Lecture 13

Code optimization

Global optimization

Hyosu Kim

School of Computer Science and Engineering

Chung-Ang University, Seoul, Korea

https://hcslab.cau.ac.kr

hskimhello@cau.ac.kr, hskim.hello@gmail.com

Intermediate code optimizer

Improves the code generated by the intermediate code generator

for optimizing the runtime performance, memory usage, and power consumption of the program, but preserving the semantics of the original program

Types of optimizations

- Local optimizations
- Global optimizations

Global optimizations

Work on a control-flow graph as a whole

- Many of the local optimization techniques can be applied globally
  - Global dead code elimination
  - Global copy propagation / common subexpression elimination

Global dead code elimination

Reminder: live variable analysis for local dead code elimination

A variable is called a live variable if it holds a value that will be needed in the future

- To know whether a variable will be used in the future or not,

  checks the statements in a basic block in a reverse order

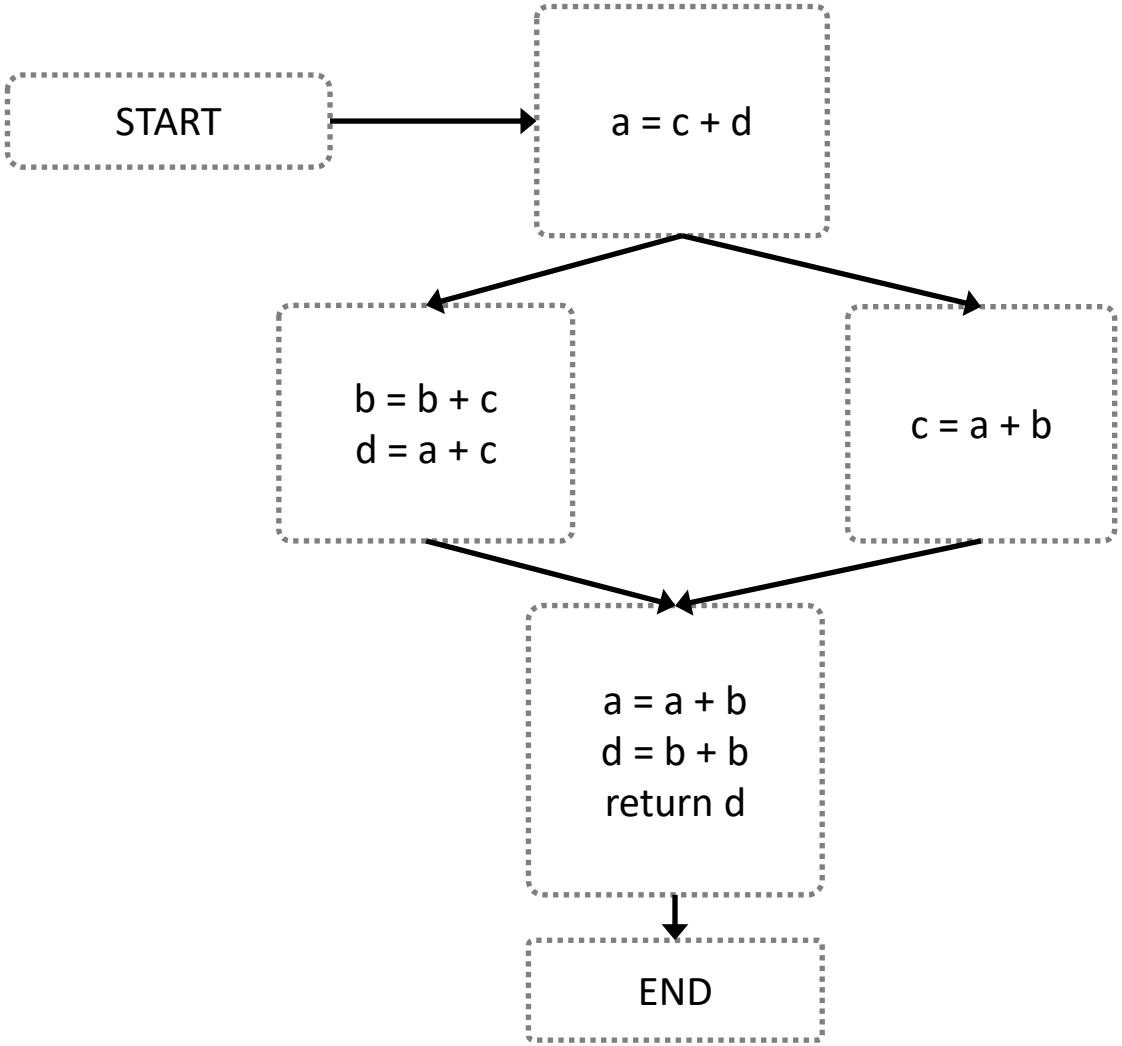
  - **Initially, some small set of variables are known to be live**

    (e.g., variables will be used in the next block)

    “This information can only be computed globally”

