Revisiting register allocation: Why and how?

The interplay of SSA, splitting, coalescing, and spilling.

Florent Bouchez, Alain Darte, and Fabrice Rastello

Joint work with colleagues at STMicroelectronics:

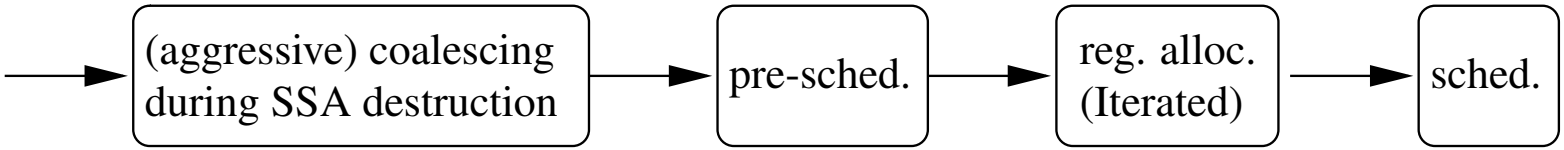
Benoit Dupont de Dinechin, Christophe Guillon, François de Ferrière.

Cours de Master
ENS-Lyon

19 Novembre 2007

Motivation

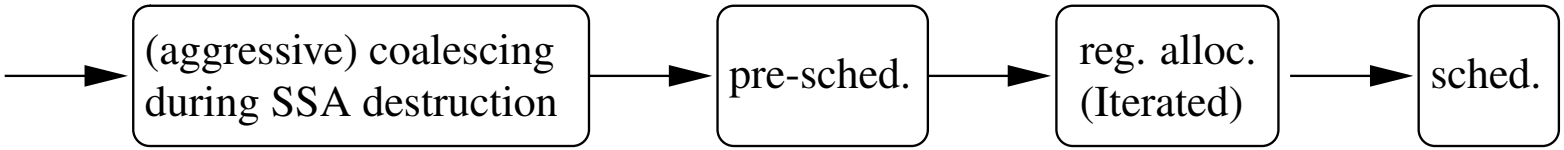
Starting point: past experience on out-of-SSA conversion.



- Too many moves if naive conversion.
- But some dramatic slow-downs if coalescing is too aggressive: ⇛ consider SSA + register allocation;
- SSA interference graphs are chordal, thus easy to color.

Motivation

Starting point: past experience on out-of-SSA conversion.



- Too many moves if naive conversion.
- But some dramatic slow-downs if coalescing is too aggressive: ⇛ consider SSA + register allocation;
- SSA interference graphs are chordal, thus easy to color.

New goals:

- Clean up register allocation theory.
- Find aggressive techniques, possibly linked to SSA.
- Improve just-in-time algorithms, both in terms of liveness analysis and representation, and register allocation.

Outline

Outline

What is register allocation?

Assign variables of a program to physical registers

- unlimited number of variables to place in:
  - a pool of limited resources (registers).
  - a pool of unlimited resources (memory).

What is register allocation?

Assign variables of a program to physical registers

- unlimited number of variables to place in:
  - a pool of limited resources (registers).
  - a pool of unlimited resources (memory).
- some architectural subtleties:
  - specific registers (sp, fp, r0, ...);
  - affinities (auto-inc, ...), register pairing (64 bits ops, ...);
  - calling conventions, distributed register banks, variable sizes, register-only instructions, ...

What is register allocation?

Assign variables of a program to physical registers

- unlimited number of variables to place in:
  - a pool of limited resources (registers).
  - a pool of unlimited resources (memory).
- some architectural subtleties:
  - specific registers (sp, fp, r0, ...);
  - affinities (auto-inc, ...), register pairing (64 bits ops, ...);
  - calling conventions, distributed register banks, variable sizes, register-only instructions, ...

Rule of the games

- fixed instruction schedule;
- insert LOADS and STORES: spill;
- add register-to-register MOVES: split;
- delete MOVES: coalesce.

Register allocation: what do we learn at school?

"NP-complete because graph coloring is NP-complete."

- variable $\Leftrightarrow$ vertex;
- interferences between variables $\Leftrightarrow$ edge;
- variable assignment $\Leftrightarrow$ graph coloring.

$\Rightarrow$ heuristics for everything: assignment (coloring), spilling (load/store insertion), and coalescing (move removal).

Register allocation: what do we learn at school?

"NP-complete because graph coloring is NP-complete."

- variable $\Leftrightarrow$ vertex;
- interferences between variables $\Leftrightarrow$ edge;
- variable assignment $\Leftrightarrow$ graph coloring.

$\Rrightarrow$ heuristics for everything: assignment (coloring), spilling (load/store insertion), and coalescing (move removal).

"But it is polynomial for a basic block." (after renaming)

- live range of a variable (unique def.) = interval;
- interference graph = interval graph.

Register allocation: what do we learn at school?

"NP-complete because graph coloring is NP-complete."

- variable $\Leftrightarrow$ vertex;
- interferences between variables $\Leftrightarrow$ edge;
- variable assignment $\Leftrightarrow$ graph coloring.

$\Rrightarrow$ heuristics for everything: assignment (coloring), spilling (load/store insertion), and coalescing (move removal).

"But it is polynomial for a basic block." (after renaming)

- live range of a variable (unique def.) = interval;
- interference graph = interval graph.

Many different register allocation schemes but, since Chaitin's original proof, no fundamental study, except Liberatore's work.

Global register allocation with graph coloring:

Chaitin et al. (1981), Briggs-Cooper-Torczon (1994), Appel-George (2001), ...

Given: k registers, interference graph, affinities (for coalescing).

Simplify remove a non-move-related vertex with degree $< k$;

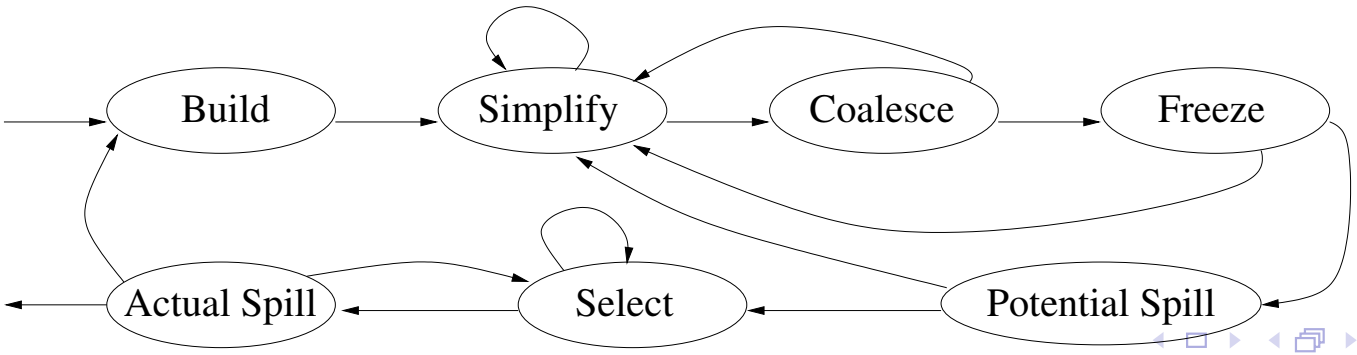
Coalesce merge 2 move-related vertices (e.g., conservatively);

Freeze give up about some moves;

Potential spill remove a vertex and push it on a stack;

Select pop a vertex and assign a color;

Actual spill if no color is found, really insert load/store.



See powerpoint slides.

So, what is confusing?

Local register allocation is polynomial?

