DEI DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

U.PORTO FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

Compilers Design and Implementation

Traditional Optimizations

Algebraic Simplification, Copy Propagation, & Constant Propagation

DEI DEPARTAMENTO DE ENGENHARIA INFORMÁTICA
U.PORTO FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

Outline

- Overview of Control-Flow/Data-Flow Analysis
- Algebraic Simplification
- Copy Propagation
- Constant Propagation

Available Expressions

- Domain
  - Set of Expressions
- Data-Flow Direction
  - Forward: Out values computed based on In values
- Data-Flow Functions:
  - OUT = gen $\cup$(IN - kill)
  - gen = { exp | exp is calculated in the Basic Block }
  - kill = { exp | $\exists$a variable v $\in$ exp that is defined in the Basic Block }
- Meet Operation
  - IN = $\cap$ OUT for all the predecessors of a Basic Block
- Initial values
  - Empty Set

DU-Chain (Reaching Definitions)

- Domain
  - Set of definitions
- Data-Flow Direction
  - Forward: Out values computed based on In values
- Data-Flow Transfer Function
  - OUT = Gen ∪(IN - Kill)
  - Gen = { x | x is defined in the Basic Block/Statement }
  - Kill = { x | LHS var. of x is redefined in the Basic Block/Statement }
- Meet Operation
  - IN = ∪ OUT for all the predecessors of a Basic Block
- Initial Values
  - Empty set

Analysis Framework

- Control-Flow Analysis
  - Identify the Structure of the Program
  - Help build Data-Flow Analysis
- Data-Flow Analysis
  - Framework to find information needed for optimizations
  - So far we found
    - Available Expressions
    - UD and DU chains
  - Now let use the information to do something interesting!!!

"Optimizations"

- Each Optimization is very simple
  - Reduces Complexity
- Multiple Optimizations are needed
- Optimizations may need to be applied multiple times

Outline

- Overview of Control-Flow/Data-Flow Analysis
- Algebraic Simplification
- Copy Propagation
- Constant Propagation

Algebraic Simplification

- Apply our knowledge from algebra, number theory etc. to simplify expressions

Algebraic Simplification

- Apply our knowledge from algebra, number theory etc. to simplify expressions
- Example
  - a + 0 ⇒ a
  - a * 1 ⇒ a
  - a / 1 ⇒ a
  - a * 0 ⇒ 0
  - 0 - a ⇒ -a
  - a + (-b) ⇒ a - b
  - -(-a) ⇒ a

Algebraic Simplification

- Apply our knowledge from algebra, number theory etc. to simplify expressions
- Example
  - $a \wedge \text{true} \Rightarrow a$
  - $a \wedge \text{false} \Rightarrow \text{false}$
  - $a \vee \text{true} \Rightarrow \text{true}$
  - $a \vee \text{false} \Rightarrow a$

Algebraic Simplification

- Apply our knowledge from algebra, number theory etc. to simplify expressions
- Example
  - a ^ 2 ⇒ a*a
  - a * 2 ⇒ a + a
  - a * 8 ⇒ a << 3

Algebraic Simplification Opportunities

- In the Code
  - Programmers are lazy to simplify expressions
  - Programs are more readable with full expressions
- After Compiler Expansion
  - Example: Array read A[8][12] will get expanded to
  - *(Abase + 4*(12 + 8*256)) which can be simplified
- After other Optimizations

Usefulness of Algebraic Simplification

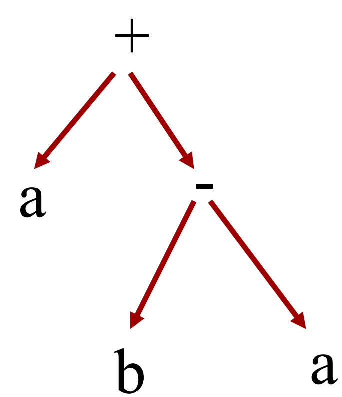
- Reduces the number of instructions
- Uses less expensive instructions
- Enable other optimizations

Implementation

- Not a Data-Flow optimization!
- Find candidates that matches the simplification rules and simplify the expression trees
- Candidates may not be obvious

Implementation

- Not a Data-Flow optimization! Why?
- Find candidates that matches the simplification rules and simplify the expression trees
- Candidates may not be obvious
  - Example
    a + b - a



U.PORTO FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO
DEI DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

Use Knowledge about Operators

- Commutative Operators
  - a **op** b = b **op** a
  -
- Associative Operators
  - (a **op** b) **op** c = b **op** (a **op** c)

Canonical Format

- Put expression trees into a canonical format
  - Sum of multiplicands
  - Example
    (a + 3) * (a + 8) * 4 ⇒ 4*a*a + 44*a + 96

Effects on the Numerical Stability

- Some algebraic simplifications may produce incorrect results

Effects on the Numerical Stability

- Some algebraic simplifications may produce incorrect results
- Example
  - (a / b)*0 + c

Effects on the Numerical Stability

- Some algebraic simplifications may produce incorrect results
- Example
  - (a / b)*0 + c
  - we can simplify this to c

Effects on the Numerical Stability

- Some algebraic simplifications may produce incorrect results
- Example
  - (a / b)*0 + c
  - we can simplify this to c
  - But what about when b = 0? should be a exception, …

Outline

- Overview of Control-Flow/Data-Flow Analysis
- Algebraic Simplification
- Copy Propagation
- Constant Propagation

Copy Propagation

- Bypass Multiple Copying
  - propagate a value directly to its use
- Example

```
a = b + c
d = a
e = d
f = d + 2*e + c
```

Copy Propagation

- Bypass Multiple Copying
  - propagate a value directly to its use
- Example

```
a = b + c
d = a
e = d
f = d + 2*e + c
```

Copy Propagation

- Bypass Multiple Copying
  - propagate a value directly to its use
- Example

```
a = b + c
d = a
e = a
f = d + 2*e + c
```

Copy Propagation

- Bypass Multiple Copying
  - propagate a value directly to its use
- Example

```
a = b + c
d = a
e = a
f = d + 2*e + c
```

Copy Propagation

- Bypass Multiple Copying
  - propagate a value directly to its use
- Example

```
a = b + c
d = a
e = a
f = a + 2*e + c
```

Copy Propagation

- Bypass Multiple Copying
  - propagate a value directly to its use
- Example

```
a = b + c
d = a
e = a
f = a + 2*e + c
```


DEI DEPARTAMENTO DE ENGENHARIA INFORMÁTICA
U.PORTO FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

Copy Propagation

- Bypass Multiple Copying
  - propagate a value directly to its use
- Example

```
a = b + c
d = a
e = a
f = a + 2*a + c
```

Copy Propagation

- Bypass Multiple Copying
  - propagate a value directly to its use
- Example

```
a = b + c
d = a
e = a
f = a + 2*a + c
```

Opportunities for Copy Propagation

- Exists in User Code
- After other Optimizations
  - Example: Algebraic simplification

Advantages of Copy Propagation

- Leads to further algebraic simplification

Advantages of Copy Propagation

- Leads to further algebraic simplification

- Example

```
a = b + c
d = a
e = a
f = a + 2*a + c
```

U.PORTO FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO
DEI DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

Advantages of Copy Propagation

- Leads to further algebraic simplification

- Example

```
a = b + c
d = a
e = a
f = a + 2*a + c
```

Advantages of Copy Propagation

- Leads to further algebraic simplification

- Example

```
a = b + c
d = a
e = a
f = 3*a + c
```

Advantages of Copy Propagation

- Reduce Instructions by Eliminating Copy Operations
  - Creates dead code that can be eliminated

Advantages of Copy Propagation

- Reduce Instructions by Eliminating Copy Operations
  - Creates dead code that can be eliminated
- Example

```
a = b + c
d = a
e = a
f = 3*a + c
```

Advantages of Copy Propagation

- Reduce Instructions by Eliminating Copy Operations
  - Creates dead code that can be eliminated
- Example

```
a = b + c



f = 3*a + c
```

How to Perform Copy Propagation ?