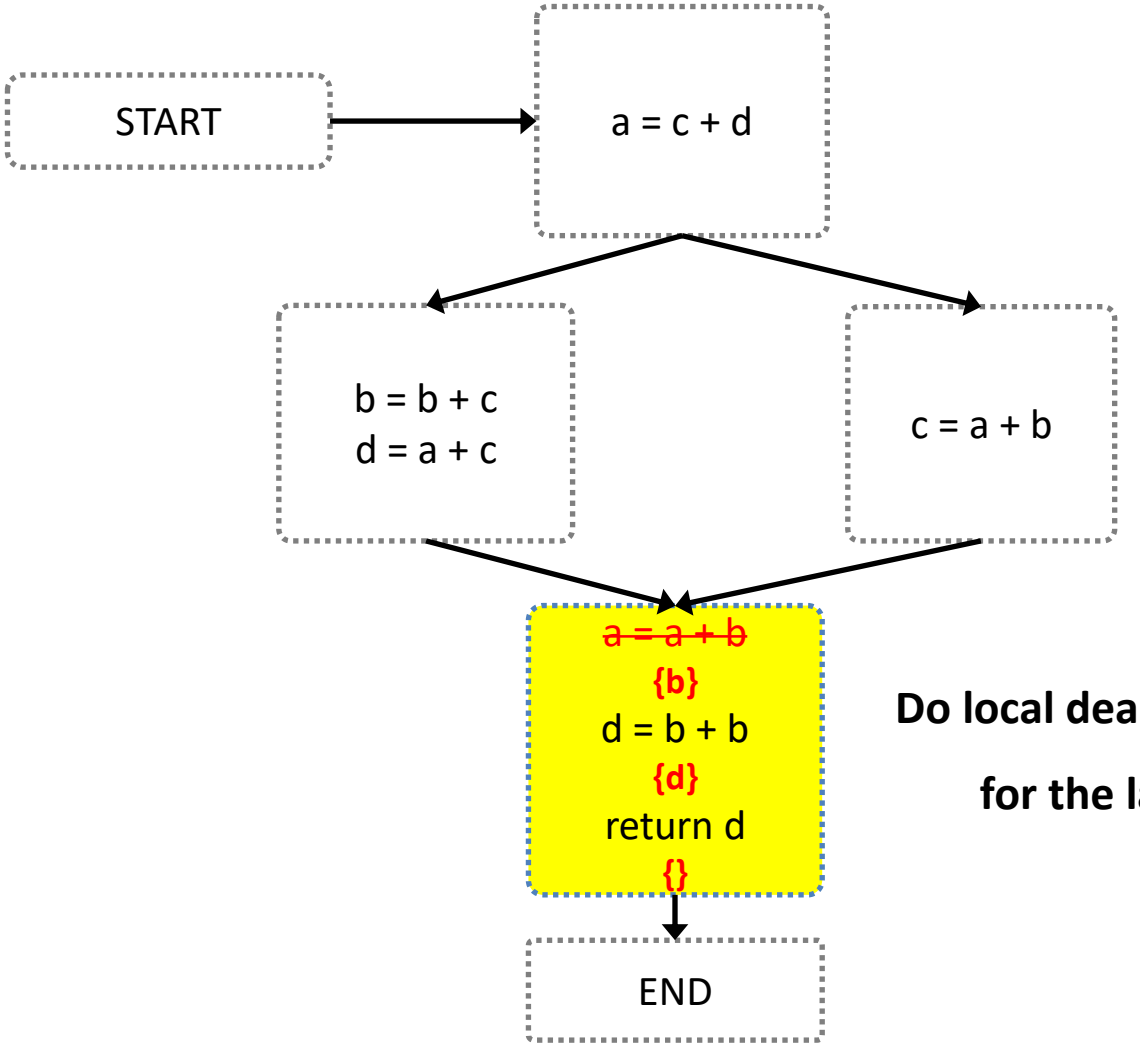
Global dead code elimination

A simple example of global live variable analysis



Global dead code elimination

A simple example of global live variable analysis

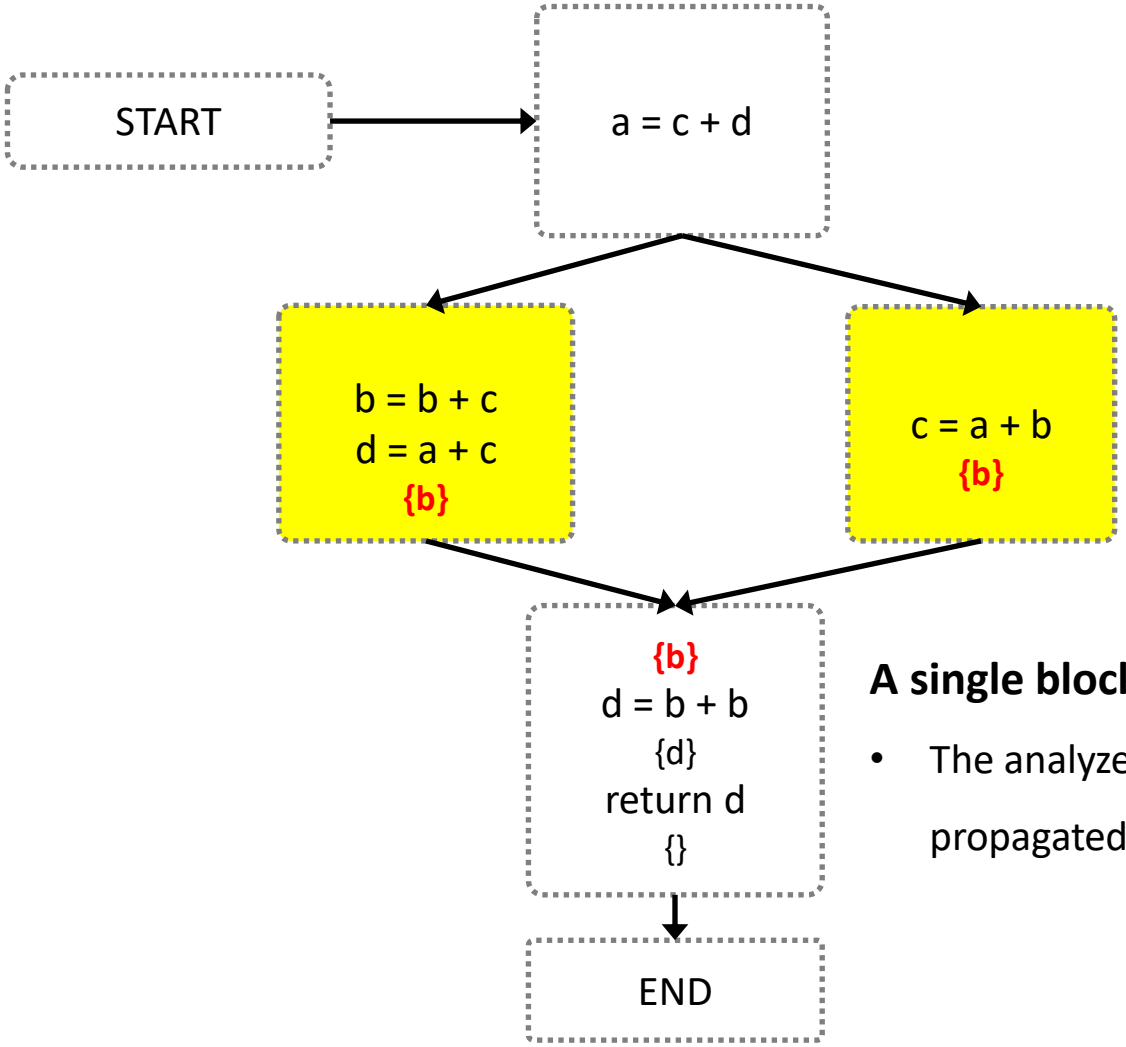


Do local dead code elimination

for the last basic block

Global dead code elimination

A simple example of global live variable analysis

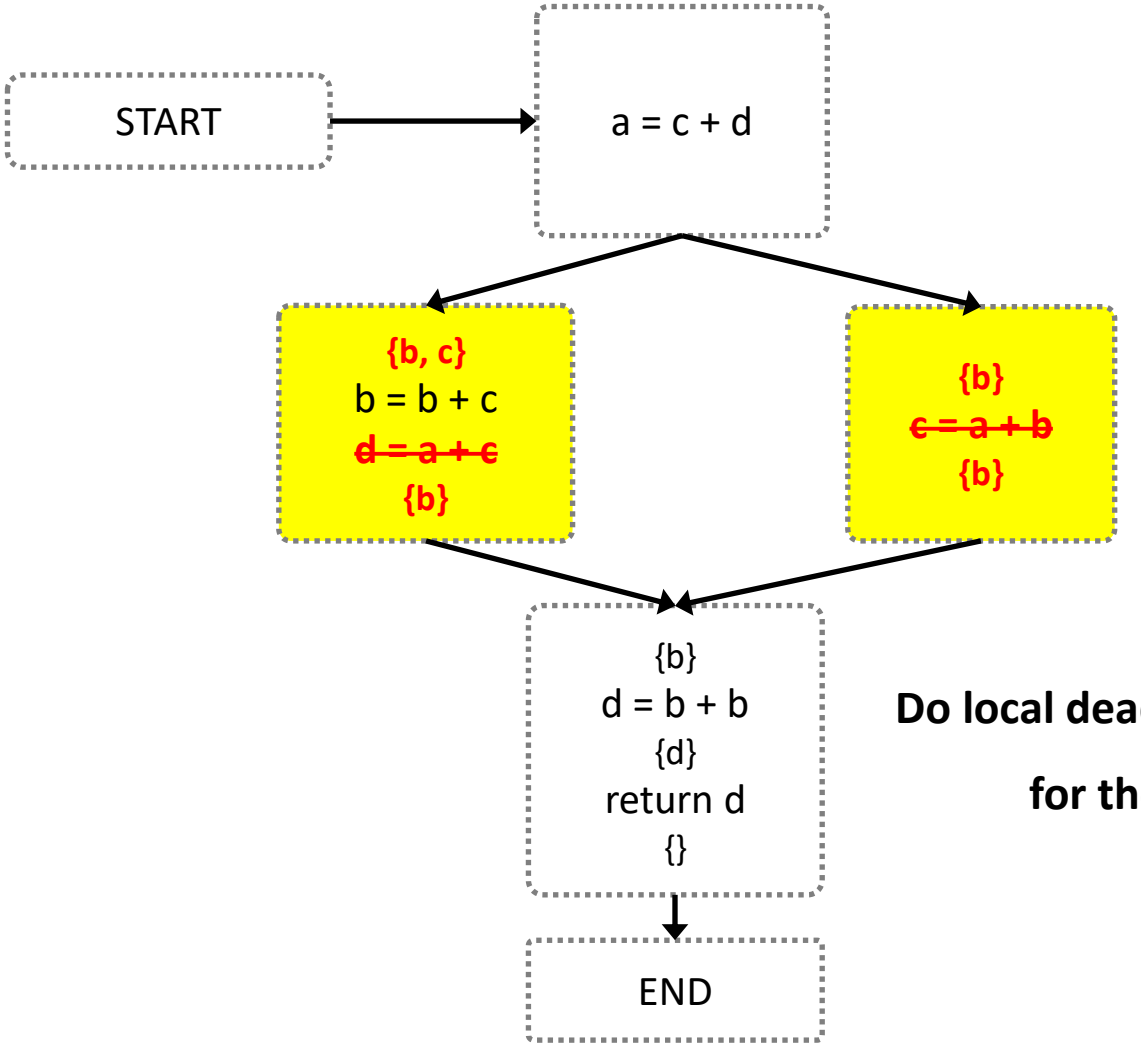


A single block can have multiple previous block

- The analyzed initial live variable set is propagated to each previous block

Global dead code elimination

A simple example of global live variable analysis

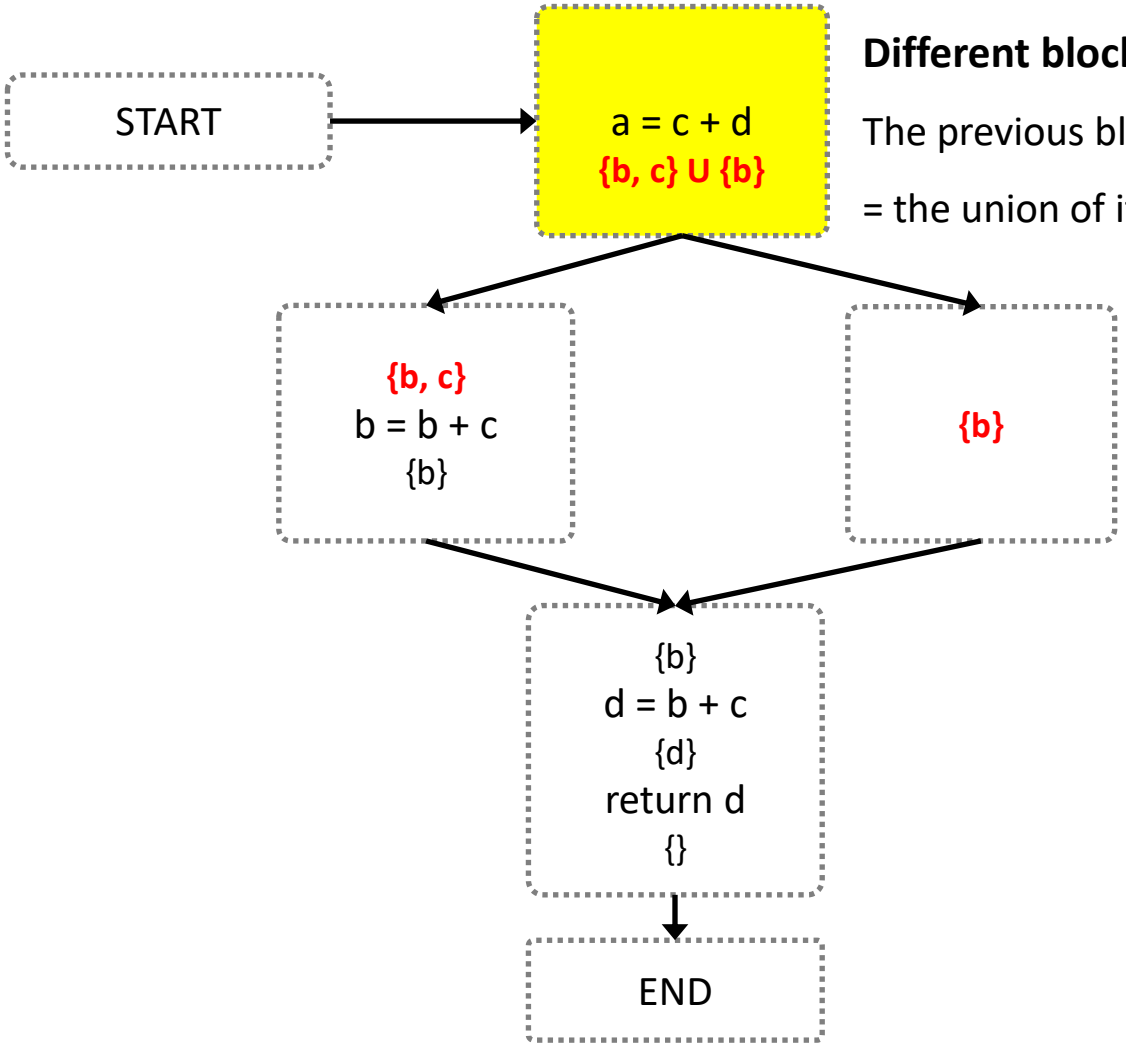


Do local dead code elimination

for the each block

Global dead code elimination

A simple example of global live variable analysis



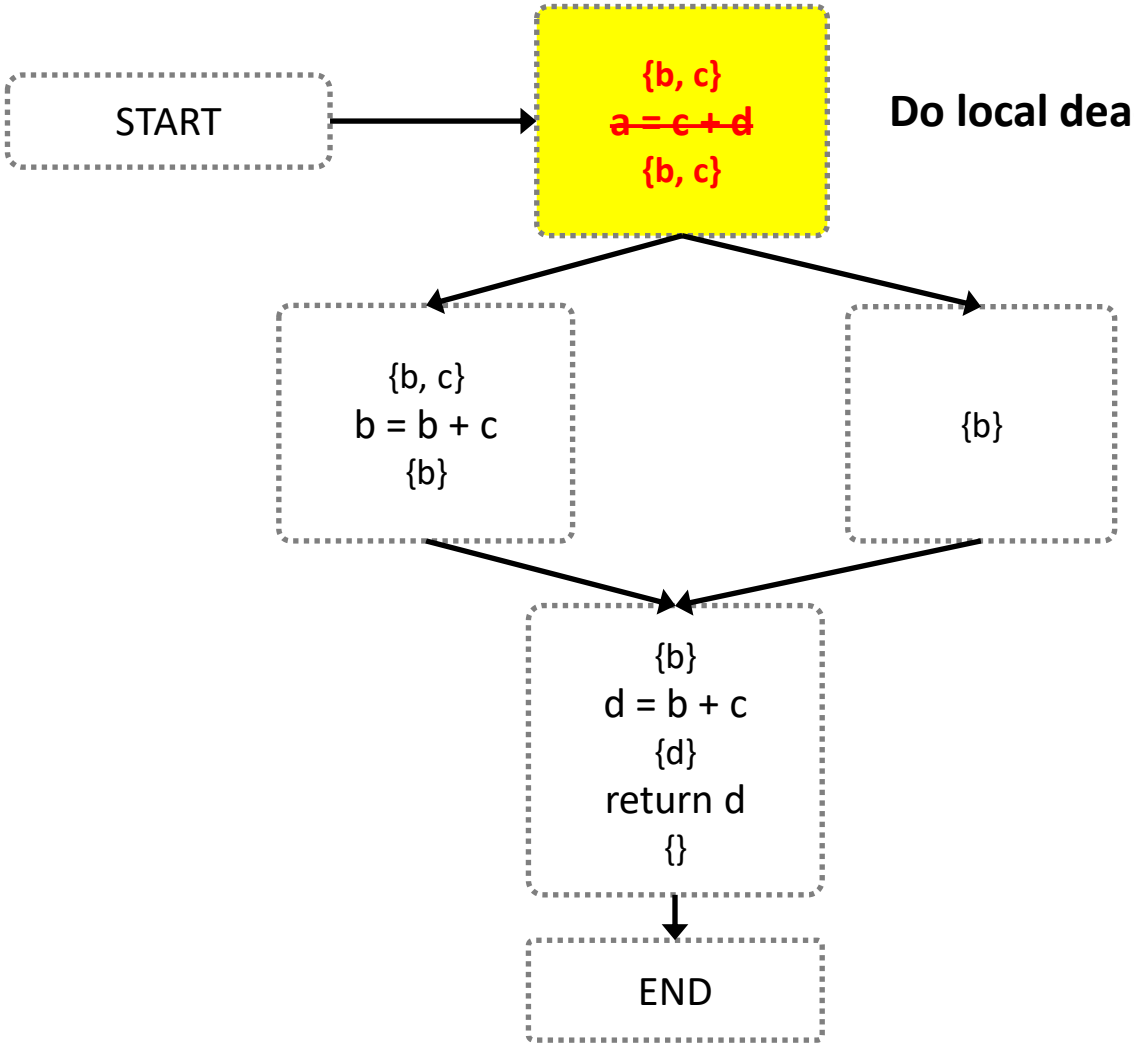
Different blocks can have the same previous block

The previous block's final live variable set

= the union of its next blocks' initial sets

Global dead code elimination

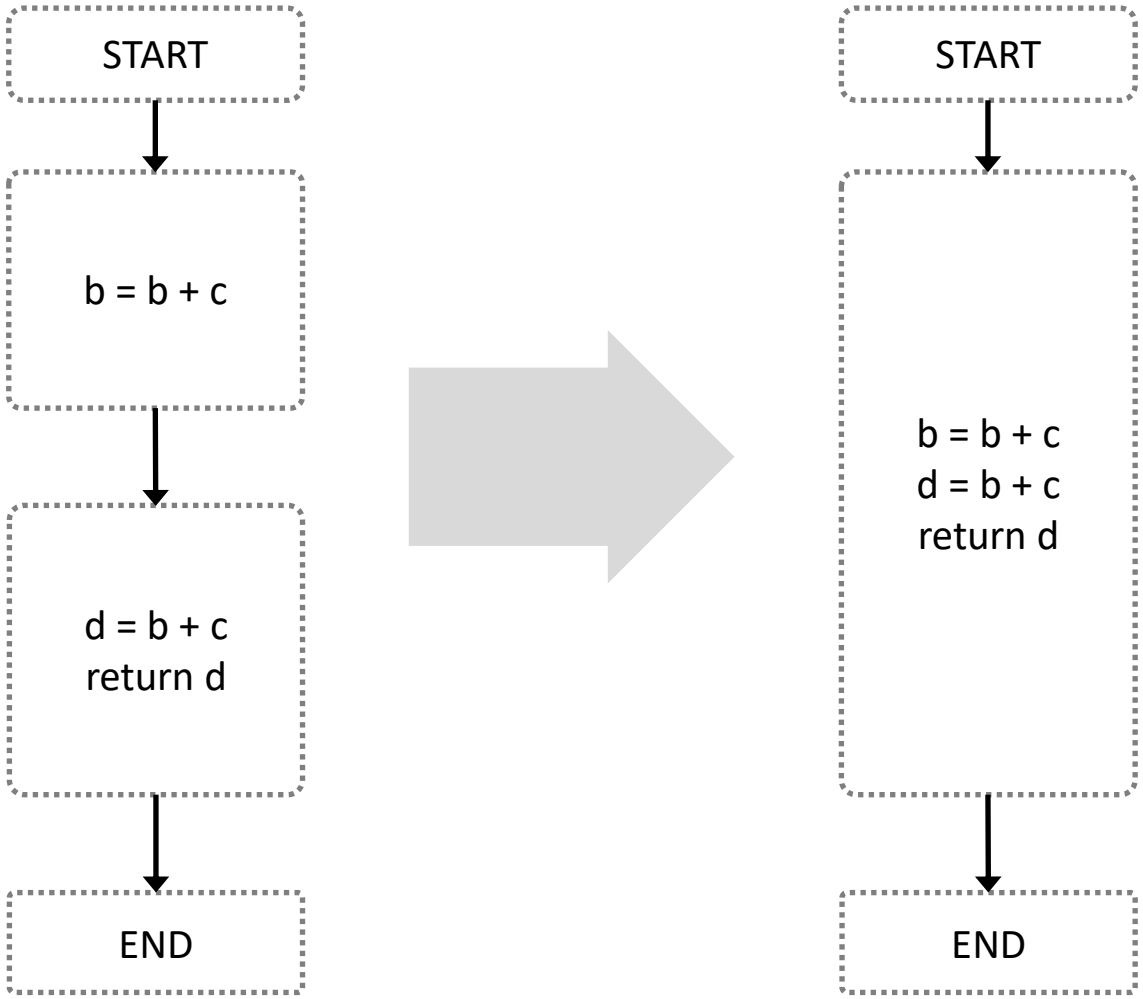
A simple example of global live variable analysis