- Yes for deciding if k registers are enough, by renaming variables to get unique definitions.
- But what if more registers are needed, i.e., if spilling is necessary? ☛ Liberatore (1999) + LCTES'07.

So, what is confusing?

Local register allocation is polynomial?

- Yes for deciding if k registers are enough, by renaming variables to get unique definitions.
- But what if more registers are needed, i.e., if spilling is necessary? ☛ Liberatore (1999) + LCTES'07.

Global register allocation is NP-complete?

- But the proof does not consider register-to-register moves (splitting)! So is it really hard to decide if k registers are enough? ☛ WDDD'06 + LCPC'06.

So, what is confusing?

Local register allocation is polynomial?

- Yes for deciding if k registers are enough, by renaming variables to get unique definitions.
- But what if more registers are needed, i.e., if spilling is necessary? ☛ Liberatore (1999) + LCTES'07.

Global register allocation is NP-complete?

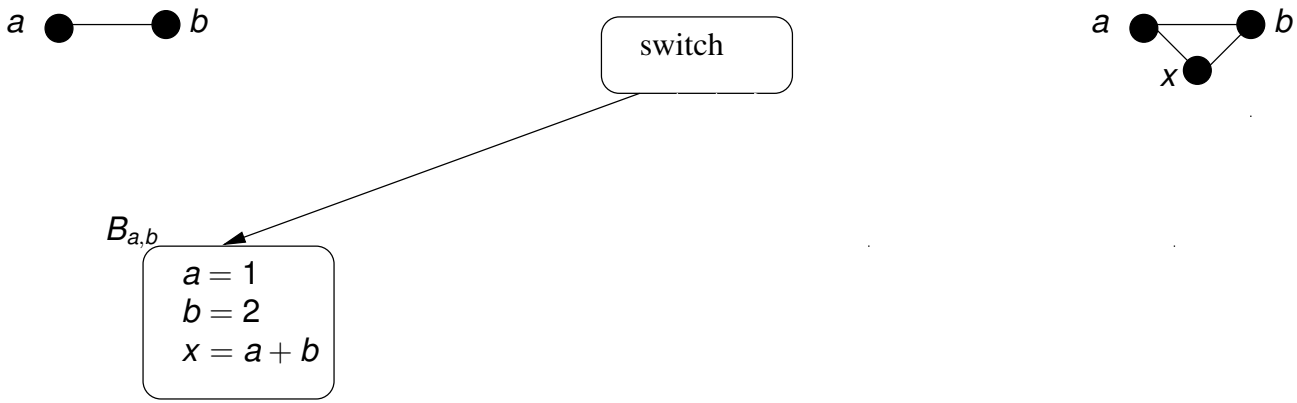
- But the proof does not consider register-to-register moves (splitting)! So is it really hard to decide if k registers are enough? ☛ WDDD'06 + LCPC'06.

What about coalescing, out-of-SSA conversion?

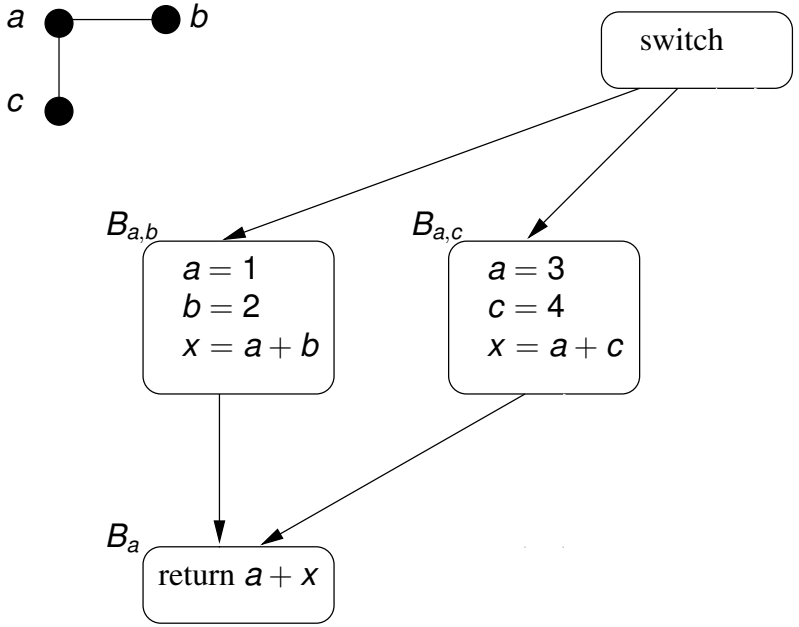
☛ Shreedar (1999), Hack (2005) + CGO'07.