- At each RHS Expression
  and for each variable `v` used in the RHS expression
  - if the variable v is defined by a statement of the form `v = u`
  - replace the variable `v` by `u`
- At each Point of the Program Need to Know
  - The variables that are equal
  - Track equal variables by keeping tuples of the form `<u,v>` in a set iff `v = u` at that point in the program (`u`, `v` are variables)

How to Perform Copy Propagation ?

- An assignment of `v = u` is still valid at a given point of the execution if and only if
  - An statement of `v = u` occurs in <u>every execution path</u> that reaches the current point
  - The variable `v` is not redefined in ***any these execution paths*** between the assign statement and the current point
  - The variable `u` is not redefined in ***any these execution paths*** between the assign statement and the current point

- A Data-Flow Problem !!!

Copy Propagation Data-Flow Problem

- Domain
  - set of tuples `<v,u>` representing a statement `v = u`
- Data-Flow Direction
  - Forward
- Data-Flow Function
  - OUT = Gen ∪ (IN - Kill)
  - Gen = { <v,u> | v = u is the statement }
  - Kill = { <v,u> | LHS var. of an assignment stmt. is either v or u }
- Meet Operation
  - IN = ⋂ OUT
- Initial Values
  - Empty Set

Example

b = a

c = b + 1

d = b

b = d + c

b = d

DEI DEPARTAMENTO DE ENGENHARIA INFORMÁTICA
U.PORTO FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

Example

gen = { <v,u> | v = u is the statement }
kill = { <v,u> | LHS var. of the assignment stmt. is either v or u }

b = a

c = b + 1

d = b

b = d + c

b = d

Example

gen = { <v,u> | v = u is the statement }
kill = { <v,u> | LHS var. of the assignment stmt. is either v or u }

	gen = { <b,a> }
b = a	
	gen = { }
c = b + 1	
	gen = { <d,b> }
d = b	
	gen = { }
b = d + c	
b = d	gen = { <b,d> }

Example

gen = { <v,u> | v = u is the statement }
kill = { <v,u> | LHS var. of the assignment stmt. is either v or u }

b = a	gen = { <b,a> }	kill = { <d,b>, <b,d> , <b,a>, <a,b>}
c = b + 1	gen = { }	kill = { <c>}
d = b	gen = { <d,b> }	kill = { <b,d>, <d,b> }
b = d + c	gen = { }	kill = { <a,b>, <b,a> , <d,b> , <b,d> }
b = d	gen = { <b,d> }	kill = { <a,b>, <b,a> , <d,b>, <b,d> }

Example

gen = { <v,u> | v = u is the statement }
kill = { <v,u> | LHS var. of the assignment stmt. is either v or u }

Statement	gen / kill
b = a	gen = { <b,a> } kill = { <d,b>, <b,d> , <b,a>, <a,b>}
c = b + 1	gen = { } kill = { <c>}
d = b	gen = { <d,b> } kill = { <b,d>, <d,b> }
b = d + c	gen = { } kill = { <a,b>, <b,a> , <d,b> , <b,d> }
b = d	gen = { <b,d> } kill = { <a,b>, <b,a> , <d,b>, <b,d> }

Kills any tuple with <c>

Example

$$OUT = gen \cup (IN - kill)$$

b = a	gen = { <b,a> } kill = { <d,b>, <b,d> , <b,a>, <a,b>}
c = b + 1	gen = { } kill = {<c> }
d = b	gen = { <d,b> } kill = { <b,d>, <d,b> }
b = d + c	gen = { } kill = { <a,b>, <b,a> , <d,b> , <b,d> }
b = d	gen = { <b,d> } kill = { <a,b>, <b,a> , <d,b>, <b,d> }

Example

OUT = gen ∪ (IN - kill)

b = a	gen = { <b,a> } kill = { <d,b>, <b,d> , <b,a>, <a,b>}	**IN = { }** **OUT = { <b,a> }**
c = b + 1	gen = { } kill = { <c>}	
d = b	gen = { <d,b> } kill = { <b,d>, <d,b> }	
b = d + c	gen = { } kill = { <a,b>, <b,a> , <d,b> , <b,d> }	
b = d	gen = { <b,d> } kill = { <a,b>, <b,a> , <d,b>, <b,d> }	

Example

OUT = gen ∪ (IN - kill)

IN = { }

b = a
gen = { <b,a> }
kill = { <d,b>, <b,d> , <b,a>, <a,b>}

IN = { <b,a> }

c = b + 1
gen = { }
kill = {<c> }

OUT = { <b,a> }

d = b
gen = { <d,b> }
kill = { <b,d>, <d,b> }

b = d + c
gen = { }
kill = { <a,b>, <b,a> , <d,b> , <b,d> }

b = d
gen = { <b,d> }
kill = { <a,b>, <b,a> , <d,b>, <b,d> }

Example

OUT = gen ∪ (IN - kill)

Statement	gen / kill	IN / OUT
		IN = { }
b = a	gen = { <b,a> } kill = { <d,b>, <b,d> , <b,a>, <a,b>}	
		IN = { <b,a> }
c = b + 1	gen = { } kill = {<c> }	
		IN = { <b,a> }
d = b	gen = { <d,b> } kill = { <b,d>, <d,b> }	
		OUT = { <b,a>, <d,b> }
b = d + c	gen = { } kill = { <a,b>, <b,a> , <d,b> , <b,d> }	
b = d	gen = { <b,d> } kill = { <a,b>, <b,a> , <d,b>, <b,d> }	

Example

OUT = gen ∪ (IN - kill)

		IN = { }
b = a	gen = { <b,a> } kill = { <d,b>, <b,d> , <b,a>, <a,b>}	
		IN = { <b,a> }
c = b + 1	gen = { } kill = {<c> }	
		IN = { <b,a> }
d = b	gen = { <d,b> } kill = { <b,d>, <d,b> }	
		IN = { <b,a>, <d,b> }
b = d + c	gen = { } kill = { <a,b>, <b,a> , <d,b> , <b,d> }	
		OUT = { }
b = d	gen = { <b,d> } kill = { <a,b>, <b,a> , <d,b>, <b,d> }	

Example

OUT = gen ∪ (IN - kill)

IN = { }

b = a
gen = { <b,a> }
kill = { <d,b>, <b,d> , <b,a>, <a,b>}

IN = { <b,a> }

c = b + 1
gen = { }
kill = { <c>}

IN = { <b,a> }

d = b
gen = { <d,b> }
kill = { <b,d>, <d,b> }

IN = { <b,a>, <d,b> }

b = d + c
gen = { }
kill = { <a,b>, <b,a> , <d,b> , <b,d> }

IN = { }

b = d
gen = { <b,d> }
kill = { <a,b>, <b,a> , <d,b>, <b,d> }

OUT = { <b,d> }

Example

	gen = { <b,a> }	{ }
b = a	kill = { <d,b>, <b,d> , <b,a>, <a,b>}	
		{ <b,a> }
	gen = { }	
c = b + 1	kill = { <c>}	
		{ <b,a> }
	gen = { <d,b> }	
d = b	kill = { <b,d>, <d,b> }	
		{ <b,a>, <d,b> }
	gen = { }	
b = d + c	kill = { <a,b>, <b,a> , <d,b> , <b,d> }	
		{ }
	gen = { <b,d> }	
b = d	kill = { <a,b>, <b,a> , <d,b>, <b,d> }	**{ <b,d> }**

Example

Statement	gen / kill	Set
		{ }
b = a	gen = { <b,a> } kill = { <d,b>, <b,d> , <b,a>, <a,b>}	
		{ <b,a> }
c = b + 1	gen = { } kill = {<c> }	
		{ <b,a> }
d = b	gen = { <d,b> } kill = { <b,d>, <d,b> }	
		{ <b,a>, <d,b> }
b = d + c	gen = { } kill = { <a,b>, <b,a> , <d,b> , <b,d> }	
		{ }
b = d	gen = { <b,d> } kill = { <a,b>, <b,a> , <d,b>, <b,d> }	
		{ <b,d> }