Do local dead code elimination

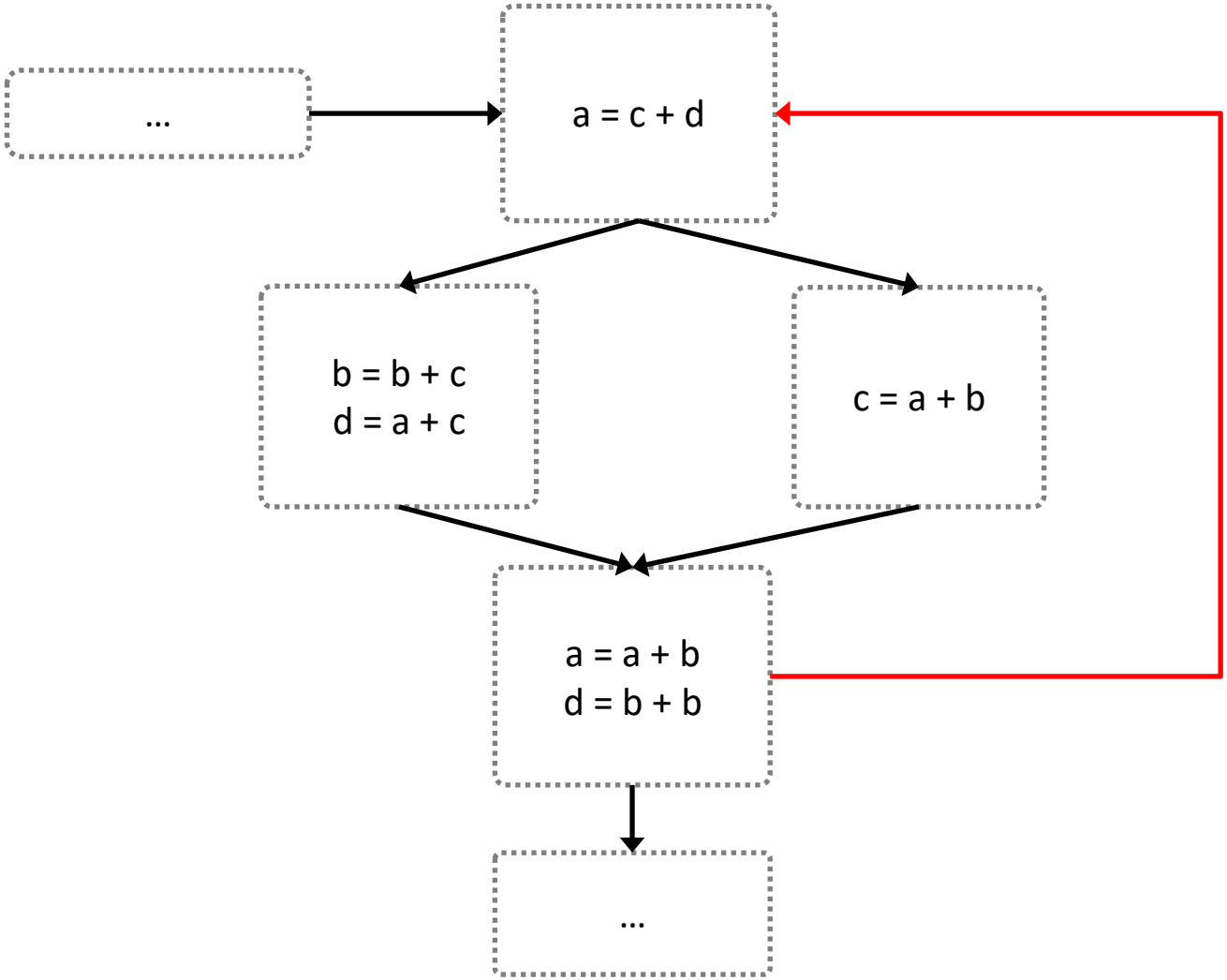
Global dead code elimination

A simple example of global live variable analysis



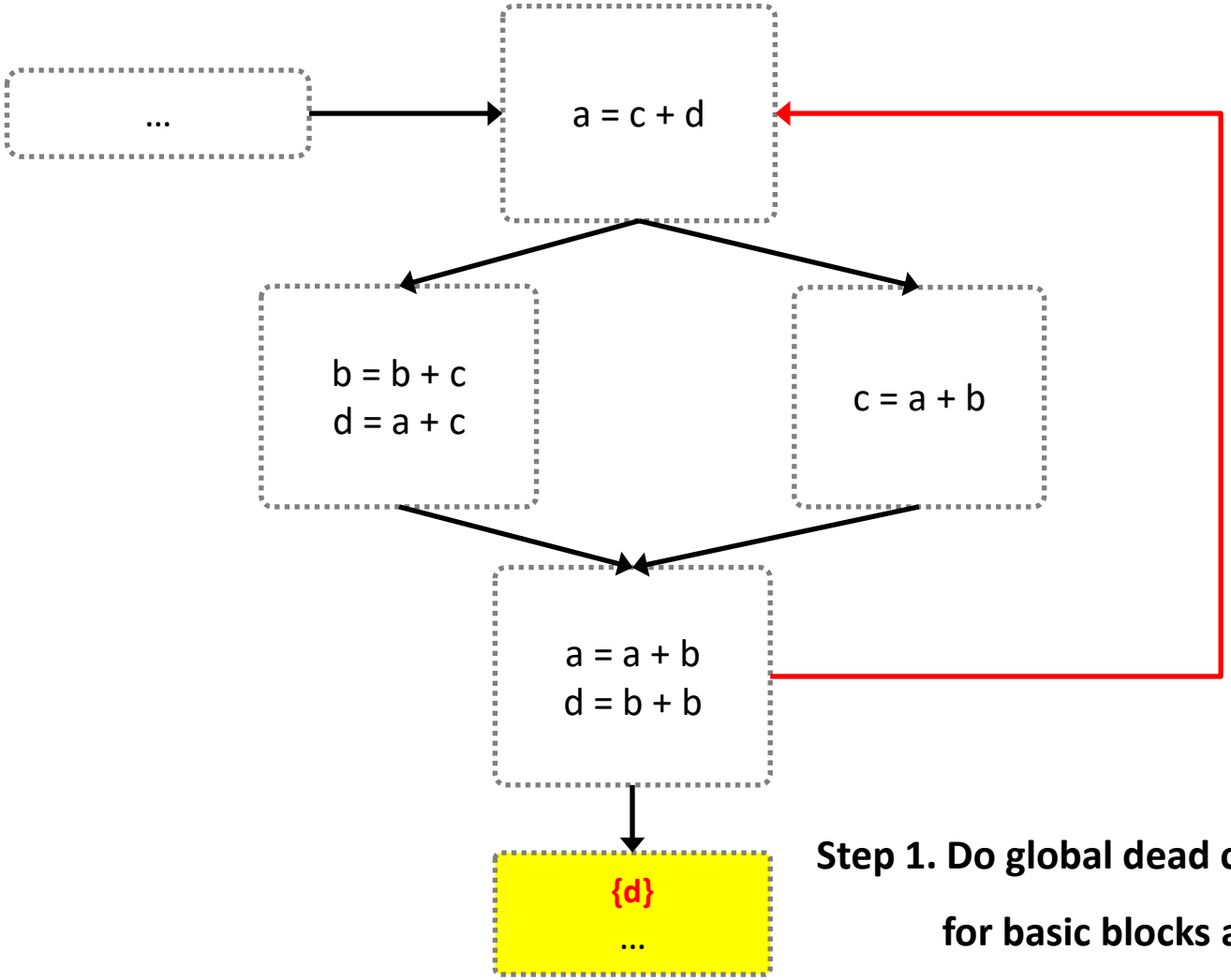
Global dead code elimination

A complicated example of global live variable analysis with loops



Global dead code elimination

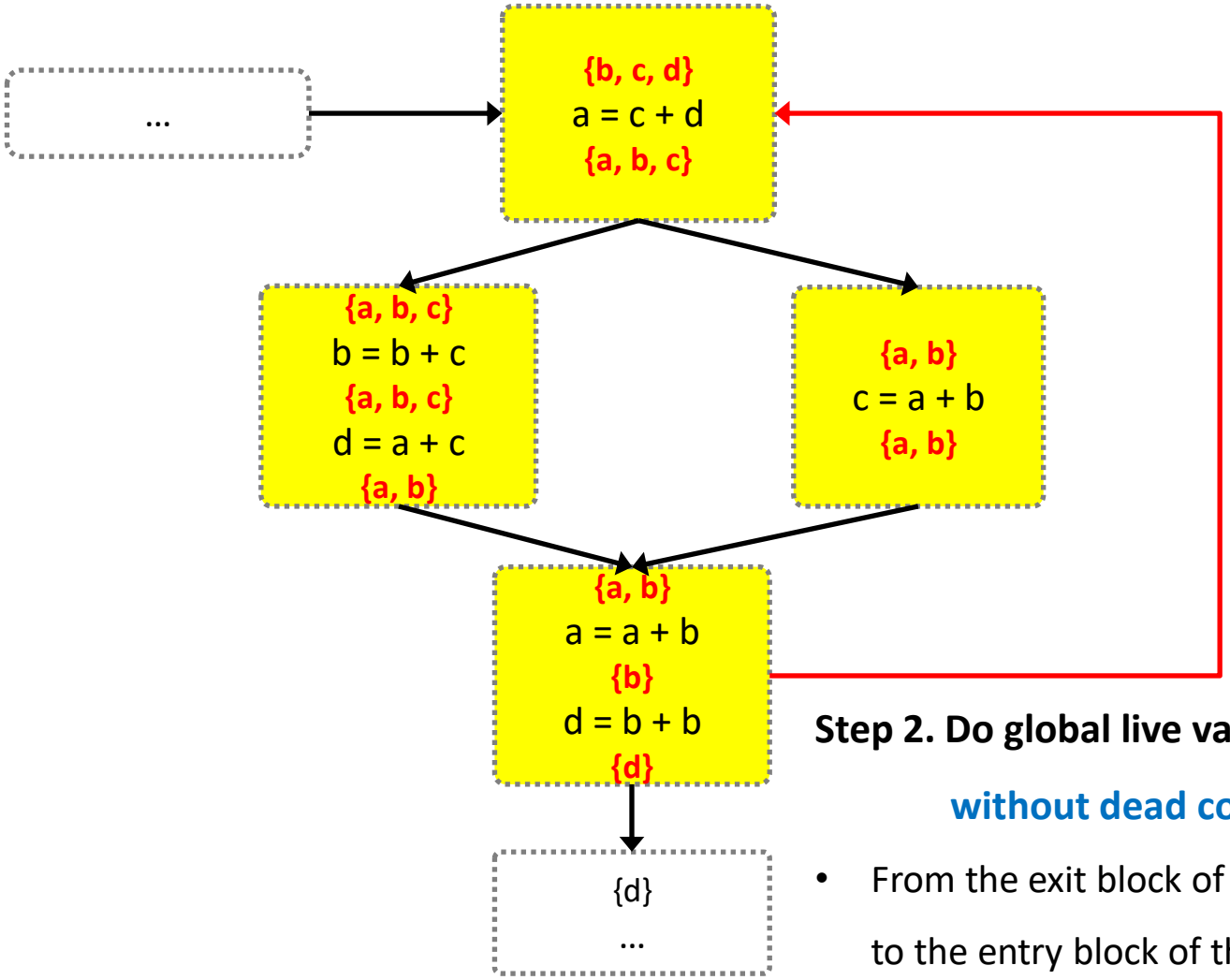
A complicated example of global live variable analysis with loops



Step 1. Do global dead code elimination for basic blocks after the loop

Global dead code elimination

A complicated example of global live variable analysis with loops



Step 2. Do global live variable analysis

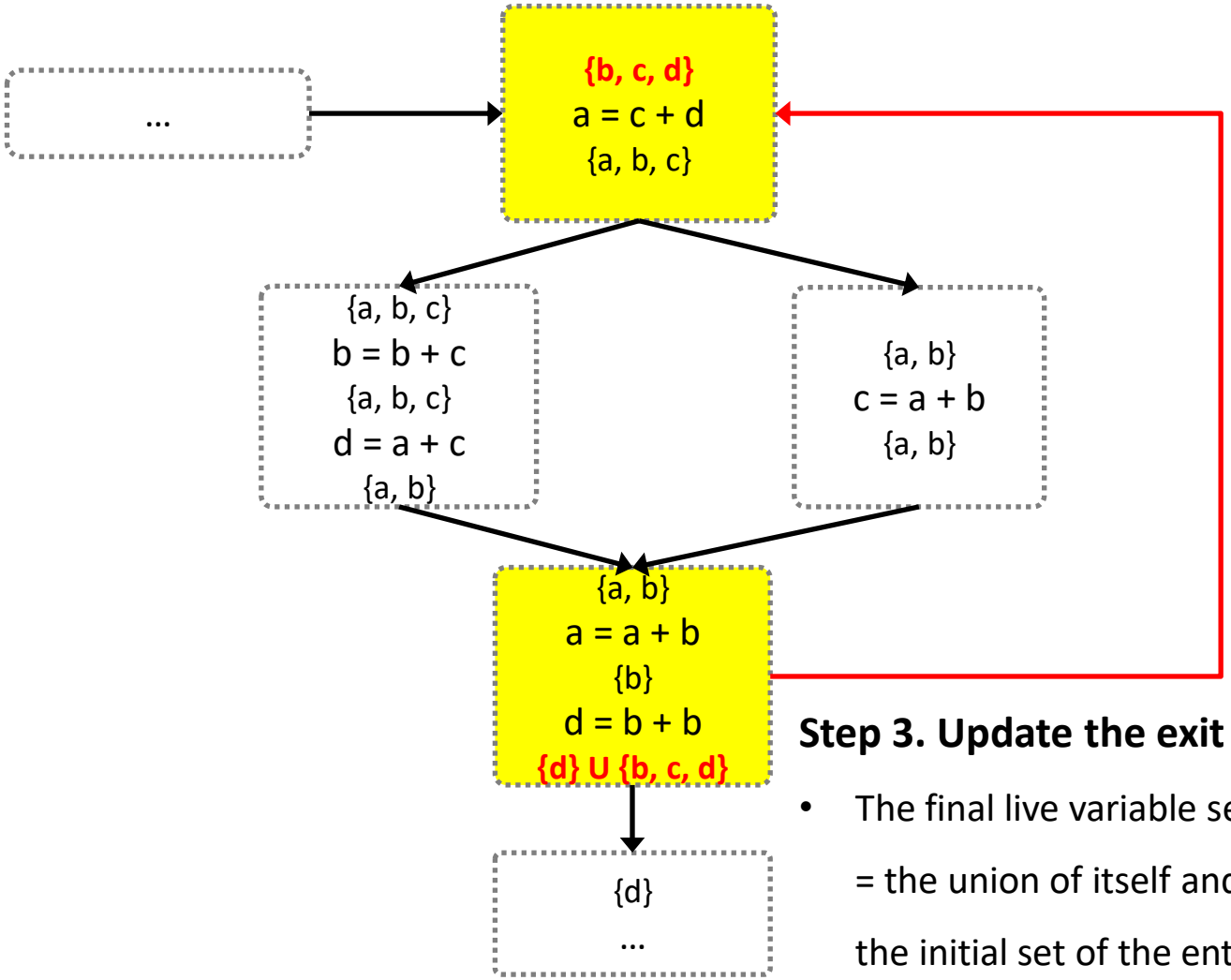
without dead code elimination

- From the exit block of the loop

  to the entry block of the loop

Global dead code elimination

A complicated example of global live variable analysis with loops

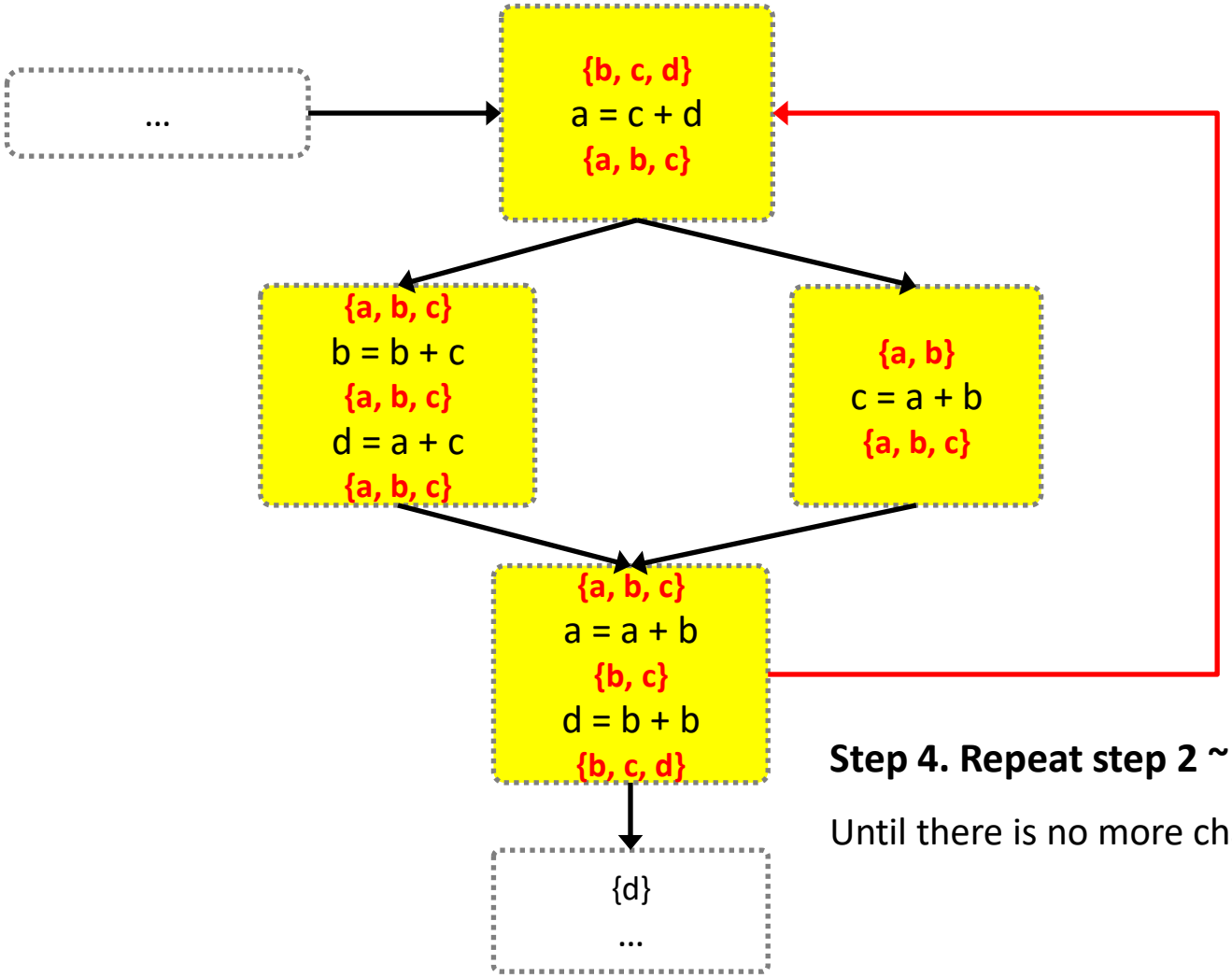


Step 3. Update the exit block of the loop

- The final live variable set of the exit block
  = the union of itself and
  the initial set of the entry block of the loop