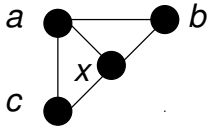
Outline

Interpretation of original proof

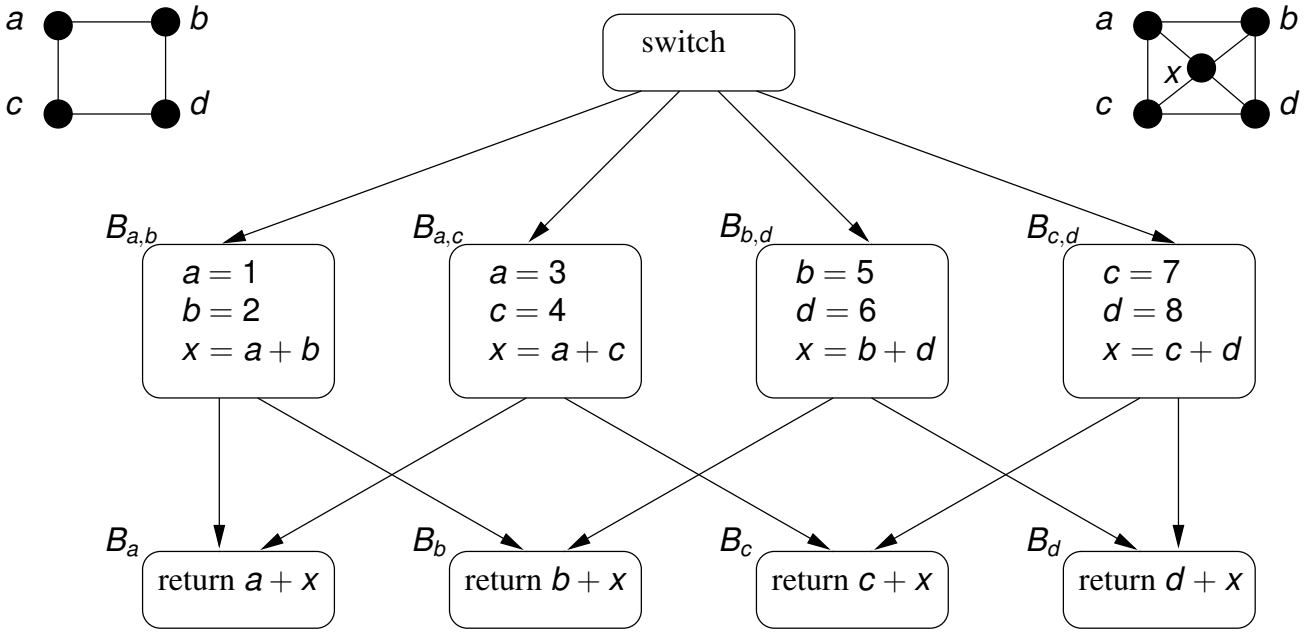


Interpretation of original proof





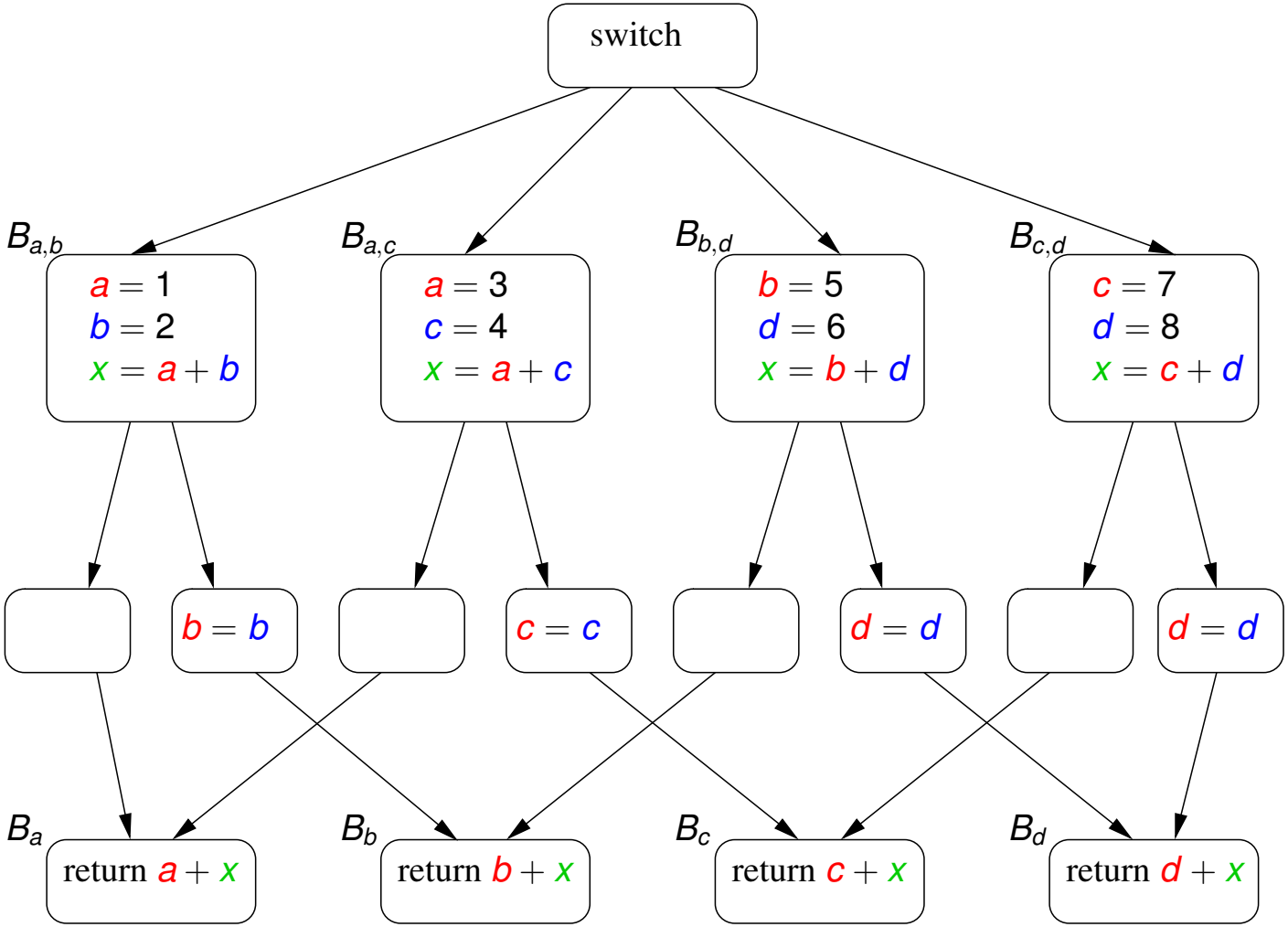
Interpretation of original proof



Chaitin et al. ⇛ NP-complete if each variable is assigned to a unique register.

Extension ⇛ if live-range splitting is allowed, remains NP-complete because of critical edges.

But proves nothing if blocks & moves can be inserted!



Maxlive: max. number of variables simultaneously live

Assume swaps, a strict program, edge splitting allowed

1. One needs Maxlive $\leq k$, so spill to get Maxlive $\leq k$.
2. Split critical edges (= add basic blocks).
3. Color each program point independently with at most Maxlive colors.
4. Use permutations to match colors (thanks to swaps).

This gives a correct assignment...

Maxlive: max. number of variables simultaneously live

Assume swaps, a strict program, edge splitting allowed

1. One needs Maxlive $\leq k$, so spill to get Maxlive $\leq k$.
2. Split critical edges (= add basic blocks).
3. Color each program point independently with at most Maxlive colors.
4. Use permutations to match colors (thanks to swaps).

This gives a correct assignment. . . but expensive in moves, even after conservative register coalescing (Appel-George).

Maxlive: max. number of variables simultaneously live

Assume swaps, a strict program, edge splitting allowed

1. One needs Maxlive $\leq k$, so spill to get Maxlive $\leq k$.
2. Split critical edges (= add basic blocks).
3. Color each program point independently with at most Maxlive colors.
4. Use permutations to match colors (thanks to swaps).

This gives a correct assignment. . . but expensive in moves, even after conservative register coalescing (Appel-George).

More promising approaches

- Basic block coloring (interval graph);
- SSA-like coloring = subtrees of a tree (chordal graph);
- Guided live-range/edge splitting + permutation motion.

Pereira & Palsberg question (FOSSACS 2006)

After results by Brisk et al., Hack et al., Bouchez et al. on SSA and register allocation, Pereira and Palsberg wondered:

> “ Can we do polynomial-time register allocation by first transforming the program to SSA form, then doing linear-time register allocation for the SSA form, and finally doing SSA elimination while maintaining the mapping from temporaries to registers? ”

Pereira & Palsberg question (FOSSACS 2006)

After results by Brisk et al., Hack et al., Bouchez et al. on SSA and register allocation, Pereira and Palsberg wondered:

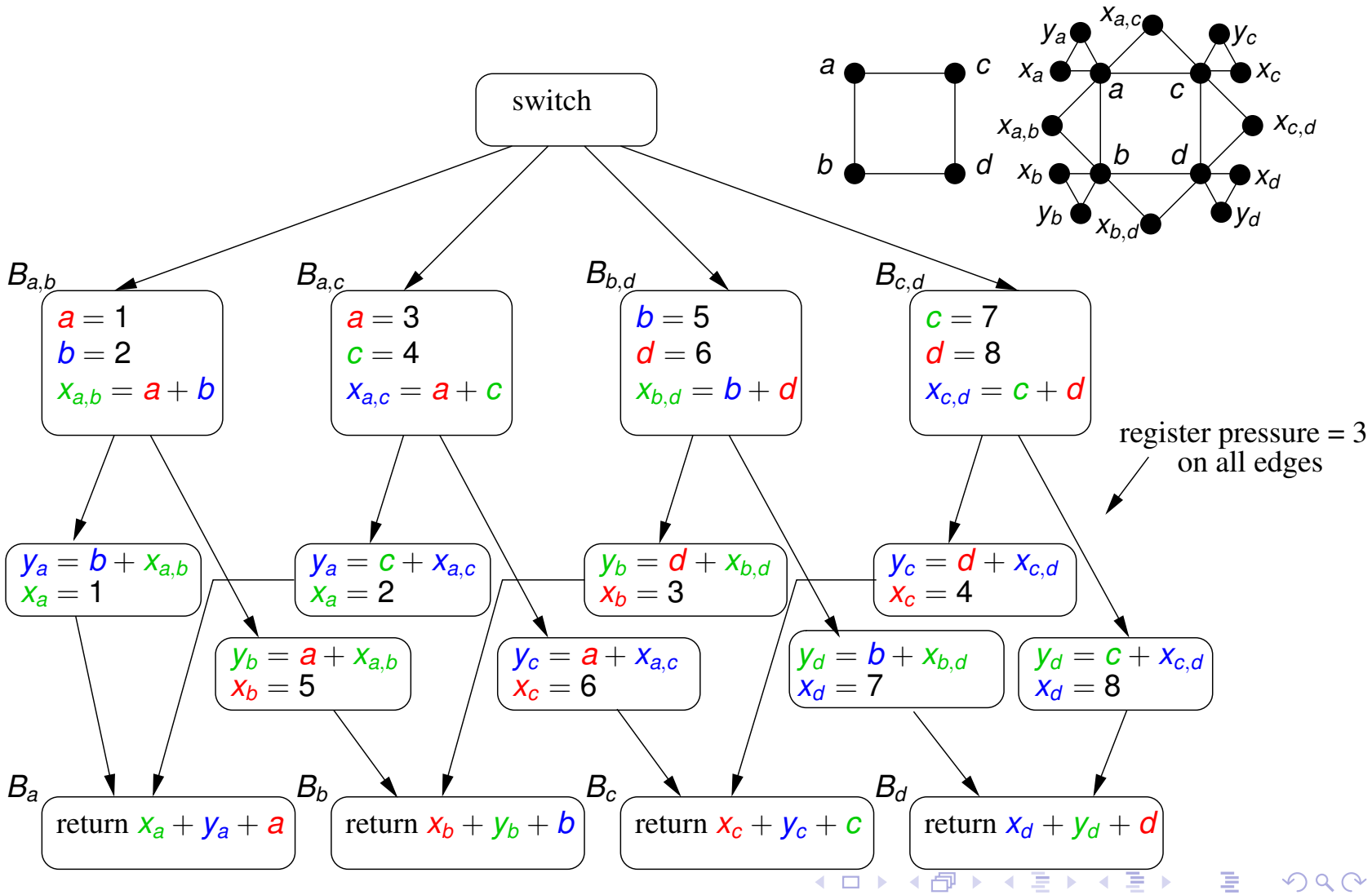
> ❝ Can we do polynomial-time register allocation by first transforming the program to SSA form, then doing linear-time register allocation for the SSA form, and finally doing SSA elimination while maintaining the mapping from temporaries to registers? ❞

☛ They show it is NP-complete if swaps are not available.

- Reduce from k-coloring circular-arc graph.
- Make sure Live $= k$ on the back edge (where SSA will split) so that a non-trivial permutation is impossible.

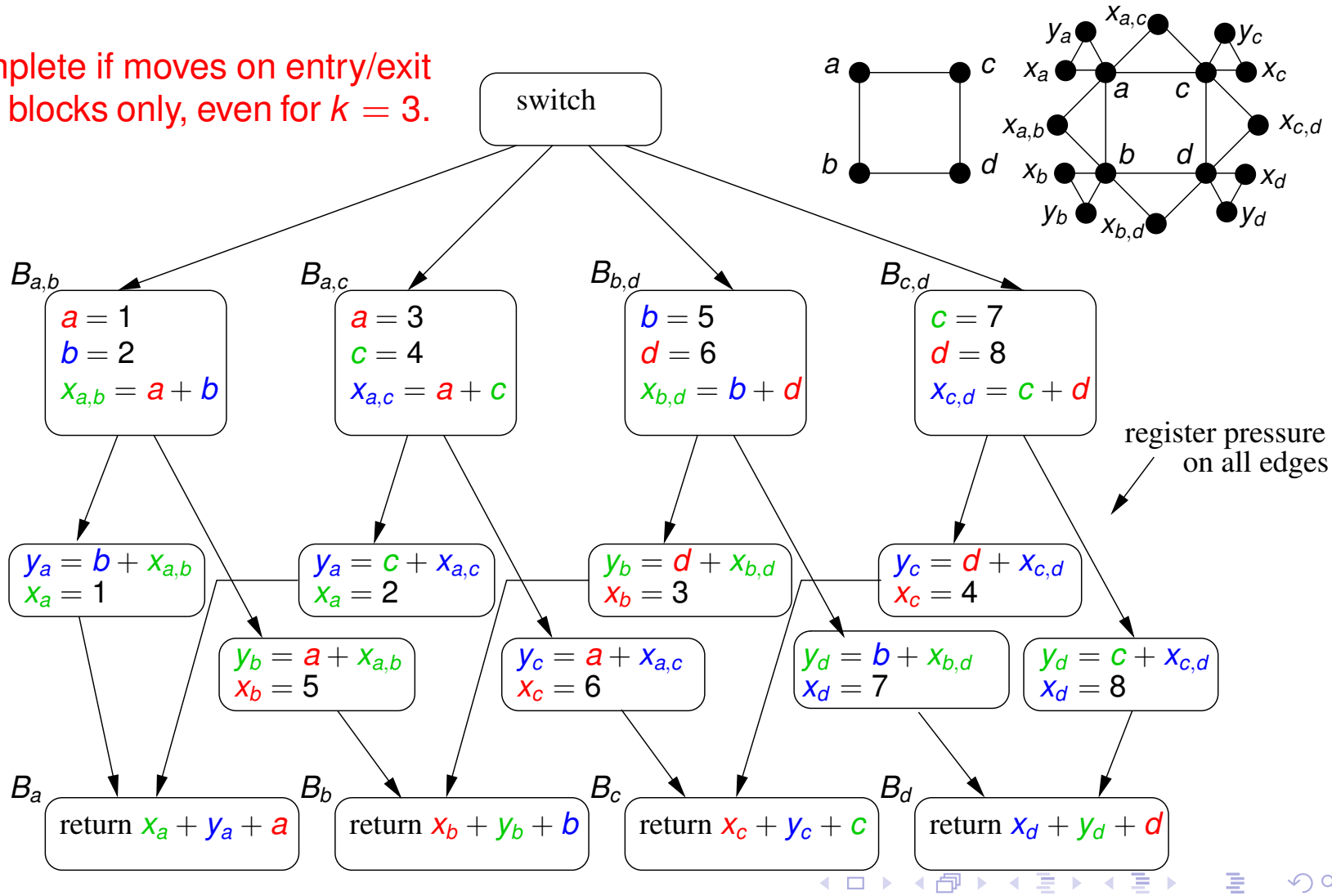
Note: polynomial for a fixed k. (See Garey, Johnson, Miller, Papadimitriou.)

Chaitin et al's variant if swaps are not available



Chaitin et al's variant if swaps are not available

NP-complete if moves on entry/exit of basic blocks only, even for $k = 3$.



If swaps are not available, what can we conclude?

NP-complete if moves on entry/exit of basic blocks only.

If swaps are not available, what can we conclude?

NP-complete if moves on entry/exit of basic blocks only.

☛ But why not inserting moves in the middle of a block?

If swaps are not available, what can we conclude?

NP-complete if moves on entry/exit of basic blocks only.

☛ But why not inserting moves in the middle of a block?

NP-complete if instructions can define two variables simultaneously.

Replace each pair of definitions such as $y_a = b + x_{a,b}$ and $x_a = 1$ by one instruction $(x_a, y_a) = f(b, x_{a,b})$.

If swaps are not available, what can we conclude?

NP-complete if moves on entry/exit of basic blocks only.

☛ But why not inserting moves in the middle of a block?

NP-complete if instructions can define two variables simultaneously.

Replace each pair of definitions such as $y_a = b + x_{a,b}$ and $x_a = 1$ by one instruction $(x_a, y_a) = f(b, x_{a,b})$.