Example

		{ }
b = a	gen = { <b,a> } kill = { <d,b>, <b,d> , <b,a>, <a,b>}	
		{ <b,a> }
c = b + 1	gen = { } kill = {<c> }	
		{ <b,a> }
d = a	gen = { <d,b> } kill = { <b,d>, <d,b> }	
		{ <b,a>, <d,b> }
b = d + c	gen = { } kill = { <a,b>, <b,a> , <d,b> , <b,d> }	
		{ }
b = d	gen = { <b,d> } kill = { <a,b>, <b,a> , <d,b>, <b,d> }	
		{ <b,d> }

Example

{ }

b = a
gen = { <b,a> }
kill = { <d,b>, <b,d> , <b,a>, <a,b>}

{ <b,a> }

c = b + 1
gen = { }
kill = { <c>}

{ <b,a> }

d = a
gen = { <d,b> }
kill = { <b,d>, <d,b> }

{ <b,a>, <d,b> }

b = d + c
gen = { }
kill = { <a,b>, <b,a> , <d,b> , <b,d> }

{ }

b = d
gen = { <b,d> }
kill = { <a,b>, <b,a> , <d,b>, <b,d> }

{ <b,d> }

Example

{ }

b = a
gen = { <b,a> }
kill = { <d,b>, <b,d> , <b,a>, <a,b>}

{ <b,a> }

c = b + 1
gen = { }
kill = { <c>}

{ <b,a> }

d = a
gen = { <d,b> }
kill = { <b,d>, <d,b> }

{ <b,a>, <d,b> }

b = b + c
gen = { }
kill = { <a,b>, <b,a> , <d,b> , <b,d> }

{ }

b = d
gen = { <b,d> }
kill = { <a,b>, <b,a> , <d,b>, <b,d> }

{ <b,d> }

Example

Statement	gen / kill	Reaching definitions
		{ }
b = a	gen = { <b,a> } kill = { <d,b>, <b,d> , <b,a>, <a,b>}	
		{ <b,a> }
c = b + 1	gen = { } kill = {<c>}	
		{ <b,a> }
d = a	gen = { <d,b> } kill = { <b,d>, <d,b> }	
		{ <b,a>, <d,b> }
b = b + c	gen = { } kill = { <a,b>, <b,a> , <d,b> , <b,d> }	
		{ }
b = d	gen = { <b,d> } kill = { <a,b>, <b,a> , <d,b>, <b,d> }	
		{ <b,d> }

Example

```
b = a

c = b + 1

d = a

b = b + c

b = d
```

Example

gen = { <v,u> | v = u is the statement }
kill = { <v,u> | LHS var. of an assignment stmt. is either v or u }

b = a	gen = { <b,a> }
c = b + 1	gen = { }
d = a	gen = { <d,a> }
b = b + c	gen = { }
b = d	gen = { <b,d> }

Example

gen = { <v,u> | v = u is the statement }
kill = { <v,u> | LHS var. of an assignment stmt. is either v or u }

Statement	gen / kill
b = a	gen = { <b,a> } kill = { <d,b>, <b,d> , <b,a>, <a,b>}
c = b + 1	gen = { } kill = {<c> }
d = a	gen = { <d,a> } kill = { <b,d>, <d,b>,<d,a>,<a,d> }
b = b + c	gen = { } kill = { <b,a> , <a,b>, <b,d>, <d,b> }
b = d	gen = { <b,d> } kill = { <b,a>,<a,b>,<b,d>,<d,b> }

Example

OUT = gen ∪ (IN - kill)

b = a	gen = { <b,a> } kill = { <d,b>, <b,d> , <b,a>, <a,b>}
c = b + 1	gen = { } kill = {<c> }
d = a	gen = { <d,a> } kill = { <b,d>, <d,b>,<d,a>,<a,d> }
b = b + c	gen = { } kill = { <b,a> , <a,b>, <b,d>, <d,b> }
b = d	gen = { <b,d> } kill = { <b,a>,<a,b>,<b,d>,<d,b> }

Example

OUT = gen ∪ (IN - kill)

IN = { }

b = a	gen = { <b,a> } kill = { <d,b>, <b,d> , <b,a>, <a,b>}

OUT = { <b,a> }

c = b + 1	gen = { } kill = {<c> }
d = a	gen = { <d,a> } kill = { <b,d>, <d,b>,<d,a>,<a,d> }
b = b + c	gen = { } kill = { <b,a> , <a,b>, <b,d>, <d,b> }
b = d	gen = { <b,d> } kill = { <b,a>,<a,b>,<b,d>,<d,b> }

Example

OUT = gen ∪ (IN - kill)

IN = { }

b = a
gen = { <b,a> }
kill = { <d,b>, <b,d> , <b,a>, <a,b>}

IN = { <b,a> }

c = b + 1
gen = { }
kill = { }

OUT = { <b,a> }

d = a
gen = { <d,a> }
kill = { <b,d>, <d,b>,<d,a>,<a,d> }

b = b + c
gen = { }
kill = { <b,a> , <a,b>, <b,d>, <d,b> }

b = d
gen = { <b,d> }
kill = { <b,a>,<a,b>,<b,d>,<d,b> }

Example

OUT = gen ∪ (IN - kill)

IN = { }

b = a
gen = { <b,a> }
kill = { <d,b>, <b,d> , <b,a>, <a,b>}

IN = { <b,a> }

c = b + 1
gen = { }
kill = { }

IN = { <b,a> }

d = a
gen = { <d,a> }
kill = { <b,d>, <d,b>,<d,a>,<a,d> }

OUT = { <d,a>,<b,a> }

b = b + c
gen = { }
kill = { <b,a> , <a,b>, <b,d>, <d,b> }

b = d
gen = { <b,d> }
kill = { <b,a>,<a,b>,<b,d>,<d,b> }

Example

$$OUT = gen \cup (IN - kill)$$

IN = { }

b = a
gen = { <b,a> }
kill = { <d,b>, <b,d> , <b,a>, <a,b>}

IN = { <b,a> }

c = b + 1
gen = { }
kill = { }

IN = { <b,a> }

d = a
gen = { <d,a> }
kill = { <b,d>, <d,b>,<d,a>,<a,d> }

IN = { <d,a>,<b,a> }

b = b + c
gen = { }
kill = { <b,a> , <a,b>, <b,d>, <d,b> }

OUT = { <d,a> }

b = d
gen = { <b,d> }
kill = { <b,a>,<a,b>,<b,d>,<d,b> }

Example

OUT = gen ∪ (IN - kill)

Statement	gen / kill	IN / OUT
		IN = { }
b = a	gen = { <b,a> } kill = { <d,b>, <b,d> , <b,a>, <a,b>}	
		IN = { <b,a> }
c = b + 1	gen = { } kill = { }	
		IN = { <b,a> }
d = a	gen = { <d,a> } kill = { <b,d>, <d,b>,<d,a>,<a,d> }	
		IN = { <d,a>,<b,a> }
b = b + c	gen = { } kill = { <b,a> , <a,b>, <b,d>, <d,b> }	
		IN = { <d,a> }
b = d	gen = { <b,d> } kill = { <b,a>,<a,b>,<b,d>,<d,b> }	
		OUT = { <d,a>,<b,d> }

Example

		{ }
b = a	gen = { <b,a> } kill = { <d,b>, <b,d> , <b,a>, <a,b>}	
		{ <b,a> }
c = b + 1	gen = { } kill = { }	
		{ <b,a> }
d = a	gen = { <d,a> } kill = { <b,d>, <d,b>,<d,a>,<a,d> }	
		{ <d,a>,<b,a> }
b = b + c	gen = { } kill = { <b,a> , <a,b>, <b,d>, <d,b> }	
		{ <d,a> }
b = d	gen = { <b,d> } kill = { <b,a>,<a,b>,<b,d>,<d,b> }	
		{ <d,a>,<b,d> }

