Dataflow - Defined Variables in Depth

Lorenzo Ceragioli

December 5, 2024

IMT Lucca

Recap: Defined Variables

Computed Value: Set of defined variables (Registers for the IR)

Analysis State: Associate a pair of values to each block (*in* and *out*)

Local Update: update the value associated with a block

- From the block itself: variables defined at the exit of the block are those defined when entering plus the ones defined by the block's commands
- From a block to the others: variables defined at beginning of a block are those defined **in every** preceding block

Global Update: all local updates until fixpoint

Then check that each instruction uses variables that are defined either at the beginning of the block or in the block before the current instruction.

Simplest: Defined Variables (a forward analysis)

Computed Value: $\mathcal{P}(R)$

Analysis State:

- Formally $dv : L \longrightarrow \mathcal{P}(R) \times \mathcal{P}(R)$
- More handy $dv_{in} : L \longrightarrow \mathcal{P}(R)$ and $dv_{out} : L \longrightarrow \mathcal{P}(R)$

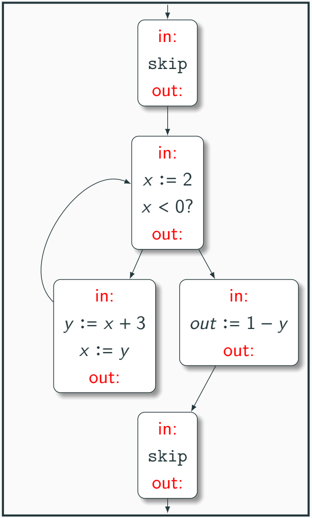
Local Update:

- $lub(dv_{out}(L)) = dv_{in}(L) \cup \{\text{variables defined in } L\}$
- $lucf(dv_{in}(L)) = \begin{cases} \{in \text{ (register for the input)}\} & \text{if } L \text{ is initial} \\ \bigcap_{(L',L) \in \text{CFG edges}} dv_{out}(L') & \text{otherwise} \end{cases}$

Global Update: $gu(dv_{in})(L) = lucf(dv_{in}(L))$ and $gu(dv_{out})(L) = lub(dv_{out}(L))$ until fixpoint

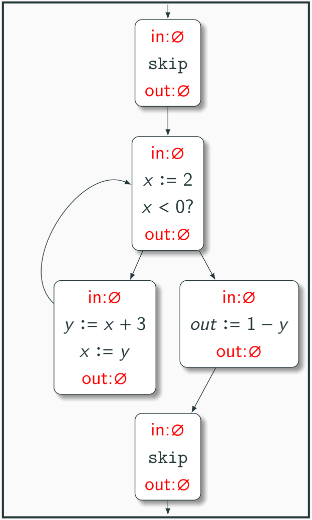
Then check each instruction in blocks.

Defined Variables – Example



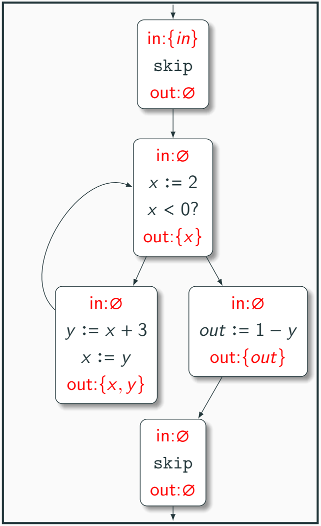
- The variable y in $out := y - 1$ is undefined!
- Notice that the analysis is very coarse grained, **we can do better!**

Defined Variables – Example



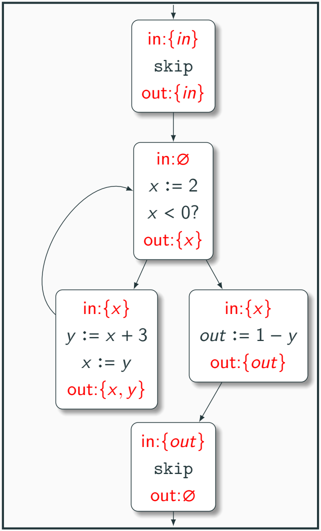
- The variable y in $out := y - 1$ is undefined!
- Notice that the analysis is very coarse grained, **we can do better!**