Global dead code elimination

A complicated example of global live variable analysis with loops

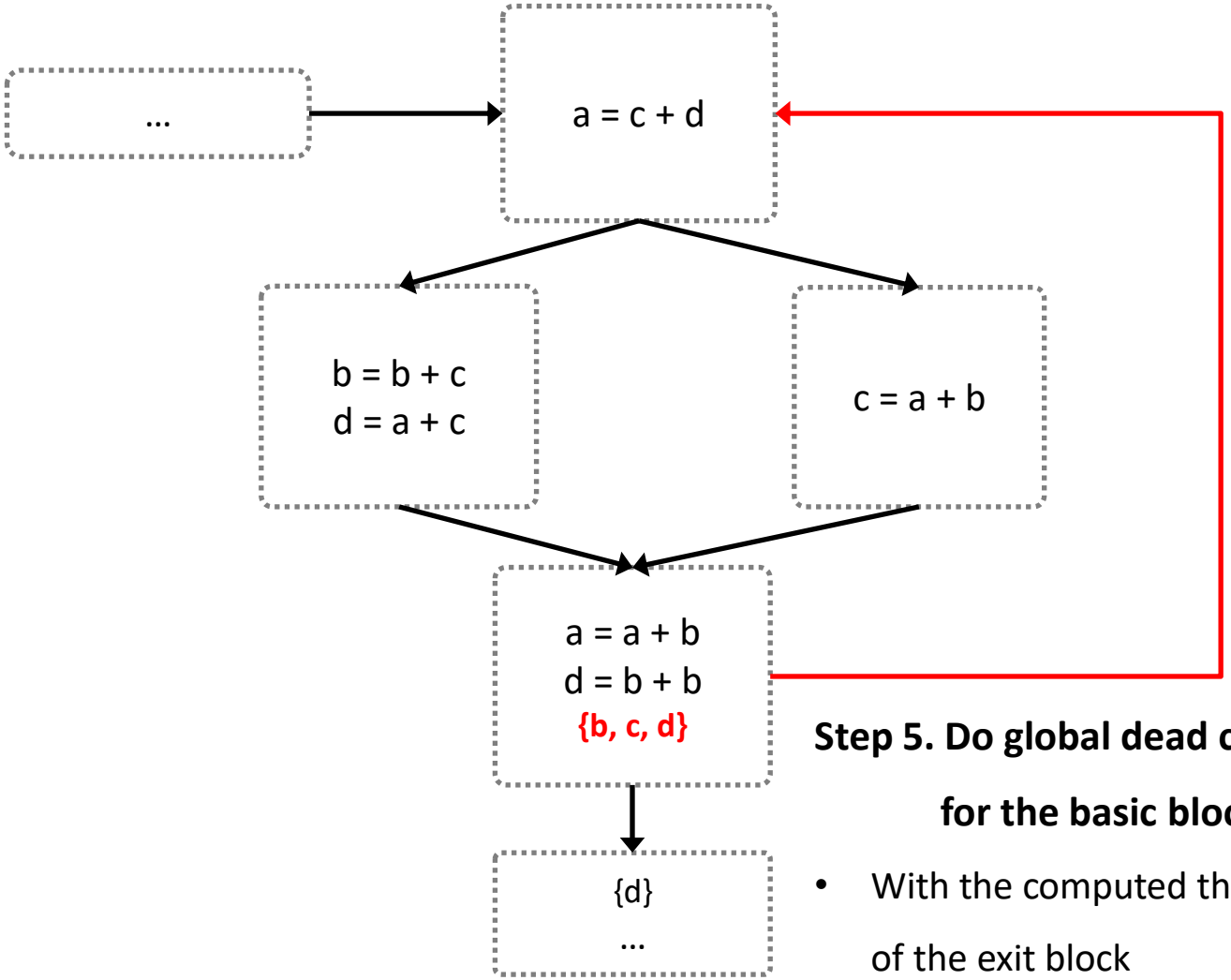


Step 4. Repeat step 2 ~ 3

Until there is no more change

Global dead code elimination

A complicated example of global live variable analysis with loops

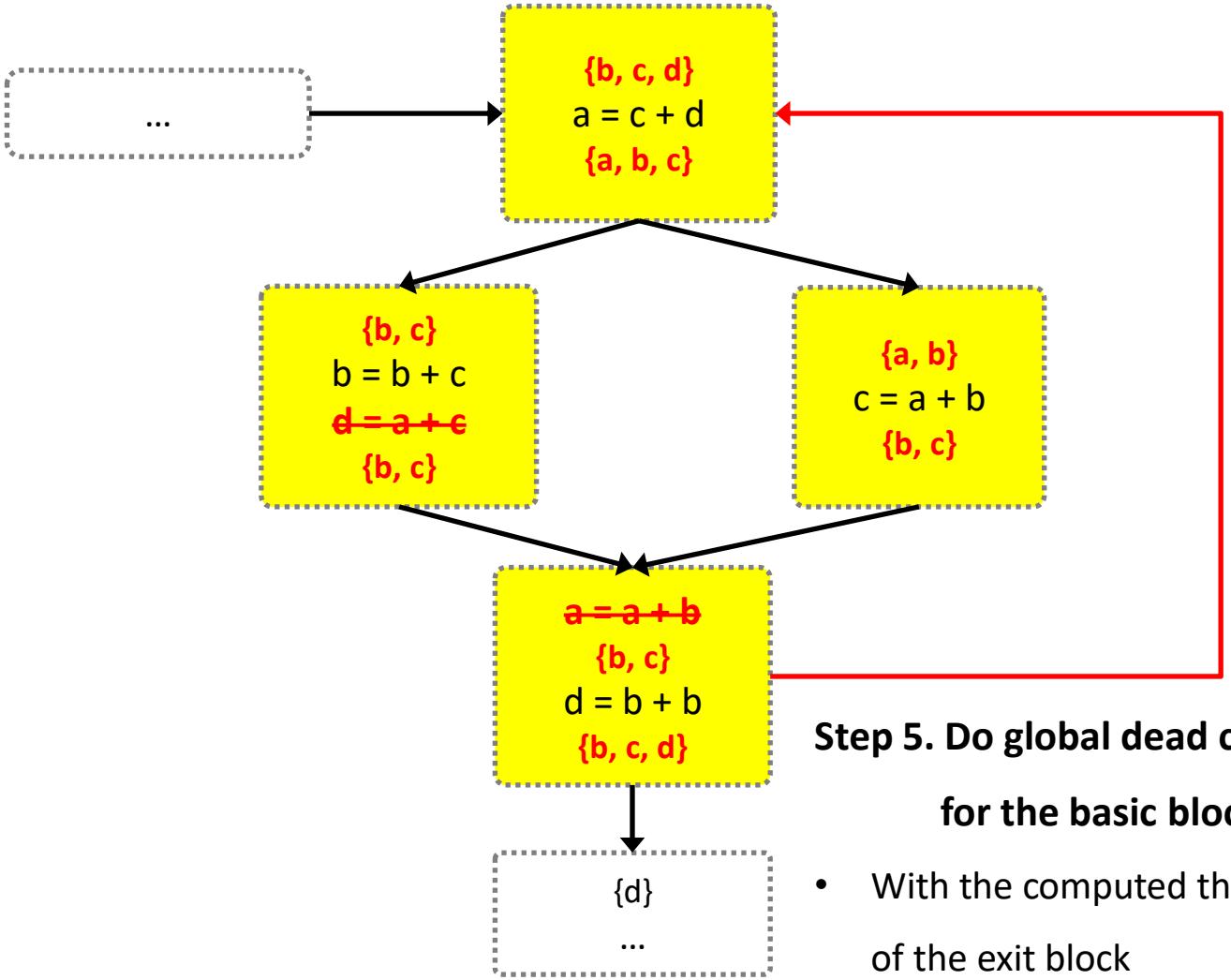


Step 5. Do global dead code elimination for the basic blocks in the loop

- With the computed the initial live variable set of the exit block

Global dead code elimination

A complicated example of global live variable analysis with loops

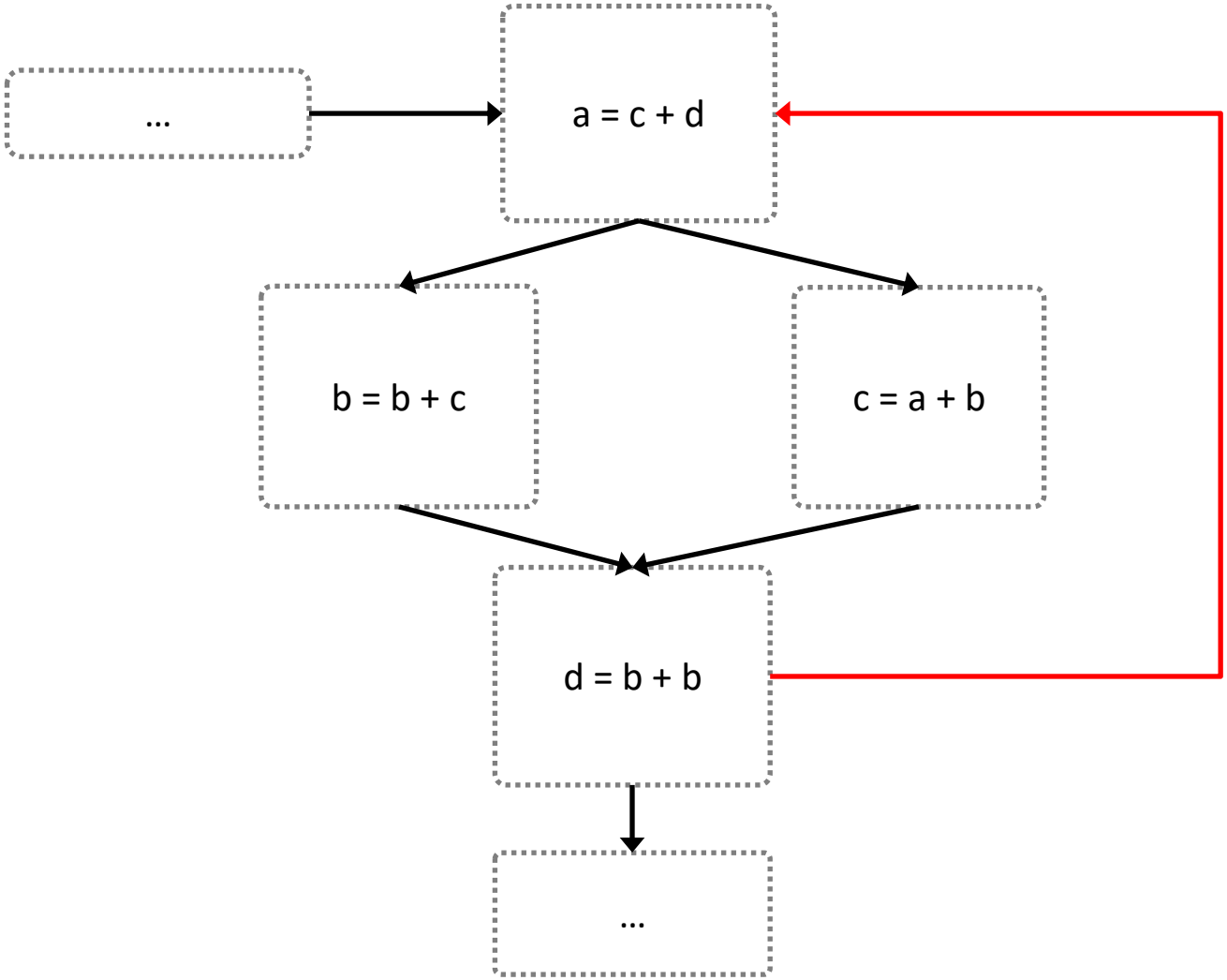


Step 5. Do global dead code elimination for the basic blocks in the loop

- With the computed the initial live variable set of the exit block

Global dead code elimination

A complicated example of global live variable analysis with loops



Global copy propagation / common subexpressions elimination

Key idea: compute available expressions globally

Reminder: available expression analysis

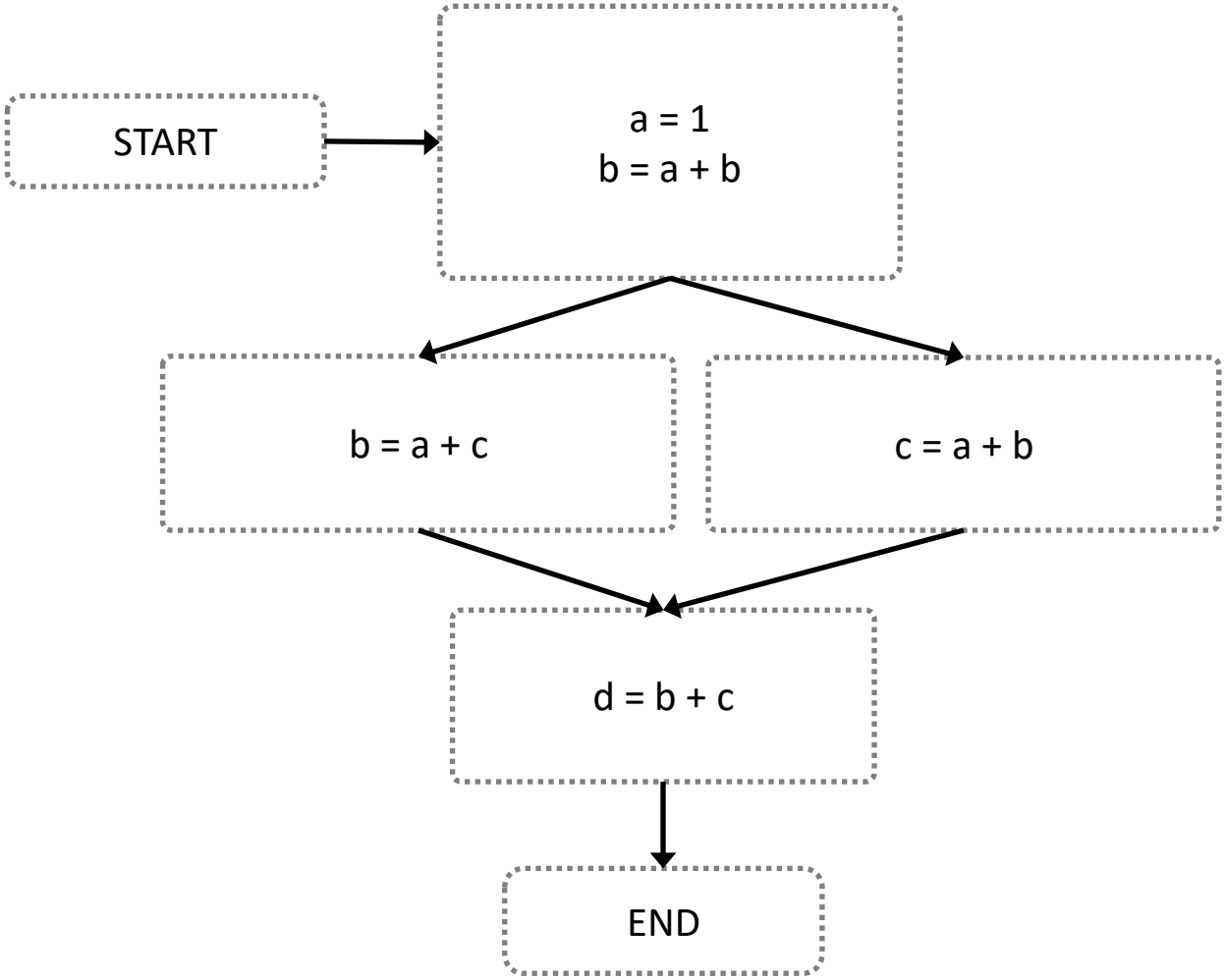
Determines for each point in a program the set of available expressions