Example

b = a

gen = { <b,a> }

kill = { <d,b>, <b,d> , <b,a>, <a,b>}

{ }

{ <b,a> }

c = b + 1

gen = { }

kill = { }

{ <b,a> }

d = a

gen = { <d,a> }

kill = { <b,d>, <d,b>,<d,a>,<a,d> }

{ <d,a>,<b,a> }

b = b + c

gen = { }

kill = { <b,a> , <a,b>, <b,d>, <d,b> }

{ <d,a> }

b = d

gen = { <b,d> }

kill = { <b,a>,<a,b>,<b,d>,<d,b> }

{ <d,a>,<b,d> }

Example

		{ }
b = a	gen = { <b,a> } kill = { <d,b>, <b,d> , <b,a>, <a,b>}	
		{ <b,a> }
c = b + 1	gen = { } kill = { }	
		{ <b,a> }
d = a	gen = { <d,a> } kill = { <b,d>, <d,b>,<d,a>,<a,d> }	
		{ <d,a>,<b,a> }
b = a + c	gen = { } kill = { <b,a> , <a,b>, <b,d>, <d,b> }	
		{ <d,a> }
b = d	gen = { <b,d> } kill = { <b,a>,<a,b>,<b,d>,<d,b> }	
		{ <d,a>,<b,d> }

Example

{ }

b = a
gen = { <b,a> }
kill = { <d,b>, <b,d> , <b,a>, <a,b>}

{ <b,a> }

c = b + 1
gen = { }
kill = { }

{ <b,a> }

d = a
gen = { <d,a> }
kill = { <b,d>, <d,b>,<d,a>,<a,d> }

{ <d,a>,<b,a> }

b = a + c
gen = { }
kill = { <b,a> , <a,b>, <b,d>, <d,b> }

{ <d,a> }

b = d
gen = { <b,d> }
kill = { <b,a>,<a,b>,<b,d>,<d,b> }

{ <d,a>,<b,d> }

Example

		{ }
b = a	gen = { <b,a> } kill = { <d,b>, <b,d> , <b,a>, <a,b>}	
		{ <b,a> }
c = b + 1	gen = { } kill = { }	
		{ <b,a> }
d = a	gen = { <d,a> } kill = { <b,d>, <d,b>,<d,a>,<a,d> }	
		{ <d,a>,<b,a> }
b = a + c	gen = { } kill = { <b,a> , <a,b>, <b,d>, <d,b> }	
		{ <d,a> }
b = a	gen = { <b,d> } kill = { <b,a>,<a,b>,<b,d>,<d,b> }	
		{ <d,a>,<b,d> }

Example

		{ }
b = a	gen = { <b,a> } kill = { <d,b>, <b,d> , <b,a>, <a,b>}	
		{ <b,a> }
c = b + 1	gen = { } kill = { }	
		{ <b,a> }
d = a	gen = { <d,a> } kill = { <b,d>, <d,b>,<d,a>,<a,d> }	
		{ <d,a>,<b,a> }
b = a + c	gen = { } kill = { <b,a> , <a,b>, <b,d>, <d,b> }	
		{ <d,a> }
b = a	gen = { <b,d> } kill = { <b,a>,<a,b>,<b,d>,<d,b> }	
		{ <d,a>,<b,d> }

DEI DEPARTAMENTO DE ENGENHARIA INFORMÁTICA
U.PORTO FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

Example

b = a

c = b + 1

d = a

b = a + c

b = a

ARE WE DONE?

YES!!

Example

b = a

c = b + 1

d = a

b = a + c

b = a

Can we do other Transformations?

Example

~~b = a~~

c = b + 1

~~d = a~~ **?**

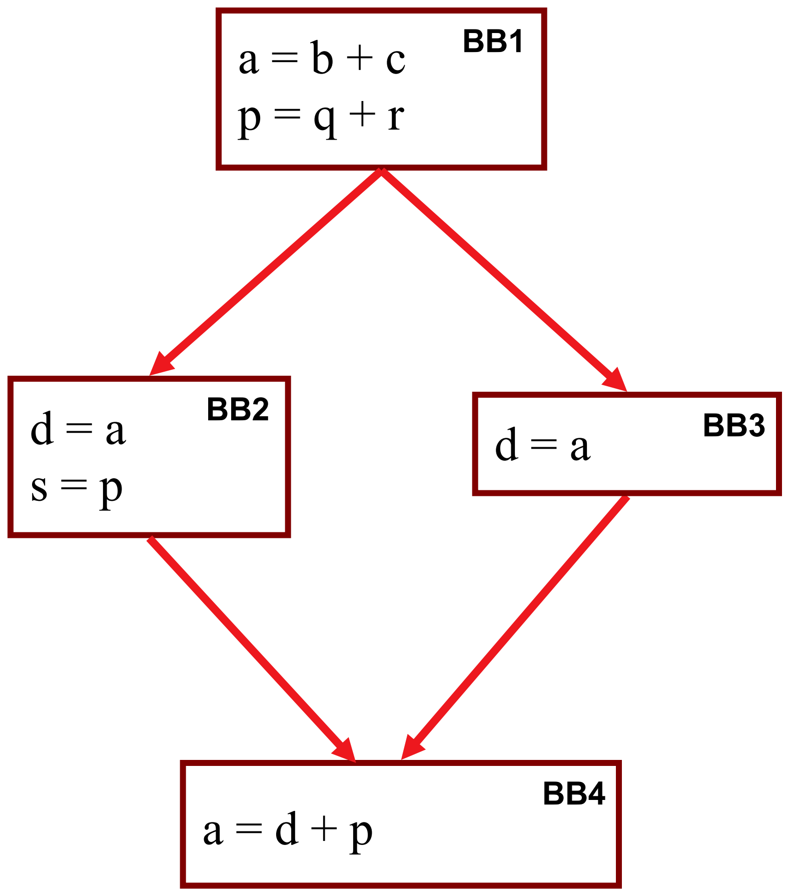
b = a + c

b = a

Can we do other Transformations?

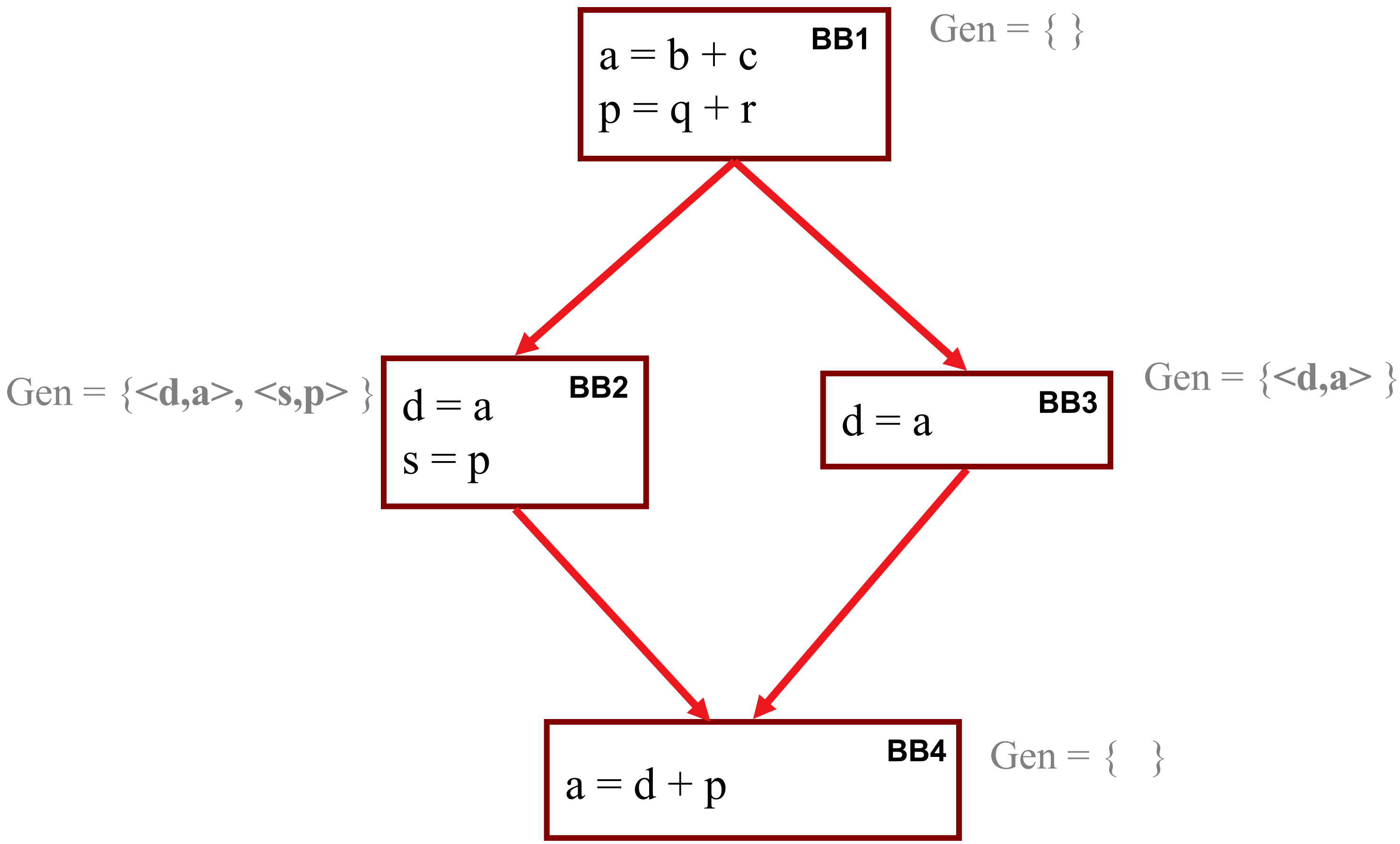
YES!!

