` (source/use/right-hand side)

```
x = input();
if (x == 42) {
   y1 = 1;
} else {
   y2 = x + 2;
}
y3 = φ(y1,y2);
print(y3);
```



Informal Semantics



Introduction of ϕ-functions:

- to fix the ambiguity; introduces y_3 which takes either y_1 or y_2
- placed at control-flow merge points i.e., head of basic-blocks that have multiple predecessors

- n parameters if it has n incoming CFG paths
- represented as $a_0 = \phi(a_1, \ldots, a_n)$

Questions on SSA

$\approx$ 5 min to answer the following questions:

Is it possible to have more than one ϕ-function in a basic block?

How can you execute code containing ϕ-functions on a machine?

Informal Semantics

- multiple ϕ-functions executed simultaneously:
  $a = \phi(a, b)$
  $b = \phi(b, a)$
- ϕ-functions not directly executable (IR only: for static analysis)
- ϕ-functions removed before assembly code generation
  ☞ copy instructions insertion

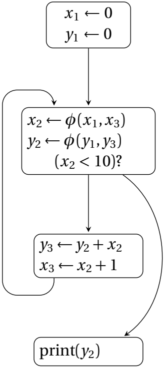
Informal Semantics

- multiple ϕ-functions executed simultaneously:
  $a = \phi(a, b)$
  $b = \phi(b, a)$
- ϕ-functions not directly executable (IR only: for static analysis)
- ϕ-functions removed before assembly code generation
  ☞ copy instructions insertion

Informal Semantic

- SSA is not Dynamic Single Assignment (DSA or SA)
- Construction: insert ϕ-function where multiple reaching defs converge; version variables x and y (integer subscripts);

```
x = 0;
y = 0;
while (x<10) {
  y = y + x;
  x = x + 1;
}
print (y);
```



Comparison with Classical Data Flow Analysis

- During actual program execution, information flows between variables
- Static analysis captures this behavior by propagating abstract information along CFG
- Can be propagated more efficiently using a functional or sparse representation such as SSA
- Constant propagation: definitions $\equiv$ set of points where information may change; associate information with variable names rather than variables $\times$ program points

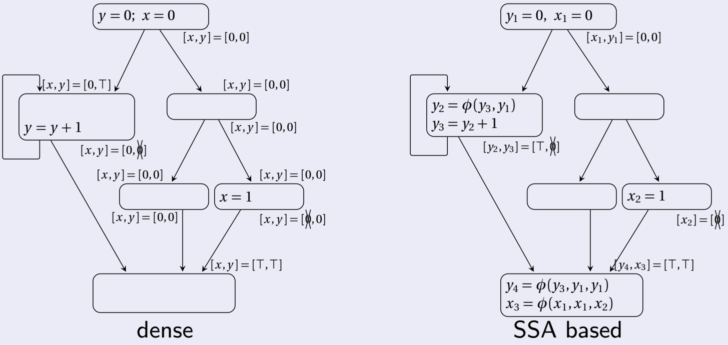
Comparison with Classical Data Flow Analysis

Null pointer analysis

Determine statically if variable can contain null value at run-time.

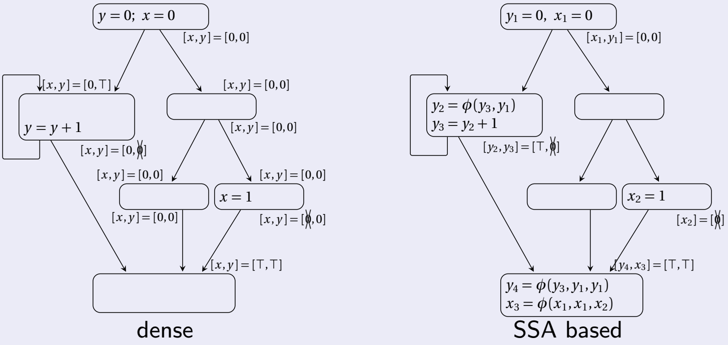
Comparison with Classical Data Flow Analysis

Null pointer analysis



Comparison with Classical Data Flow Analysis

Null pointer analysis

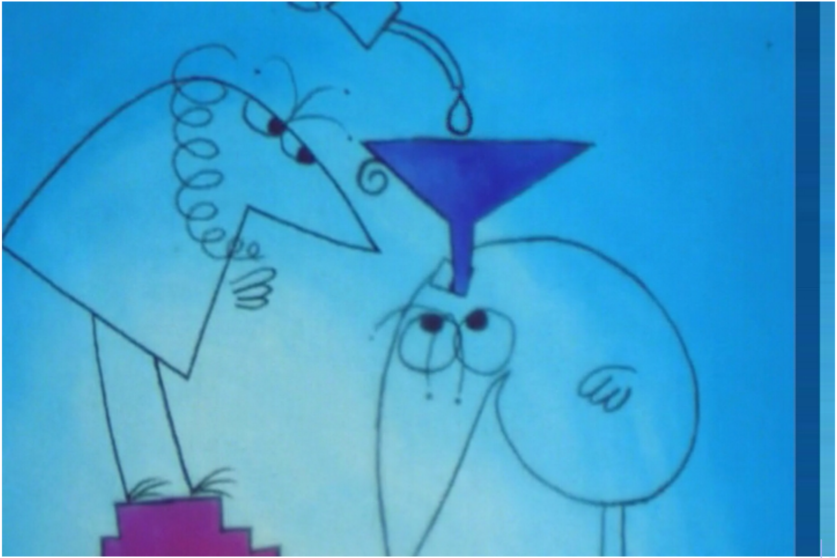


- Propagates from defs to uses (via def-use links); avoid program points where information does not change or not relevant
- Results are more compact

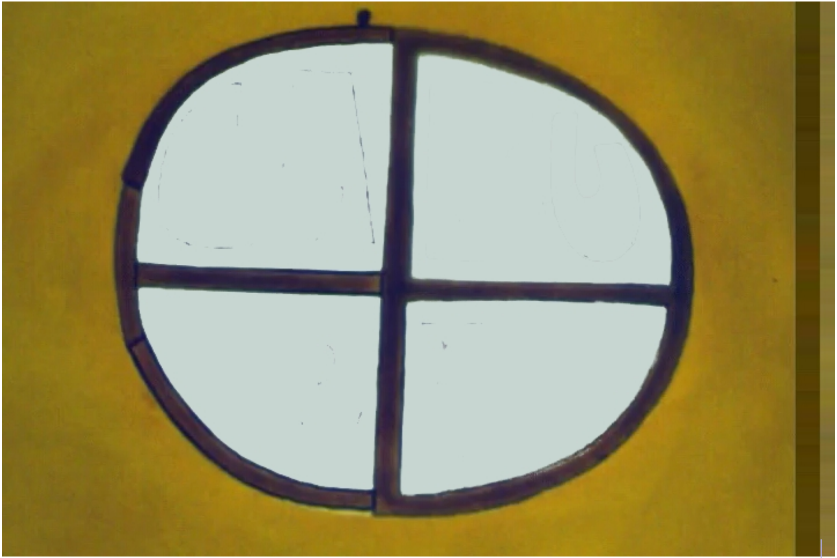
Outline

French folklore: *Les Shadoks*

Shadoks have a very small brain



Shadoks have a very small brain

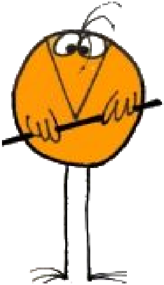


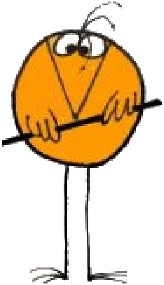
Shadoks have a very small brain

Shadoks have a very small brain

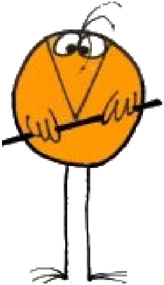
Shadoks have a very small brain

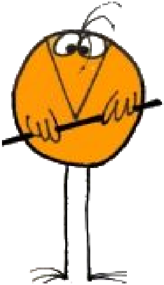
Shadoks have a very small brain

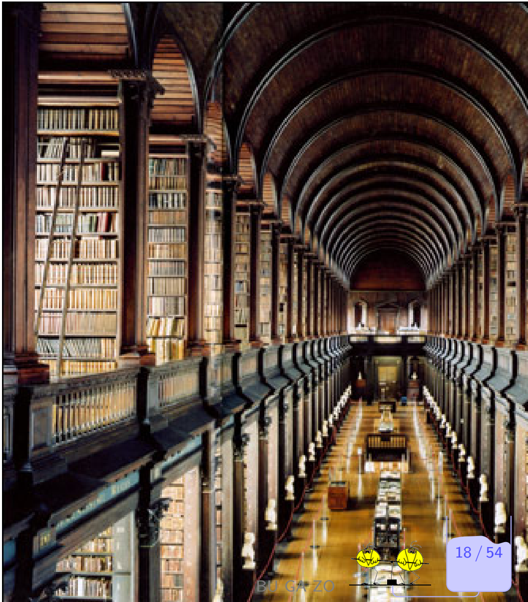




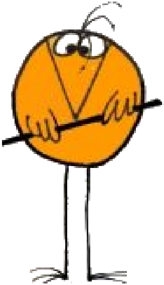
The shadoks at the library







The shadoks at the library



The shadoks at the library

The shadoks at the library

The shadoks at the library

The shadoks at the library

The shadoks at the library

The shadoks at the library

The shadoks at the library

The shadoks at the library

The shadoks at the library

The shadoks at the library

The shadoks at the library

The shadoks at the library

The shadoks at the library

The shadoks at the library

The shadoks at the library

The shadoks at the library

The shadoks at the library

The shadoks at the library

The shadoks at the library

The shadoks at the library

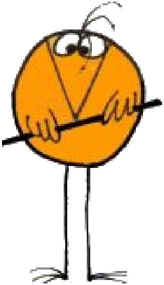
The shadoks at the library

The shadoks at the library

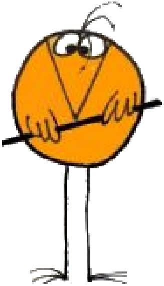
The shadoks at the library

The shadoks at the library

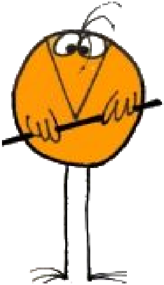
The shadoks at the library

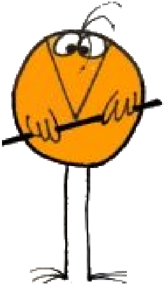


The shadoks at the library

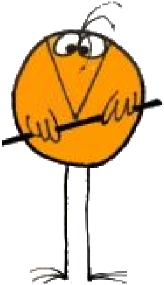


The shadoks at the library





The shadoks at the library



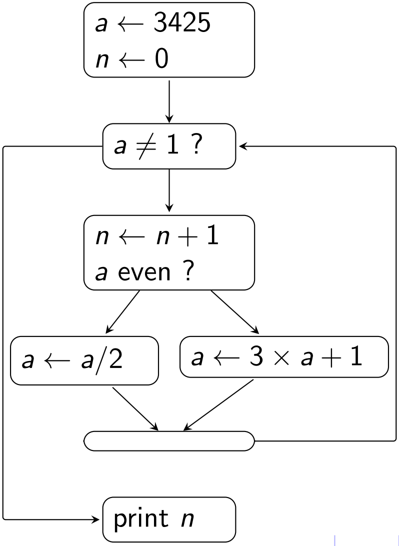
What is register allocation?

Assign variables to memory locations

- Registers: , , , . . .
- Memory: infinite

Rules of the game

- two interfering variables
  ➠ different registers
- not enough registers
  ➠ spill to memory



What is register allocation?

Assign variables to memory locations

- Registers: , , , . . .
- Memory: infinite

Rules of the game

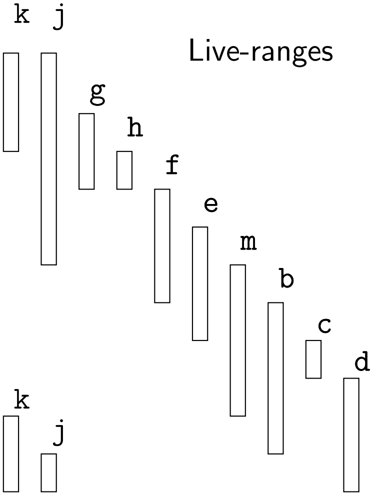
- two interfering variables
  ➠ different registers
- not enough registers
  ➠ spill to memory

Plus constraints:

- register constraints
- pre-colored variables
- register pairing, aliasing,. . .

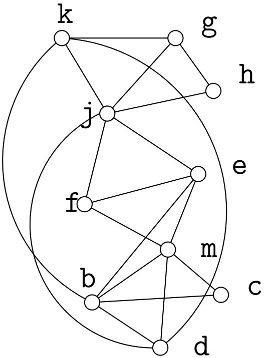
Chaitin et al. model

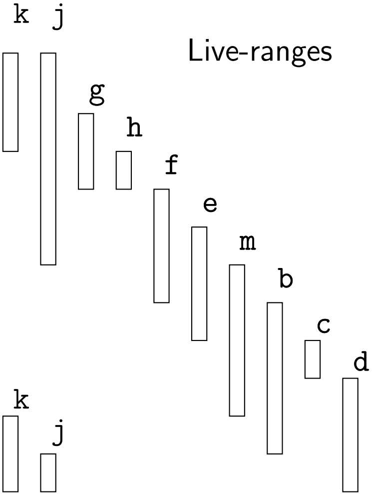
```
Live-in:  k j
   g  := mem[j+12]
   h  := k-1
   f  := g+h
   e  := mem[j+8]
   m  := mem[j+16]
   b  := mem[f]
   c  := e+8
   d  := c
   k  := m+4
   j  := b
Live-out:  d k j
```



Chaitin et al. model

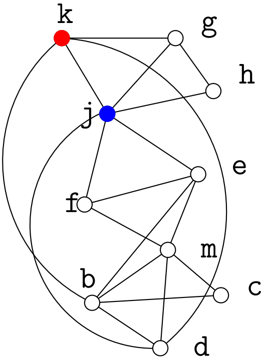
Interference graph



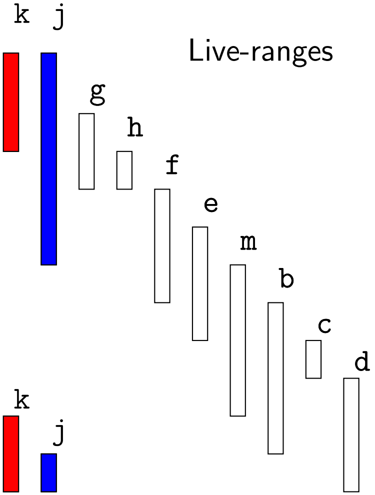


Chaitin et al. model

Interference graph



Live-ranges



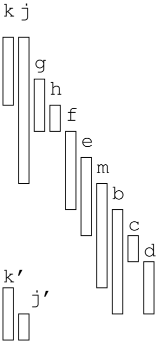
Coloring a basic block

```
Live-in: k j
g  := mem[j+12]
h  := k-1
f  := g+h
e  := mem[j+8]
m  := mem[j+16]
b  := mem[f]
c  := e+8
d  := c
k'  := m+4
j'  := b+d
Live-out: k' j'
```

- MAXLIVE $\leq r$
- Linear scan

Coloring a basic block

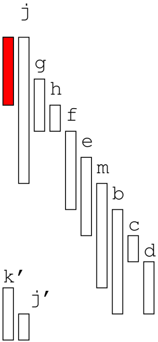
```
Live-in: k j
g  := mem[j+12]
h  := k-1
f  := g+h
e  := mem[j+8]
m  := mem[j+16]
b  := mem[f]
c  := e+8
d  := c
k’  := m+4
j’  := b+d
Live-out: k’ j’
```



- MAXLIVE $\leq r$
- Linear scan

Coloring a basic block

```
Live-in: k j
g  := mem[j+12]
h  := k-1
f  := g+h
e  := mem[j+8]
m  := mem[j+16]
b  := mem[f]
c  := e+8
d  := c
k'  := m+4
j'  := b+d
Live-out: k' j'
```



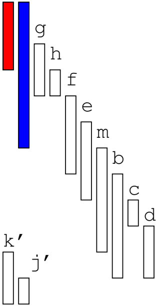
- MAXLIVE $\leq r$
- Linear scan

Coloring a basic block

Live-in: k j

```
g  := mem[j+12]
h  := k-1
f  := g+h
e  := mem[j+8]
m  := mem[j+16]
b  := mem[f]
c  := e+8
d  := c
k'  := m+4
j'  := b+d
```

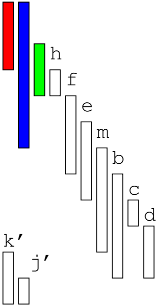
Live-out: k' j'



- MAXLIVE $\leq r$
- Linear scan

Coloring a basic block

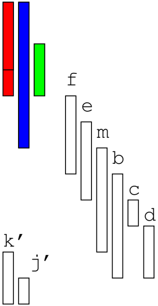
```
Live-in: k j
g  := mem[j+12]
h  := k-1
f  := g+h
e  := mem[j+8]
m  := mem[j+16]
b  := mem[f]
c  := e+8
d  := c
k’  := m+4
j’  := b+d
Live-out: k’ j’
```



- MAXLIVE $\leq r$
- Linear scan

Coloring a basic block

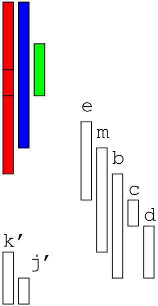
```
Live-in: k j
g  := mem[j+12]
h  := k-1
f  := g+h
e  := mem[j+8]
m  := mem[j+16]
b  := mem[f]
c  := e+8
d  := c
k'  := m+4
j'  := b+d
Live-out: k' j'
```



- MAXLIVE $\leq r$
- Linear scan

Coloring a basic block

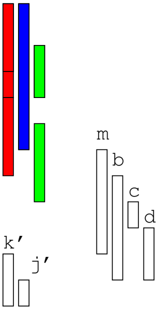
```
Live-in: k j
g  := mem[j+12]
h  := k-1
f  := g+h
e  := mem[j+8]
m  := mem[j+16]
b  := mem[f]
c  := e+8
d  := c
k'  := m+4
j'  := b+d
Live-out: k' j'
```



- MAXLIVE $\leq r$
- Linear scan

Coloring a basic block

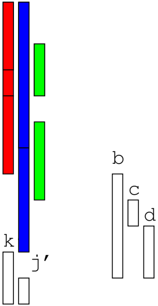
```
Live-in: k j
g  := mem[j+12]
h  := k-1
f  := g+h
e  := mem[j+8]
m  := mem[j+16]
b  := mem[f]
c  := e+8
d  := c
k'  := m+4
j'  := b+d
Live-out: k' j'
```



- MAXLIVE $\leq r$
- Linear scan

Coloring a basic block

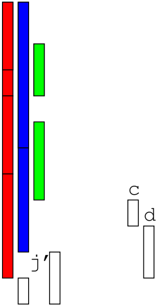
```
Live-in: k j
g  := mem[j+12]
h  := k-1
f  := g+h
e  := mem[j+8]
m  := mem[j+16]
b  := mem[f]
c  := e+8
d  := c
k'  := m+4
j'  := b+d
Live-out: k' j'
```



- MAXLIVE $\leq r$
- Linear scan

Coloring a basic block

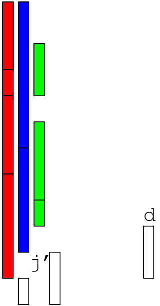
```
Live-in: k j
g  := mem[j+12]
h  := k-1
f  := g+h
e  := mem[j+8]
m  := mem[j+16]
b  := mem[f]
c  := e+8
d  := c
k'  := m+4
j'  := b+d
Live-out: k' j'
```



- MAXLIVE $\leq r$
- Linear scan

Coloring a basic block

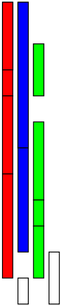
```
Live-in: k j
g  := mem[j+12]
h  := k-1
f  := g+h
e  := mem[j+8]
m  := mem[j+16]
b  := mem[f]
c  := e+8
d  := c
k'  := m+4
j'  := b+d
Live-out: k' j'
```



- MAXLIVE $\leq r$
- Linear scan

Coloring a basic block

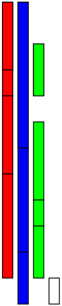
```
Live-in: k j
g  := mem[j+12]
h  := k-1
f  := g+h
e  := mem[j+8]
m  := mem[j+16]
b  := mem[f]
c  := e+8
d  := c
k'  := m+4
j'  := b+d
Live-out: k' j'
```



- MAXLIVE $\leq r$
- Linear scan

Coloring a basic block

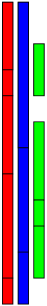
```
Live-in: k j
g := mem[j+12]
h := k-1
f := g+h
e := mem[j+8]
m := mem[j+16]
b := mem[f]
c := e+8
d := c
k' := m+4
j' := b+d
Live-out: k' j'
```



- MAXLIVE $\leq r$
- Linear scan

Coloring a basic block

```
Live-in: k j
g  := mem[j+12]
h  := k-1
f  := g+h
e  := mem[j+8]
m  := mem[j+16]
b  := mem[f]
c  := e+8
d  := c
k'  := m+4
j'  := b+d
Live-out: k' j'
```

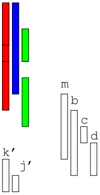


- MAXLIVE $\leq r$
- Linear scan

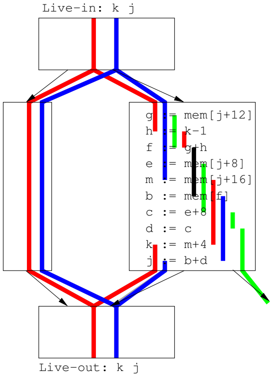
“Spilling easier on a BB than on a general CFG”

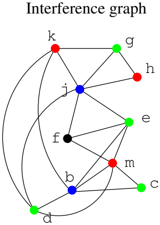
Basic Block

```
Live-in: k j
g  := mem[j+12]
h  := k-1
f  := g+h
e  := mem[j+8]
m  := mem[j+16]
b  := mem[f]
c  := e+8
d  := c
k’  := m+4
j’  := b+d
Live-out: k’ j’
```



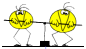
General control flow graph





- Coloring test
- Greedy coloring

Demo: greedy coloring in ubigraph



Coloring the interference graph: the greedy way

Register allocation is modeled as coloring the interference graph of the program.