- Initially, no expressions are available
- Whenever we check a statement **a = operation (e.g., a = b + c)**
  - Any expression holding **a** is invalidated!!
  - The new expression **a = operation** becomes available

Three-address code	Available expressions
(before executing t0 = a* a)	{}
t0 = a * a	
(after t0 = a * a, before t0 = a + b)	{t0 = a * a}
t0 = a + b	
(after t0 = a + b)	{**t0 = a + b**}

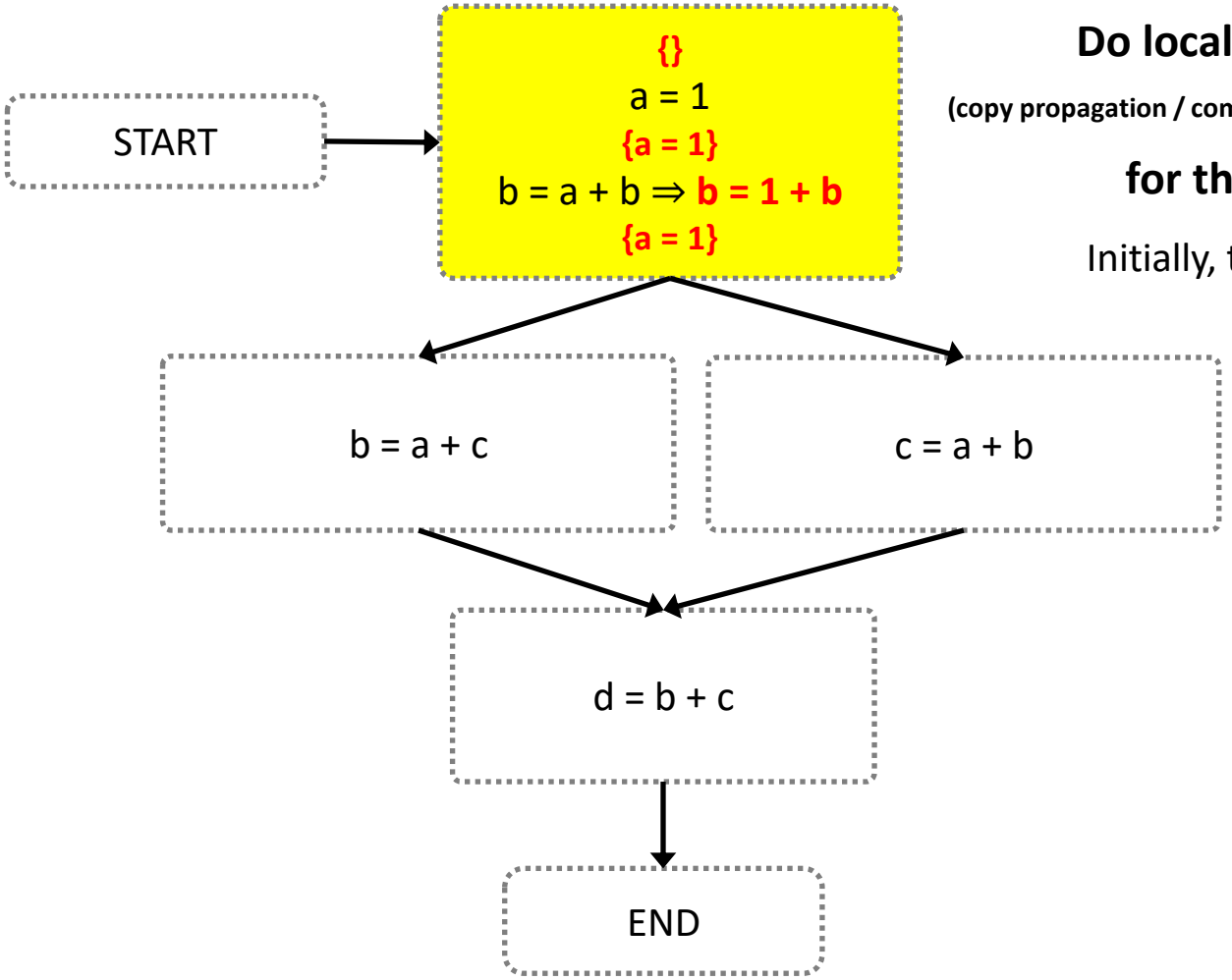
Global copy propagation / common subexpressions elimination

A simple example



Global copy propagation / common subexpressions elimination

A simple example



Do local optimizations

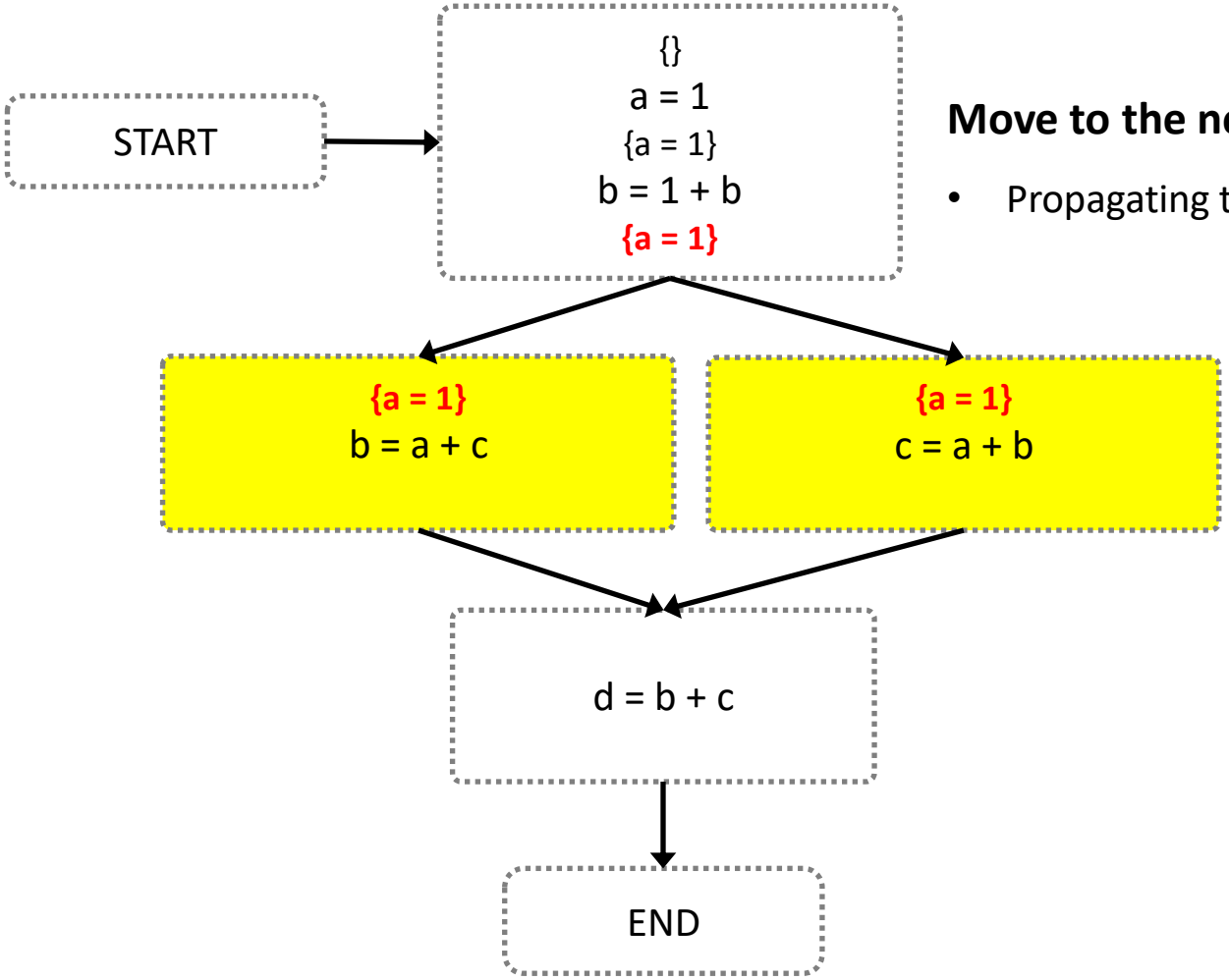
(copy propagation / common subexpressions elimination)