Another Example



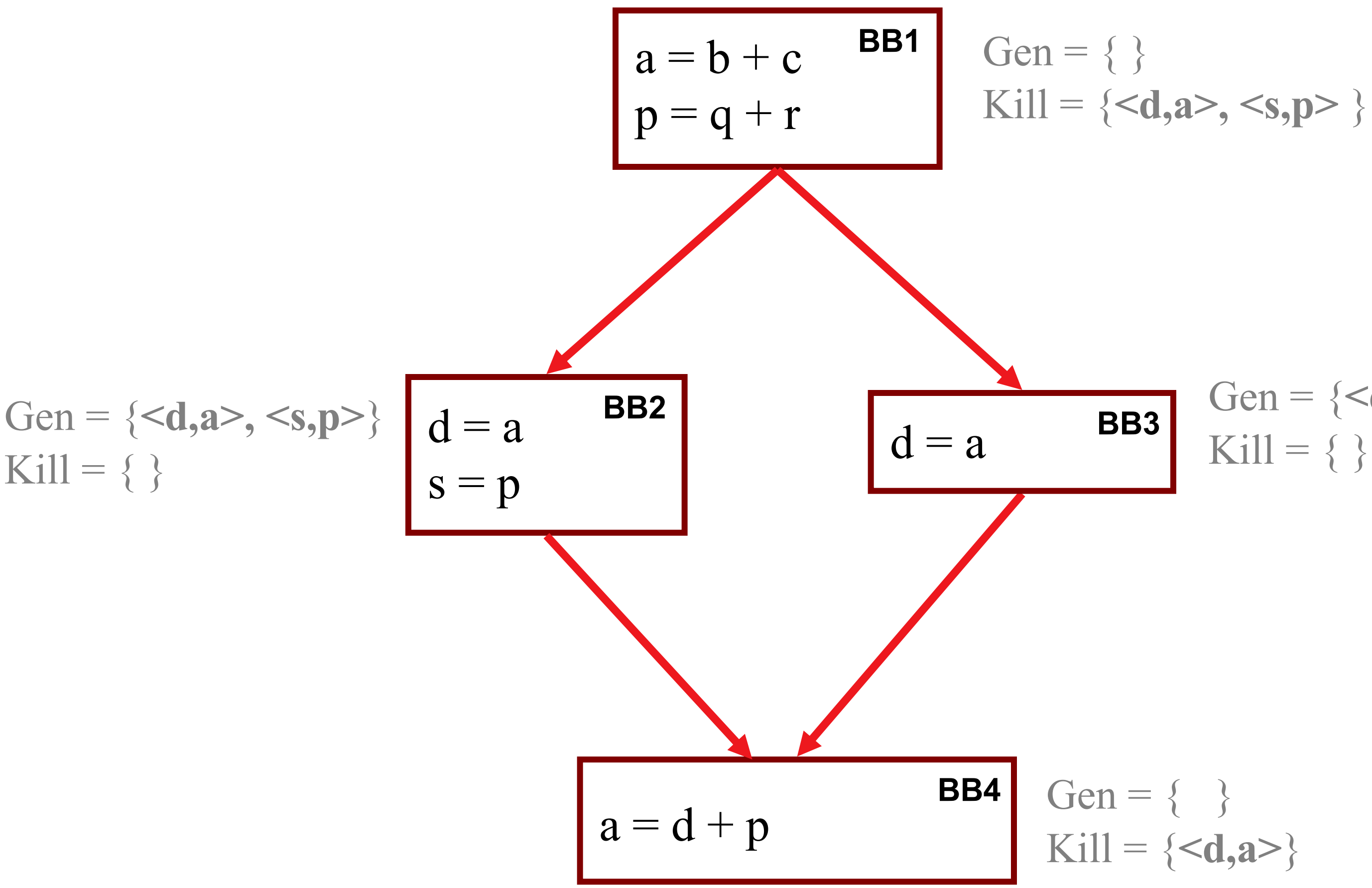
Another Example

gen = { <v,u> | v = u is the statement }
kill = { <v,u> | LHS var. of an assignment stmt. is either v or u }



Another Example

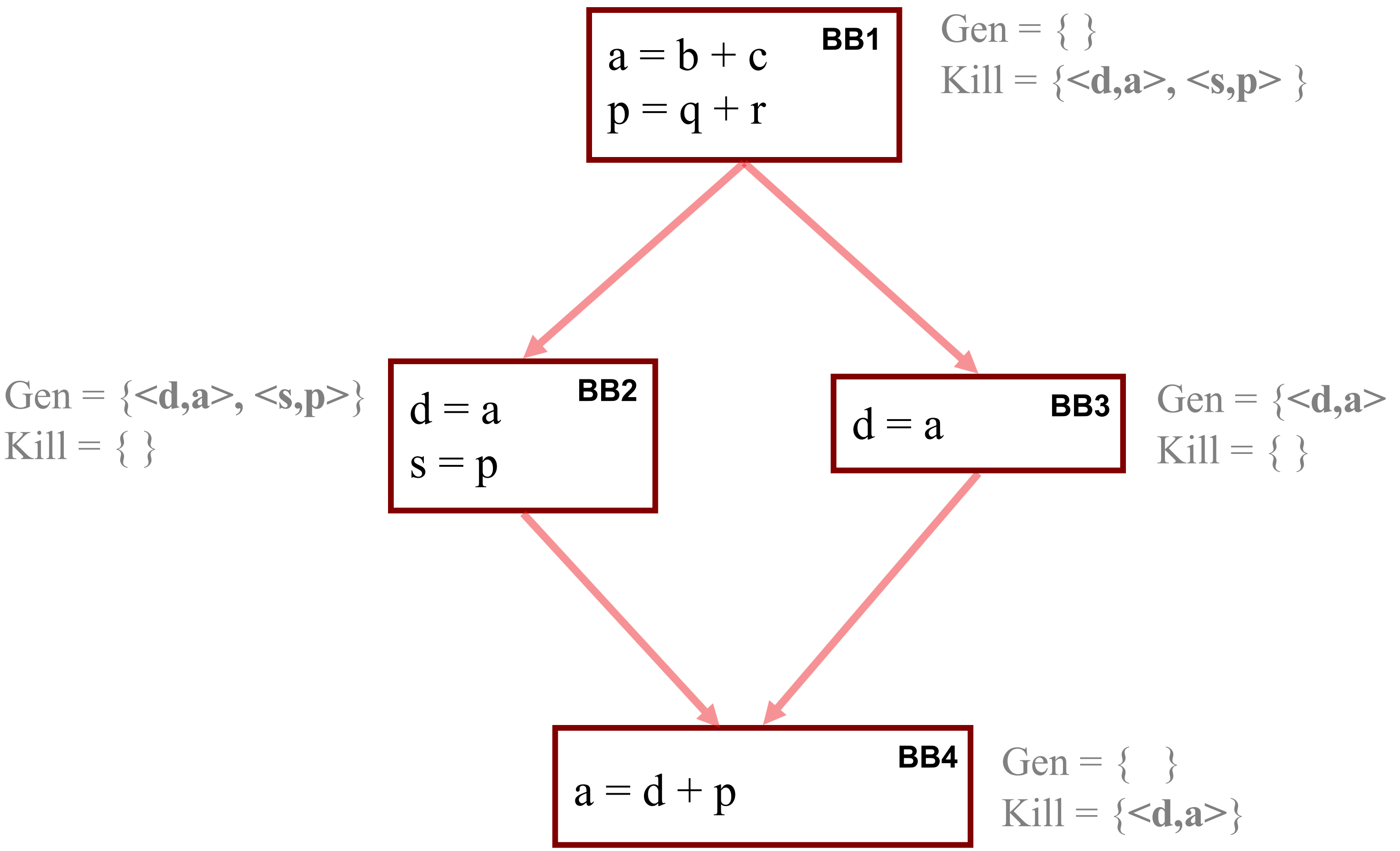
gen = { <v,u> | v = u is the statement }
kill = { <v,u> | LHS var. of an assignment stmt. is either v or u }