Problem

Graph-k-coloring is *NP-complete* (for $k \geq 3$), and any interference graph can arise in programs. *(Chaitin et al.'s proof)*
➠ register allocation is NP-complete in this model.

Coloring the interference graph: the greedy way

Register allocation is modeled as coloring the interference graph of the program.

Problem

Graph-k-coloring is *NP-complete* (for $k \geq 3$), and any interference graph can arise in programs. *(Chaitin et al.'s proof)*
➠ register allocation is NP-complete in this model.

A greedy coloring heuristic is used: *Chaitin et al.'s* greedy scheme.

▸ Greedy scheme

If coloring fails, usually spill.

Coloring the interference graph: the greedy way

Register allocation is modeled as coloring the interference graph of the program.

Problem

Graph-k-coloring is *NP-complete* (for $k \geq 3$), and any interference graph can arise in programs. *(Chaitin et al.'s proof)*
➠ register allocation is NP-complete in this model.

A greedy coloring heuristic is used: *Chaitin et al.'s* greedy scheme.

▸ Greedy scheme

If coloring fails, usually spill.
Disadvantages:

- The scheme might fail even when there is a solution.
- A variable is supposed to be in exactly one register.

Coloring the interference graph: the greedy way

Register allocation is modeled as coloring the interference graph of the program.

Problem

Graph-k-coloring is *NP-complete* (for $k \geq 3$), and any interference graph can arise in programs. *(Chaitin et al.'s proof)*
➠ register allocation is NP-complete in this model.

A greedy coloring heuristic is used: *Chaitin et al.'s* greedy scheme.

▸ Greedy scheme

If coloring fails, usually spill.
Disadvantages:

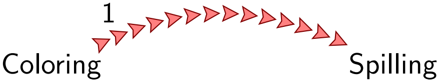
- The scheme might fail even when there is a solution.
  ➠ need to spill more than necessary
- A variable is supposed to be in exactly one register.
  ➠ restriction on the coloring

Coloring, spilling are inter-dependent

Coloring

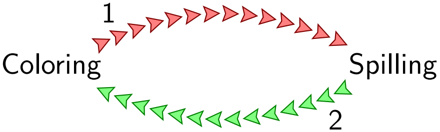
Spilling

Coloring, spilling are inter-dependent



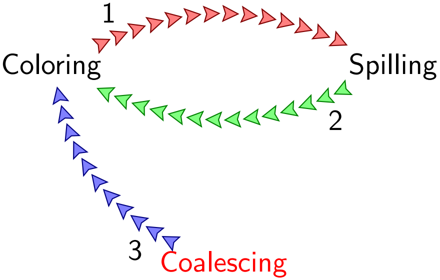
1. coloring fails

Coloring, spilling are inter-dependent



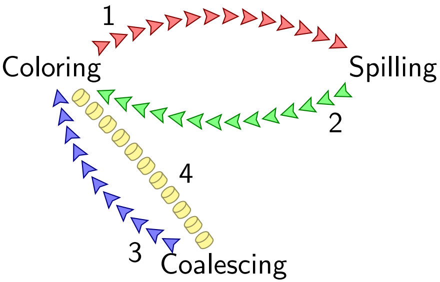
1. coloring fails
2. less nodes to color
   change in code
   (load/store)

Coloring, spilling and coalescing are inter-dependent



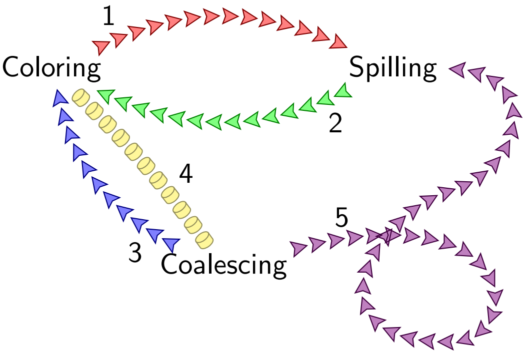
1. coloring fails
2. less nodes to color
   change in code
   (load/store)
3. decrease degree of
   neighbors

Coloring, spilling and coalescing are inter-dependent



1. coloring fails
2. less nodes to color
   change in code
   (load/store)
3. decrease degree of
   neighbors
4. coalescing = giving same
   color to two nodes

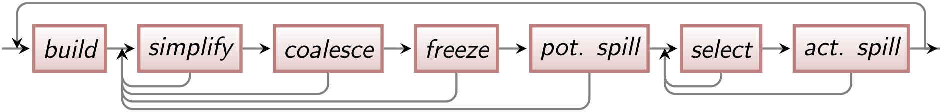
Coloring, spilling and coalescing are inter-dependent



1. coloring fails
2. less nodes to color
   change in code
   (load/store)
3. decrease degree of
   neighbors
4. coalescing = giving same
   color to two nodes
5. spilling a coalesced node
   is more expensive

Graph coloring allocators in one phase

- Chaitin-Briggs allocator (Briggs, Cooper, Torczon)
- Iterated Register Coalescing (Appel, George)



Drawbacks of register allocation in one phase

- code more complicated to maintain
- improvements must take the whole allocator into account
- harder to "prioritize" a problem

Separating register allocation in two phases

Allows to optimize problems separately:

- priority is given to spilling
- then, coloring/coalescing (without “useless spills”)

How to separate register allocation in two phases?

Here comes the SSA form…

Theorem

The interference graph of a program under strict SSA form is chordal.

Separating register allocation in two phases

Allows to optimize problems separately:

- priority is given to spilling
- then, coloring/coalescing (without "useless spills")

How to separate register allocation in two phases?

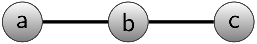
Here comes the SSA form...

Theorem

The interference graph of a program under strict SSA form is chordal.

Definition (Chordal graph)

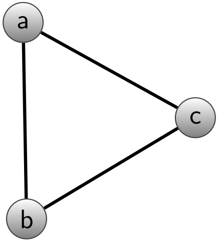
A graph is chordal iff every cycle of size ≥ 4 has a chord.



Chordal graphs

Definition (Chordal graph)

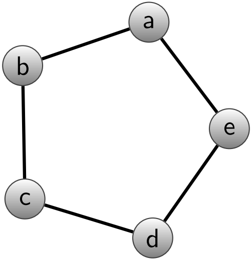
A graph is chordal iff every cycle of size ≥ 4 has a chord.



Chordal graphs

Definition (Chordal graph)

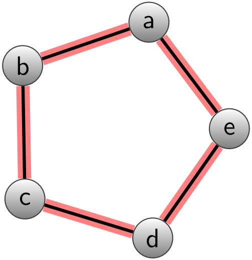
A graph is chordal iff every cycle of size ≥ 4 has a chord.



Chordal graphs

Definition (Chordal graph)

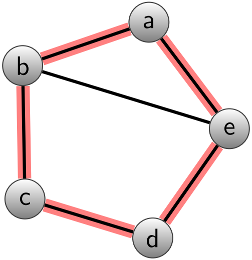
A graph is chordal iff every cycle of size ≥ 4 has a chord.



Chordal graphs

Definition (Chordal graph)

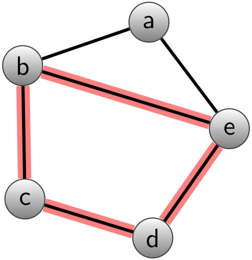
A graph is chordal iff every cycle of size ≥ 4 has a chord.



Chordal graphs

Definition (Chordal graph)

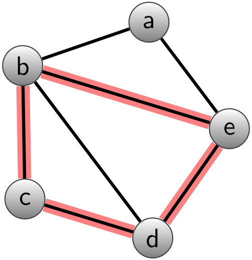
A graph is chordal iff every cycle of size ≥ 4 has a chord.



Chordal graphs

Definition (Chordal graph)

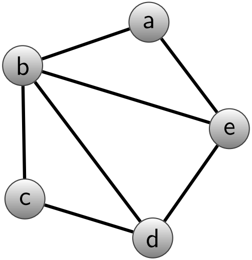
A graph is chordal iff every cycle of size ≥ 4 has a chord.



Chordal graphs

Definition (Chordal graph)

A graph is chordal iff every cycle of size ≥ 4 has a chord.



Chordal graphs are *perfect* and *easy to color*.

Static Single Assignment

SSA : exactly one *textual* definition per variable

Static Single Assignment

SSA : exactly one *textual* definition per variable

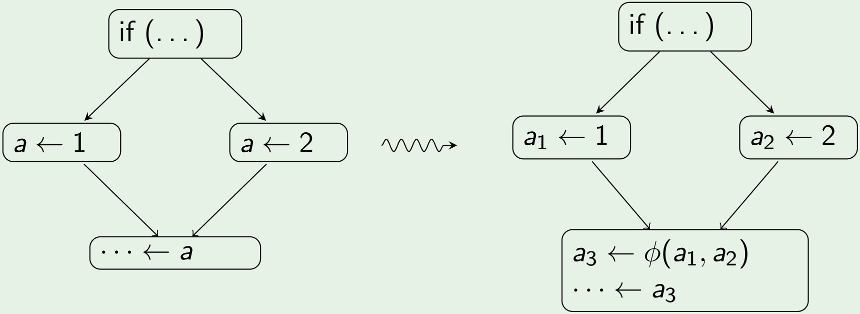
Example (Straight code converted to SSA form)

$$
\begin{array}{c}
a \leftarrow \ldots \\
\vdots \\
\ldots \leftarrow a \\
\vdots \\
a \leftarrow \ldots \\
\vdots \\
\ldots \leftarrow a
\end{array}
\quad \rightsquigarrow \quad
\begin{array}{c}
a_1 \leftarrow \ldots \\
\vdots \\
\ldots \leftarrow a_1 \\
\vdots \\
a_2 \leftarrow \ldots \\
\vdots \\
\ldots \leftarrow a_2
\end{array}
$$

Static Single Assignment

SSA : exactly one *textual* definition per variable

Example (Conditional code converted to SSA form)

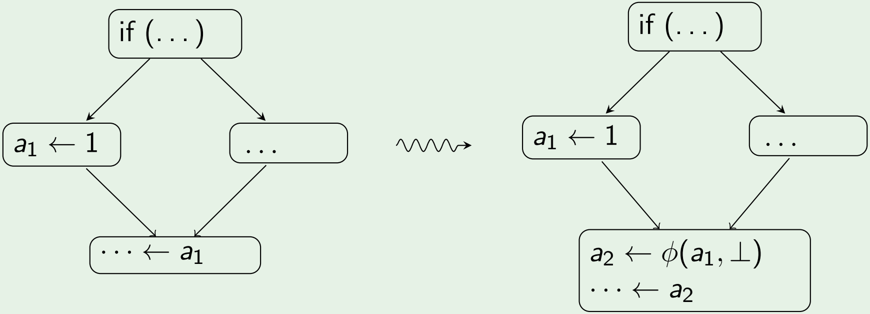


Static Single Assignment

SSA : exactly one *textual* definition per variable

strictness : SSA where the definition always dominates its uses

Example (strict SSA or SSA with dominance property)

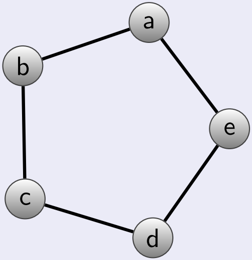


Proof that strict SSA interference graphs are chordal

Theorem

The interference graph of a program under strict SSA form is chordal.

Proof.



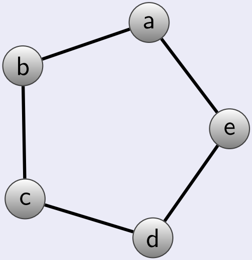
□

Proof that strict SSA interference graphs are chordal

Theorem

The interference graph of a program under strict SSA form is chordal.

Proof.



- dominance property

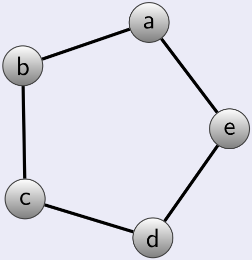
□

Proof that strict SSA interference graphs are chordal

Theorem

The interference graph of a program under strict SSA form is chordal.

Proof.



- dominance property
- a and b interfere $\Rightarrow$ $def(a)$ dominates $def(b)$ *(or the converse)*

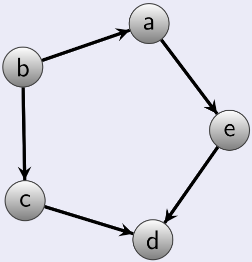
$\square$

Proof that strict SSA interference graphs are chordal

Theorem

The interference graph of a program under strict SSA form is chordal.

Proof.



- dominance property
- a and b interfere $\Rightarrow$ $def(a)$ dominates $def(b)$ *(or the converse)*
- direct each edge with dominance

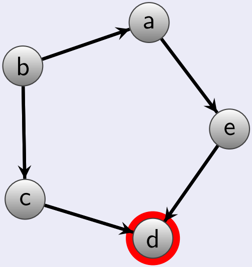
□

Proof that strict SSA interference graphs are chordal

Theorem

The interference graph of a program under strict SSA form is chordal.

Proof.



- dominance property
- a and b interfere $\Rightarrow$ $def(a)$ dominates $def(b)$ *(or the converse)*
- direct each edge with dominance
- $def(d)$ is dominated by $def(c)$ and $def(e)$

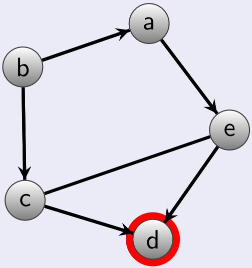
□

Proof that strict SSA interference graphs are chordal

Theorem

The interference graph of a program under strict SSA form is chordal.

Proof.

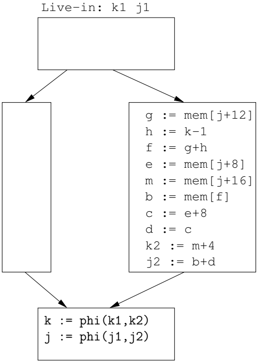


- dominance property
- a and b interfere $\Rightarrow$ $def(a)$ dominates $def(b)$ *(or the converse)*
- direct each edge with dominance
- $def(d)$ is dominated by $def(c)$ and $def(e)$
- c and e are live at $def(d)$

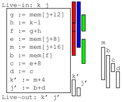
□

“Under SSA: the dominance tree”

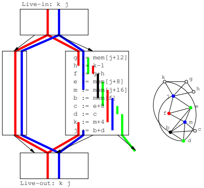
Static single assignment form



BB



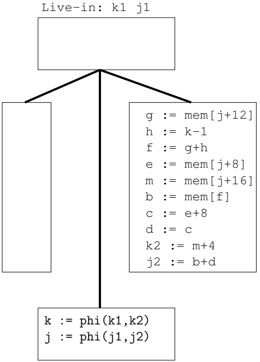
CFG



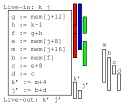
- MAXLIVE $\leq r$
- Tree scan

“Under SSA: the dominance tree”

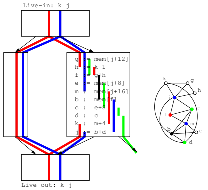
Static single assignment form



BB



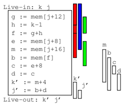
CFG



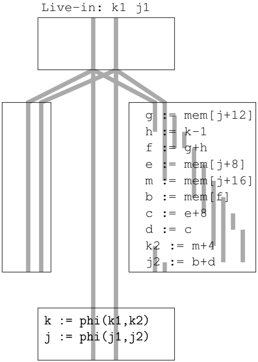
- MAXLIVE $\leq r$
- Tree scan

"Under SSA: the dominance tree"

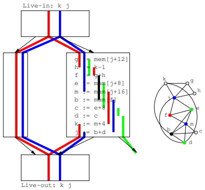
BB



Static single assignment form



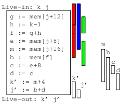
CFG



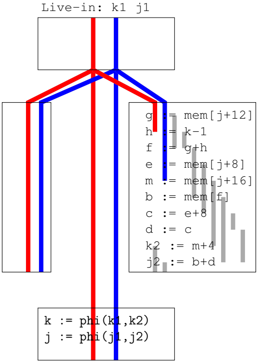
- MAXLIVE $\leq r$
- Tree scan

“Under SSA: the dominance tree”

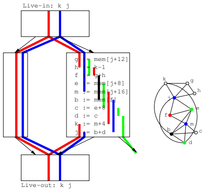
BB



Static single assignment form



CFG

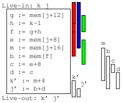


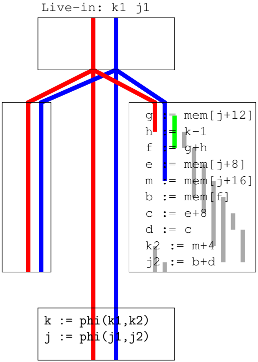
- MAXLIVE $\leq r$
- Tree scan

“Under SSA: the dominance tree”

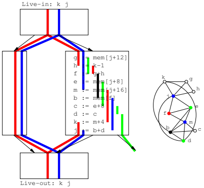
Static single assignment form

BB

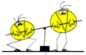




CFG

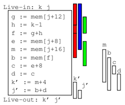


- MAXLIVE $\leq r$
- Tree scan

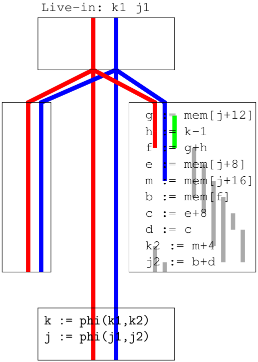


“Under SSA: the dominance tree”

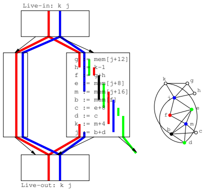
BB



Static single assignment form



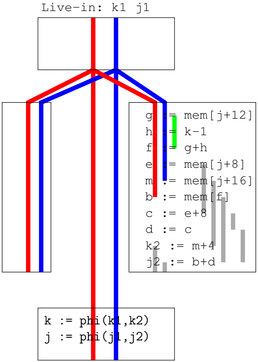
CFG



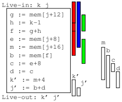
- MAXLIVE $\leq r$
- Tree scan

"Under SSA: the dominance tree"

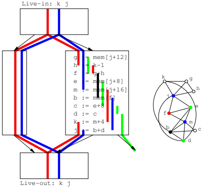
Static single assignment form



BB



CFG

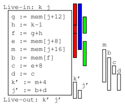


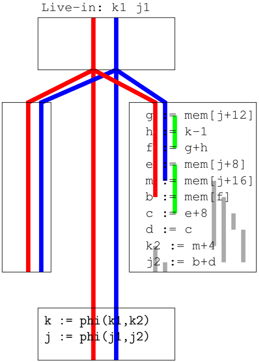
- MAXLIVE $\leq r$
- Tree scan

"Under SSA: the dominance tree"

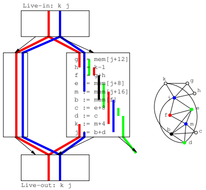
Static single assignment form

BB





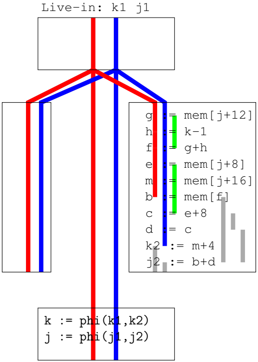
CFG



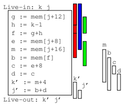
- MAXLIVE $\leq r$
- Tree scan

“Under SSA: the dominance tree”

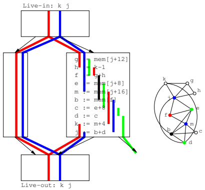
Static single assignment form



BB



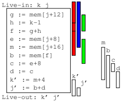
CFG



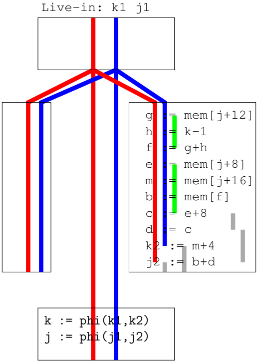
- MAXLIVE $\leq r$
- Tree scan

“Under SSA: the dominance tree”

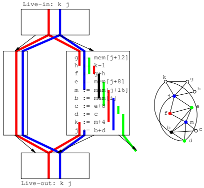
BB



Static single assignment form



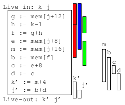
CFG



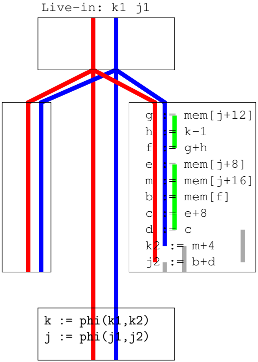
- MAXLIVE $\leq r$
- Tree scan

“Under SSA: the dominance tree”

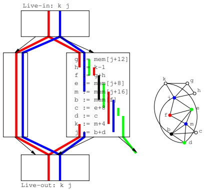
BB



Static single assignment form



CFG

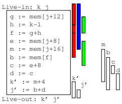


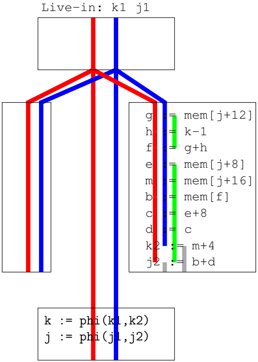
- MAXLIVE $\leq r$
- Tree scan

“Under SSA: the dominance tree”

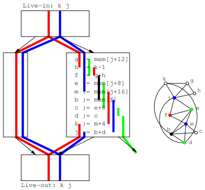
Static single assignment form

BB

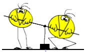




CFG

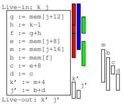


- MAXLIVE $\leq r$
- Tree scan

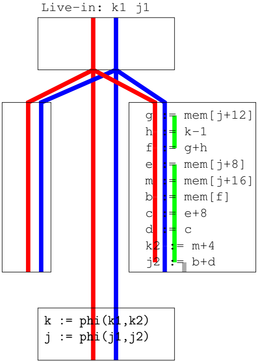


"Under SSA: the dominance tree"

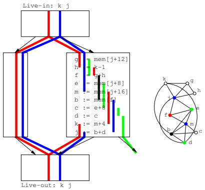
BB



Static single assignment form



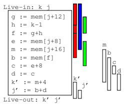
CFG



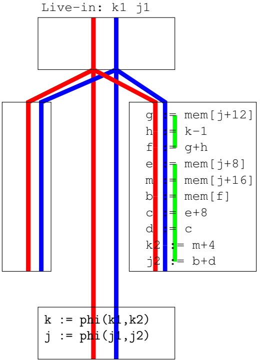
- MAXLIVE $\leq r$
- Tree scan

“Under SSA: the dominance tree”

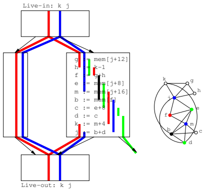
BB



Static single assignment form



CFG



- MAXLIVE $\leq r$
- Tree scan

Strict SSA programs are easy to color

Chordal graphs are *perfect graphs*, hence easy to color. We proved more:

Theorem

Chordal graphs are colorable using Chaitin et al. greedy scheme.
They are greedy-k-colorable.