Defined Variables – Example



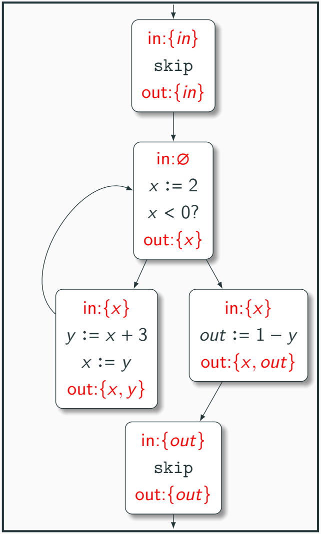
- The variable y in $out := y - 1$ is undefined!
- Notice that the analysis is very coarse grained, **we can do better!**

Defined Variables – Example



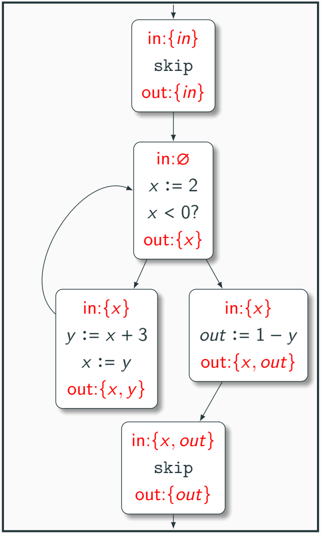
- The variable y in $out := y - 1$ is undefined!
- Notice that the analysis is very coarse grained, **we can do better!**

Defined Variables – Example



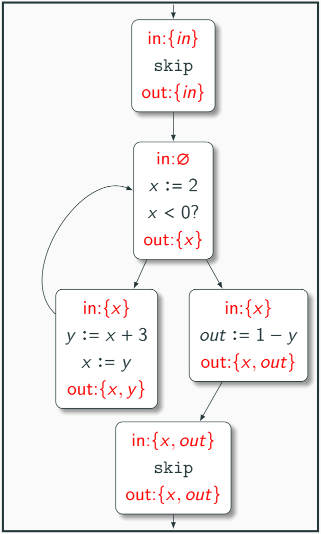
- The variable y in $out := y - 1$ is undefined!
- Notice that the analysis is very coarse grained, **we can do better!**

Defined Variables – Example



- The variable y in $out := y - 1$ is undefined!
- Notice that the analysis is very coarse grained, **we can do better!**

Defined Variables – Example



- The variable y in $out := y - 1$ is undefined!
- Notice that the analysis is very coarse grained, **we can do better!**

Recall: Correctness and Completeness of the Analysis

- **correctness** means that every variable that is **deemed defined by the analysis** is **actually defined**
  (recall, in each transition system obtained by computing the small-step semantics for a given input, when the execution reach the given instruction or block)
- the analysis always returning x_0 associating each block with the empty set, i.e. deeming that no variable is defined, is a correct (but useless) analysis
- **completeness** means that every variable that is **actually defined** is **deemed defined by the analysis**
- no analysis can be correct and complete for some properties – we must approximate

fixpointS

- our global update function *gu* defines correctness of the analysis
- every fixpoint ($\hat{x}$ such that $gu(\hat{x}) = \hat{x}$) is correct, none is complete
- the nearest fixpoint to a complete analysis is our best approximation!
- the least fixpoint $\hat{x}_{min}$ is smaller that the maximal fixpoint $\hat{x}_{max}$

$$x_0 \subseteq \hat{x}_{min} \subseteq \hat{x}_{max} \subseteq \text{actually defined variables}$$

How to Compute Fixpoints – Recap

Note, we have a finite CPO with top $\top$ and bottom $\bot$ (a finite lattice), and gu is monotone (and thus complete).

Our CPO is of functions $L \longrightarrow \mathcal{P}(R) \times \mathcal{P}(R)$ (assuming a finite R)

- $s_1 \sqsubseteq s_2$ if for any $l \in L$, $X_1 \subseteq X_2$ and $Y_1 \subseteq Y_2$ where $s_1(l) = (X_1, Y_1)$ and $s_2(l) = (X_2, Y_2)$
- $\bot$ is the function associating every label l with $(\emptyset, \emptyset)$
- $\top$ is the function associating every label l with (R, R)

Fixpoints for
$gu : (L \longrightarrow \mathcal{P}(R) \times \mathcal{P}(R)) \longrightarrow (L \longrightarrow \mathcal{P}(R) \times \mathcal{P}(R))$

Kleene's Theorem: $\hat{x}_{min} = \bigsqcup_n gu^n(\bot)$ $\qquad \hat{x}_{max} = \bigsqcap_n gu^n(\top)$