☛ But, often, either swaps are available or such instructions have low register pressure (ex: function call, 64 bits load).

If swaps are not available, what can we conclude?

NP-complete if moves on entry/exit of basic blocks only.

☛ But why not inserting moves in the middle of a block?

NP-complete if instructions can define two variables simultaneously.

Replace each pair of definitions such as $y_a = b + x_{a,b}$ and $x_a = 1$ by one instruction $(x_a, y_a) = f(b, x_{a,b})$.

☛ But, often, either swaps are available or such instructions have low register pressure (ex: function call, 64 bits load).

Polynomial if instructions have only one result! Greedy traversal along the control-flow graph where Live $= k$.

If swaps are not available, what can we conclude?

NP-complete if moves on entry/exit of basic blocks only.

☛ But why not inserting moves in the middle of a block?

NP-complete if instructions can define two variables simultaneously.

Replace each pair of definitions such as $y_a = b + x_{a,b}$ and $x_a = 1$ by one instruction $(x_a, y_a) = f(b, x_{a,b})$.

☛ But, often, either swaps are available or such instructions have low register pressure (ex: function call, 64 bits load).

Polynomial if instructions have only one result! Greedy traversal along the control-flow graph where Live $= k$.

So, NP-completeness did not disappear, it was simply not there! The proof of Chaitin et al. does not say anything about register allocation with live-range splitting and critical edge splitting.

On the complexity of register allocation

☛ If moves are more suitable than loads and stores, it is in general easy to decide if some spilling is necessary or not.

Spill test Chaitin (degree $\geq k$) ➙ Briggs (potential spill) ➙ Appel-George (iterated) ➙ Biased coloring ➙ Optimal test

But register allocation remains difficult

- When critical edges cannot be split or code is not strict. But compilers often go through strict SSA and almost always split critical edges. . .
- Because optimal spilling is (almost) always hard.
- Because optimal coalescing is, in most cases, NP-complete. But improvements are possible.

Outline

Key concepts and properties

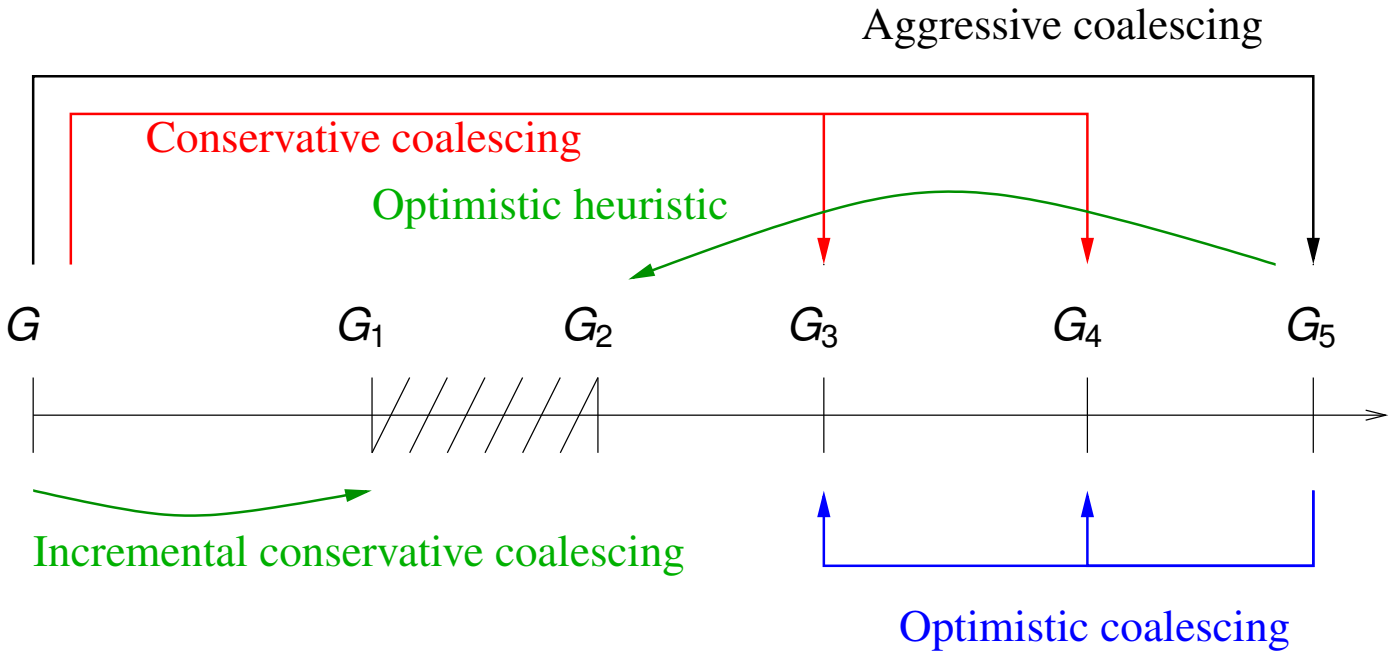
k-colorable NP-complete to decide;

Greedy-k-colorable If all vertices can be "simplified" ($d^o \leq k$);

k-chordal intersection of subtrees of a tree, # clique $\leq k$.

- Interference graph of strict SSA programs (with parallel copies) are chordal;
- k-chordal graphs are greedy-k-colorable;
- Aggressive coalescing, unlike conservative coalescing, does not preserve k-colorability.

Links between different approaches



G: Initial graph, k-colorable or greedy-k-colorable
G_1: obtained by incremental conservative coalescing, greedy-k-colorable
G_2: obtained by optimistic de-coalescing, greedy-k-colorable
G_3: optimally coalesced greedy-k-colorable
G_4: optimally coalesced k-colorable
G_5: obtained by aggressive coalescing

Main complexity results

G interference graph, G_f graph after coalescing.

Aggressive coalescing NP-complete, even if G is chordal or greedy-3-colorable.

Conservative coalescing NP-complete even if G is greedy-2-colorable, one requires G_f to be chordal or greedy-3-colorable, and only affinities can be merged.

Incremental conservative coalescing (Briggs, George)
NP-complete if G is arbitrary. Polynomial if G is chordal.
Open if G is greedy-k-colorable.

Optimistic coalescing (Park & Moon) = conservative de-coalescing
NP-complete even if G is chordal and $k = 4$.

And experiments show that there is space for improvements!

Complexity of spill everywhere

Spill everywhere *without holes*

	$\Omega \leq k$	$\Omega \leq r$	$\Omega \leftarrow \Omega - 1$
Interval graph	$\mathbb{P}$	$\mathbb{P}$ Belady/ILP	$\mathbb{P}$ dyn. prog.
Chordal graph	$\mathbb{P}$ dyn. prog.	$\mathbb{NP}$	$\mathbb{NP}$ 3-exact cover