for the first block

Initially, the set is empty

Global copy propagation / common subexpressions elimination

A simple example

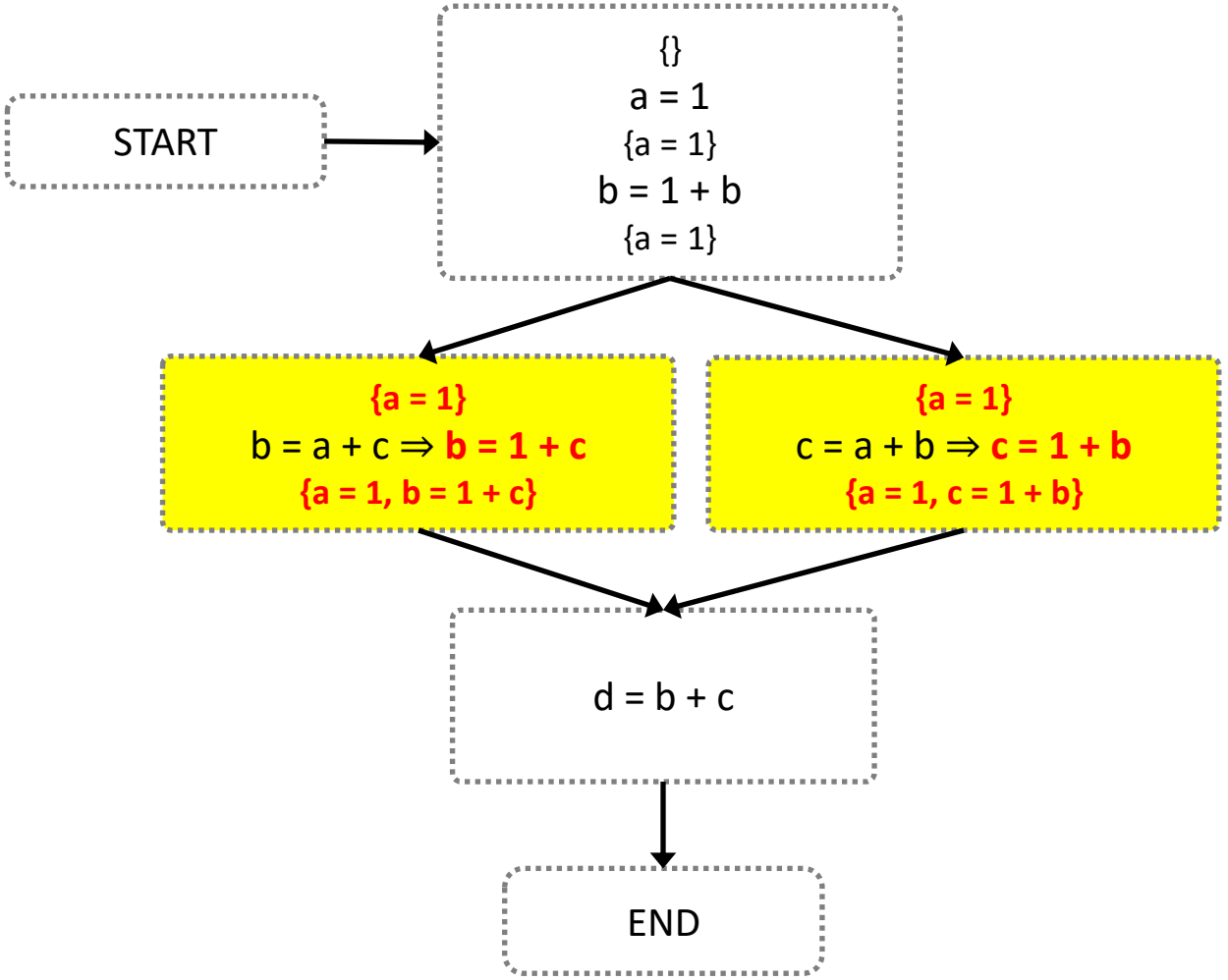


Move to the next blocks

- Propagating the final result to the next blocks

Global copy propagation / common subexpressions elimination

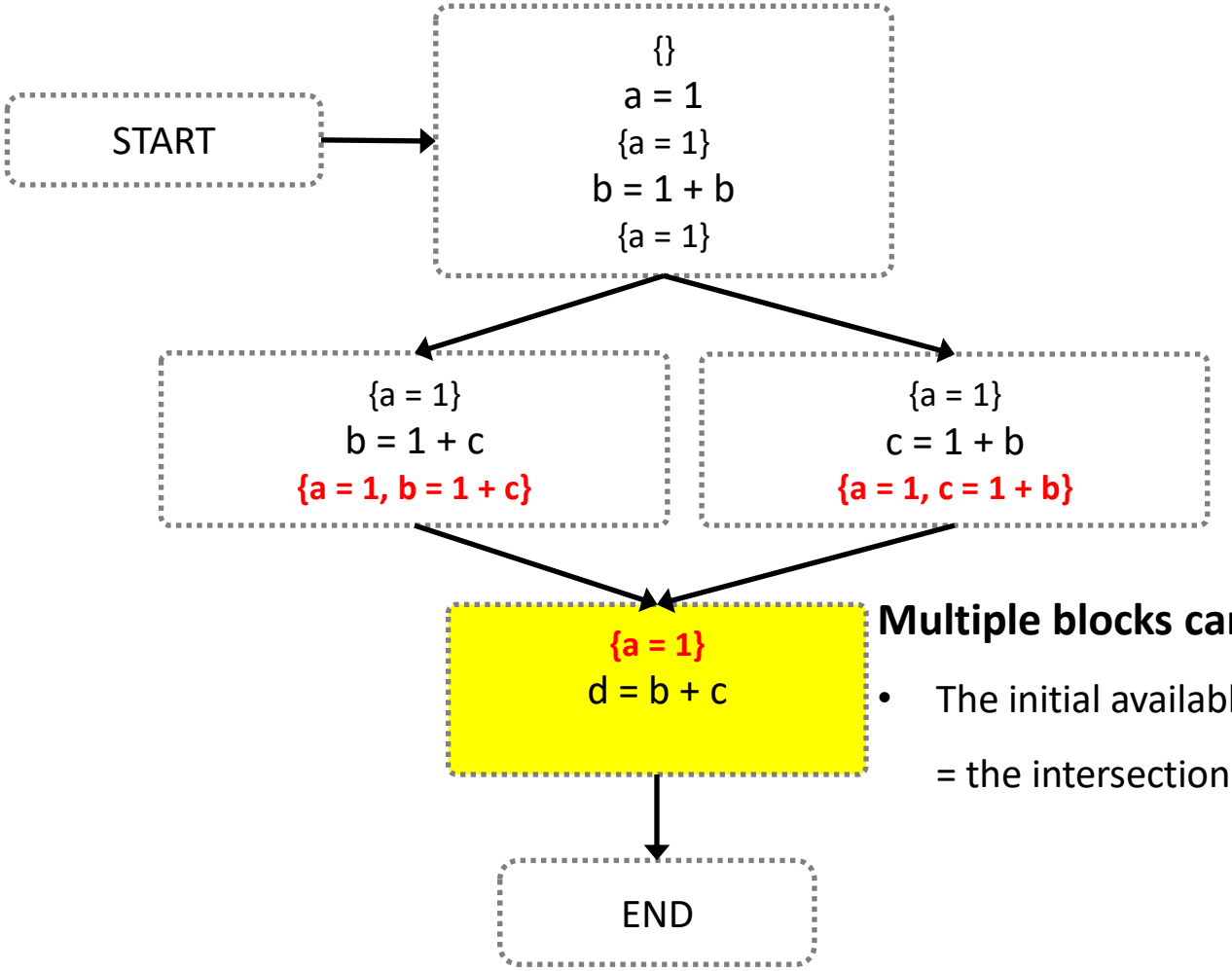
A simple example



Do optimizations locally for each block

Global copy propagation / common subexpressions elimination

A simple example

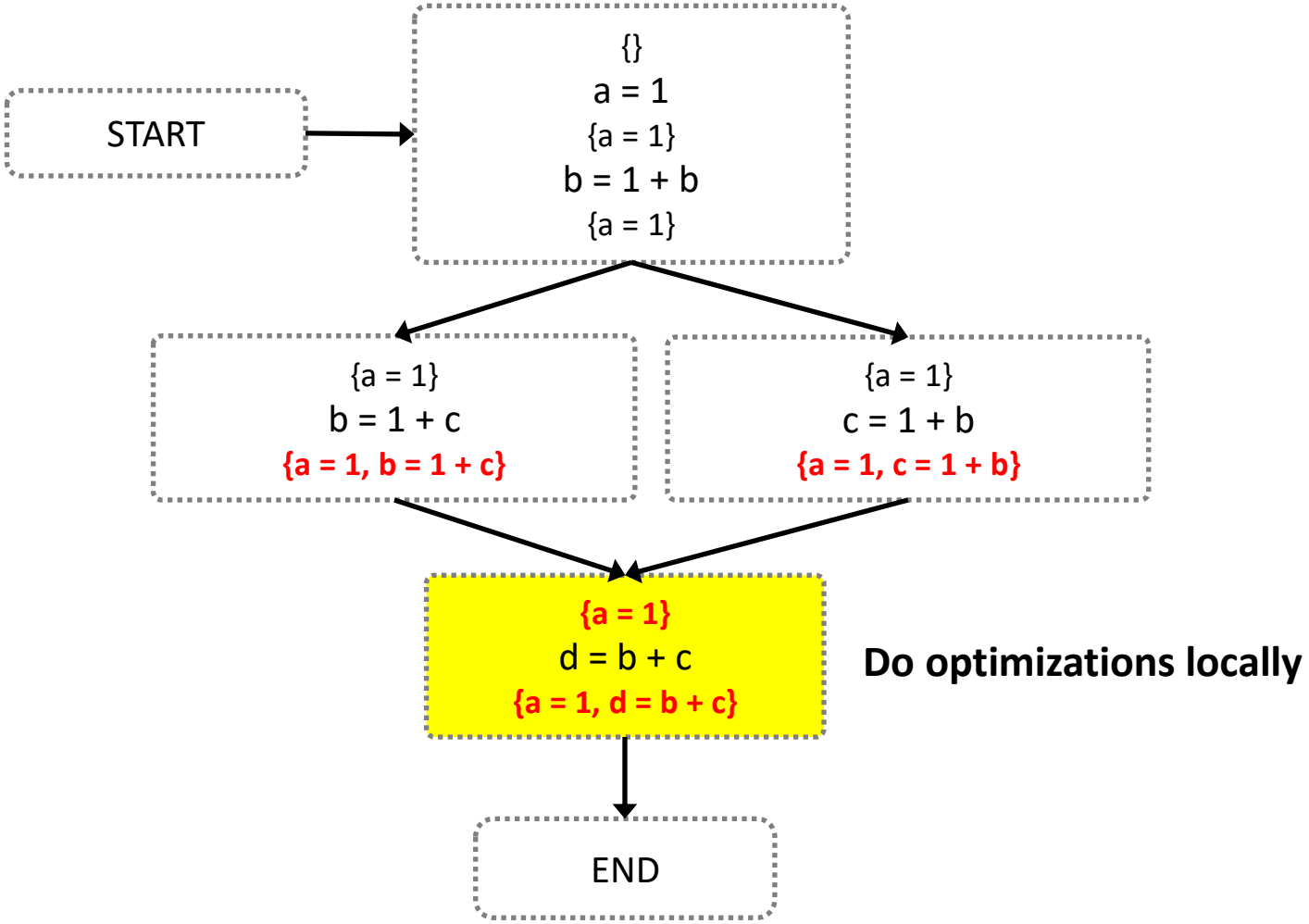


Multiple blocks can have the same next block

- The initial available expression set of the next block
  = the intersection of its previous blocks' results

Global copy propagation / common subexpressions elimination

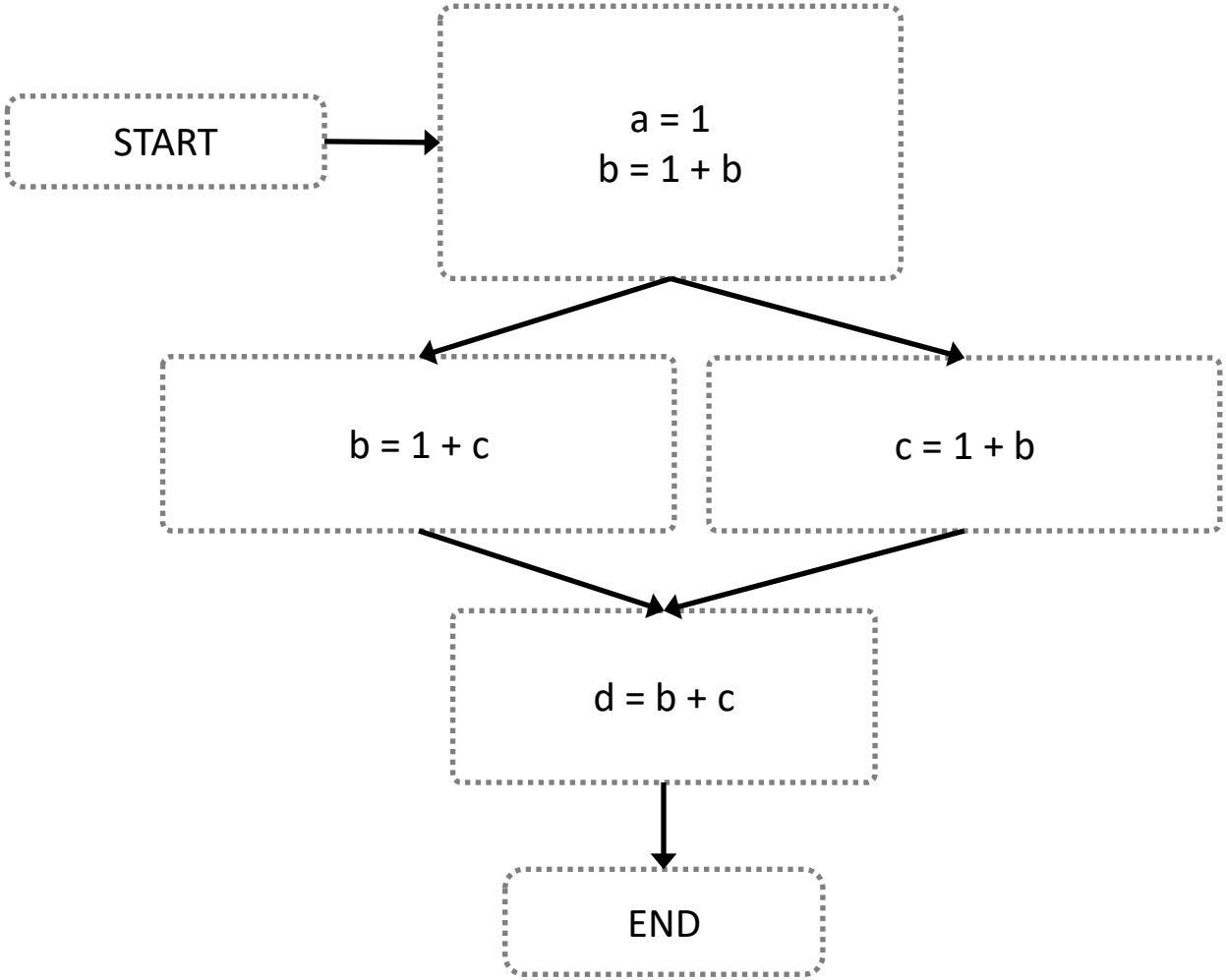
A simple example



Do optimizations locally

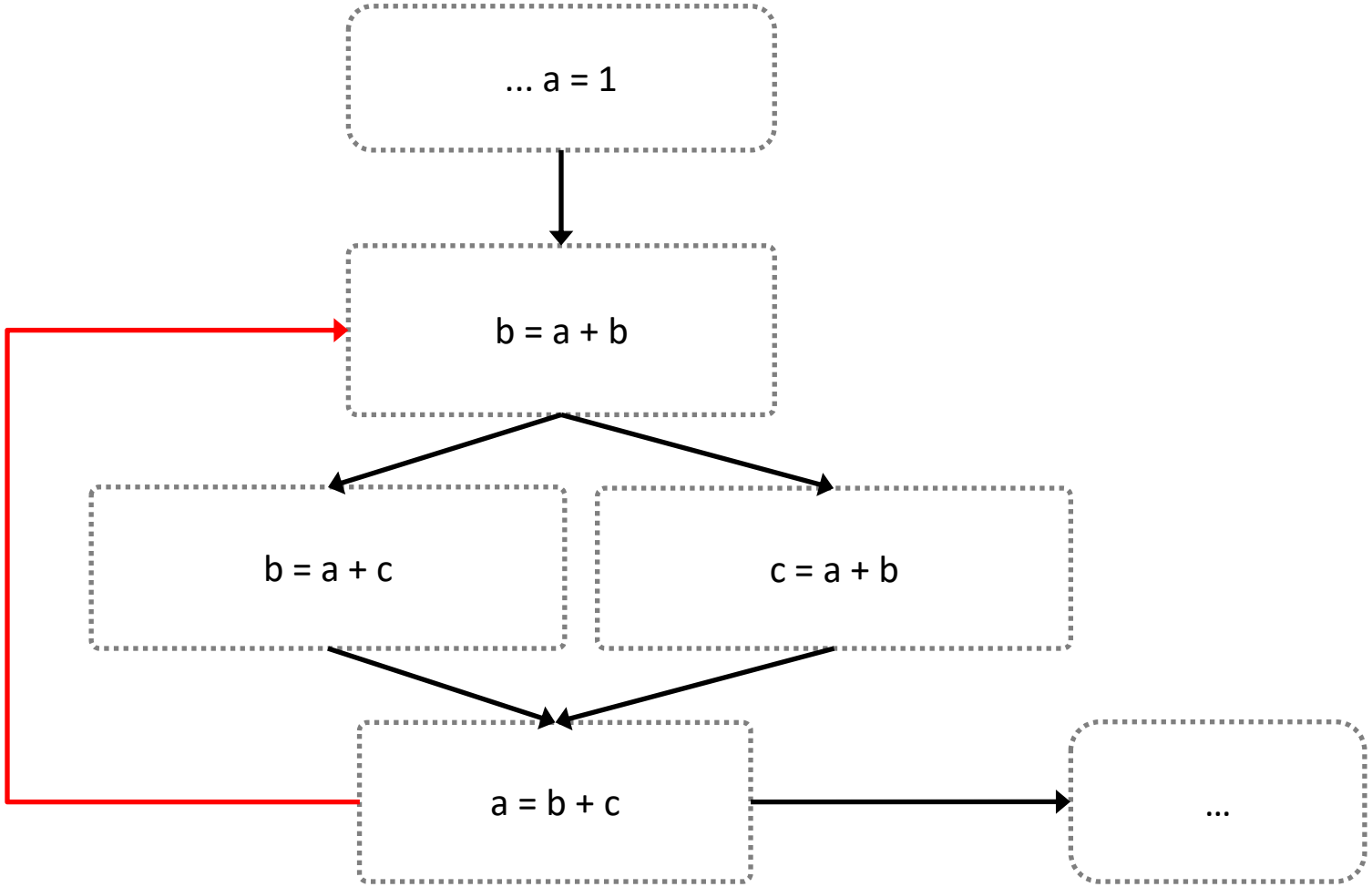
Global copy propagation / common subexpressions elimination