General program:
NP-complete

strict SSA program:
greedy-k-colorable

Strict SSA programs are easy to color

Chordal graphs are *perfect graphs*, hence easy to color. We proved more:

Theorem

Chordal graphs are colorable using Chaitin et al. greedy scheme. They are greedy-k-colorable.

General program:	strict SSA program:
NP-complete	greedy-k-colorable

Under strict SSA, Maxlive, the maximum number of simultaneously live variables, is the coloring indicator:

$$\text{Maxlive} \leq R$$

Register allocation in two phases

Using Maxlive, it seems possible to use a very simple register allocation scheme:

1. spill variables until Maxlive $\leq R$
2. transform program into strict SSA form
3. allocate variables using R registers
4. go out of colored SSA form

Register allocation in two phases

Using Maxlive, it seems possible to use a very simple register allocation scheme:

1. spill variables until Maxlive $\leq R$
2. transform program into strict SSA form
3. allocate variables using R registers
4. go out of colored SSA form

Questions

SSA seems to transform an NP-complete problem into polynomial one...
Where is the complexity now? What else is simplified?

What is coalescing?

Goal of coalescing

Removing the register-to-register copies [`move` $a \leftarrow b$]

Numerous `move` due to:

- live-range splitting to avoid spilling

What is coalescing?

Goal of coalescing

Removing the register-to-register copies [`move` $a \leftarrow b$]

Numerous `move` due to:

- live-range splitting to avoid spilling
- register constraints

$$\boxed{\begin{array}{l} a \leftarrow \ldots \\ b \leftarrow \ldots \\ c \leftarrow f(a, b) \end{array}} \leadsto \boxed{\begin{array}{l} a \leftarrow \ldots \\ b \leftarrow \ldots \\ \text{move } R_0, a \\ \text{move } R_1, b \\ \text{call } f \\ \text{move } c, R_0 \end{array}}$$

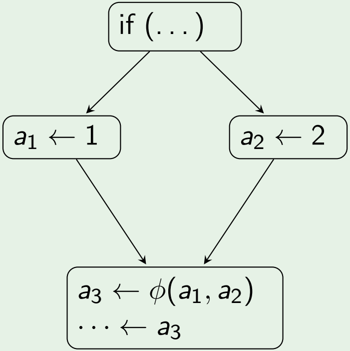
What is coalescing?

Goal of coalescing

Removing the register-to-register copies [`move` $a \leftarrow b$]

Numerous `move` due to:

- live-range splitting to avoid spilling
- register constraints
- SSA destruction



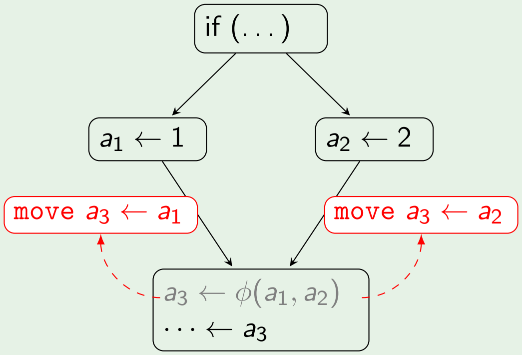
What is coalescing?

Goal of coalescing

Removing the register-to-register copies [`move` $a \leftarrow b$]

Numerous `move` due to:

- live-range splitting to avoid spilling
- register constraints
- SSA destruction



Traditional modeling of the coalescing problem

Given an instruction [`move` $a \leftarrow b$]

Fact I Giving the same color to both a and b saves the instruction.

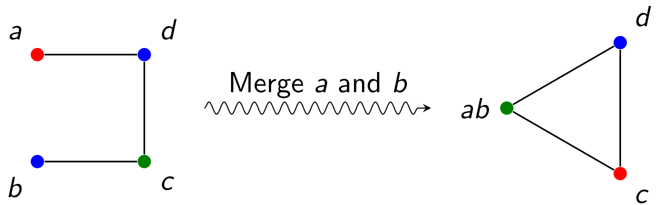
Fact II Merging nodes a and b forces them to have the same color.

Traditional modeling of the coalescing problem

Given an instruction [`move` $a \leftarrow b$]

Fact I Giving the same color to both a and b saves the instruction.

Fact II Merging nodes a and b forces them to have the same color.



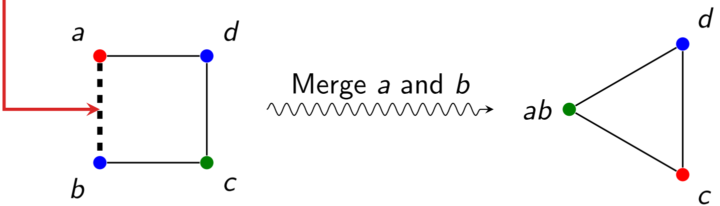
Traditional modeling of the coalescing problem

Given an instruction [`move` $a \leftarrow b$]

Fact I Giving the same color to both a and b saves the instruction.

Fact II Merging nodes a and b forces them to have the same color.

Idea Express this as an "affinity" between a and b in the interference graph to drive the algorithm.



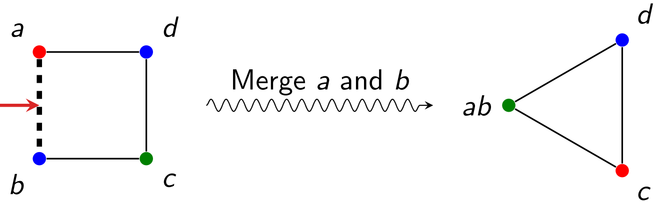
Traditional modeling of the coalescing problem

Given an instruction [`move` $a \leftarrow b$]

Fact I Giving the same color to both a and b saves the instruction.

Fact II Merging nodes a and b forces them to have the same color.

Idea Express this as an "affinity" between a and b in the interference graph to drive the algorithm.



We work on graphs instead of programs.

Demo: split graph in ubigraph

Different coalescing problems

Aggressive

Conservative

Incremental

Optimistic

Different coalescing problems

Aggressive

Coalesce as many affinities as possible.

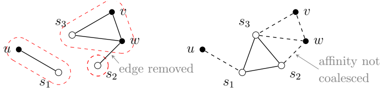
Conservative

Incremental

Optimistic

Different coalescing problems

Aggressive



multiway-cut

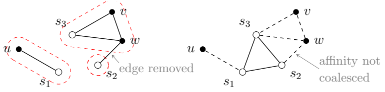
Conservative

Incremental

Optimistic

Different coalescing problems

Aggressive



multiway-cut

Conservative

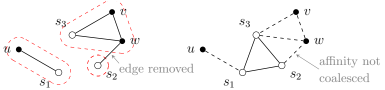
Coalesce as many affinities as possible but remains k-colorable.

Incremental

Optimistic

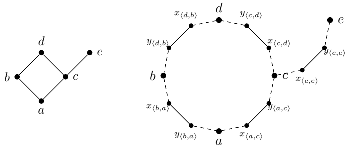
Different coalescing problems

Aggressive



multiway-cut

Conservative



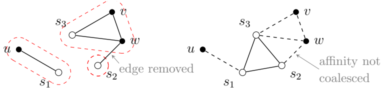
graph k-coloring

Incremental

Optimistic

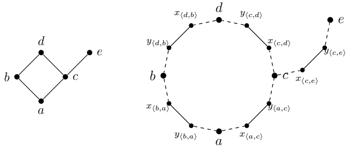
Different coalescing problems

Aggressive



multiway-cut

Conservative



graph k-coloring

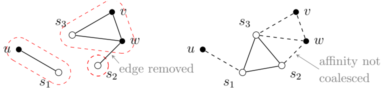
Incremental

Optimistic

Perform aggressive coalescing, then de-coalescing to get k-colorable.

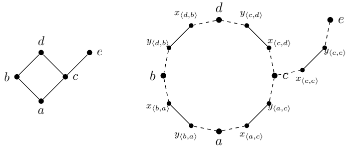
Different coalescing problems

Aggressive



multiway-cut

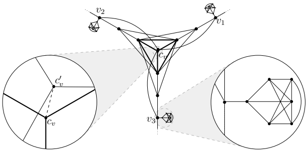
Conservative



graph k-coloring

Incremental

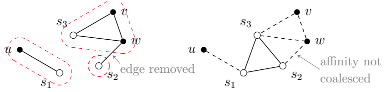
Optimistic



3-vertex-cover

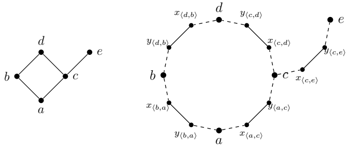
Different coalescing problems

Aggressive



multiway-cut

Conservative

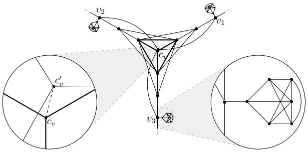


graph k-coloring

Incremental

Coalesce *one* affinity while staying k-colorable.

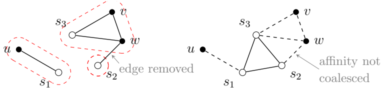
Optimistic



3-vertex-cover

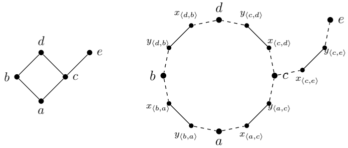
Different coalescing problems

Aggressive



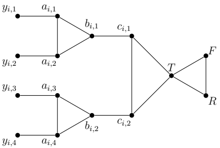
multiway-cut

Conservative



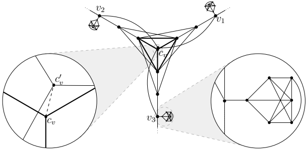
graph k-coloring

Incremental



3-SAT

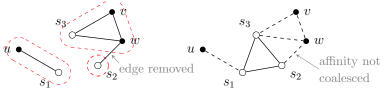
Optimistic



3-vertex-cover

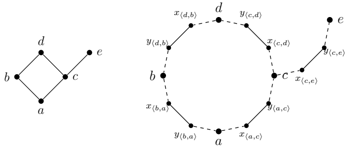
Different coalescing problems

Aggressive



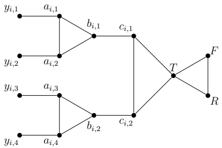
multiway-cut

Conservative



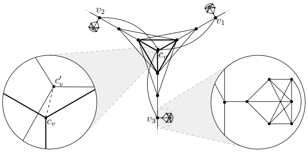
graph k-coloring

Incremental



3-SAT

Optimistic



3-vertex-cover

NP-complete for general graphs.

Polynomial for chordal graphs.

The problem of incremental coalescing

The goal of incremental is to perform conservative coalescing by coalescing affinities one by one.

Problem (Incremental coalescing)

Given a k-colorable graph G and two nodes x and y, is it possible to color G such that x and y have the same color?

Theorem

The incremental coalescing problem is NP-complete.

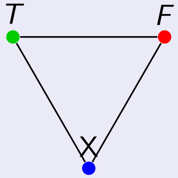
Incremental coalescing is NP-complete in the general case

Reduction from 3-SAT. *(Similar to reduction of graph-3-coloring from 3-SAT).*

Example on $(x \vee y \vee \bar{z} \vee w) \wedge \cdots \wedge (\bar{x} \vee z \vee \bar{y} \vee u)$.

First, equivalence between graph-3-coloring and 4-SAT.

- 3 nodes for `True`, `False`, and `X`



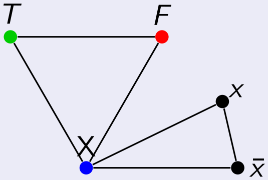
Incremental coalescing is NP-complete in the general case

Reduction from 3-SAT. *(Similar to reduction of graph-3-coloring from 3-SAT).*

Example on $(x \vee y \vee \bar{z} \vee w) \wedge \cdots \wedge (\bar{x} \vee z \vee \bar{y} \vee u)$.

First, equivalence between graph-3-coloring and 4-SAT.

- 3 nodes for `True`, `False`, and `X`
- 2 nodes for each variable: v and $\bar{v}$



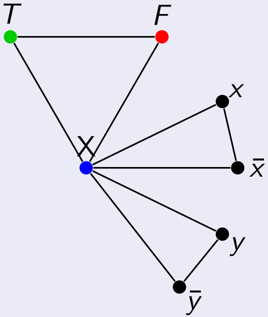
Incremental coalescing is NP-complete in the general case

Reduction from 3-SAT. *(Similar to reduction of graph-3-coloring from 3-SAT).*

Example on $(x \vee y \vee \bar{z} \vee w) \wedge \cdots \wedge (\bar{x} \vee z \vee \bar{y} \vee u)$.

First, equivalence between graph-3-coloring and 4-SAT.

- 3 nodes for `True`, `False`, and `X`
- 2 nodes for each variable: v and $\bar{v}$



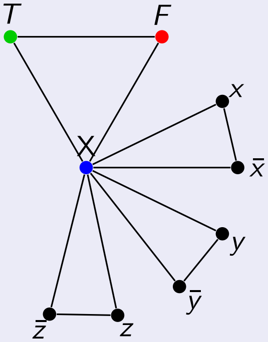
Incremental coalescing is NP-complete in the general case

Reduction from 3-SAT. *(Similar to reduction of graph-3-coloring from 3-SAT).*

Example on $(x \vee y \vee \bar{z} \vee w) \wedge \cdots \wedge (\bar{x} \vee z \vee \bar{y} \vee u)$.

First, equivalence between graph-3-coloring and 4-SAT.

- 3 nodes for `True`, `False`, and `X`
- 2 nodes for each variable: v and $\bar{v}$



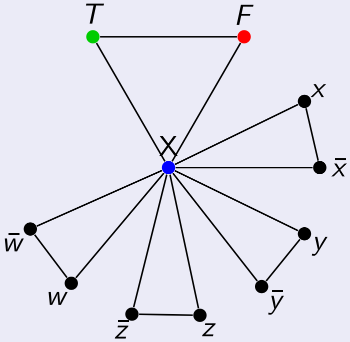
Incremental coalescing is NP-complete in the general case

Reduction from 3-SAT. *(Similar to reduction of graph-3-coloring from 3-SAT).*

Example on $(x \vee y \vee \bar{z} \vee w) \wedge \cdots \wedge (\bar{x} \vee z \vee \bar{y} \vee u)$.

First, equivalence between graph-3-coloring and 4-SAT.

- 3 nodes for `True`, `False`, and `X`
- 2 nodes for each variable: v and $\bar{v}$



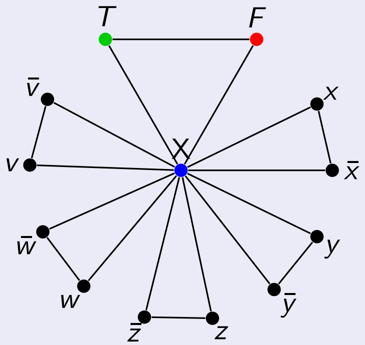
Incremental coalescing is NP-complete in the general case

Reduction from 3-SAT. *(Similar to reduction of graph-3-coloring from 3-SAT).*

Example on $(x \vee y \vee \bar{z} \vee w) \wedge \cdots \wedge (\bar{x} \vee z \vee \bar{y} \vee u)$.

First, equivalence between graph-3-coloring and 4-SAT.

- 3 nodes for `True`, `False`, and `X`
- 2 nodes for each variable: v and $\bar{v}$



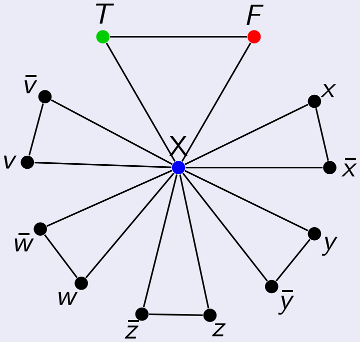
Incremental coalescing is NP-complete in the general case

Reduction from 3-SAT. *(Similar to reduction of graph-3-coloring from 3-SAT).*

Example on $(x \vee y \vee \bar{z} \vee w) \wedge \cdots \wedge (\bar{x} \vee z \vee \bar{y} \vee u)$.

First, equivalence between graph-3-coloring and 4-SAT.

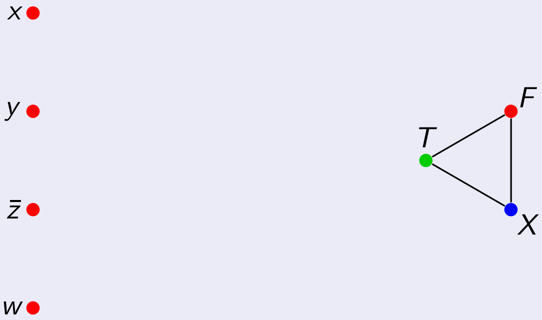
- 3 nodes for `True`, `False`, and `X`
- 2 nodes for each variable: v and $\bar{v}$
- ...and a widget to forbid every variable of a clause to be false



Incremental coalescing is NP-complete in the general case

Reduction from 3-SAT. *(Similar to reduction of graph-3-coloring from 3-SAT).*

Example on $(x \vee y \vee \bar{z} \vee w) \wedge \cdots \wedge (\bar{x} \vee z \vee \bar{y} \vee u)$.

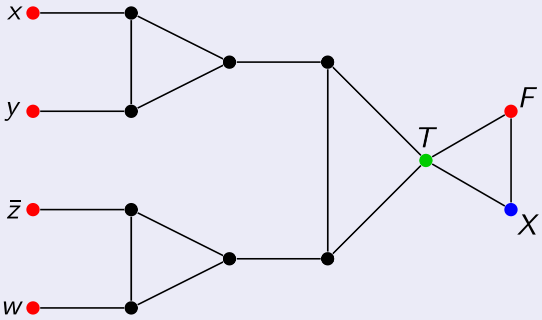


Need a widget that is 3-colorable only if not all 4 variables are false.

Incremental coalescing is NP-complete in the general case

Reduction from 3-SAT. *(Similar to reduction of graph-3-coloring from 3-SAT).*

Example on $(x \vee y \vee \bar{z} \vee w) \wedge \cdots \wedge (\bar{x} \vee z \vee \bar{y} \vee u)$.

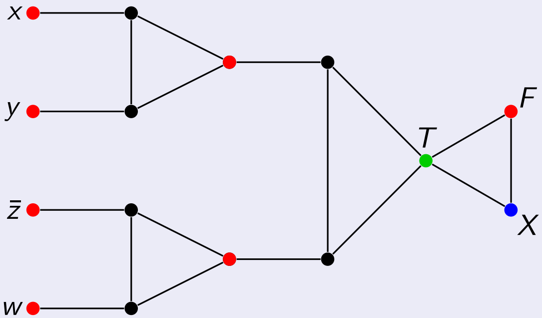


Need a widget that is 3-colorable only if not all 4 variables are false.

Incremental coalescing is NP-complete in the general case

Reduction from 3-SAT. *(Similar to reduction of graph-3-coloring from 3-SAT).*

Example on $(x \vee y \vee \bar{z} \vee w) \wedge \cdots \wedge (\bar{x} \vee z \vee \bar{y} \vee u)$.

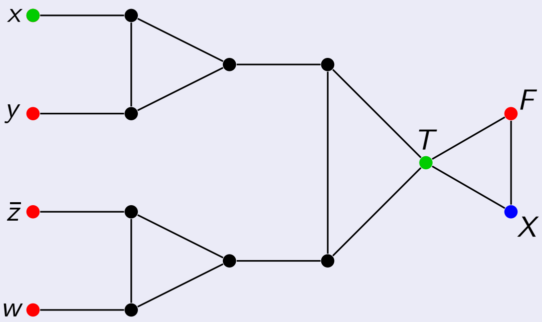


If all 4 variables are false, not 3-colorable.

Incremental coalescing is NP-complete in the general case

Reduction from 3-SAT. *(Similar to reduction of graph-3-coloring from 3-SAT).*

Example on $(x \vee y \vee \bar{z} \vee w) \wedge \cdots \wedge (\bar{x} \vee z \vee \bar{y} \vee u)$.

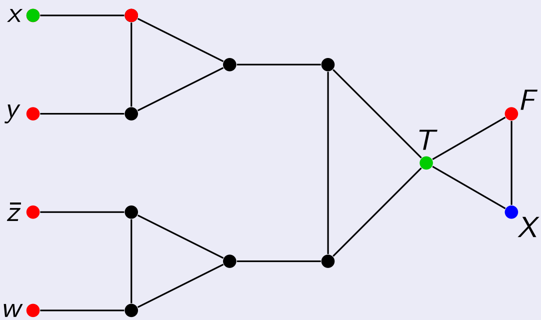


If at least one variable is true

Incremental coalescing is NP-complete in the general case

Reduction from 3-SAT. *(Similar to reduction of graph-3-coloring from 3-SAT).*

Example on $(x \vee y \vee \bar{z} \vee w) \wedge \cdots \wedge (\bar{x} \vee z \vee \bar{y} \vee u)$.