Another Example

$$IN = \cap\ OUT$$

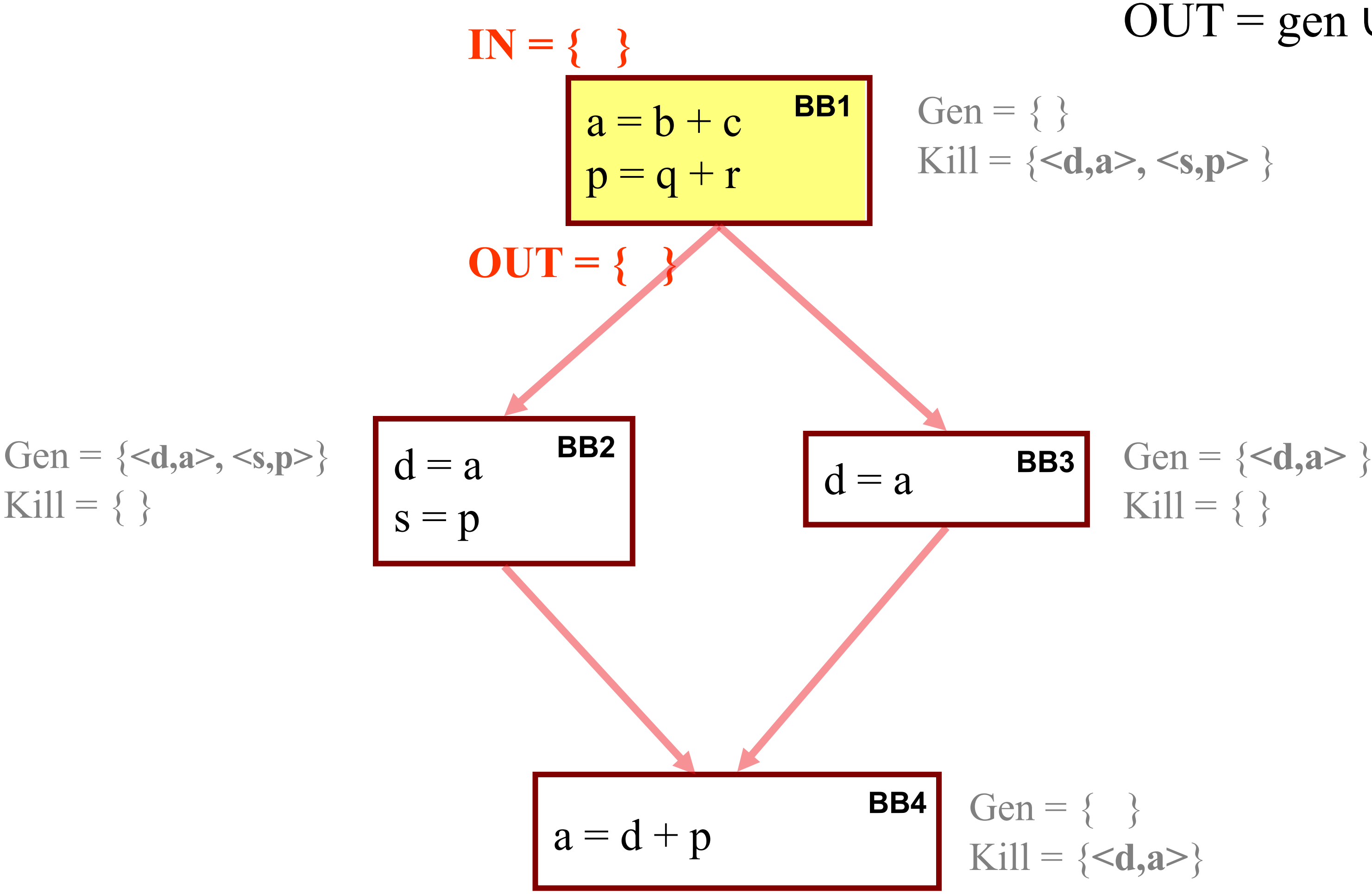
$$OUT = gen \cup (IN - kill)$$



Another Example

$IN = \cap\ OUT$

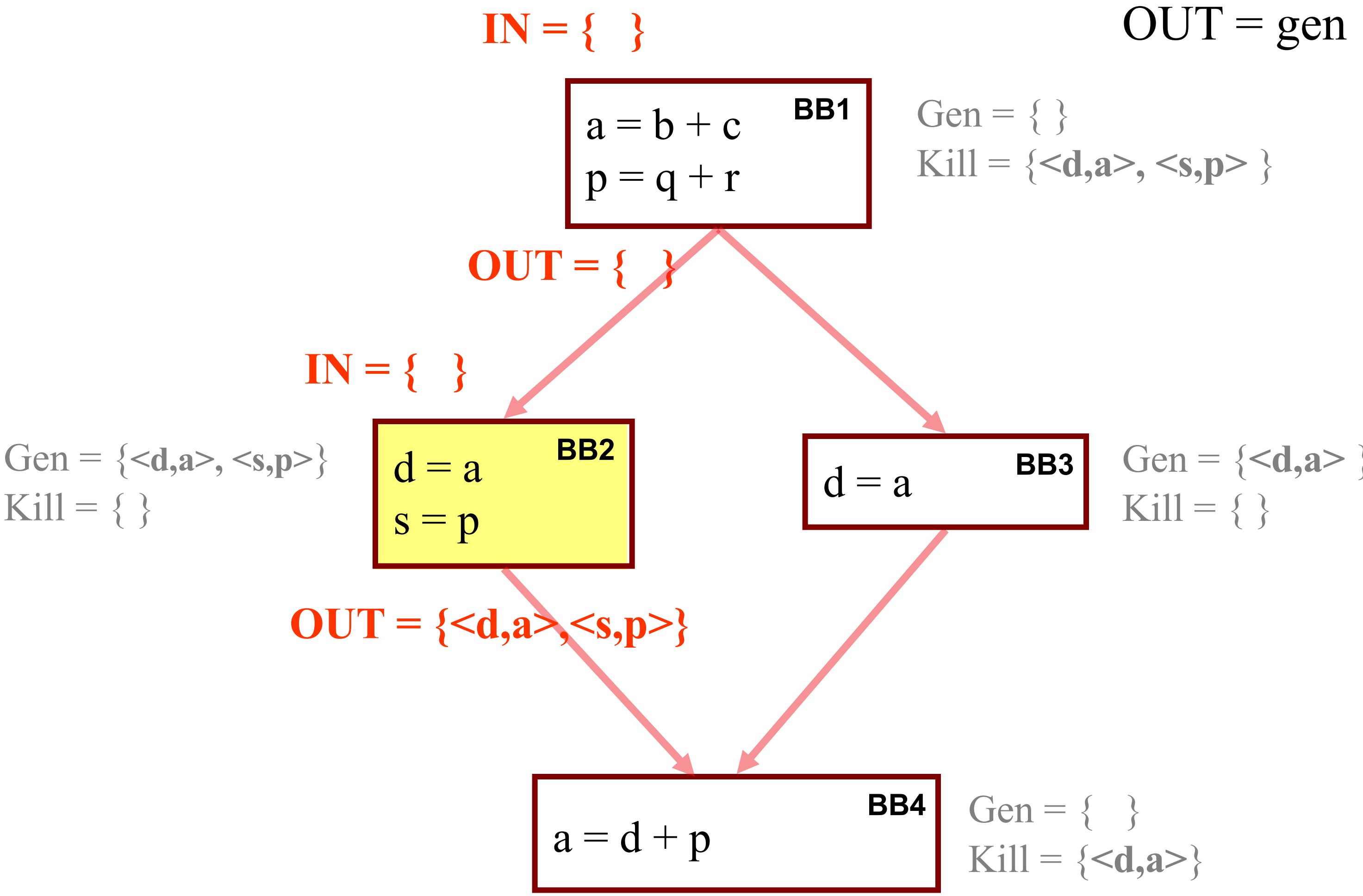