A simple example



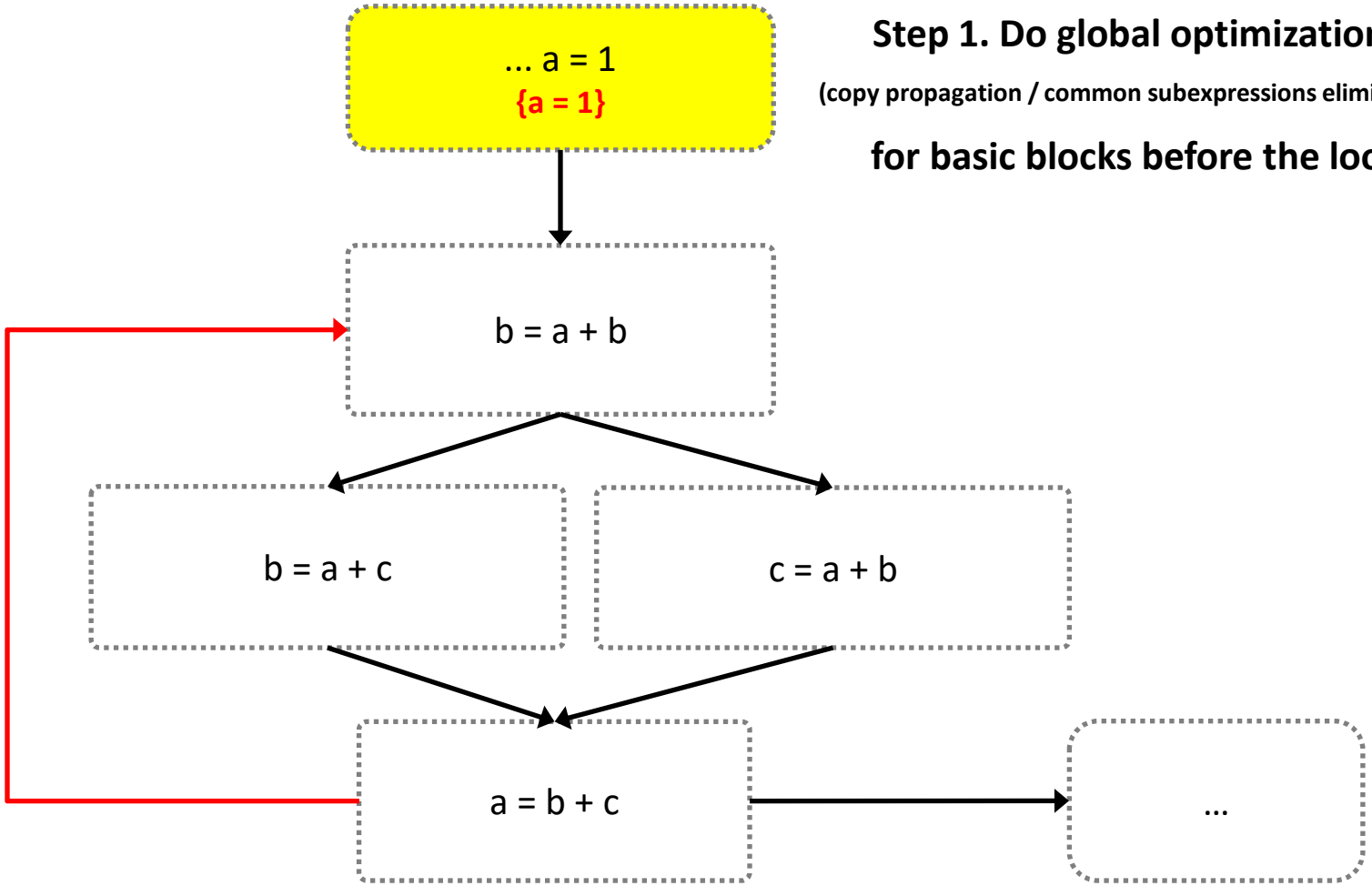
Global copy propagation / common subexpressions elimination

A complicated example with loop



Global copy propagation / common subexpressions elimination

A complicated example with loop



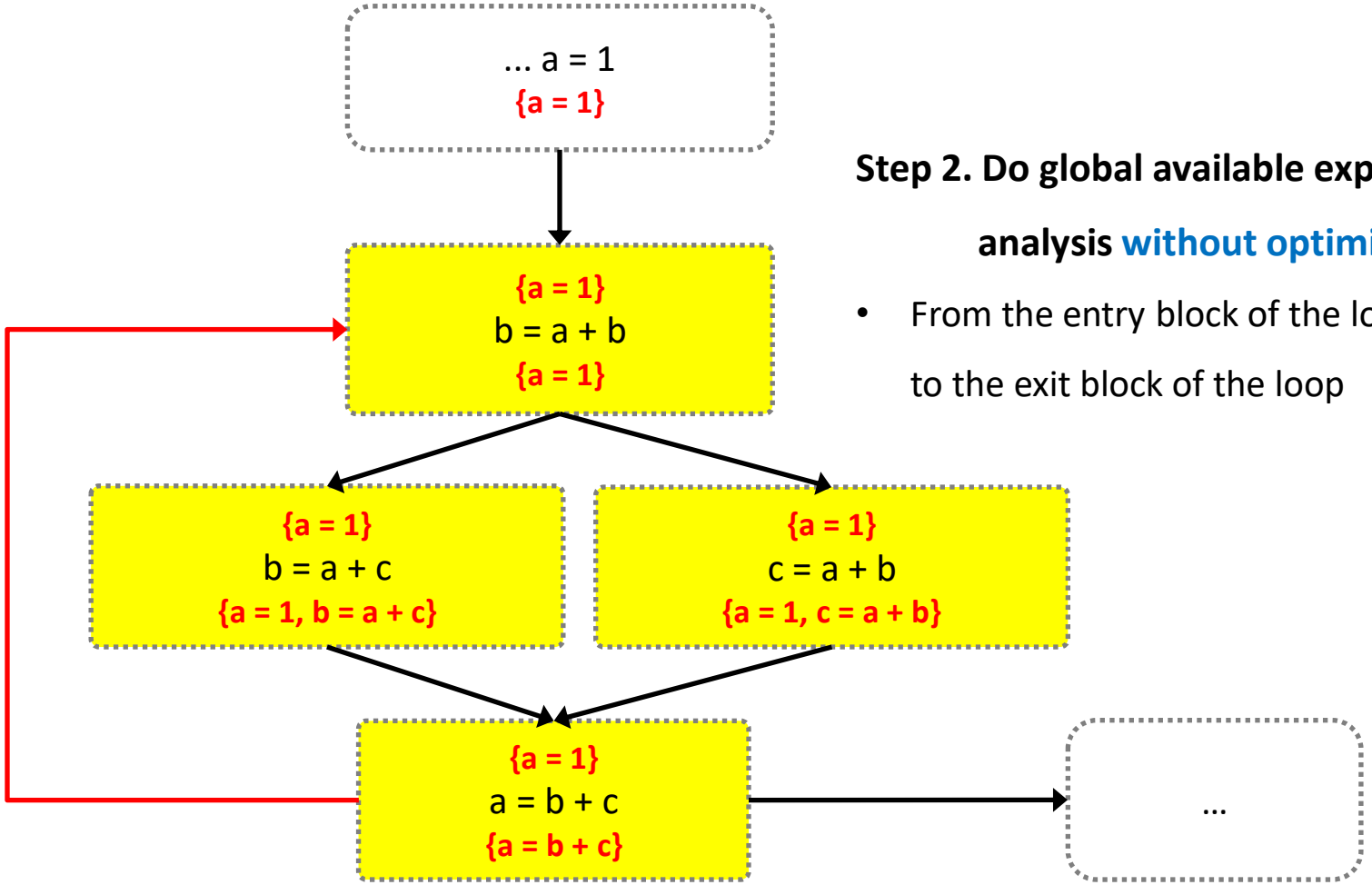
Step 1. Do global optimizations

(copy propagation / common subexpressions elimination)