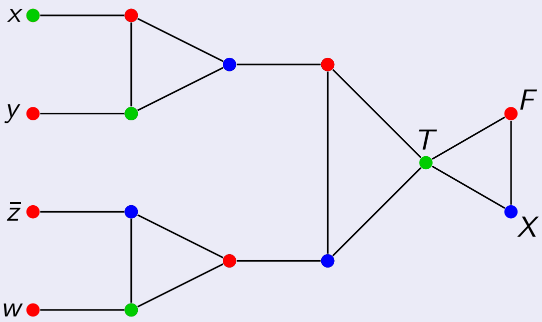


If at least one variable is true

Incremental coalescing is NP-complete in the general case

Reduction from 3-SAT. *(Similar to reduction of graph-3-coloring from 3-SAT).*

Example on $(x \vee y \vee \bar{z} \vee w) \wedge \cdots \wedge (\bar{x} \vee z \vee \bar{y} \vee u)$.



If at least one variable is true, 3-colorable.

Incremental coalescing is NP-complete in the general case

Reduction from 3-SAT. *(Similar to reduction of graph-3-coloring from 3-SAT).*

Example on $(x \vee y \vee \bar{z} \vee w) \wedge \cdots \wedge (\bar{x} \vee z \vee \bar{y} \vee u)$.

Now, transform 3-SAT instance into 4-SAT by adding x_0 to every clause:

$$(y \vee \bar{z} \vee w) \wedge \cdots \wedge (z \vee \bar{y} \vee u)$$

becomes

$$(x_0 \vee y \vee \bar{z} \vee w) \wedge \cdots \wedge (x_0 \vee z \vee \bar{y} \vee u)$$

Clearly, $x_0 = \texttt{True}$ satisfies the equation (i.e., the graph is 3-colorable).

Incremental coalescing is NP-complete in the general case

Reduction from 3-SAT. *(Similar to reduction of graph-3-coloring from 3-SAT).*

Example on $(x \vee y \vee \bar{z} \vee w) \wedge \cdots \wedge (\bar{x} \vee z \vee \bar{y} \vee u)$.

Now, transform 3-SAT instance into 4-SAT by adding x_0 to every clause:

$$(y \vee \bar{z} \vee w) \wedge \cdots \wedge (z \vee \bar{y} \vee u)$$

becomes

$$(x_0 \vee y \vee \bar{z} \vee w) \wedge \cdots \wedge (x_0 \vee z \vee \bar{y} \vee u)$$

Clearly, $x_0 =$`True` satisfies the equation (i.e., the graph is 3-colorable).

Now, ask x_0 and `False` to be coalesced...

□

Incremental coalescing is NP-complete in the general case

Reduction from 3-SAT. *(Similar to reduction of graph-3-coloring from 3-SAT).*

Example on $(x \vee y \vee \bar{z} \vee w) \wedge \cdots \wedge (\bar{x} \vee z \vee \bar{y} \vee u)$.

To conclude:

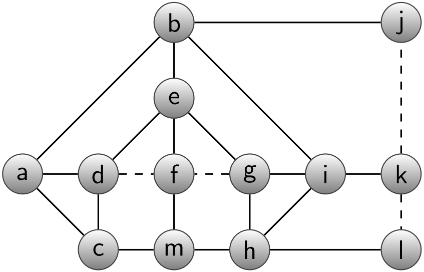
3-SAT is true	$\iff$	4-SAT is true with $x_0 = \texttt{False}$
	$\iff$	graph is 3-colorable with x_0 in red/$\texttt{False}$
	$\iff$	incremental coalescing of x_0 with $\texttt{False}$ is possible

$\square$

Incremental conservative coalescing

Finding the optimal subset of affinities is hard.

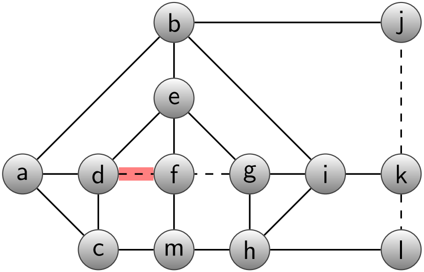
Algorithms do incremental conservative coalescing.



Incremental conservative coalescing

Finding the optimal subset of affinities is hard.

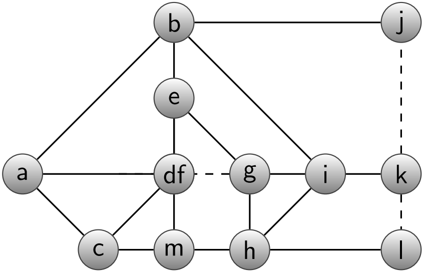
Algorithms do incremental conservative coalescing.



Incremental conservative coalescing

Finding the optimal subset of affinities is hard.

Algorithms do incremental conservative coalescing.

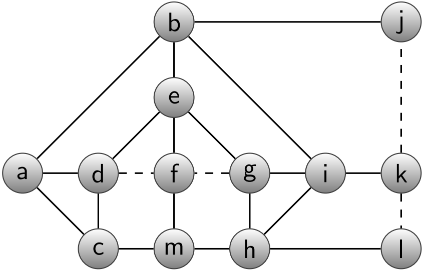


Not greedy-3-colorable

Incremental conservative coalescing

Finding the optimal subset of affinities is hard.

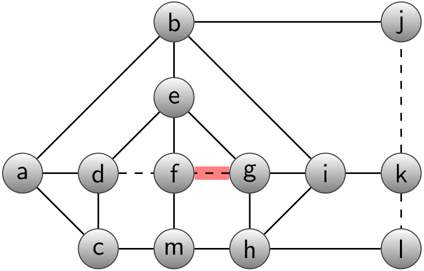
Algorithms do incremental conservative coalescing.



Incremental conservative coalescing

Finding the optimal subset of affinities is hard.

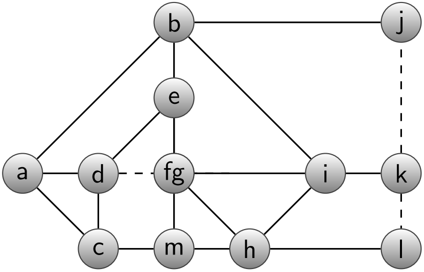
Algorithms do incremental conservative coalescing.



Incremental conservative coalescing

Finding the optimal subset of affinities is hard.

Algorithms do incremental conservative coalescing.

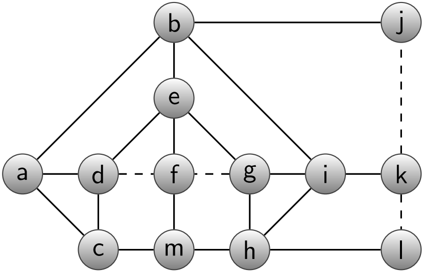


Not greedy-3-colorable

Incremental conservative coalescing

Finding the optimal subset of affinities is hard.

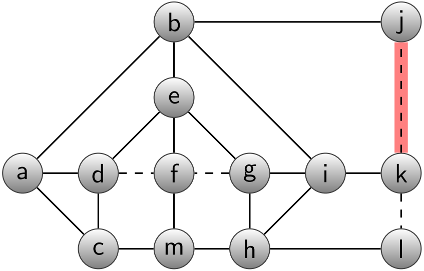
Algorithms do incremental conservative coalescing.



Incremental conservative coalescing

Finding the optimal subset of affinities is hard.

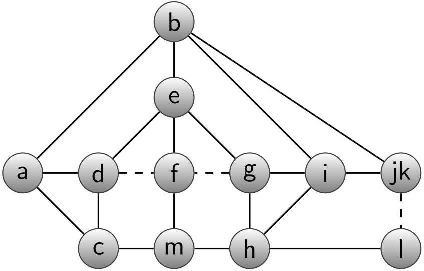
Algorithms do incremental conservative coalescing.



Incremental conservative coalescing

Finding the optimal subset of affinities is hard.

Algorithms do incremental conservative coalescing.

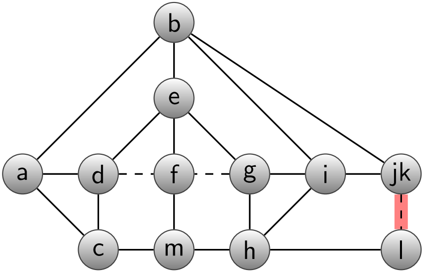


greedy-3-colorable

Incremental conservative coalescing

Finding the optimal subset of affinities is hard.

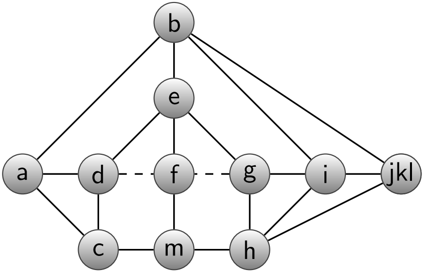
Algorithms do incremental conservative coalescing.



Incremental conservative coalescing

Finding the optimal subset of affinities is hard.

Algorithms do incremental conservative coalescing.

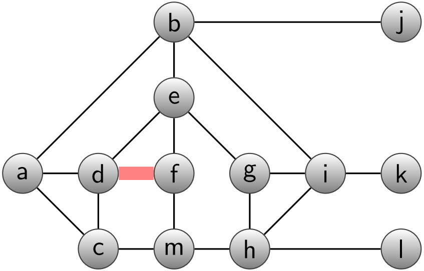


greedy-3-colorable

A gap

Incremental conservative is not optimal.

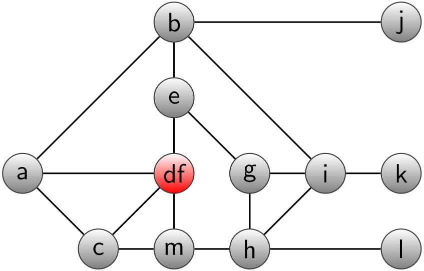
Greedy-k-colorable test might be stuck. Multiple node merging necessary to stay Greedy-k-colorable.



A gap

Incremental conservative is not optimal.

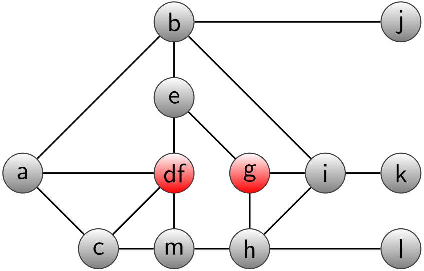
Greedy-k-colorable test might be stuck. Multiple node merging necessary to stay Greedy-k-colorable.



A gap

Incremental conservative is not optimal.

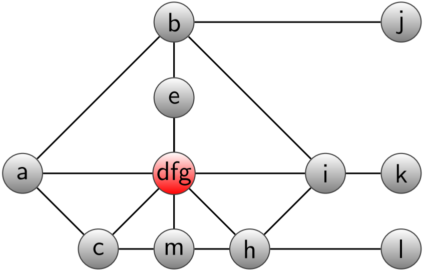
Greedy-k-colorable test might be stuck. Multiple node merging necessary to stay Greedy-k-colorable.



A gap

Incremental conservative is not optimal.

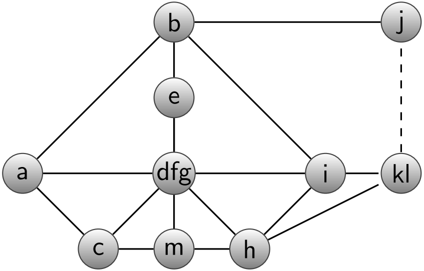
Greedy-k-colorable test might be stuck. Multiple node merging necessary to stay Greedy-k-colorable.



A gap

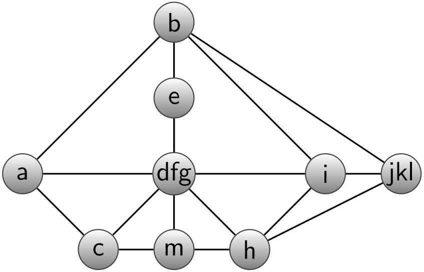
Incremental conservative is not optimal.

Greedy-k-colorable test might be stucked. Multiple node merging necessary to stay Greedy-k-colorable.

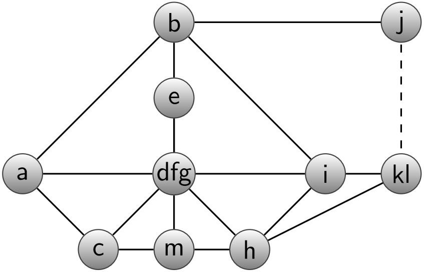


Aggressive + decoalescing

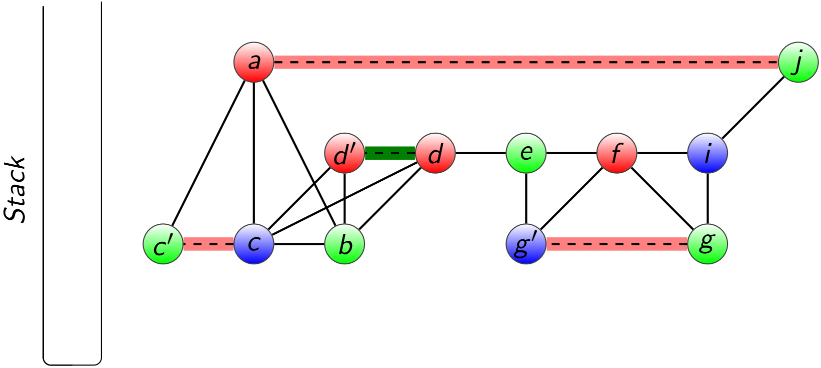
Aggressive + de-coalescing scheme: start from a completely aggressively coalesced graph, give up with some move until it gets Greedy-k-colorable again.



Aggressive + de-coalescing scheme: start from a completely aggressively coalesced graph, give up with some move until it gets Greedy-k-colorable again.



Back to our colored graph

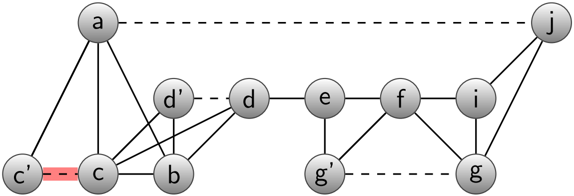


Coalescing two nodes

- Briggs

Briggs

Resulting node has $< k$ high-degree neighbours

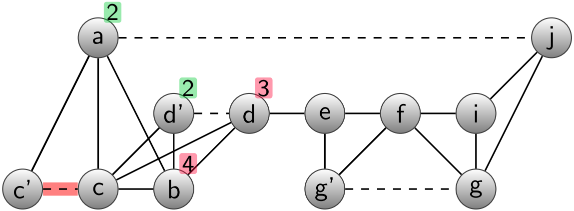


Coalescing two nodes

- Briggs

Briggs

Resulting node has $< k$ high-degree neighbours

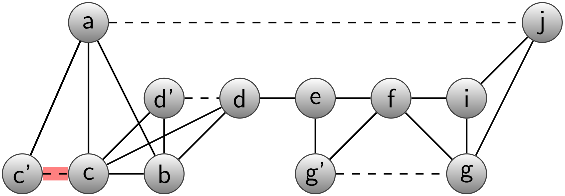


Coalescing two nodes

- Briggs

Briggs

Resulting node has $< k$ high-degree neighbours

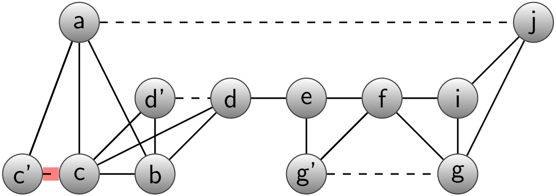


Coalescing two nodes

- Briggs

Briggs

Resulting node has $< k$ high-degree neighbours

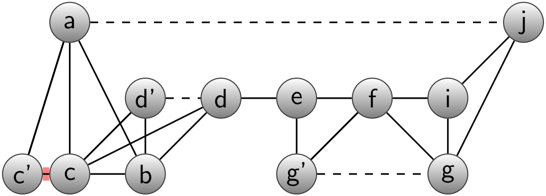


Coalescing two nodes

- Briggs

Briggs

Resulting node has $< k$ high-degree neighbours

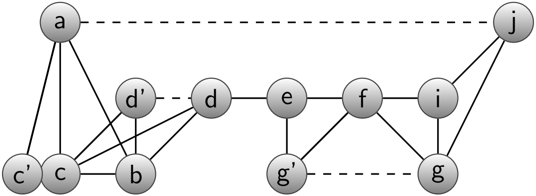


Coalescing two nodes

- Briggs

Briggs

Resulting node has $< k$ high-degree neighbours

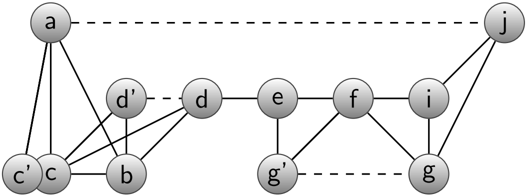


Coalescing two nodes

- Briggs

Briggs

Resulting node has $< k$ high-degree neighbours

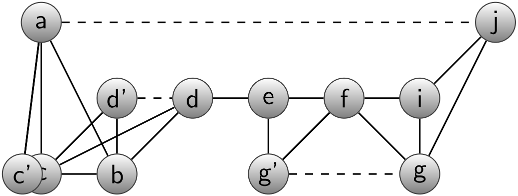


Coalescing two nodes

- Briggs

Briggs

Resulting node has $< k$ high-degree neighbours

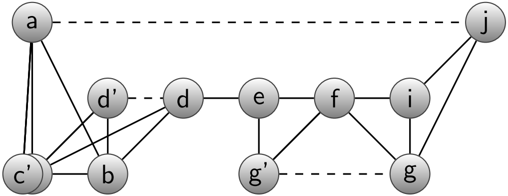


Coalescing two nodes

- Briggs

Briggs

Resulting node has $< k$ high-degree neighbours

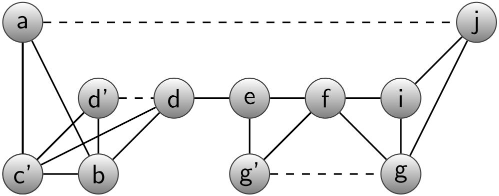


Coalescing two nodes

- Briggs

Briggs

Resulting node has $< k$ high-degree neighbours

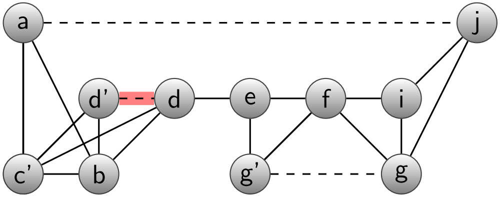


Coalescing two nodes

- Briggs
- George

George

All high-degree neighbours are neighbours of the other node

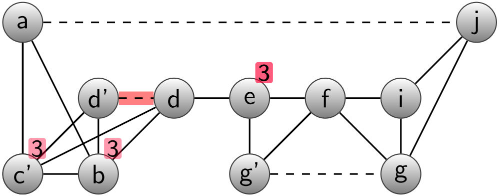


Coalescing two nodes

- Briggs
- George

George

All high-degree neighbours are neighbours of the other node

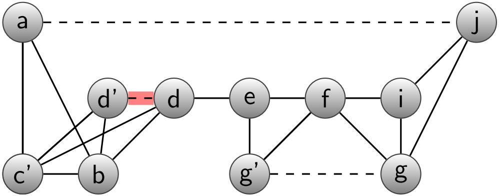


Coalescing two nodes

- Briggs
- George

George

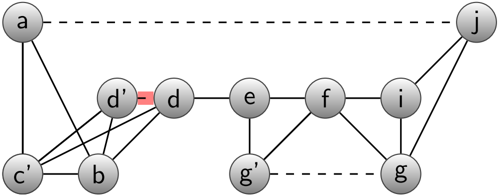
All high-degree neighbours are neighbours of the other node



- Briggs
- George

George

All high-degree neighbours are neighbours of the other node

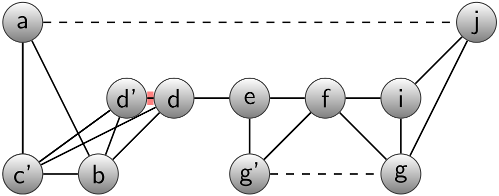


Coalescing two nodes

- Briggs
- George

George

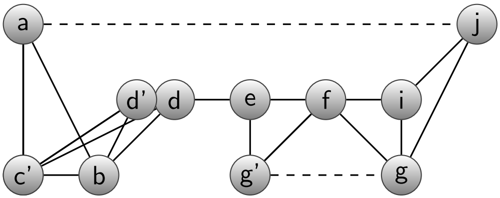
All high-degree neighbours are neighbours of the other node