$OUT = gen \cup (IN - kill)$



Another Example

$$IN = \cap\ OUT$$

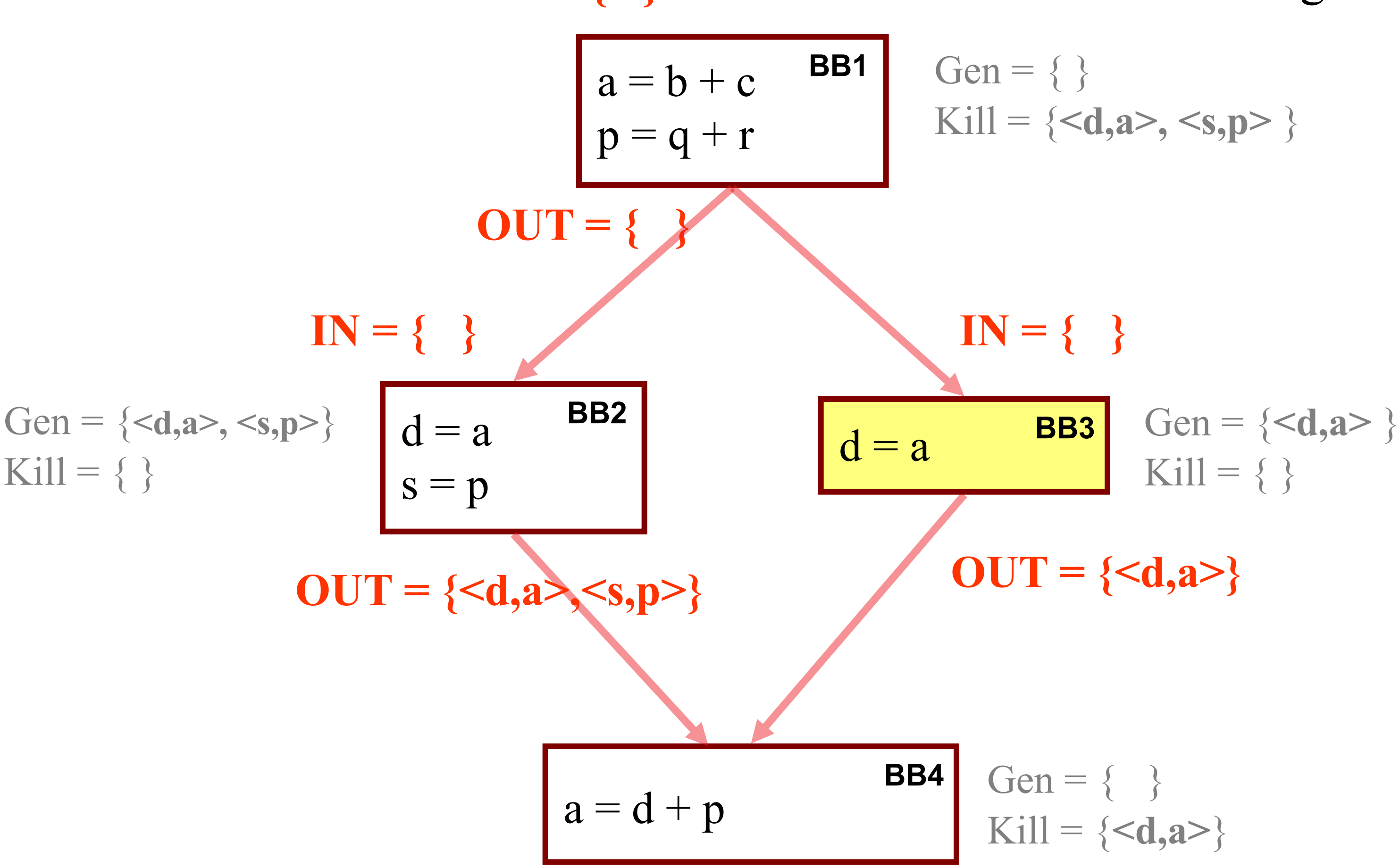
$$OUT = gen \cup (IN - kill)$$



Another Example

$IN = \cap\ OUT$

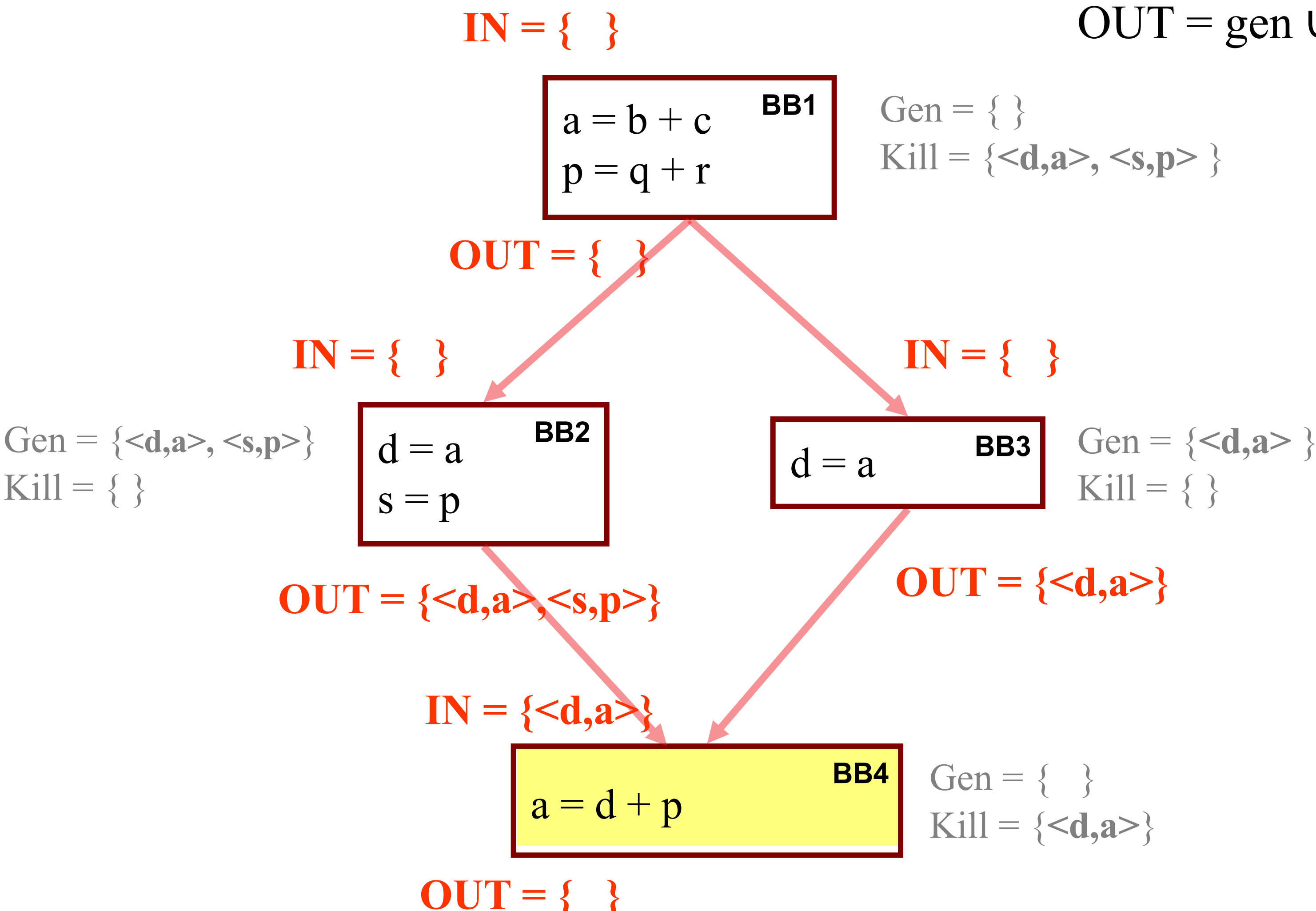