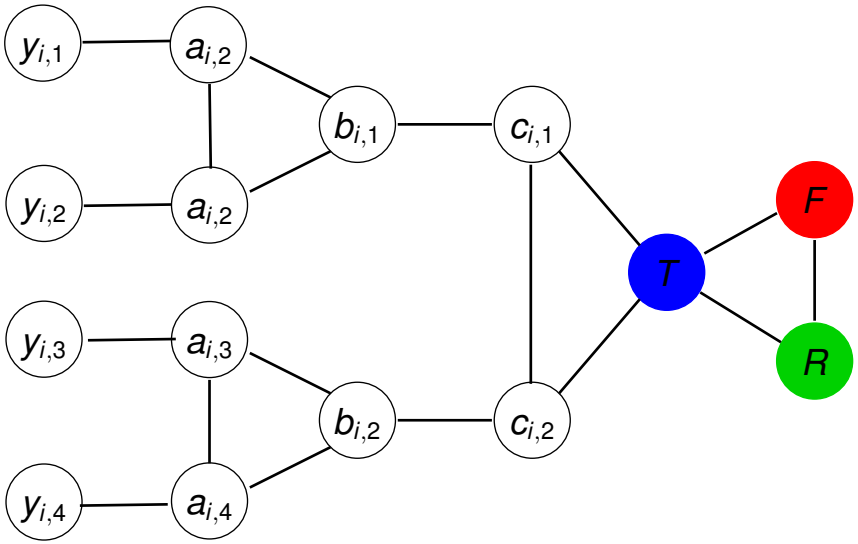
Spill everywhere *with holes on a basic block*

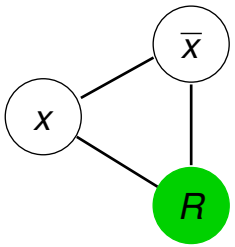
	$\Omega \leq k$	$\Omega \leq r$	$\Omega \leftarrow \Omega - 1$
h **fixed**	$\mathbb{P}$	$\mathbb{NP}$ stable set [a]	$\mathbb{P}$ dyn. prog.
h **not bounded**	$\mathbb{P}$ dyn. prog.	$\mathbb{NP}$	$\mathbb{NP}$ set cover

[a] Open for $h = 1$

Incremental coalescing for a general graph

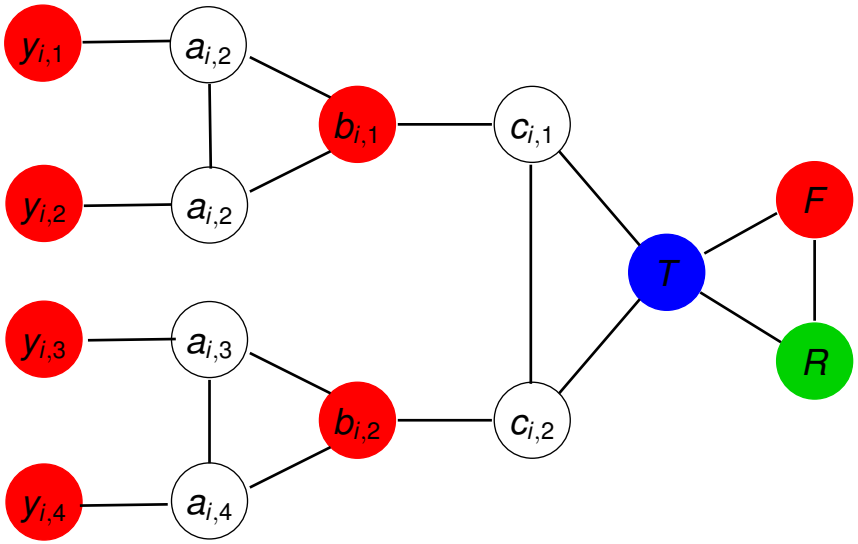
Construction for one clause:

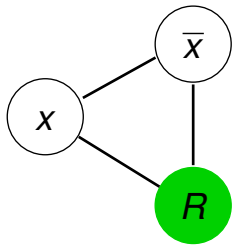




Incremental coalescing for a general graph

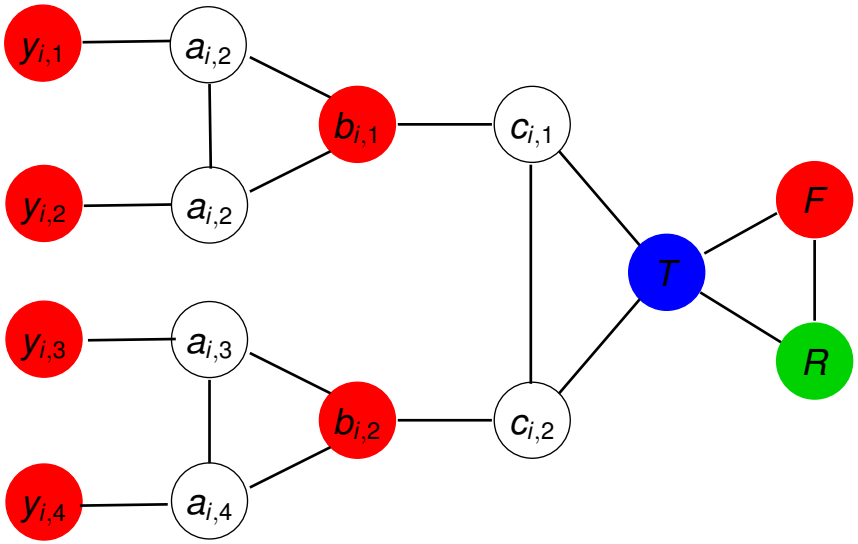
Colorable iff one of the literals is true:

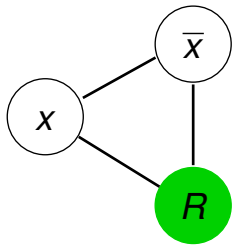




Incremental coalescing for a general graph

Colorable iff one of the literals is true:

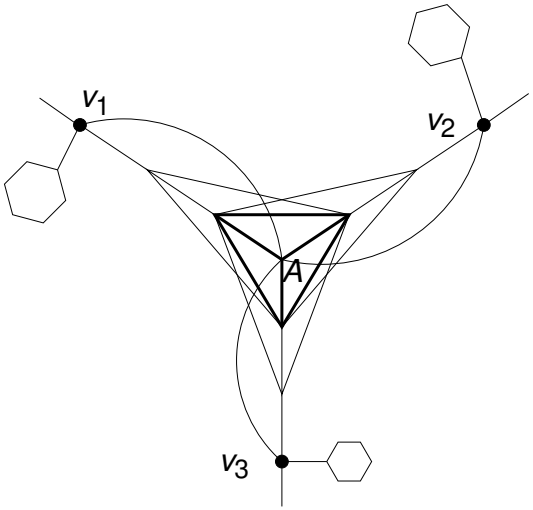




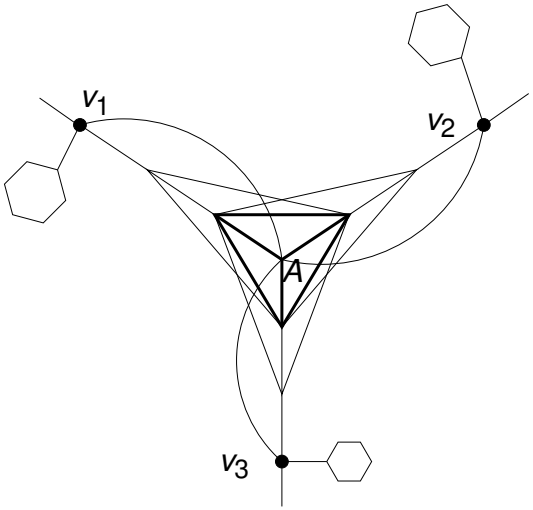
Trick

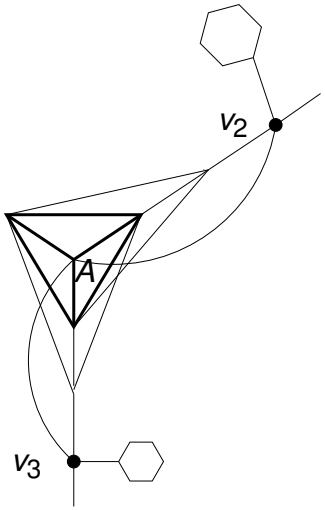
- Replace each $y_{i,1} \vee y_{i,2} \vee y_{i,3}$ by $y_{i,1} \vee y_{i,2} \vee y_{i,3} \vee x_0$.
- 3SAT formula $\Rightarrow$ 4SAT formula.
- Coalescing x_0 with FALSE is possible iff 3SAT is TRUE.