for basic blocks before the loop

Global copy propagation / common subexpressions elimination

A complicated example with loop

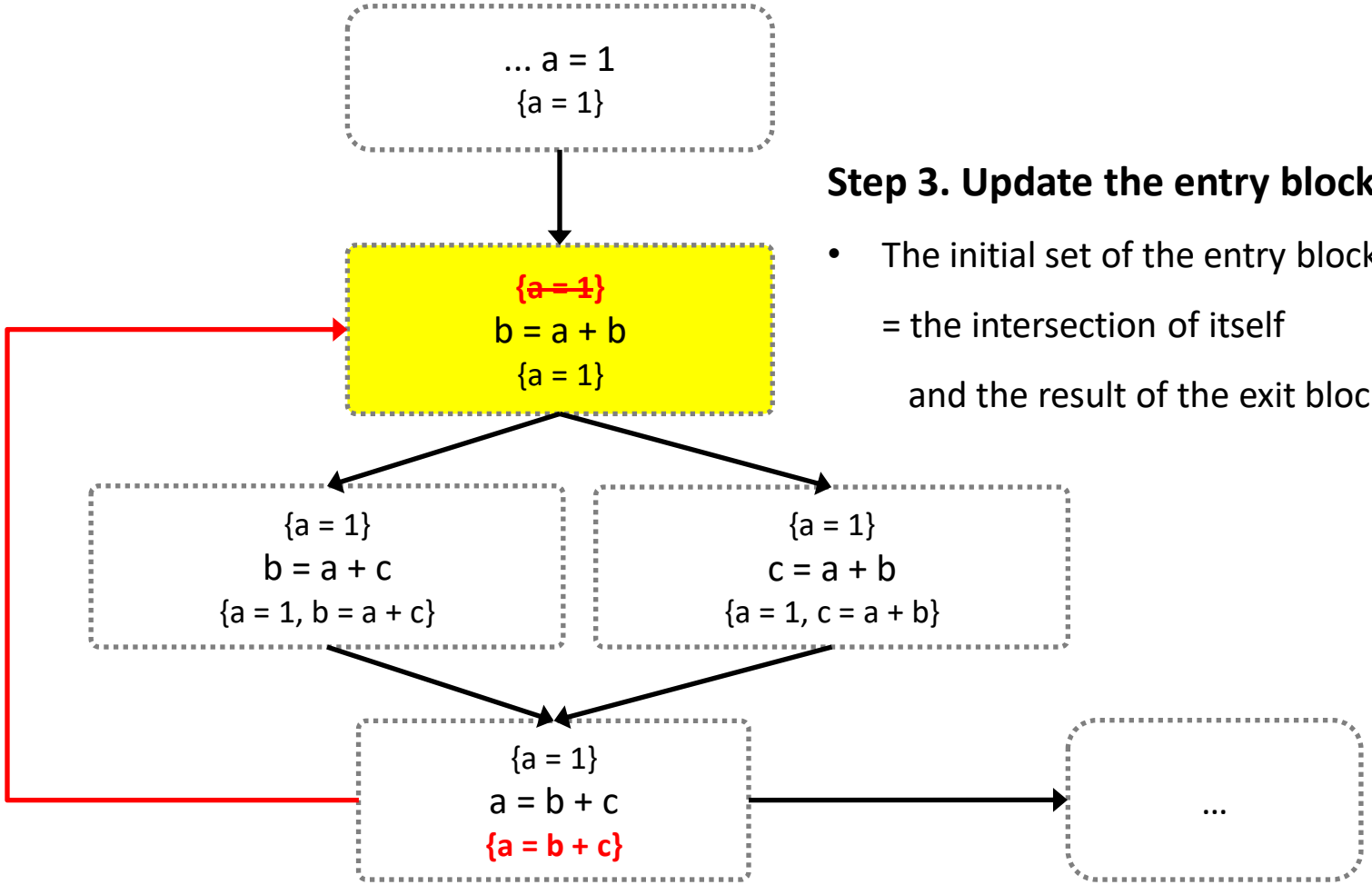


Step 2. Do global available expression analysis without optimizations

- From the entry block of the loop to the exit block of the loop

Global copy propagation / common subexpressions elimination

A complicated example with loop



Step 3. Update the entry block of the loop

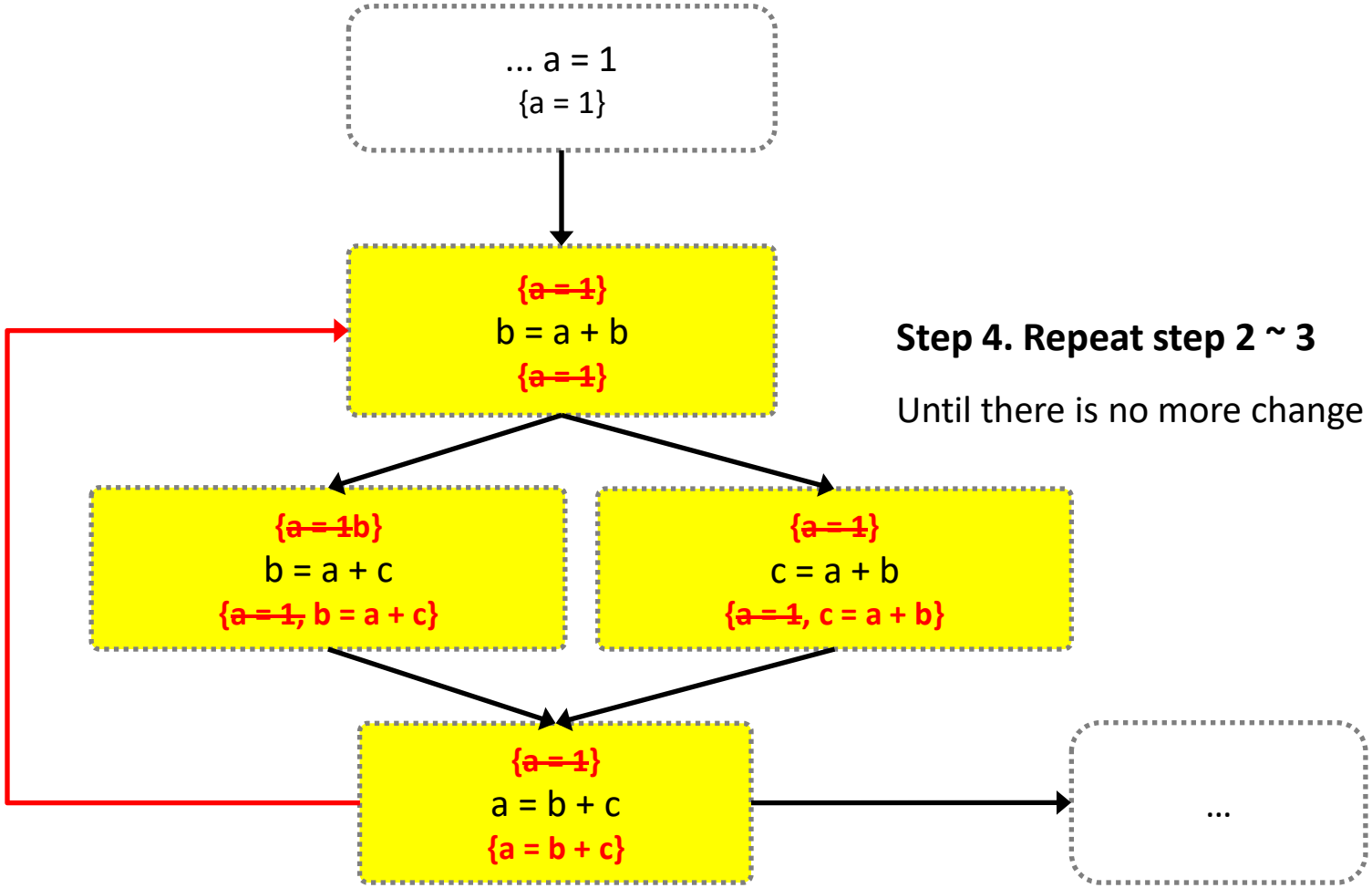
- The initial set of the entry block

  = the intersection of itself

  and the result of the exit block of the loop

Global copy propagation / common subexpressions elimination

A complicated example with loop

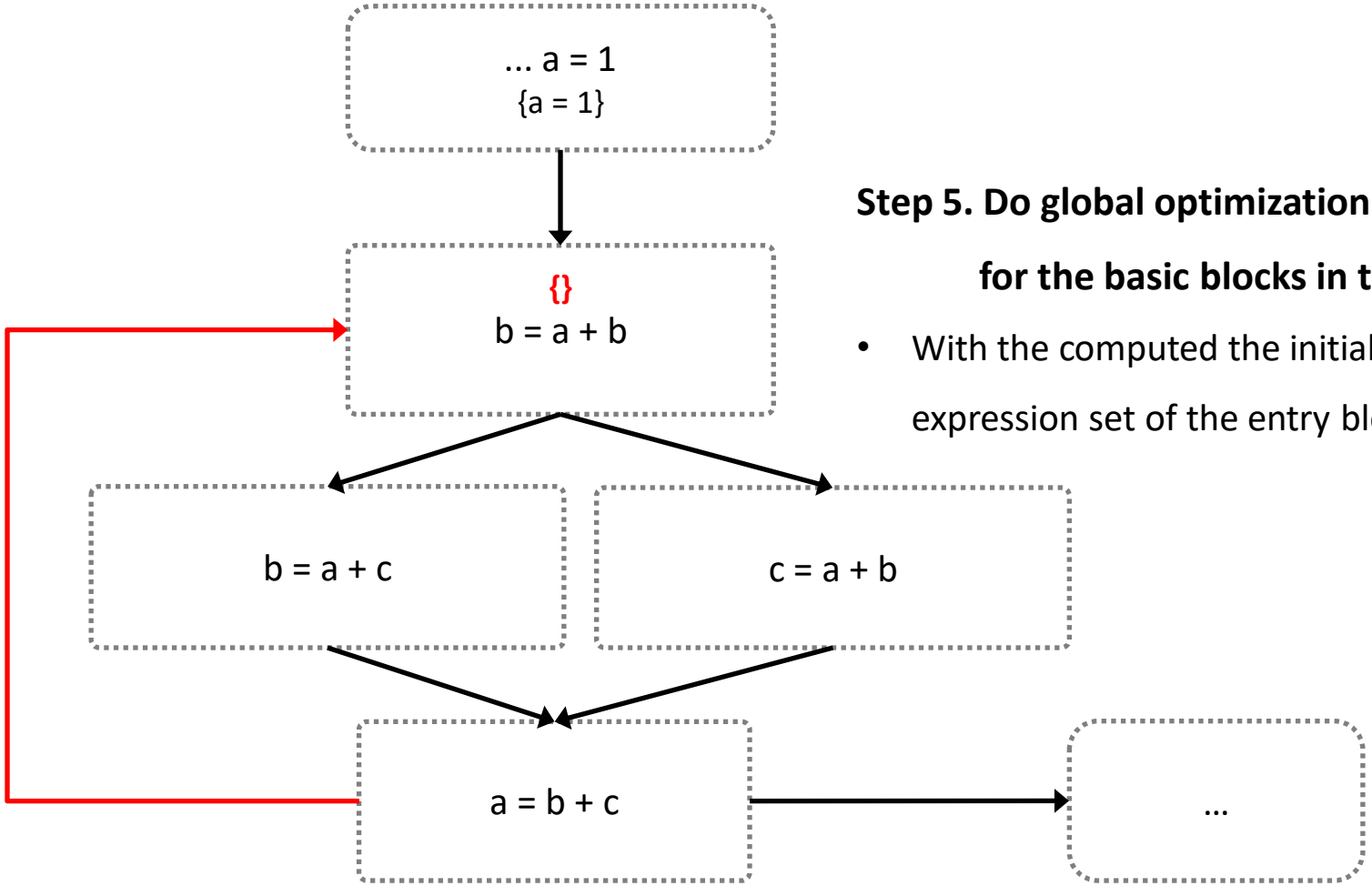


Step 4. Repeat step 2 ~ 3

Until there is no more change

Global copy propagation / common subexpressions elimination

A complicated example with loop

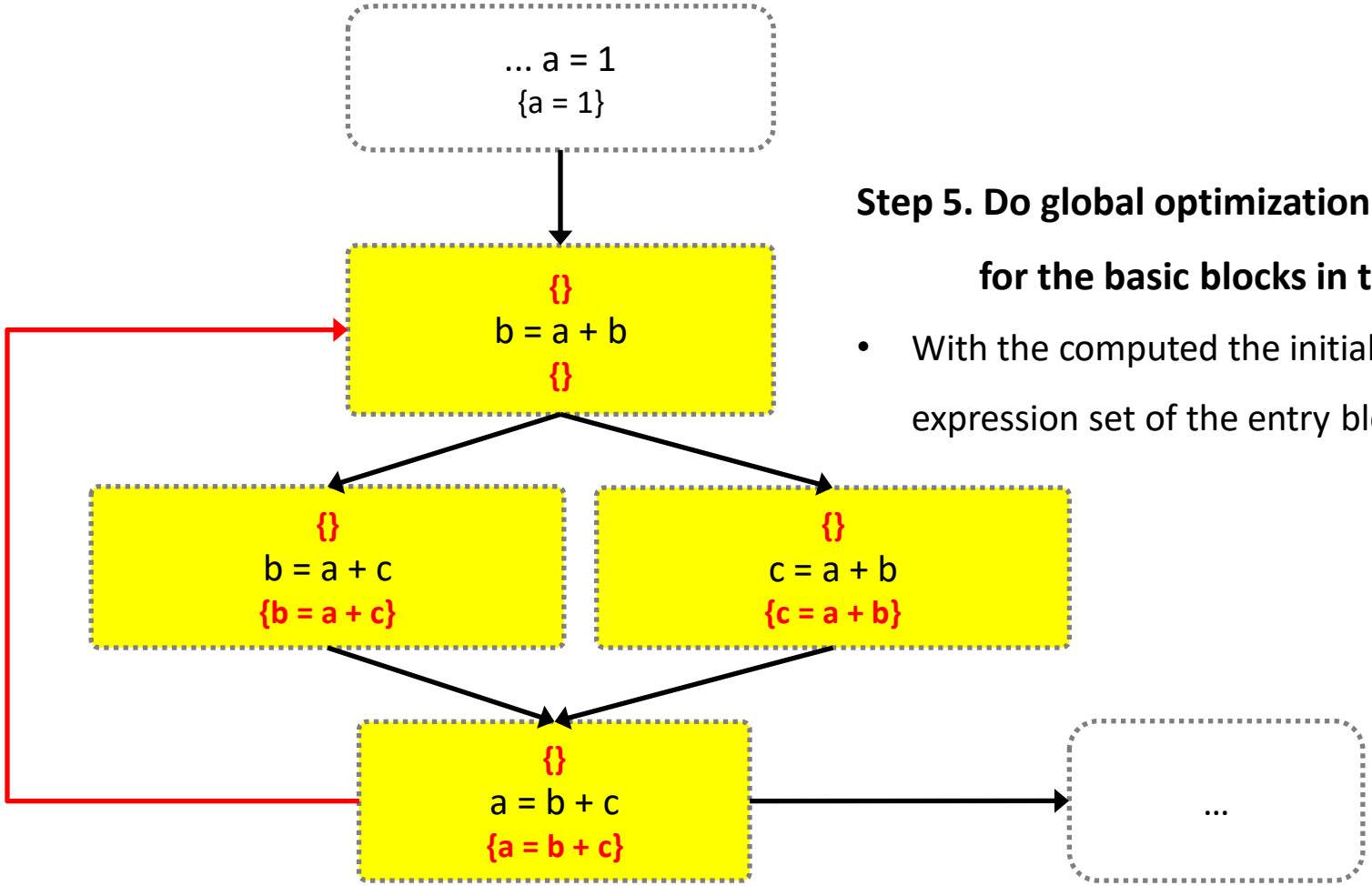


Step 5. Do global optimizations for the basic blocks in the loop

- With the computed the initial available expression set of the entry block

Global copy propagation / common subexpressions elimination

A complicated example with loop

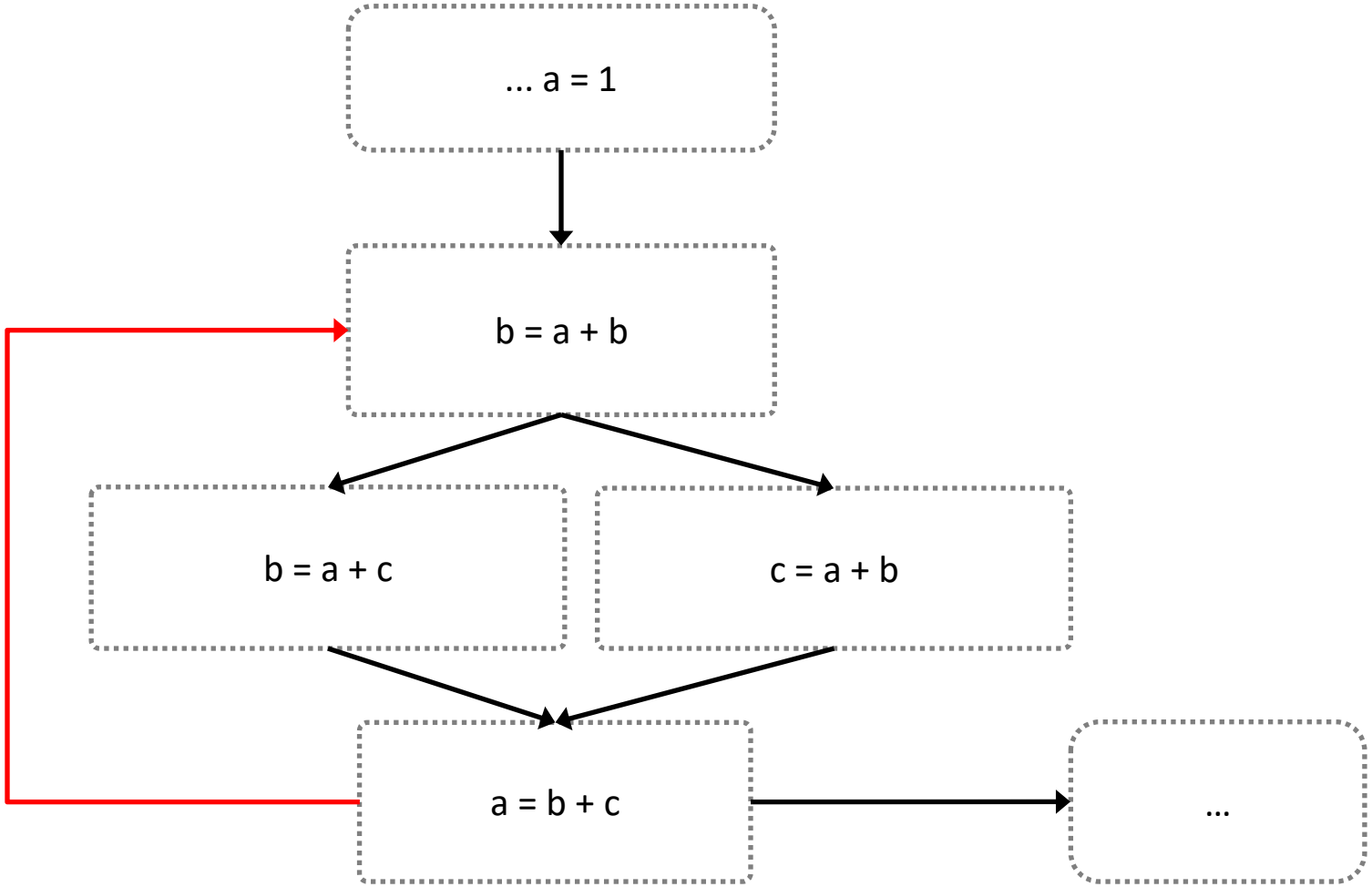


Step 5. Do global optimizations for the basic blocks in the loop

- With the computed the initial available expression set of the entry block

Global copy propagation / common subexpressions elimination

A complicated example with loop



Summary: global optimizations

Work on a control-flow graph as a whole

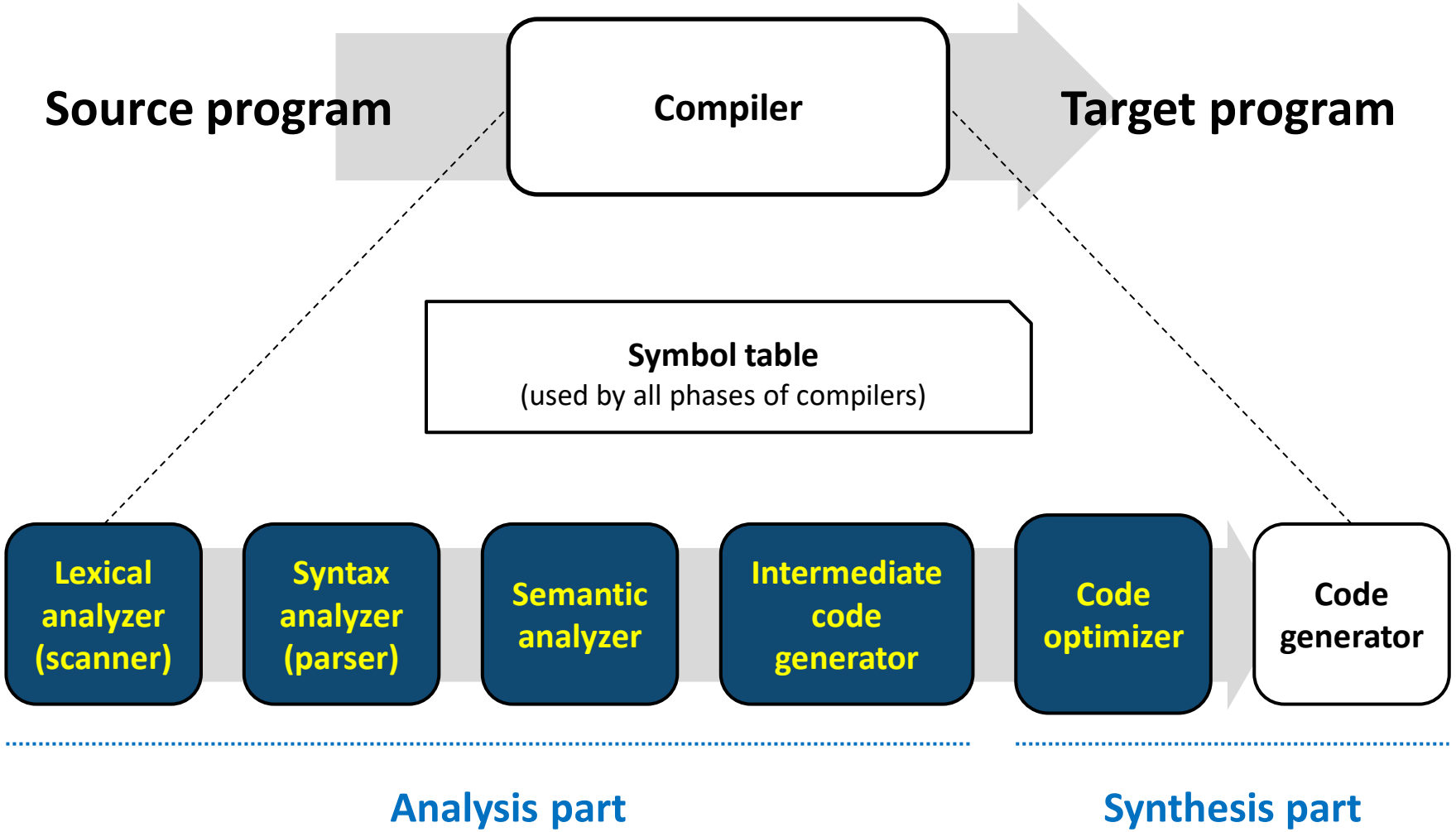
Many of the local optimization techniques can be applied globally

- Global copy/constant propagation
- Global dead code elimination

Some optimizations are possible in global analysis that aren't possible locally

e.g., code motion: moving code from one basic block into another to avoid unnecessary computations

Summary


Source program
Compiler
Target program
Symbol table
(used by all phases of compilers)
Lexical analyzer (scanner)
Syntax analyzer (parser)
Semantic analyzer
Intermediate code generator
Code optimizer
Code generator
Analysis part
Synthesis part