- Briggs
- George

George

All high-degree neighbours are neighbours of the other node

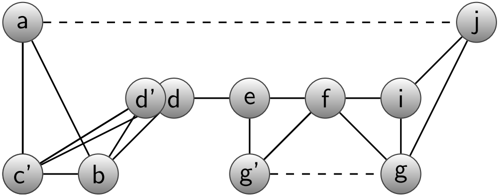


Coalescing two nodes

- Briggs
- George

George

All high-degree neighbours are neighbours of the other node

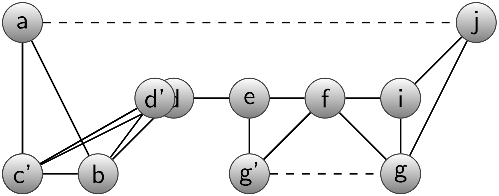


Coalescing two nodes

- Briggs
- George

George

All high-degree neighbours are neighbours of the other node

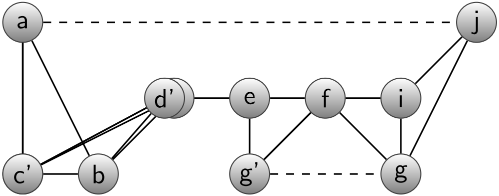


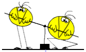
Coalescing two nodes

- Briggs
- George

George

All high-degree neighbours are neighbours of the other node



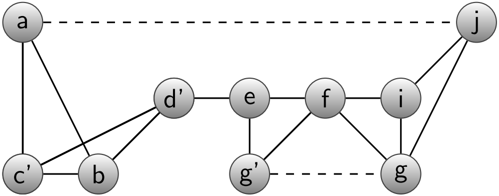


Coalescing two nodes

- Briggs
- George

George

All high-degree neighbours are neighbours of the other node

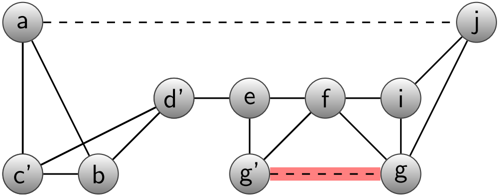


Coalescing two nodes

- Briggs
- George
- Brute-force

Brute-force

Merge the nodes and check if resulting graph is greedy-k-colorable

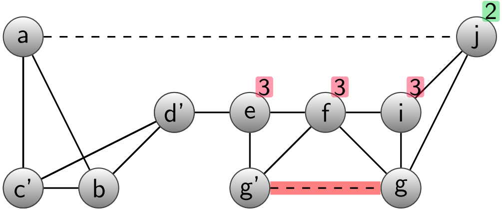


Coalescing two nodes

- Briggs
- George
- Brute-force

Brute-force

Merge the nodes and check if resulting graph is greedy-k-colorable

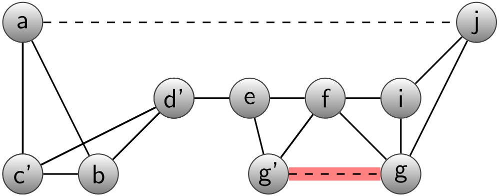


Coalescing two nodes

- Briggs
- George
- Brute-force

Brute-force

Merge the nodes and check if resulting graph is greedy-k-colorable

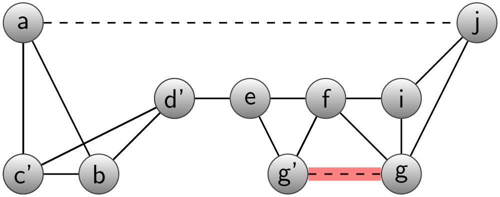


Coalescing two nodes

- Briggs
- George
- Brute-force

Brute-force

Merge the nodes and check if resulting graph is greedy-k-colorable

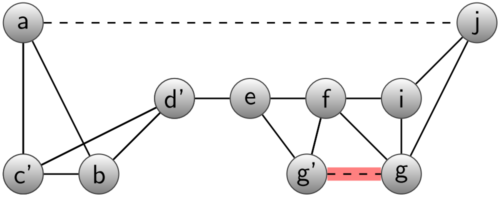


Coalescing two nodes

- Briggs
- George
- Brute-force

Brute-force

Merge the nodes and check if resulting graph is greedy-k-colorable

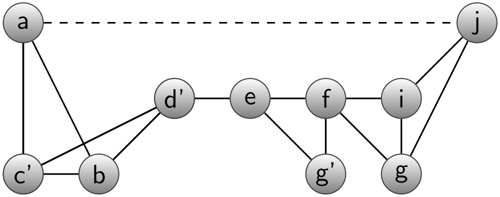


Coalescing two nodes

- Briggs
- George
- Brute-force

Brute-force

Merge the nodes and check if resulting graph is greedy-k-colorable

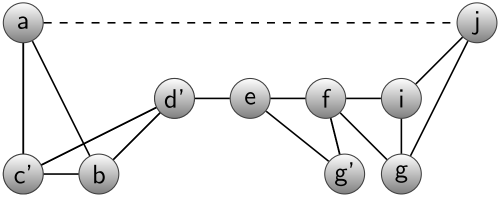


Coalescing two nodes

- Briggs
- George
- Brute-force

Brute-force

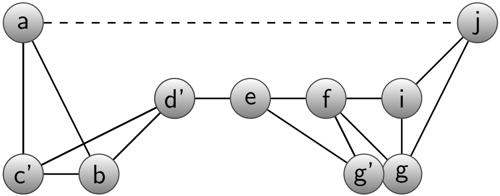
Merge the nodes and check if resulting graph is greedy-k-colorable



- Briggs
- George
- Brute-force

Brute-force

Merge the nodes and check if resulting graph is greedy-k-colorable

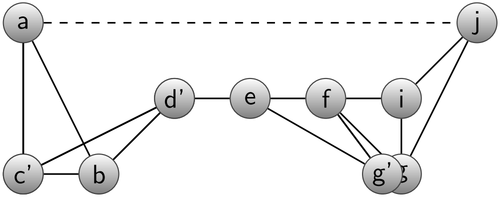


Coalescing two nodes

- Briggs
- George
- Brute-force

Brute-force

Merge the nodes and check if resulting graph is greedy-k-colorable

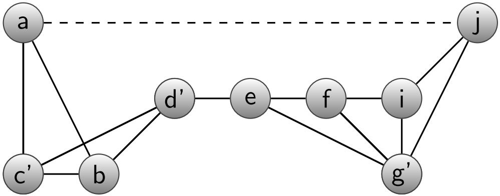


Coalescing two nodes

- Briggs
- George
- Brute-force

Brute-force

Merge the nodes and check if resulting graph is greedy-k-colorable

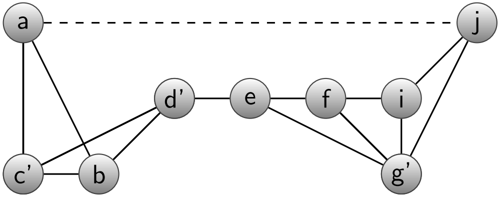


Coalescing two nodes

- Briggs
- George
- Brute-force
- Chordal

Chordal

Relies on optimal incremental coalescing for interval graphs. *(May need to merge other nodes to get a greedy-k-colorable graph.)*

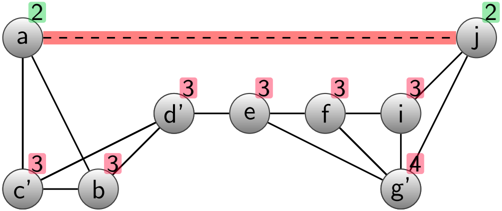


Coalescing two nodes

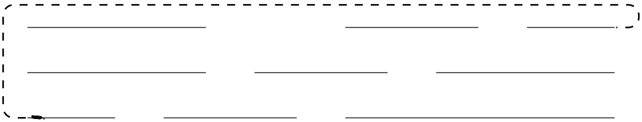
- Briggs
- George
- Brute-force
- Chordal

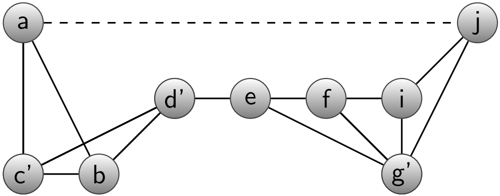
Chordal

Relies on optimal incremental coalescing for interval graphs. *(May need to merge other nodes to get a greedy-k-colorable graph.)*

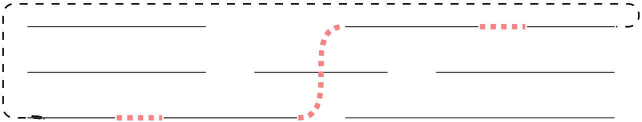


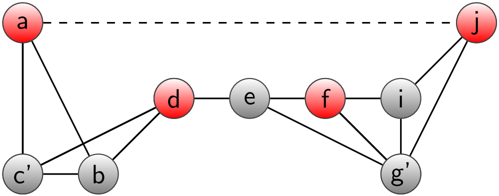
Coalescing two nodes



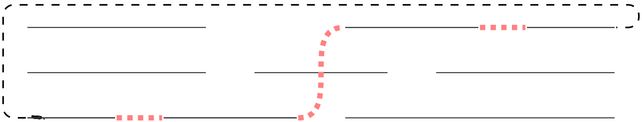


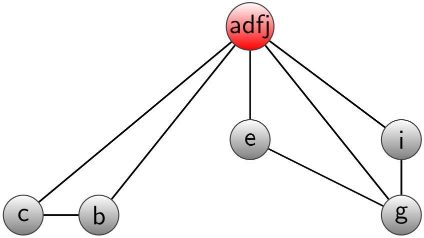
Coalescing two nodes





Coalescing two nodes. . . with additional merges





Incremental coalescing for chordal graphs

Problem (Incremental coalescing for chordal graphs)

Given a k-colorable chordal graph G and two nodes x and y. Is it possible to color G such that x and y have the same color?

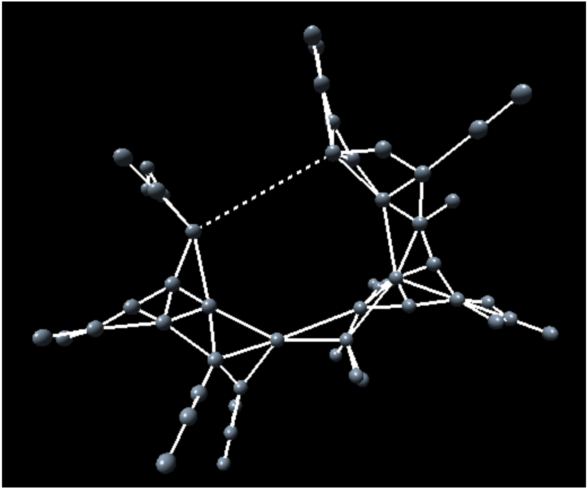
This problem is polynomial!

Moreover, if the answer is yes, it is possible to modify G so that x and y are merged and G stays chordal.

The same question with greedy-k-colorable graphs is still open.

Example on a chordal graph

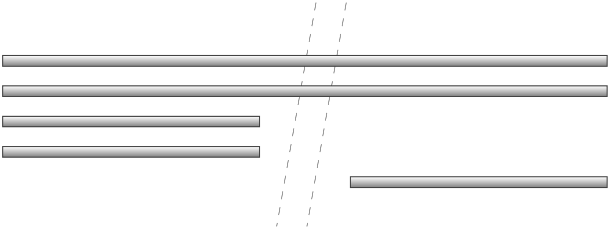
Let us consider a 3-colorable chordal graph.



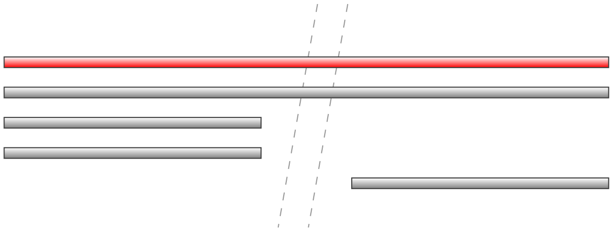
Demo: chordal graph in ubigraph

Splitting the tree

What happens at one point on the path, color-wise?

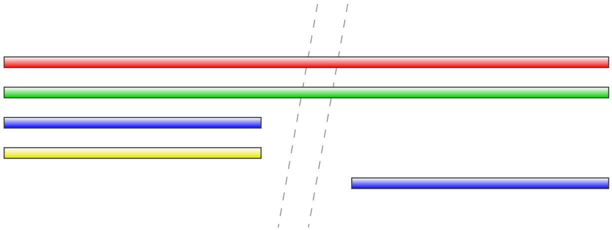


What happens at one point on the path, color-wise?



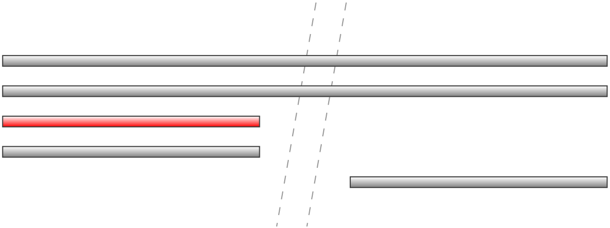
Splitting the tree

What happens at one point on the path, color-wise?



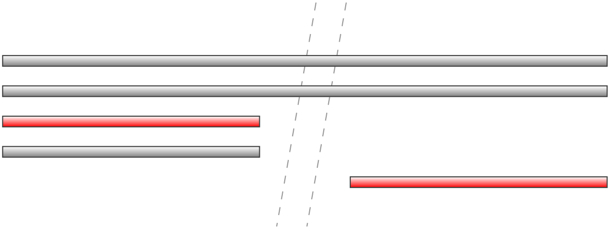
Splitting the tree

What happens at one point on the path, color-wise?



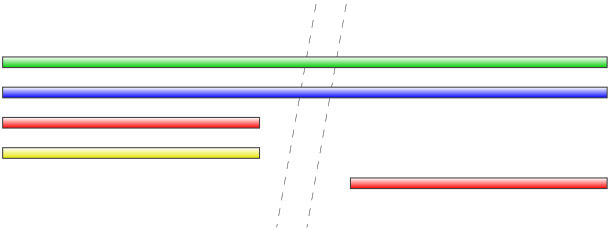
Splitting the tree

What happens at one point on the path, color-wise?



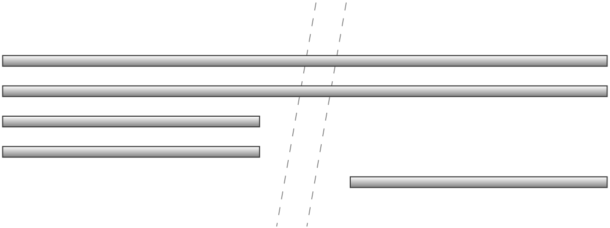
Splitting the tree

What happens at one point on the path, color-wise?



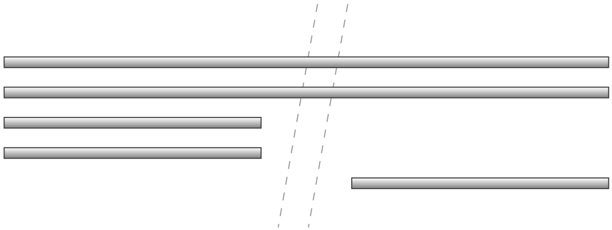
Splitting the tree

What happens at one point on the path, color-wise?



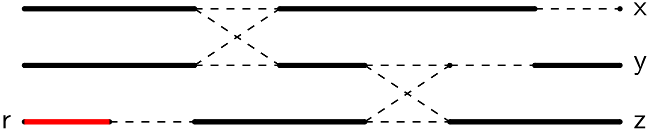
Splitting the tree

What happens at one point on the path, color-wise?

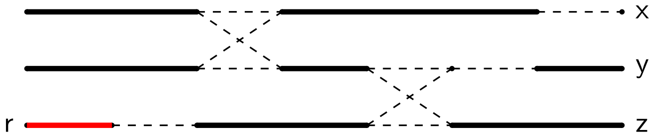


Except for the “live-through” variables, the two parts of the tree are independent.

Finding a path on the subtrees



Finding a path on the subtrees

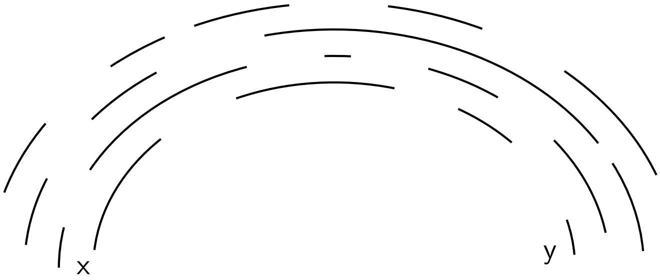


There exists a 3-coloring in which r and y have the same color.
Idem for r and z.

But there is no coloring in which r and x have the same color.

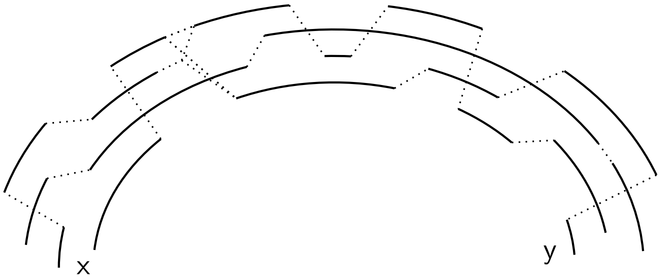
Finding a path on the interval graph

Once the branches are pruned, an interval graph remains.



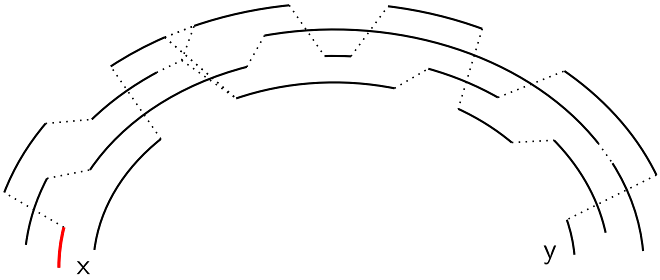
Finding a path on the interval graph

Once the branches are pruned, an interval graph remains.



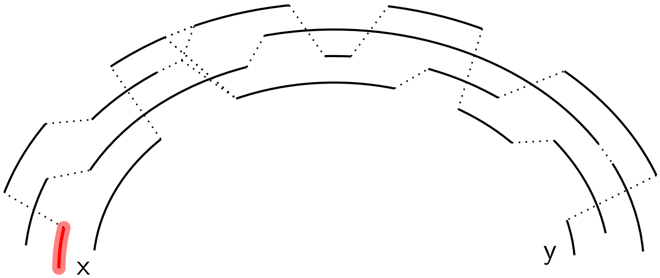
Finding a path on the interval graph

Once the branches are pruned, an interval graph remains.



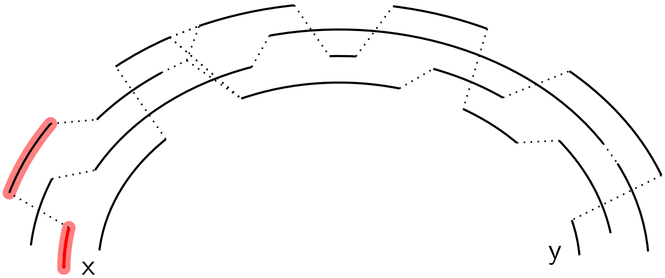
Finding a path on the interval graph

Once the branches are pruned, an interval graph remains.



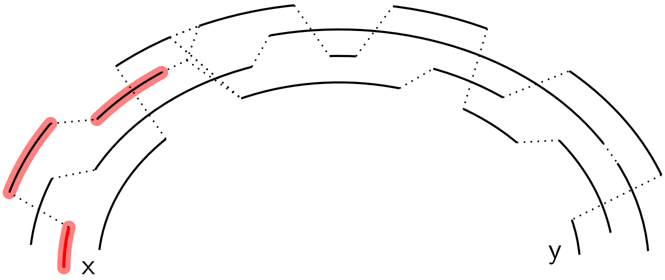
Finding a path on the interval graph

Once the branches are pruned, an interval graph remains.



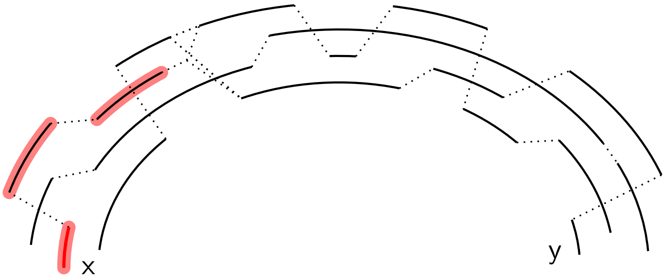
Finding a path on the interval graph

Once the branches are pruned, an interval graph remains.



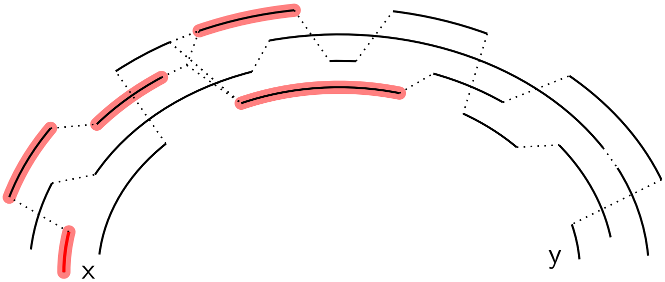
Finding a path on the interval graph

Once the branches are pruned, an interval graph remains.



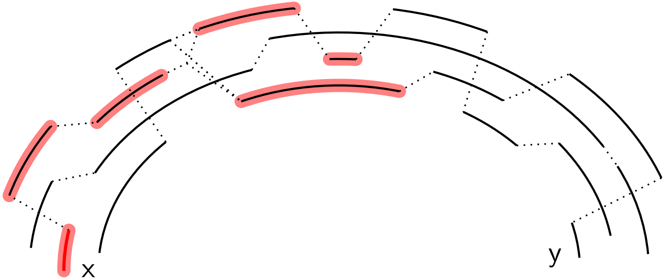
Finding a path on the interval graph

Once the branches are pruned, an interval graph remains.



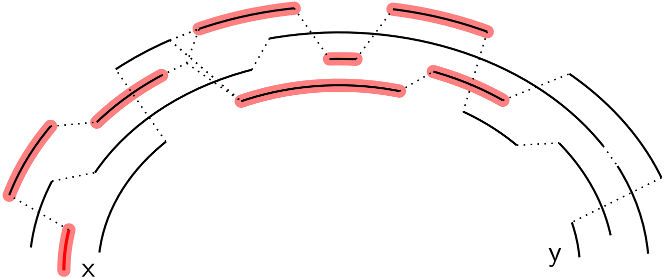
Finding a path on the interval graph

Once the branches are pruned, an interval graph remains.



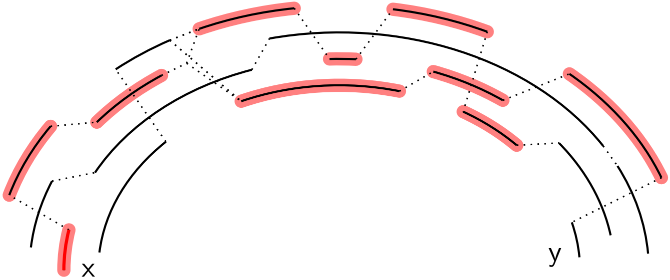
Finding a path on the interval graph

Once the branches are pruned, an interval graph remains.



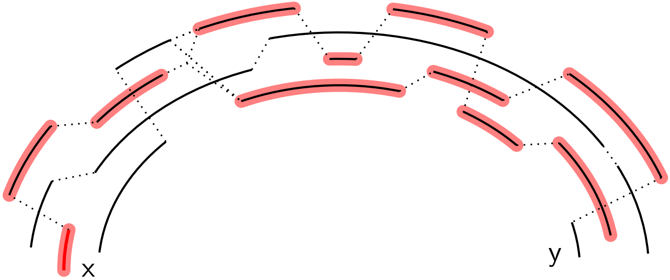
Finding a path on the interval graph

Once the branches are pruned, an interval graph remains.