Optimistic coalescing: from vertex cover, degree ≤ 3

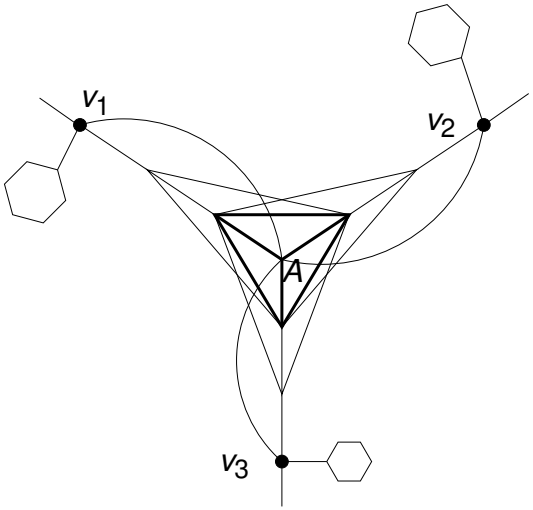


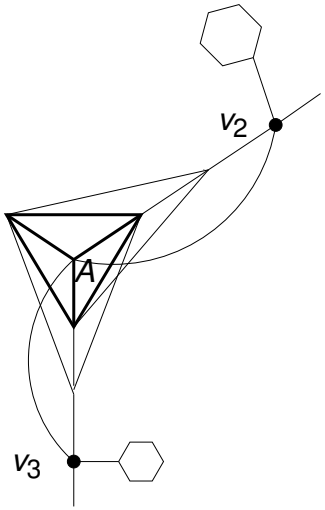
Optimistic coalescing: from vertex cover, degree ≤ 3

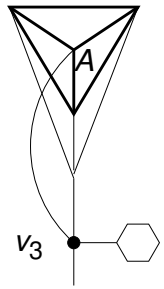




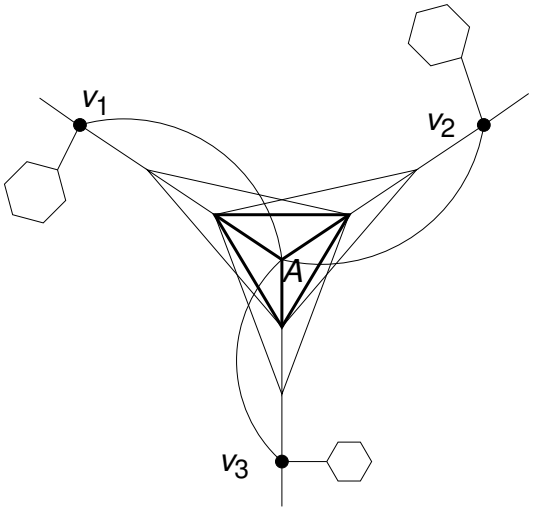
Optimistic coalescing: from vertex cover, degree ≤ 3

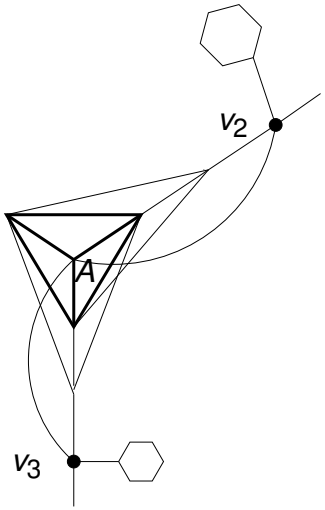


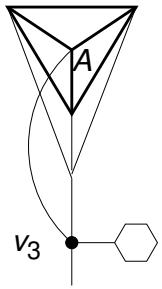




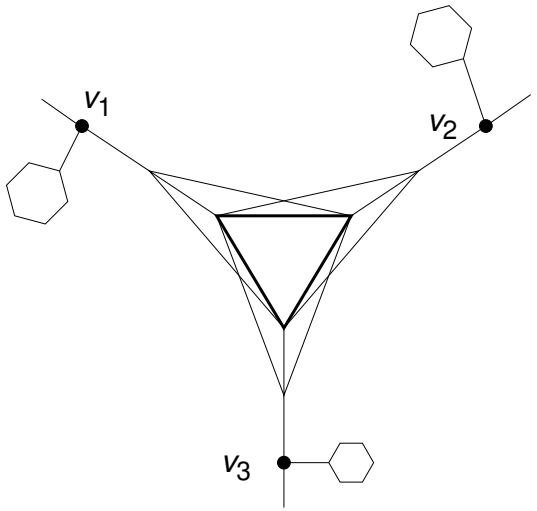
Optimistic coalescing: from vertex cover, degree ≤ 3



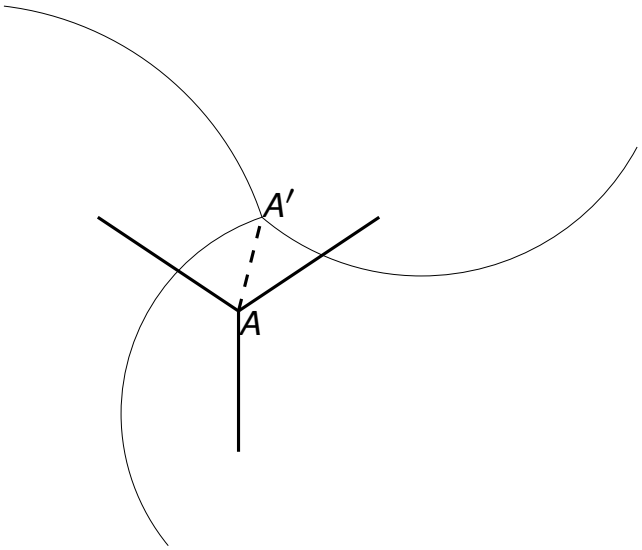


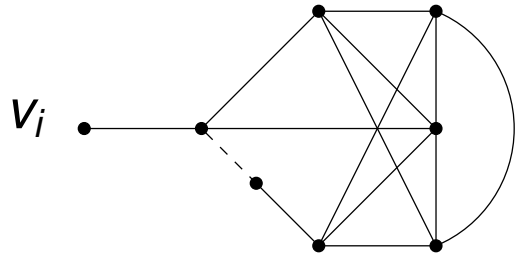


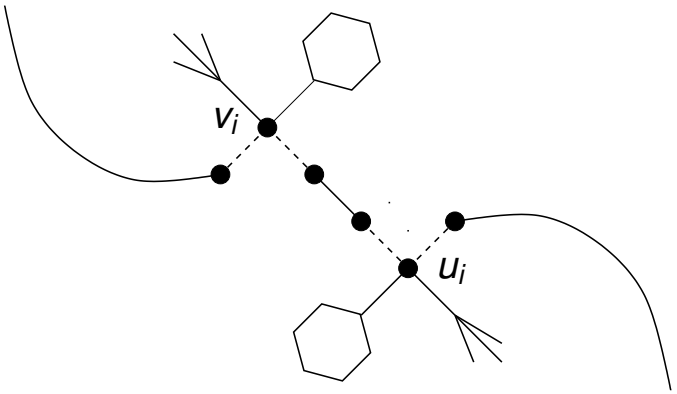
Remove central node ➙



Reduction for optimistic coalescing (Cont'd)







Outline

Conclusions and future works

Not everything is hard SSA changes the rule of the game. Splitting and coalescing need to be considered differently.