Exploiting Finiteness

Kleene's Theorem: $\hat{x}_{min} = \bigsqcup_n gu^n(\bot) \qquad \hat{x}_{max} = \bigsqcap_n gu^n(\top)$

For $\hat{x}_{min}$ we are actually computing the values

$\bot, gu(\bot), gu(gu(\bot))\ldots$ until we find $gu^n(\bot) = gu^{n+1}(\bot) = \hat{x}_{min}$

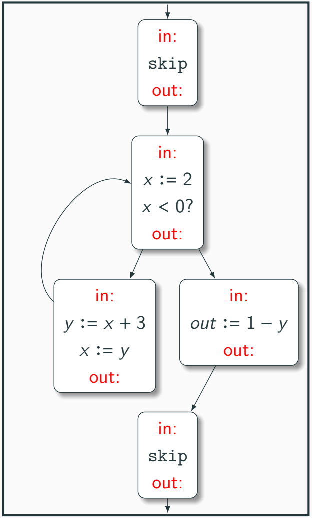
- we reach such a $gu^n(\bot)$ because the CPO is finite
- we avoid computing $\bigsqcup_n$ because:
  - $\bot \sqsubseteq gu(\bot)$ by definition of $\bot$
  - $gu^m(\bot) \sqsubseteq gu^{m+1}(\bot)$ for every m by monotonicity, hence

    $$\bot \sqsubseteq gu(\bot) \sqsubseteq gu(gu(\bot)) \ \ldots \ gu^{n-1}(\bot) \sqsubseteq gu^n(\bot)$$

  - $x \sqcup x' = x'$ if $x \sqsubseteq x'$

Warning: this is because of our domain, does not hold in general

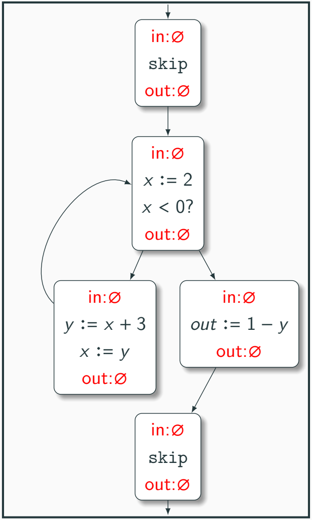
Defined Variables – Example



Procedure:

-
-
-
-
-

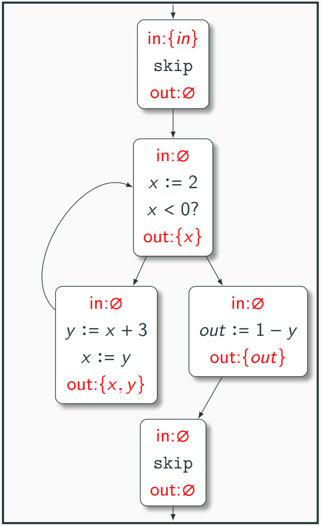
Defined Variables – Example



Procedure:

- We start with the $\perp$ of our CPO
-
-
-
-

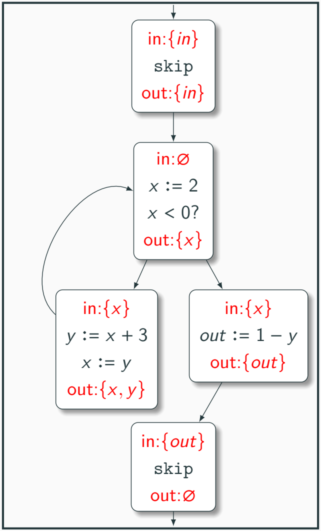
Defined Variables – Example



Procedure:

- We start with the $\perp$ of our CPO
- We compute $gu(\perp)$
-
-
-

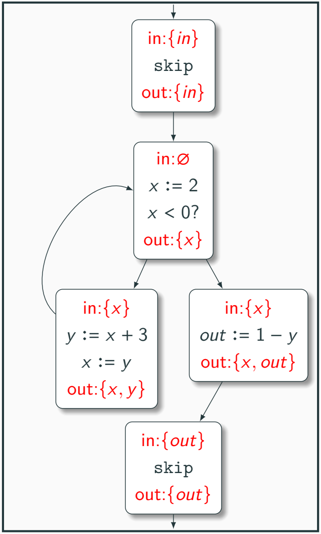
Defined Variables – Example



Procedure:

- We start with the $\perp$ of our CPO
- We compute $gu(\perp)$
- Then $gu(gu(\perp))$
-
-