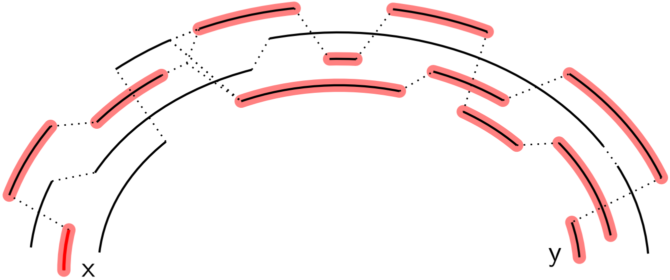
Finding a path on the interval graph

Once the branches are pruned, an interval graph remains.



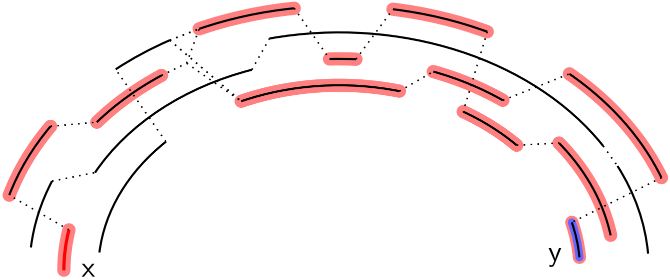
Finding a path on the interval graph

Once the branches are pruned, an interval graph remains.



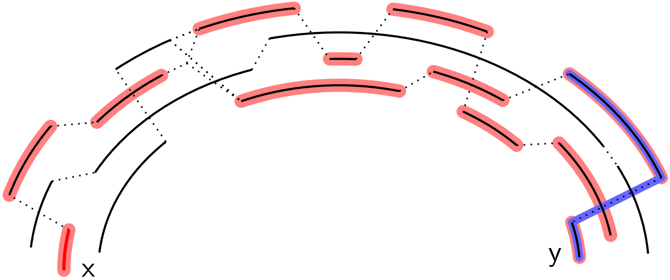
Finding a path on the interval graph

Once the branches are pruned, an interval graph remains.



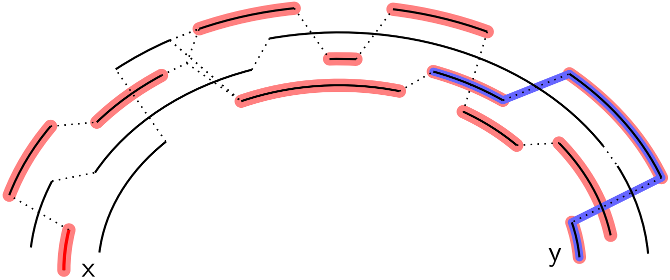
Finding a path on the interval graph

Once the branches are pruned, an interval graph remains.



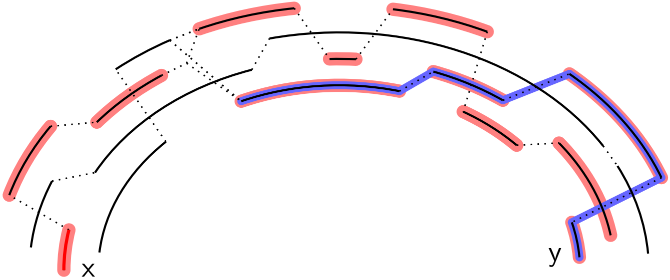
Finding a path on the interval graph

Once the branches are pruned, an interval graph remains.



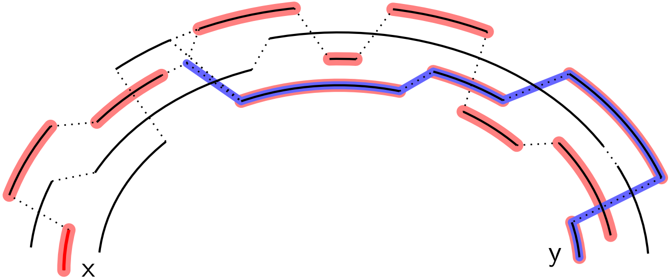
Finding a path on the interval graph

Once the branches are pruned, an interval graph remains.



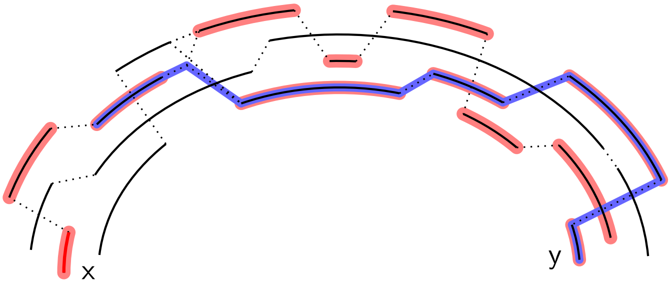
Finding a path on the interval graph

Once the branches are pruned, an interval graph remains.



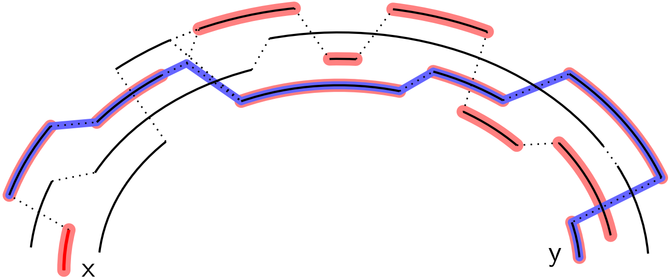
Finding a path on the interval graph

Once the branches are pruned, an interval graph remains.



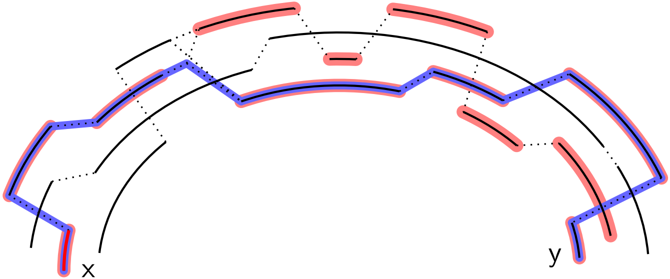
Finding a path on the interval graph

Once the branches are pruned, an interval graph remains.



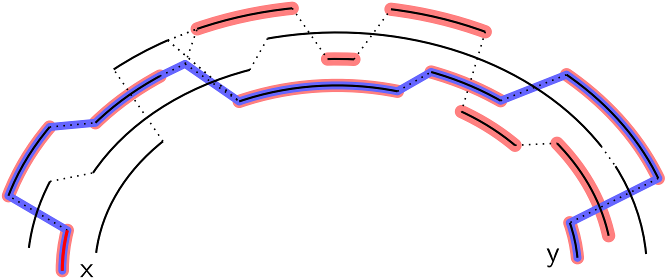
Finding a path on the interval graph

Once the branches are pruned, an interval graph remains.



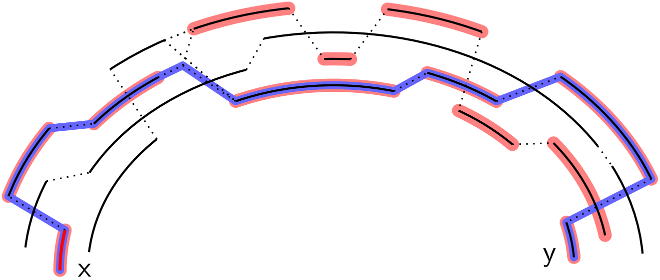
Finding a path on the interval graph

Once the branches are pruned, an interval graph remains.



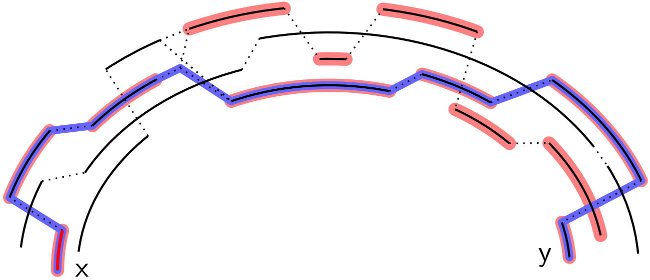
Finding a path on the interval graph

Once the branches are pruned, an interval graph remains.



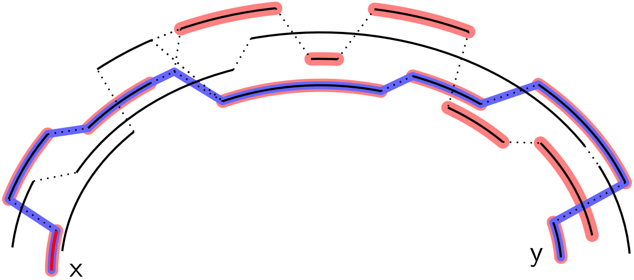
Finding a path on the interval graph

Once the branches are pruned, an interval graph remains.



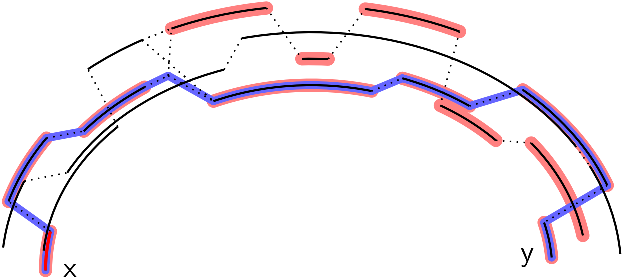
Finding a path on the interval graph

Once the branches are pruned, an interval graph remains.



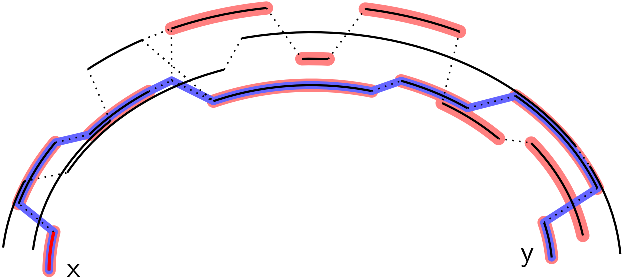
Finding a path on the interval graph

Once the branches are pruned, an interval graph remains.



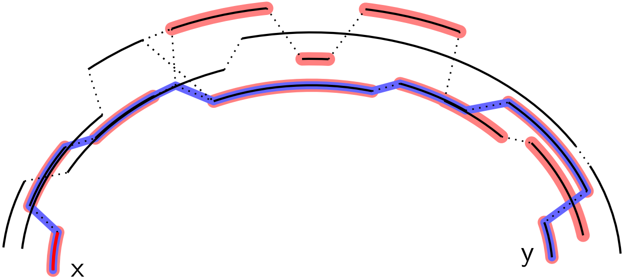
Finding a path on the interval graph

Once the branches are pruned, an interval graph remains.



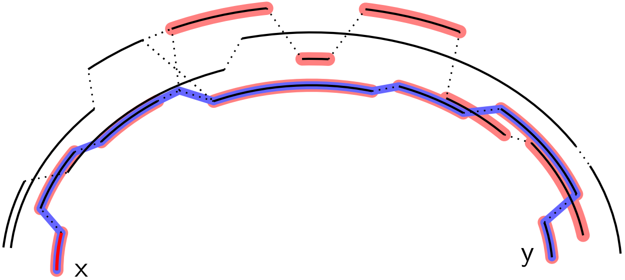
Finding a path on the interval graph

Once the branches are pruned, an interval graph remains.



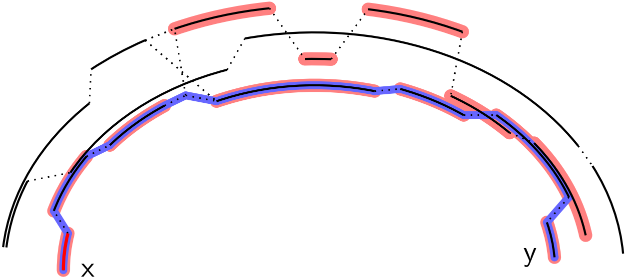
Finding a path on the interval graph

Once the branches are pruned, an interval graph remains.



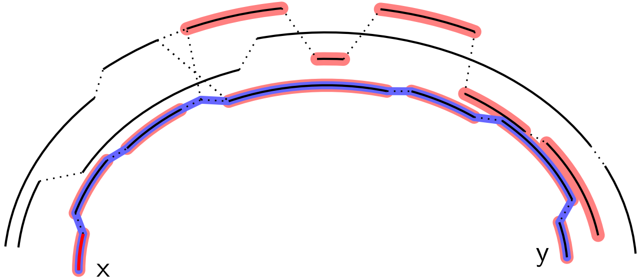
Finding a path on the interval graph

Once the branches are pruned, an interval graph remains.



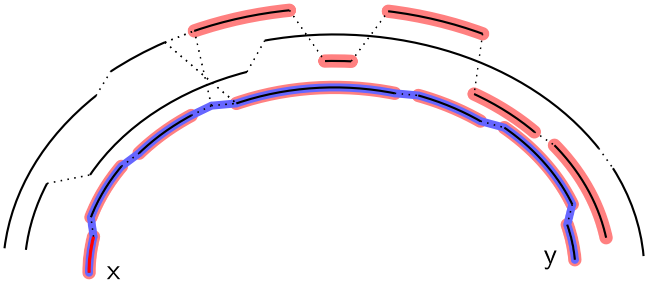
Finding a path on the interval graph

Once the branches are pruned, an interval graph remains.



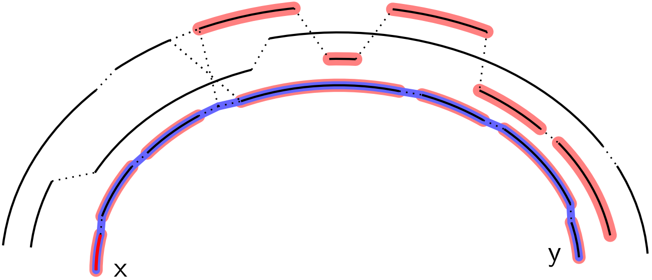
Finding a path on the interval graph

Once the branches are pruned, an interval graph remains.



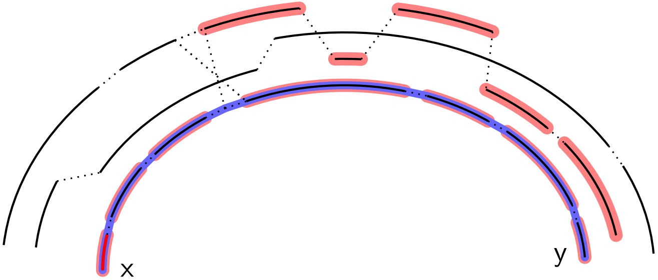
Finding a path on the interval graph

Once the branches are pruned, an interval graph remains.



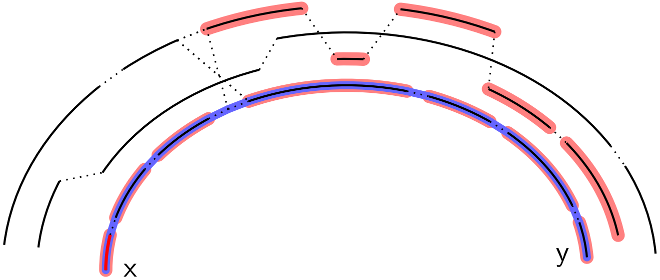
Finding a path on the interval graph

Once the branches are pruned, an interval graph remains.



Finding a path on the interval graph

Once the branches are pruned, an interval graph remains.



A simple algorithmic strategy for chordal incremental coalescing

Building the representation of a chordal graph as subtrees of a tree is painful.

We have devised an algorithmic strategy that works directly on the graph, using the same ideas as in Chaitin et al.'s greedy coloring algorithm.

Demonstration of coalescing

Demonstration of conservative coalescing on graph #311 of the "Coalescing Challenge." *(Appel&George)*

Theoretical limits of coalescing schemes

- Conservative rules (e.g., Briggs & George)

- Optimistic coalescing (e.g., Park & Moon)

Theoretical limits of coalescing schemes

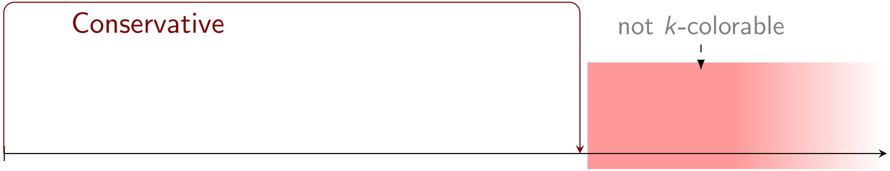
- Conservative rules (e.g., Briggs & George)
  ➠ *incremental coalescing*
- Optimistic coalescing (e.g., Park & Moon)
  ➠ *aggressive coalescing* + *de-coalescing*

Theoretical limits of coalescing schemes

- Conservative rules (e.g., Briggs & George)
  ➠ *incremental coalescing*
- Optimistic coalescing (e.g., Park & Moon)
  ➠ *aggressive coalescing + de-coalescing*

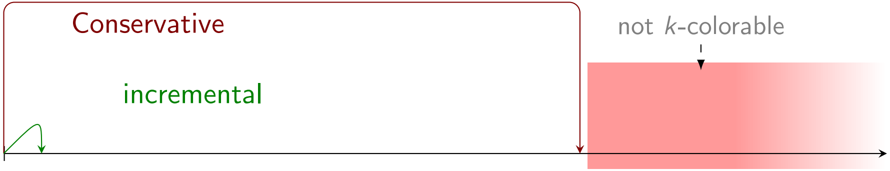
Theoretical limits of coalescing schemes

- Conservative rules (e.g., Briggs & George)
  ➠ *incremental coalescing*
- Optimistic coalescing (e.g., Park & Moon)
  ➠ *aggressive coalescing + de-coalescing*



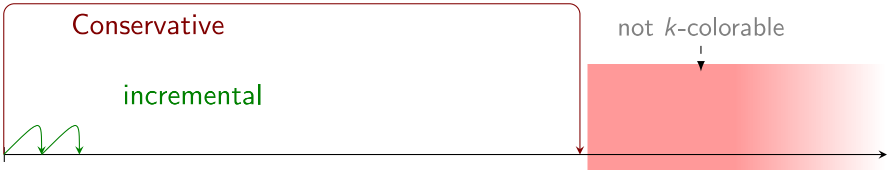
Theoretical limits of coalescing schemes

- Conservative rules (e.g., Briggs & George)
  ➠ *incremental coalescing*
- Optimistic coalescing (e.g., Park & Moon)
  ➠ *aggressive coalescing + de-coalescing*



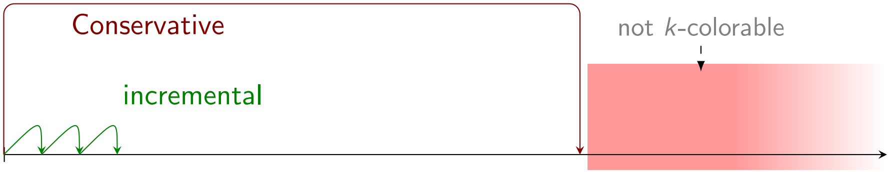
Theoretical limits of coalescing schemes

- Conservative rules (e.g., Briggs & George)
  ➠ *incremental coalescing*
- Optimistic coalescing (e.g., Park & Moon)
  ➠ *aggressive coalescing + de-coalescing*



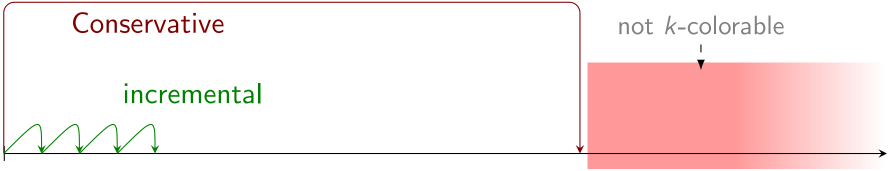
Theoretical limits of coalescing schemes

- Conservative rules (e.g., Briggs & George)
  ➠ *incremental coalescing*
- Optimistic coalescing (e.g., Park & Moon)
  ➠ *aggressive coalescing + de-coalescing*



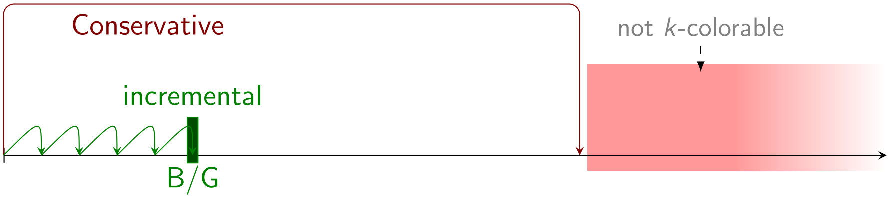
Theoretical limits of coalescing schemes

- Conservative rules (e.g., Briggs & George)
  ➠ *incremental coalescing*
- Optimistic coalescing (e.g., Park & Moon)
  ➠ *aggressive coalescing + de-coalescing*



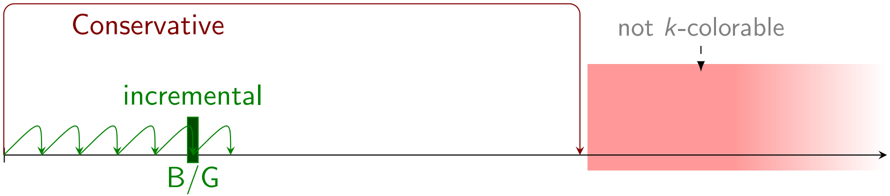
Theoretical limits of coalescing schemes

- Conservative rules (e.g., Briggs & George)
  ➠ *incremental coalescing*
- Optimistic coalescing (e.g., Park & Moon)
  ➠ *aggressive coalescing + de-coalescing*



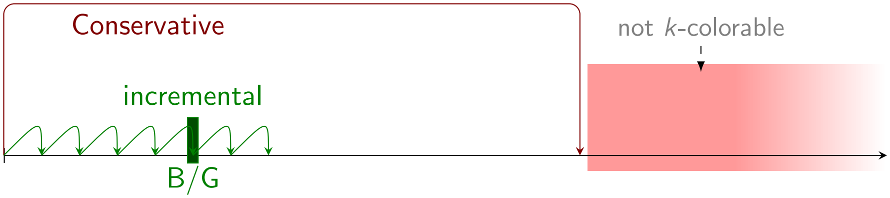
Theoretical limits of coalescing schemes