Maxlive $\leq k$ is a good test for deciding if spilling is necessary. Even iterated register coalescing overspills.

Splitting (some) critical edges does not seem to be a problem.

Spilling is hard (what/where to spill?) even under SSA.

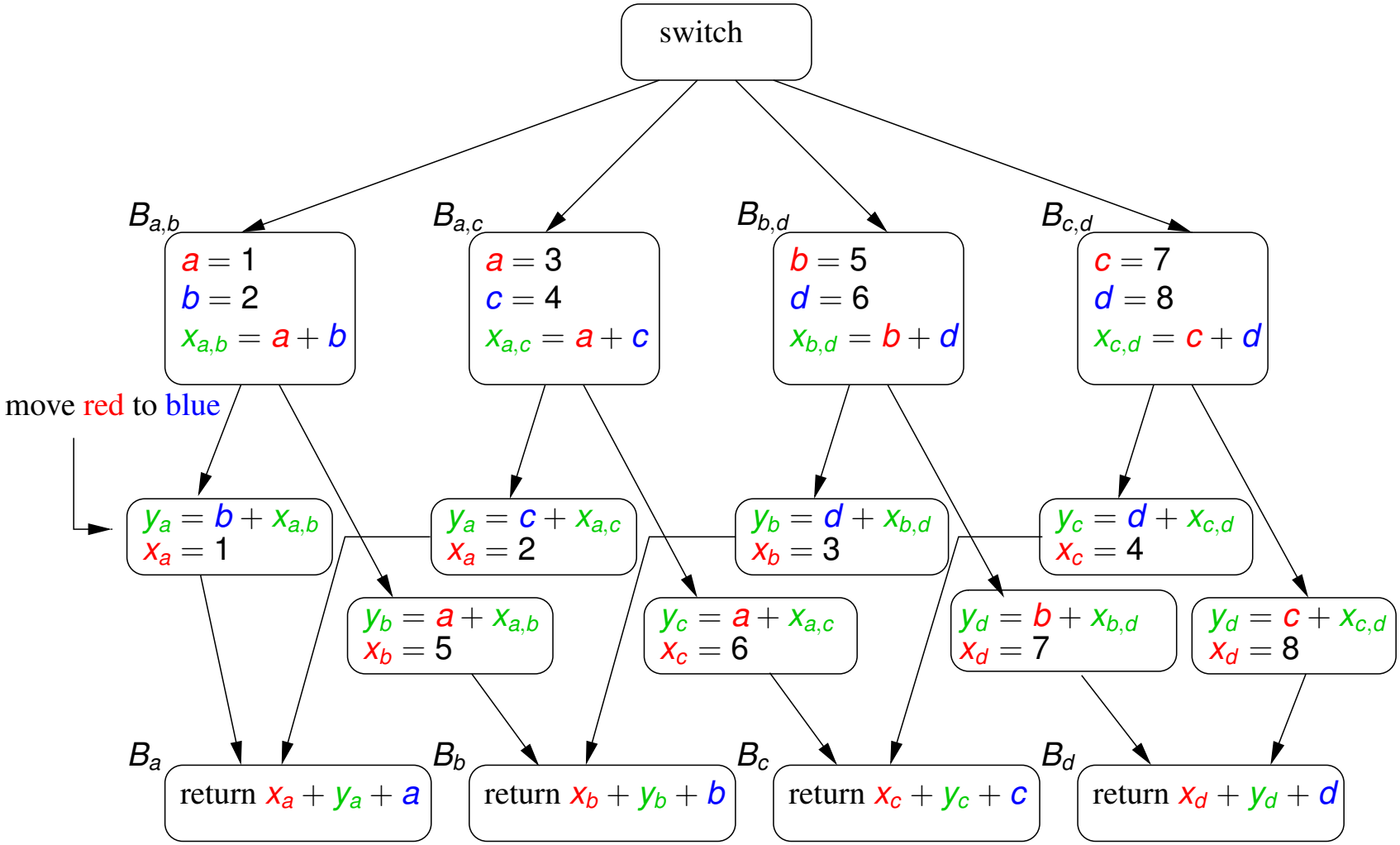
Coalescing is still challenging even with "nice" graphs. Good optimistic heuristics are possible and mandatory.

More experiments need to be done for exploring this new view and tradeoffs between spilling & coalescing.

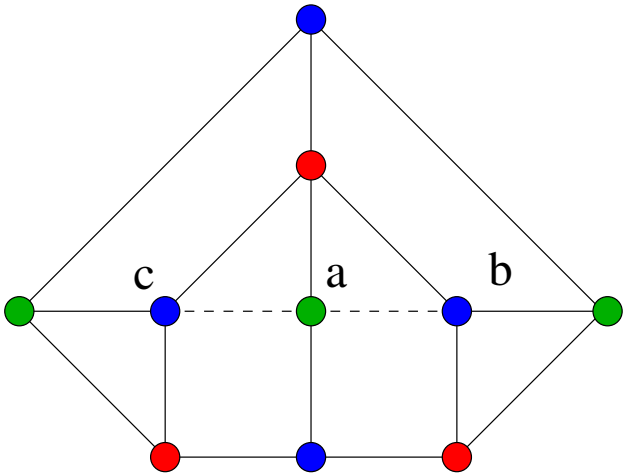
Just-in-time compilation can take advantage of this study.

That's all!
Any questions?

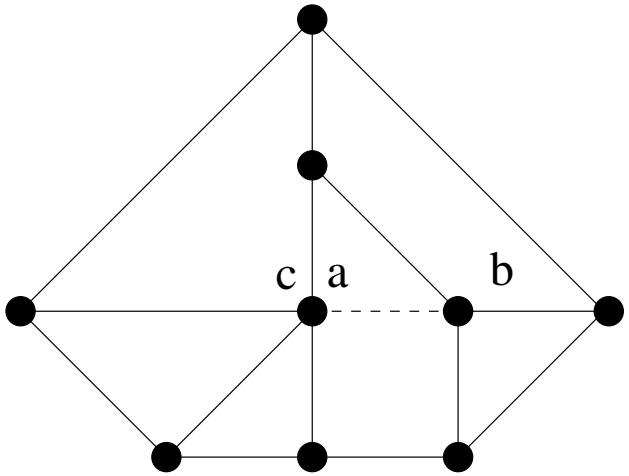
If moves can be anywhere, the proof is broken.



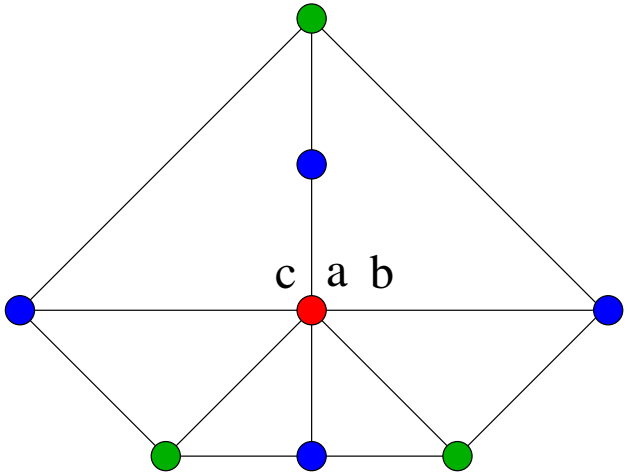
Failure of simple incremental conservative coalescing



greedy-3-colorable



greedy-4-colorable



greedy-3-colorable