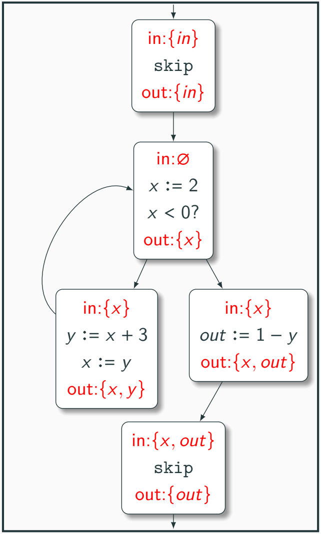
Defined Variables – Example



Procedure:

- We start with the $\perp$ of our CPO
- We compute $gu(\perp)$
- Then $gu(gu(\perp))$
- ...
-

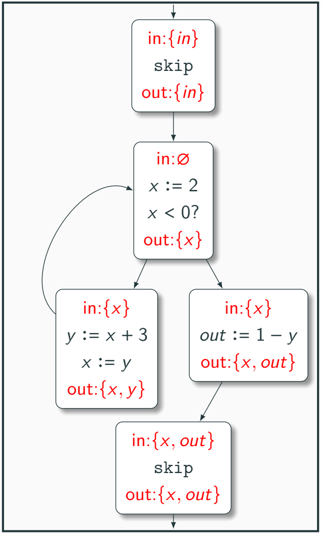
Defined Variables – Example



Procedure:

- We start with the $\bot$ of our CPO
- We compute $gu(\bot)$
- Then $gu(gu(\bot))$
- ...
-

Defined Variables – Example



Procedure:

- We start with the $\perp$ of our CPO
- We compute $gu(\perp)$
- Then $gu(gu(\perp))$
- ...
- We reach a fixpoint, guaranteed to be the minimal one!

Computing the Greatest Fixpoint

Kleene's Theorem: $\hat{x}_{min} = \bigsqcup_n gu^n(\bot)$ $\quad$ $\hat{x}_{max} = \bigsqcap_n gu^n(\top)$

For $\hat{x}_{max}$ we compute

$\top, gu(\top), gu(gu(\top))$... until we find $gu^n(\top) = gu^{n+1}(\top) = \hat{x}_{max}$

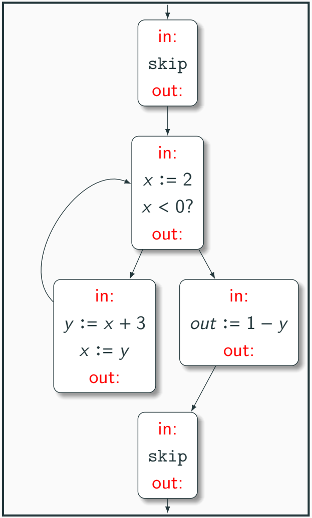
- we reach such a $gu^n(\top)$ because the CPO is finite
- we avoid computing $\bigsqcap_n$ because:
  - $gu(\top) \sqsubseteq \top$ by definition of $\top$
  - $gu^{m+1}(\top) \sqsubseteq gu^m(\top)$ for every m by monotonicity, hence

    $$\top \sqsupseteq gu(\top) \sqsupseteq gu(gu(\top)) \ \ldots \ gu^{n-1}(x) \sqsupseteq gu^n(x)$$

  - $x \sqcap x' = x'$ if $x \sqsupseteq x'$

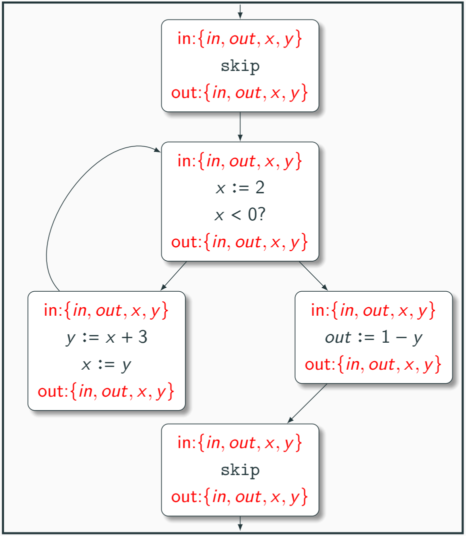
Warning: this is because of our domain, does not hold in general

Defined Variables – A better approximation



Procedure:

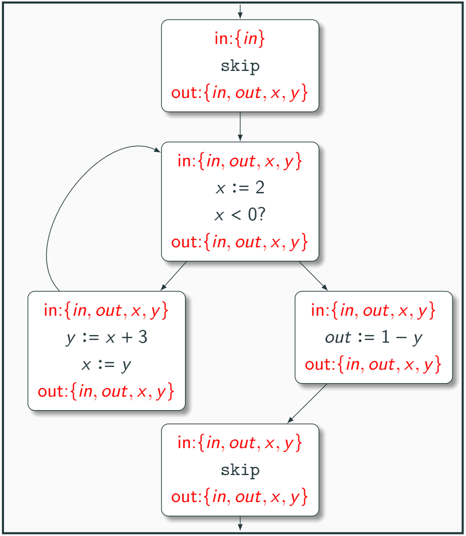
Defined Variables – A better approximation



Procedure:

- We start with the $\top$ of our CPO!

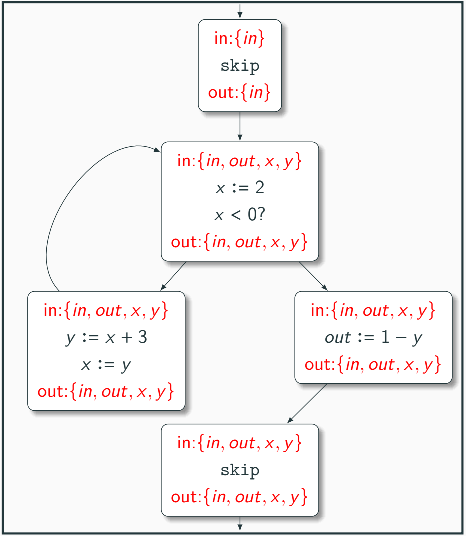
Defined Variables – A better approximation



Procedure:

- We start with the $\top$ of our CPO!
- We compute $gu(\top)$
-
-
-

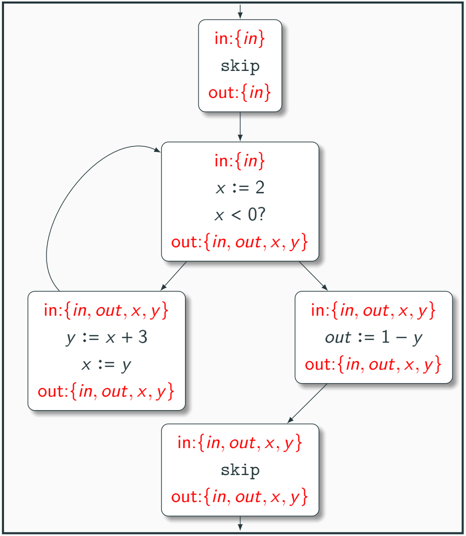
Defined Variables – A better approximation



Procedure:

- We start with the $\top$ of our CPO!
- We compute $gu(\top)$
- Then $gu(gu(\top))$

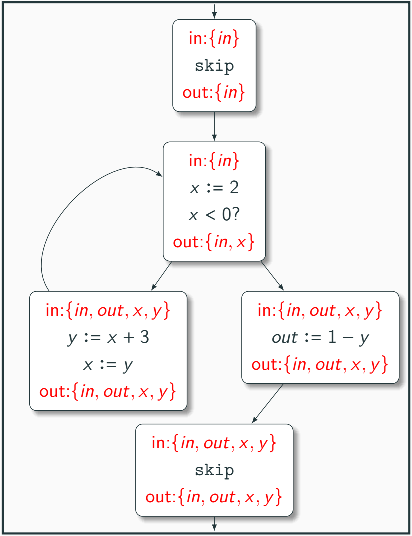
Defined Variables – A better approximation



Procedure:

- We start with the $\top$ of our CPO!
- We compute $gu(\top)$
- Then $gu(gu(\top))$
- ...
-

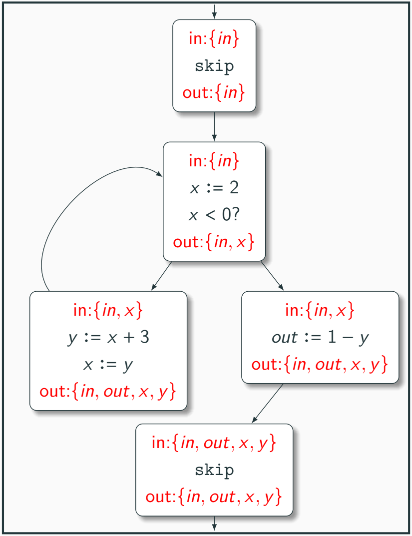
Defined Variables – A better approximation