- Conservative rules (e.g., Briggs & George)
  ➠ *incremental coalescing*
- Optimistic coalescing (e.g., Park & Moon)
  ➠ *aggressive coalescing + de-coalescing*



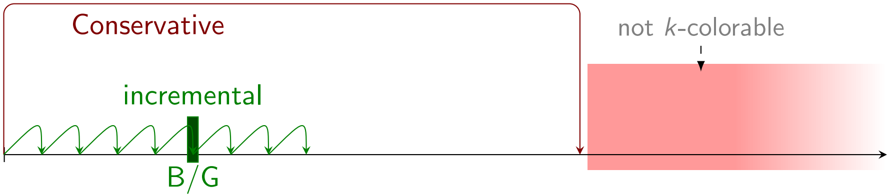
Theoretical limits of coalescing schemes

- Conservative rules (e.g., Briggs & George)
  ➠ *incremental coalescing*
- Optimistic coalescing (e.g., Park & Moon)
  ➠ *aggressive coalescing + de-coalescing*



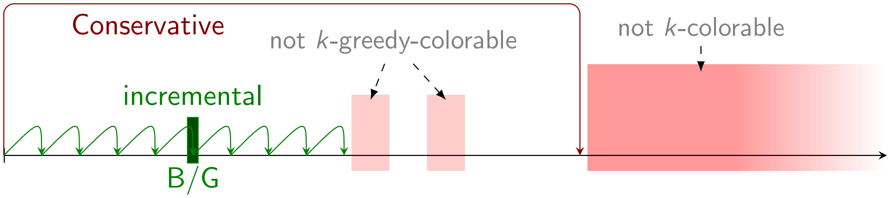
Theoretical limits of coalescing schemes

- Conservative rules (e.g., Briggs & George)
  ➠ *incremental coalescing*
- Optimistic coalescing (e.g., Park & Moon)
  ➠ *aggressive coalescing + de-coalescing*



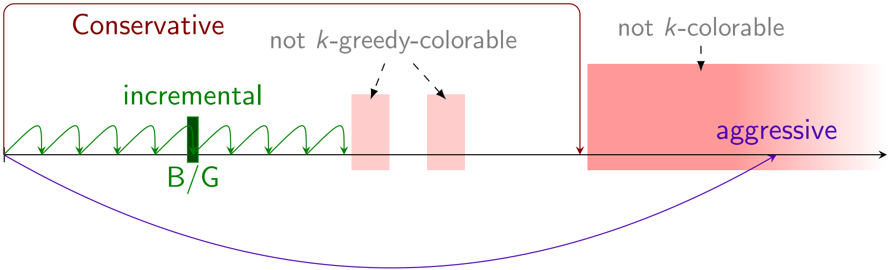
Theoretical limits of coalescing schemes

- Conservative rules (e.g., Briggs & George)
  ➠ *incremental coalescing*
- Optimistic coalescing (e.g., Park & Moon)
  ➠ *aggressive coalescing + de-coalescing*



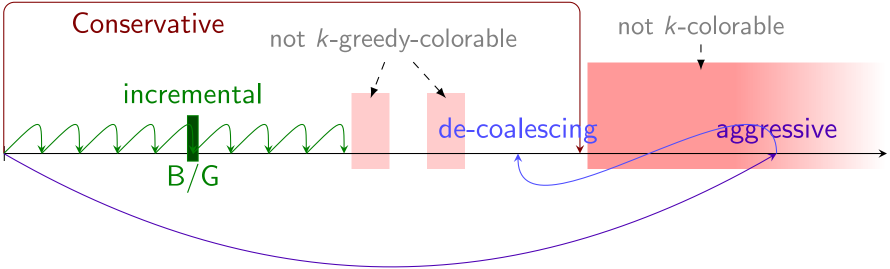
Theoretical limits of coalescing schemes

- Conservative rules (e.g., Briggs & George)
  ➠ *incremental coalescing*
- Optimistic coalescing (e.g., Park & Moon)
  ➠ *aggressive coalescing + de-coalescing*



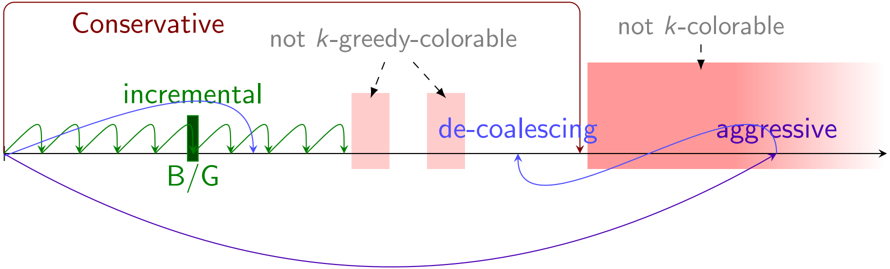
Theoretical limits of coalescing schemes

- Conservative rules (e.g., Briggs & George)
  ➠ *incremental coalescing*
- Optimistic coalescing (e.g., Park & Moon)
  ➠ *aggressive coalescing + de-coalescing*



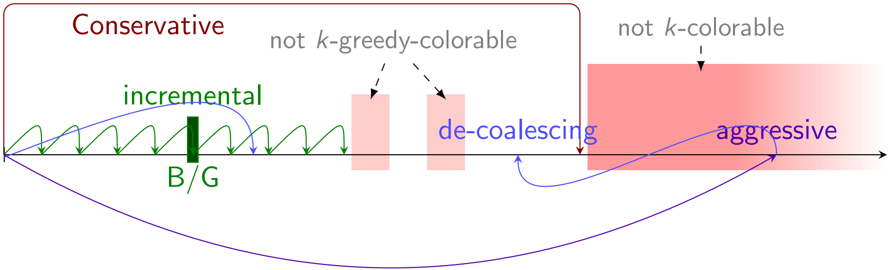
Theoretical limits of coalescing schemes

- Conservative rules (e.g., Briggs & George)
  ➠ *incremental coalescing*
- Optimistic coalescing (e.g., Park & Moon)
  ➠ *aggressive coalescing + de-coalescing*



Theoretical limits of coalescing schemes

- Conservative rules (e.g., Briggs & George)
  ➠ *incremental coalescing* — *NP-complete (not greedy-check)*
- Optimistic coalescing (e.g., Park & Moon)
  ➠ *aggressive coalescing + de-coalescing* — *both NP-complete*



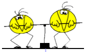
To summarize...

SSA Form is a powerfull property for compilers.

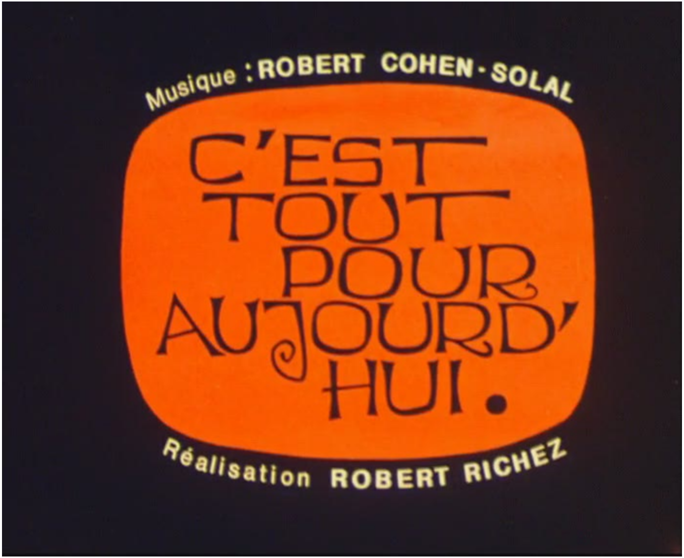
Register allocation under SSA can be separated into two clean phases:

1. spilling
2. coloring/coalescing

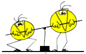
Bonus: what will be written left to the pumping shadoks in next slide?



That's all for today

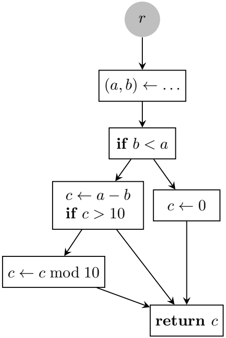


Answer: MEU BU BU



Control-flow graph (CFG)

Basic blocks sequence of consecutive statements
Edges control flow (jumps or fall-through)



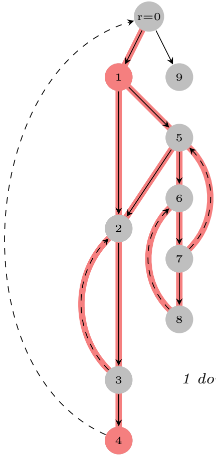
$(a, b) \leftarrow \dots$
if $b < a$ **then**
  $c \leftarrow a - b$
  if $c > 10$ **then**
    $c \leftarrow c \bmod 10$
  endif
else
  $c \leftarrow 0$
endif
return c

Back

Tree-shape. Dominance

Dominance relation

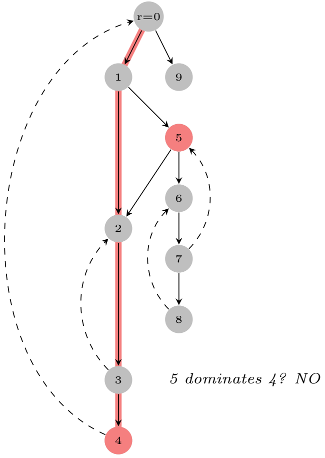
- a single entry node r.
- each node reachable from r.
- a dominates b if every path from r to b contains a.



Tree-shape. Dominance

Dominance relation

- a single entry node r.
- each node reachable from r.
- a dominates b if every path from r to b contains a.



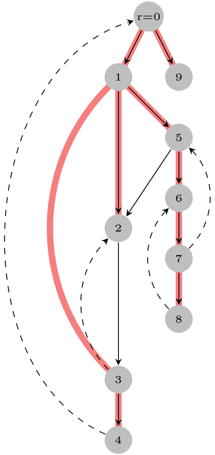
Tree-shape. Dominance

Dominance relation

- a single entry node r.
- each node reachable from r.
- a dominates b if every path from r to b contains a.

Properties

- The dominance relation induces a tree.



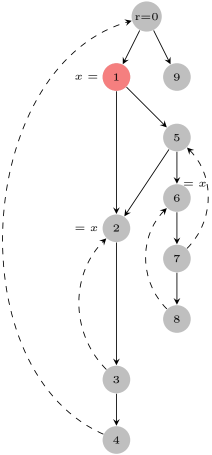
Static Single Assignment with dominance property

Strict code

Every path from *r* to a *use* traverses a definition

Strict SSA

- SSA: only *one* definition *textually* per variable
- Strict: the definition dominates all uses



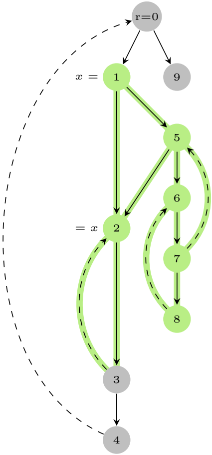
Liveness: sub-tree of a tree

The live-range of an SSA variable is

the set of program points

between the definition and a use

(*without going through the definition again*)

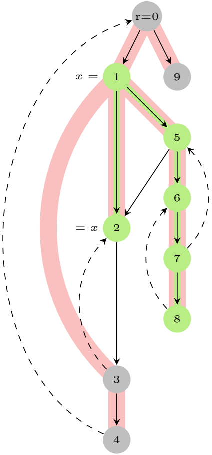


Liveness: sub-tree of a tree

The live-range of an SSA variable is

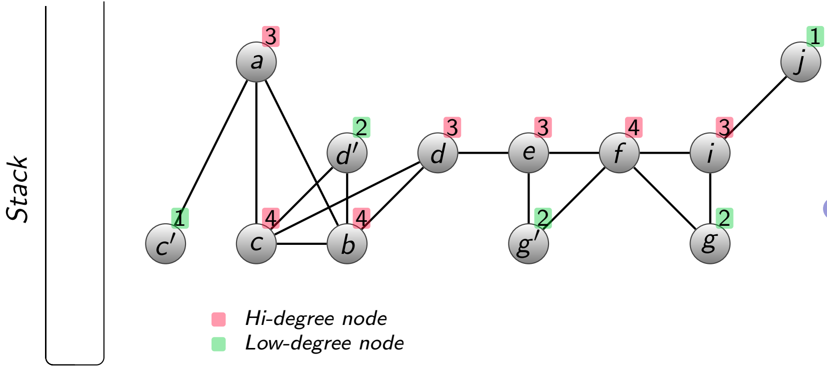
the set of program points
between the definition and a use
(*without going through the definition again*)

- the definition dominates the entire live-range
- the live-range is a sub-tree of the dominance-tree



k-colorability is hard to check, but greedy-k-colorability is easy.

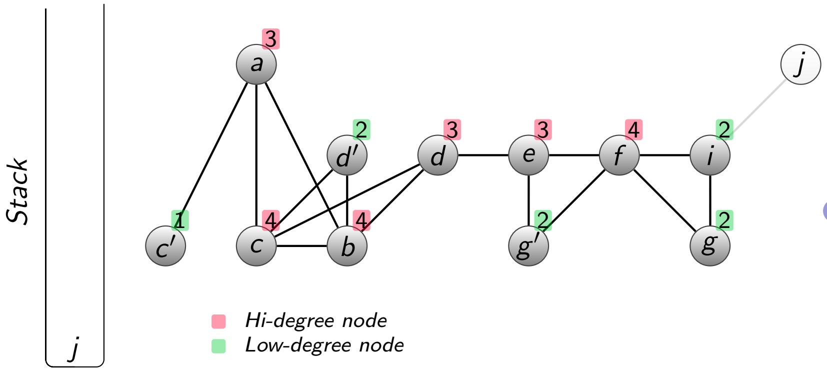
Check greedy-k-colorability: simplify nodes with $< k$ neighbors.



Back

k-colorability is hard to check, but greedy-k-colorability is easy.

Check greedy-k-colorability: simplify nodes with $< k$ neighbors.

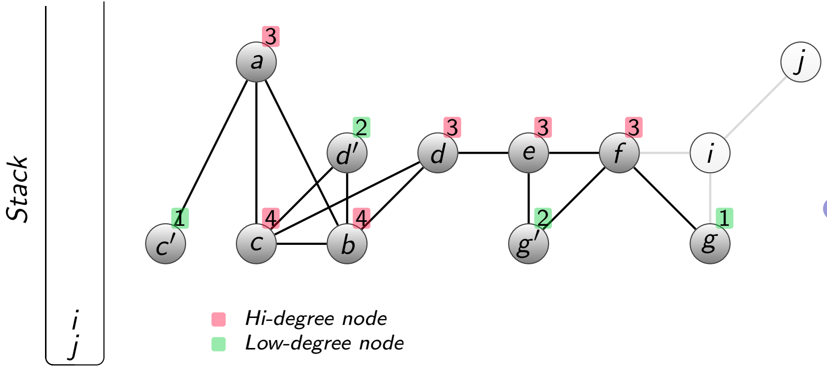


Back

Greedy-k-colorable graphs

k-colorability is hard to check, but greedy-k-colorability is easy.

Check greedy-k-colorability: simplify nodes with $< k$ neighbors.

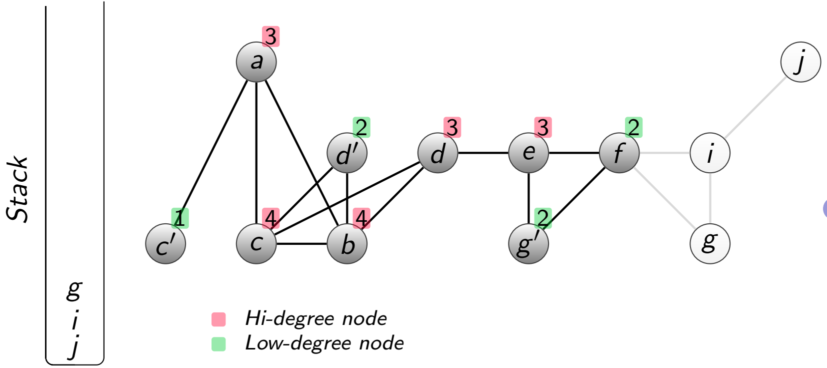


Back

Greedy-k-colorable graphs

k-colorability is hard to check, but greedy-k-colorability is easy.

Check greedy-k-colorability: simplify nodes with $< k$ neighbors.

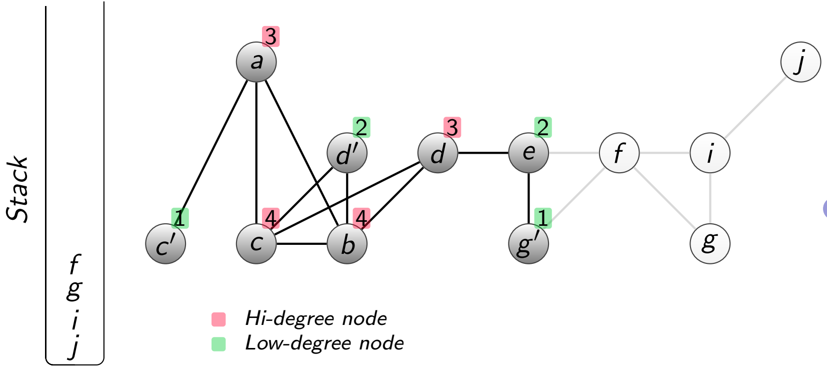


Back

Greedy-k-colorable graphs

k-colorability is hard to check, but greedy-k-colorability is easy.

Check greedy-k-colorability: simplify nodes with $< k$ neighbors.

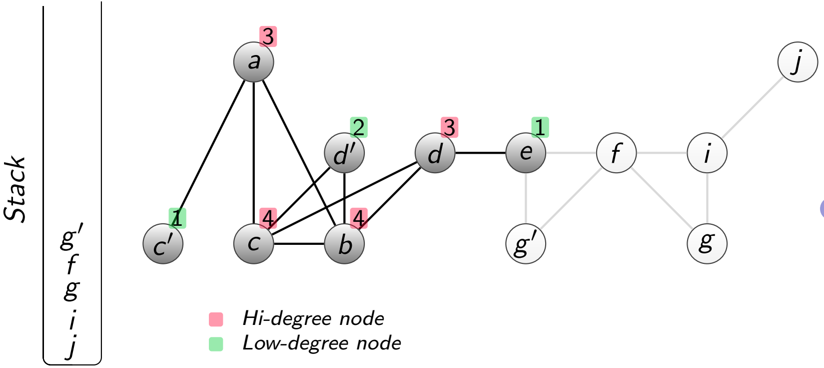


Back

Greedy-k-colorable graphs

k-colorability is hard to check, but greedy-k-colorability is easy.

Check greedy-k-colorability: simplify nodes with $< k$ neighbors.

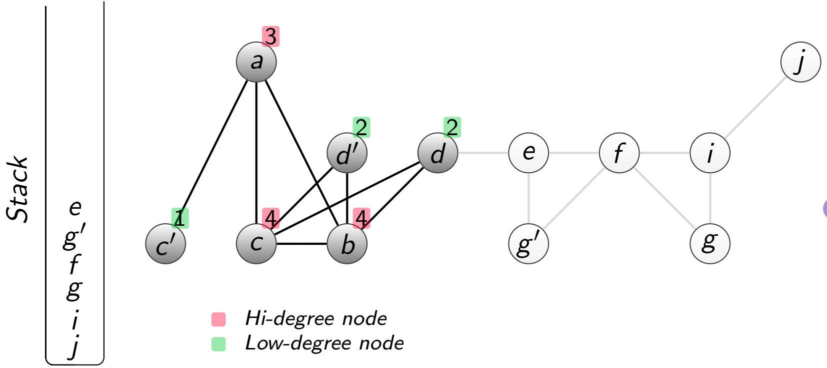


Back

Greedy-k-colorable graphs

k-colorability is hard to check, but greedy-k-colorability is easy.

Check greedy-k-colorability: simplify nodes with $< k$ neighbors.

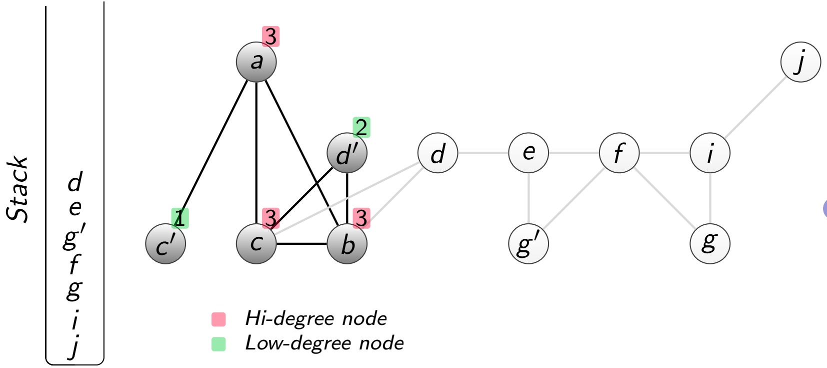


Back

Greedy-k-colorable graphs

k-colorability is hard to check, but greedy-k-colorability is easy.

Check greedy-k-colorability: simplify nodes with $< k$ neighbors.

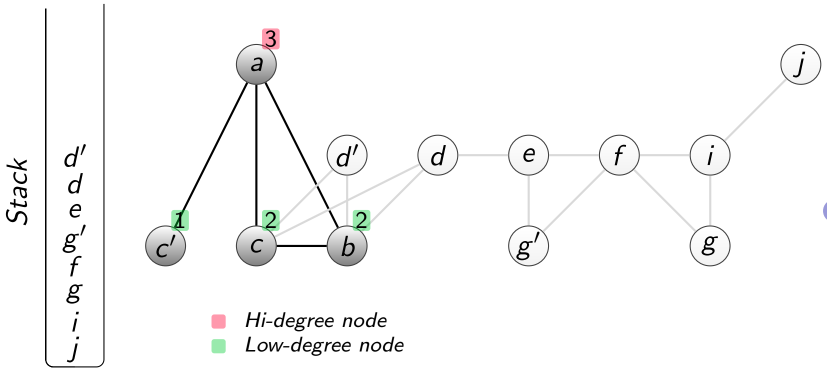


Back

Greedy-k-colorable graphs

k-colorability is hard to check, but greedy-k-colorability is easy.