Procedure:

- We start with the $\top$ of our CPO!
- We compute $gu(\top)$
- Then $gu(gu(\top))$
- ...
-

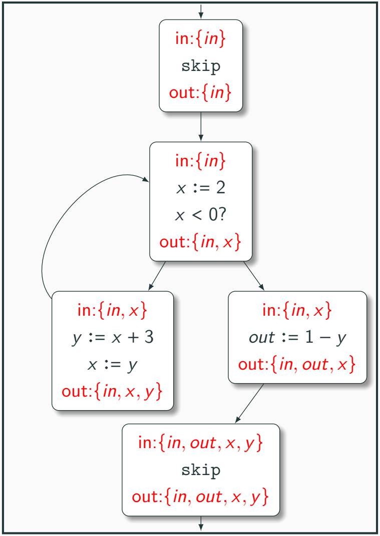
Defined Variables – A better approximation



Procedure:

- We start with the $\top$ of our CPO!
- We compute $gu(\top)$
- Then $gu(gu(\top))$
- ...
-

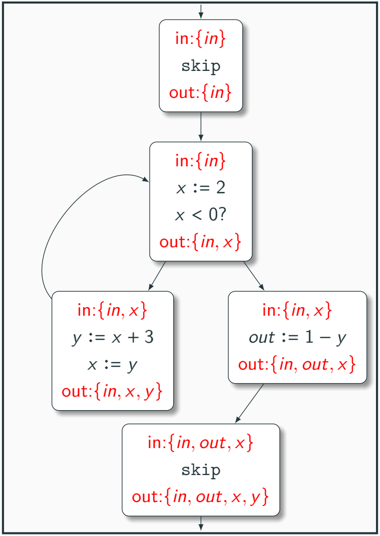
Defined Variables – A better approximation



Procedure:

- We start with the $\top$ of our CPO!
- We compute $gu(\top)$
- Then $gu(gu(\top))$
- ...
-

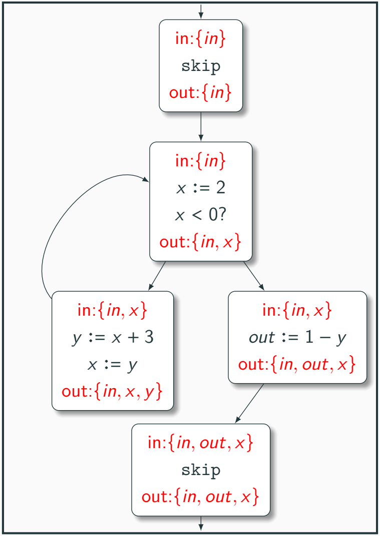
Defined Variables – A better approximation



Procedure:

- We start with the $\top$ of our CPO!
- We compute $gu(\top)$
- Then $gu(gu(\top))$
- ...
-

Defined Variables – A better approximation



Procedure:

- We start with the $\top$ of our CPO!
- We compute $gu(\top)$
- Then $gu(gu(\top))$
- ...
- We reach a fixpoint, guaranteed to be the maximal one!

Why Greatest Fixpoint for Defined Variables

Safety defines when an analysis is acceptable for us:

- Definite Variables is a "definite" analysis $\longrightarrow$ safety is correctness
  - we are happy only if all variables deemed defined by the analysis are actually defined
  - some of them may be deemed **not defined** incorrectly, but that is acceptable
  - (sometimes we will refuse to execute programs that are correct but we will never execute a faulty one)
- all fixpoints are correct (safe), we want the maximal which is the nearest to completeness

Why Least Fixpoint for Live Variables

Safety defines when an analysis is acceptable for us:

- Live Variables is a "possible" analysis $\longrightarrow$ safety is completeness
  - we are happy only if all variables that are actually live are deemed live by the analysis
  - some of them may be deemed **live** incorrectly, but that is acceptable
  - (acceptable because we use the information for guiding optimization: we will treat variables deemed live as still important for the program. Even if sometimes they are not really important, the optimization still preserves the semantics of the program)
- All fixpoints are complete (safe), we want the minimal which is the nearest to correctness

Project Fragment

- Write a function for checking that no register is ever used before being initialized with some value in a MiniRISC CFG (mind the initial register *in* which is always initialized, and *out* which is always used – if you prefer, you can perform this task on the MiniImp CFG of the program)
- **Edit:** better to use the greatest fixpoint, but the least is fine