Check greedy-k-colorability: simplify nodes with $< k$ neighbors.

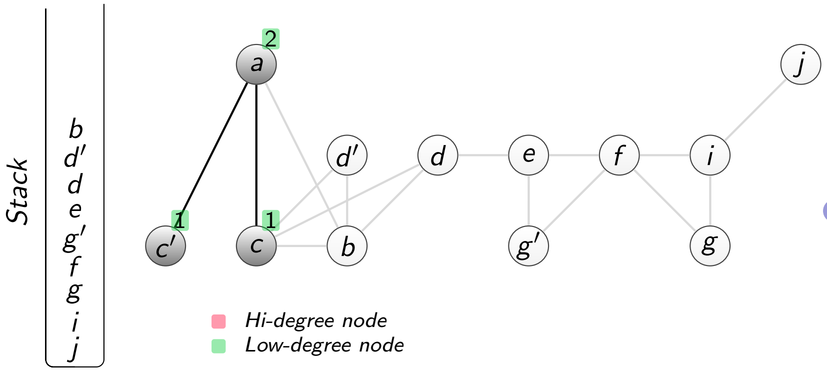


Back

Greedy-k-colorable graphs

k-colorability is hard to check, but greedy-k-colorability is easy.

Check greedy-k-colorability: simplify nodes with $< k$ neighbors.

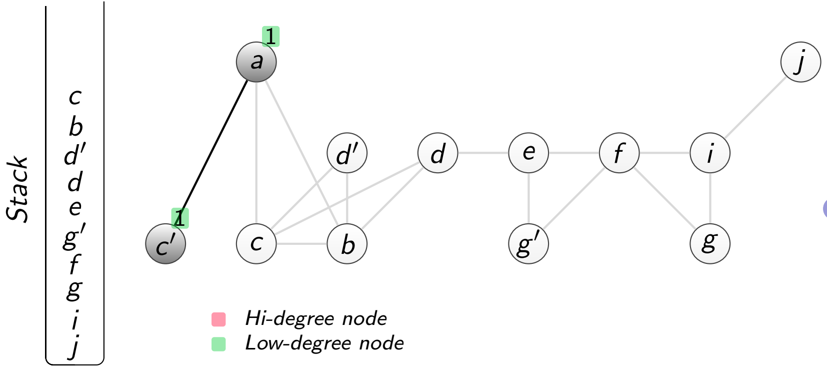


Back

Greedy-k-colorable graphs

k-colorability is hard to check, but greedy-k-colorability is easy.

Check greedy-k-colorability: simplify nodes with $< k$ neighbors.

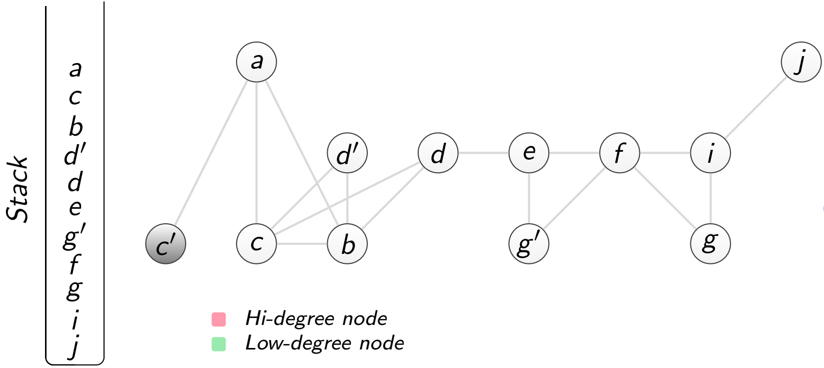


Back

Greedy-k-colorable graphs

k-colorability is hard to check, but greedy-k-colorability is easy.

Check greedy-k-colorability: simplify nodes with $< k$ neighbors.

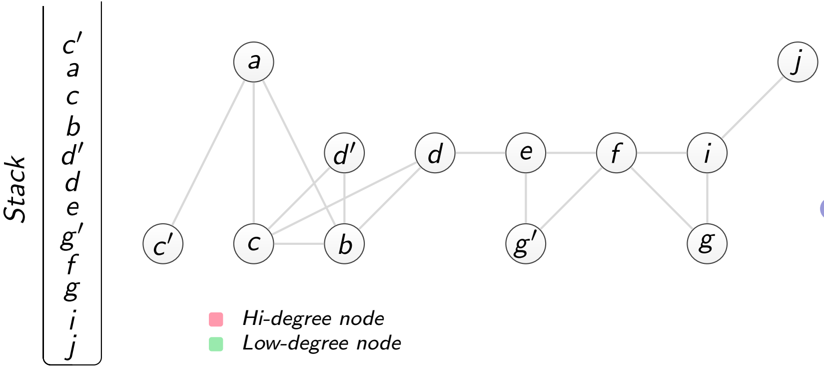


Back

Greedy-k-colorable graphs

k-colorability is hard to check, but greedy-k-colorability is easy.

Check greedy-k-colorability: simplify nodes with $< k$ neighbors.

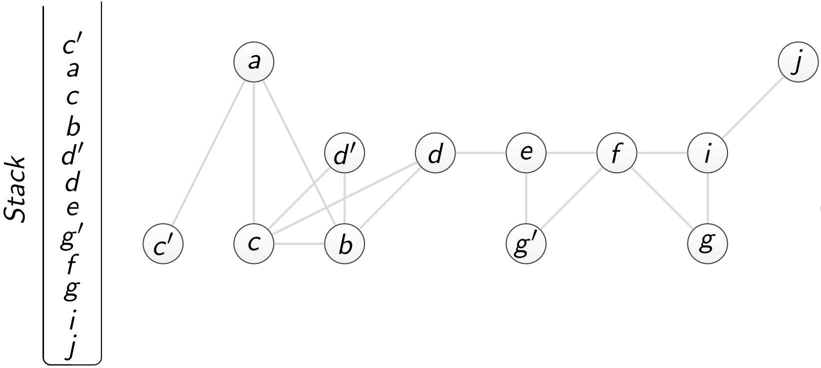


Back

Greedy-k-colorable graphs

k-colorability is hard to check, but greedy-k-colorability is easy.

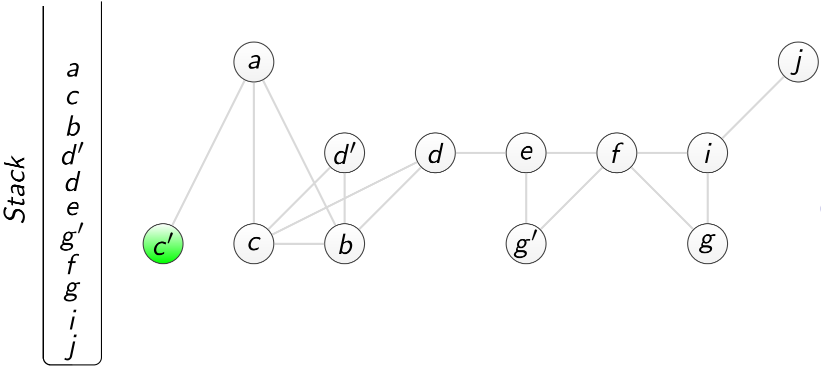
Check greedy-k-colorability: simplify nodes with $< k$ neighbors.



Back

k-colorability is hard to check, but greedy-k-colorability is easy.

Check greedy-k-colorability: simplify nodes with $< k$ neighbors.

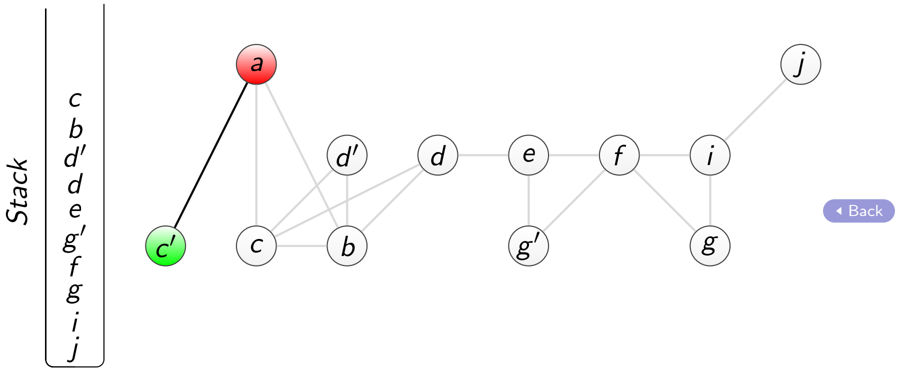


Back

Greedy-k-colorable graphs

k-colorability is hard to check, but greedy-k-colorability is easy.

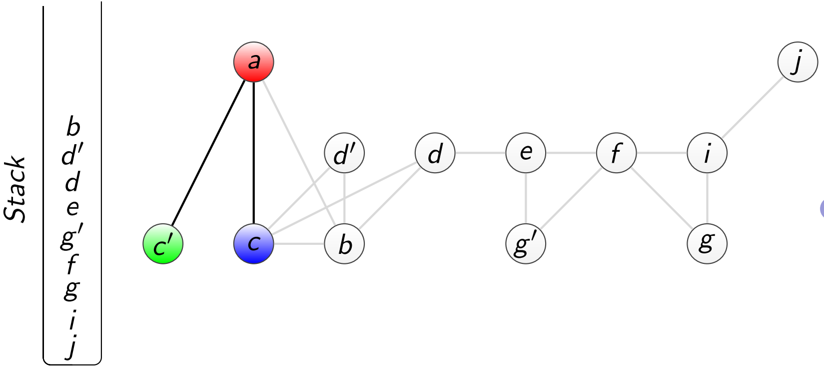
Check greedy-k-colorability: simplify nodes with $< k$ neighbors.



Greedy-k-colorable graphs

k-colorability is hard to check, but greedy-k-colorability is easy.

Check greedy-k-colorability: simplify nodes with $< k$ neighbors.

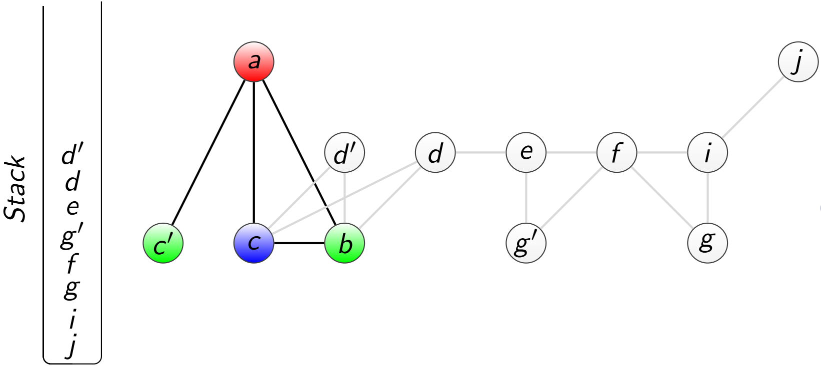


Back

Greedy-k-colorable graphs

k-colorability is hard to check, but greedy-k-colorability is easy.

Check greedy-k-colorability: simplify nodes with $< k$ neighbors.

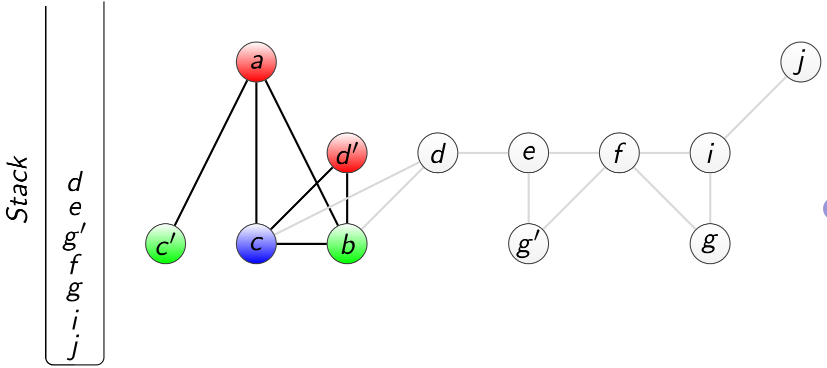


Back

Greedy-k-colorable graphs

k-colorability is hard to check, but greedy-k-colorability is easy.

Check greedy-k-colorability: simplify nodes with $< k$ neighbors.

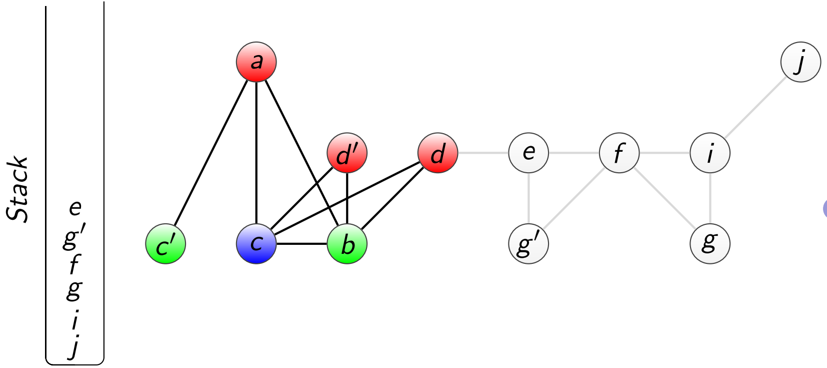


Back

Greedy-k-colorable graphs

k-colorability is hard to check, but greedy-k-colorability is easy.

Check greedy-k-colorability: simplify nodes with $< k$ neighbors.

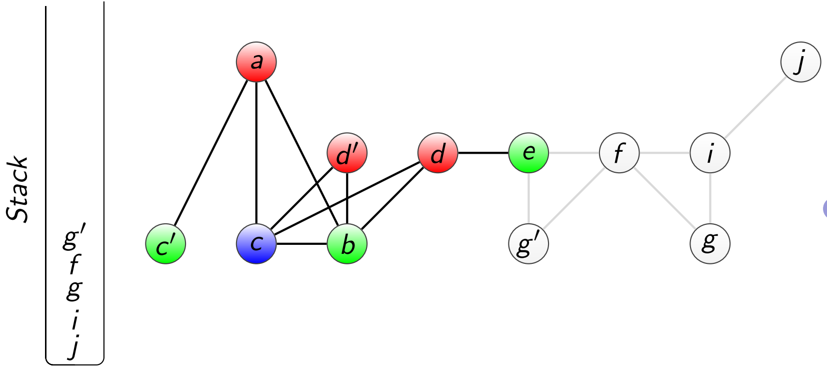


Back

Greedy-k-colorable graphs

k-colorability is hard to check, but greedy-k-colorability is easy.

Check greedy-k-colorability: simplify nodes with $< k$ neighbors.

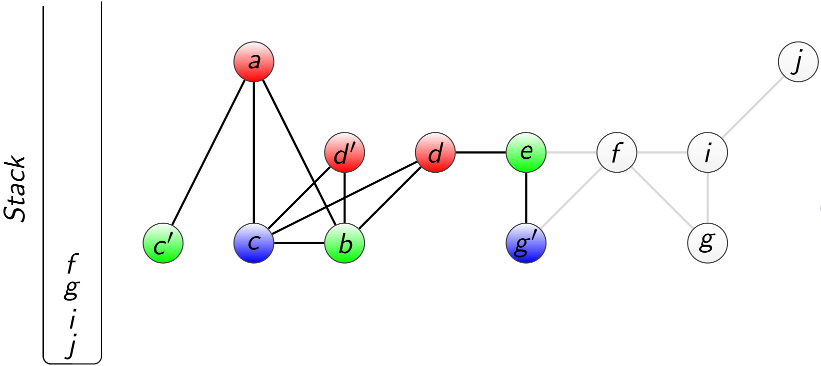


Back

Greedy-k-colorable graphs

k-colorability is hard to check, but greedy-k-colorability is easy.

Check greedy-k-colorability: simplify nodes with $< k$ neighbors.

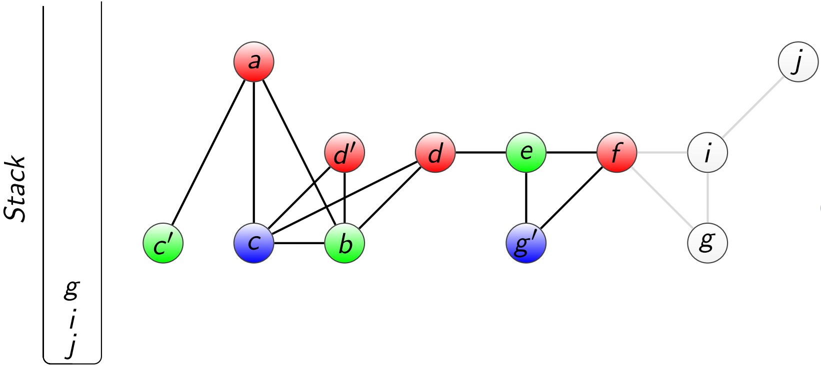


Back

Greedy-k-colorable graphs

k-colorability is hard to check, but greedy-k-colorability is easy.

Check greedy-k-colorability: simplify nodes with $< k$ neighbors.

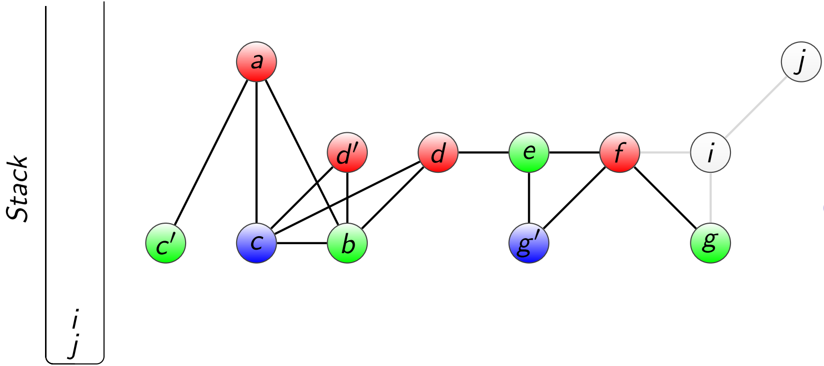


Back

Greedy-k-colorable graphs

k-colorability is hard to check, but greedy-k-colorability is easy.

Check greedy-k-colorability: simplify nodes with $< k$ neighbors.

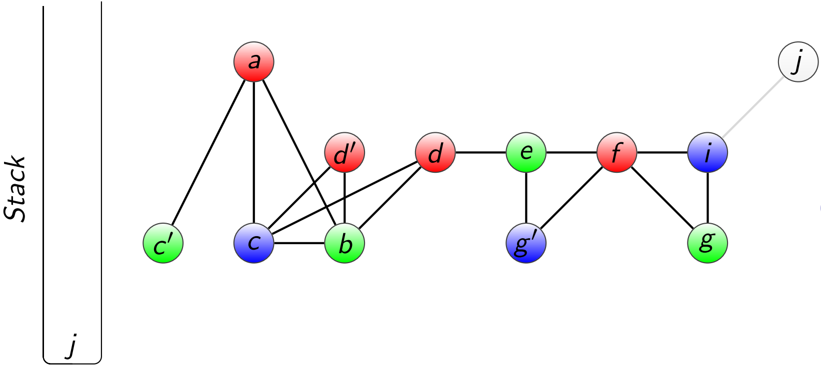


Back

Greedy-k-colorable graphs

k-colorability is hard to check, but greedy-k-colorability is easy.

Check greedy-k-colorability: simplify nodes with $< k$ neighbors.

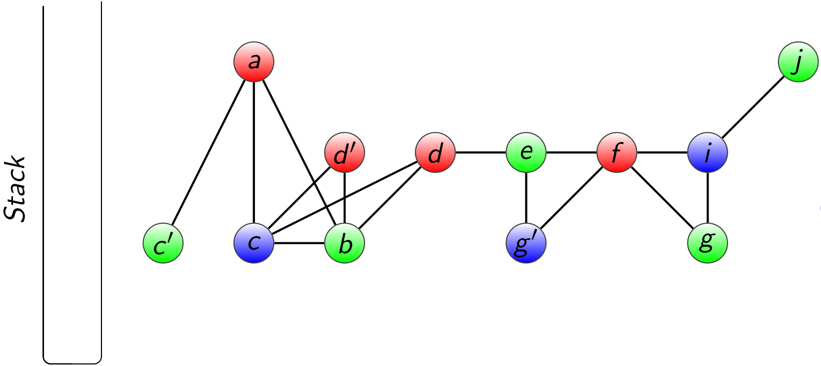


Back

Greedy-k-colorable graphs

k-colorability is hard to check, but greedy-k-colorability is easy.

Check greedy-k-colorability: simplify nodes with $< k$ neighbors.

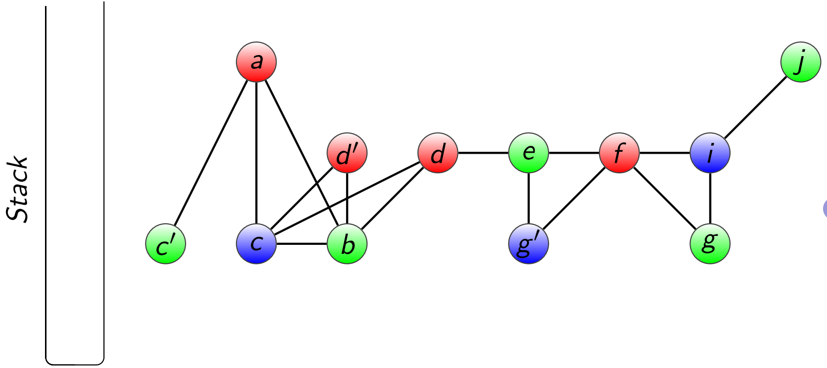


Back

Greedy-k-colorable graphs

k-colorability is hard to check, but greedy-k-colorability is easy.

Check greedy-k-colorability: simplify nodes with $< k$ neighbors.



Back