$OUT = gen \cup (IN - kill)$



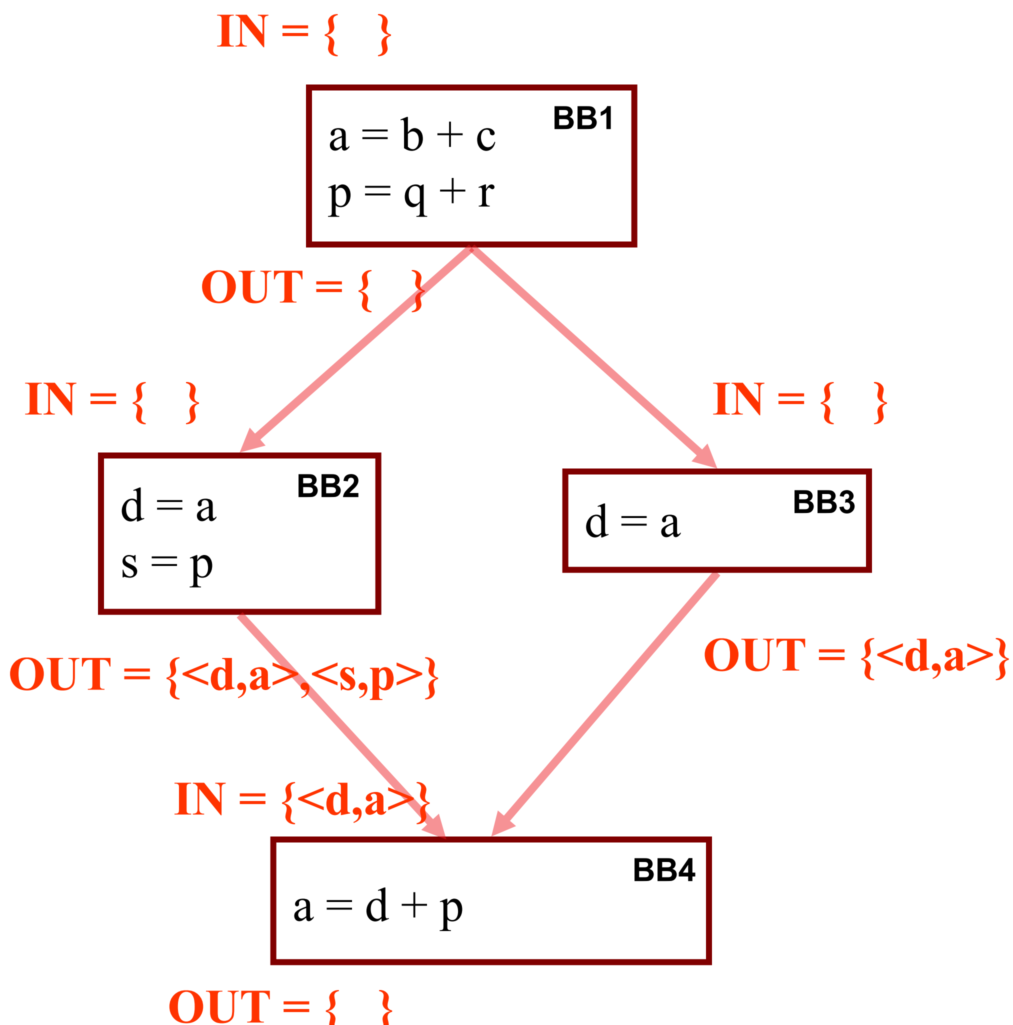
Another Example

$IN = \cap \, OUT$

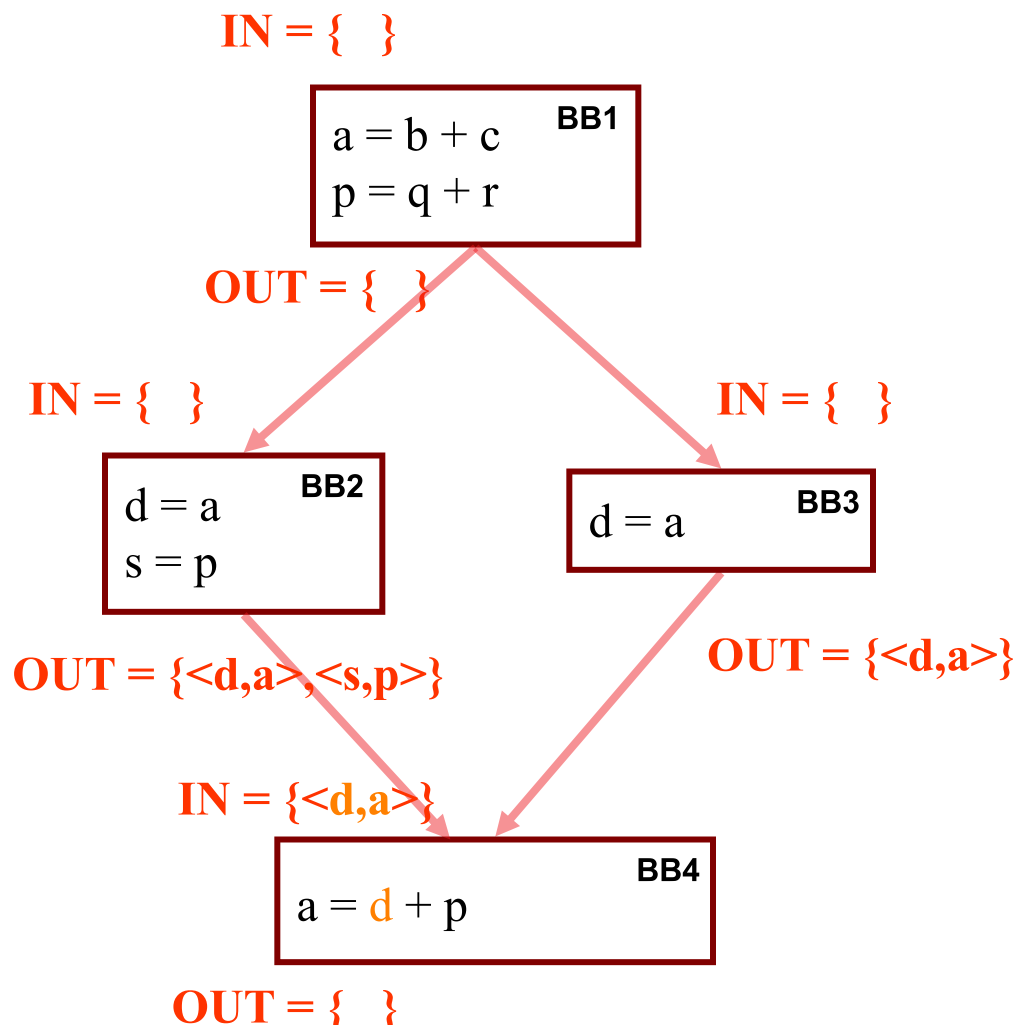
$OUT = gen \cup (IN - kill)$



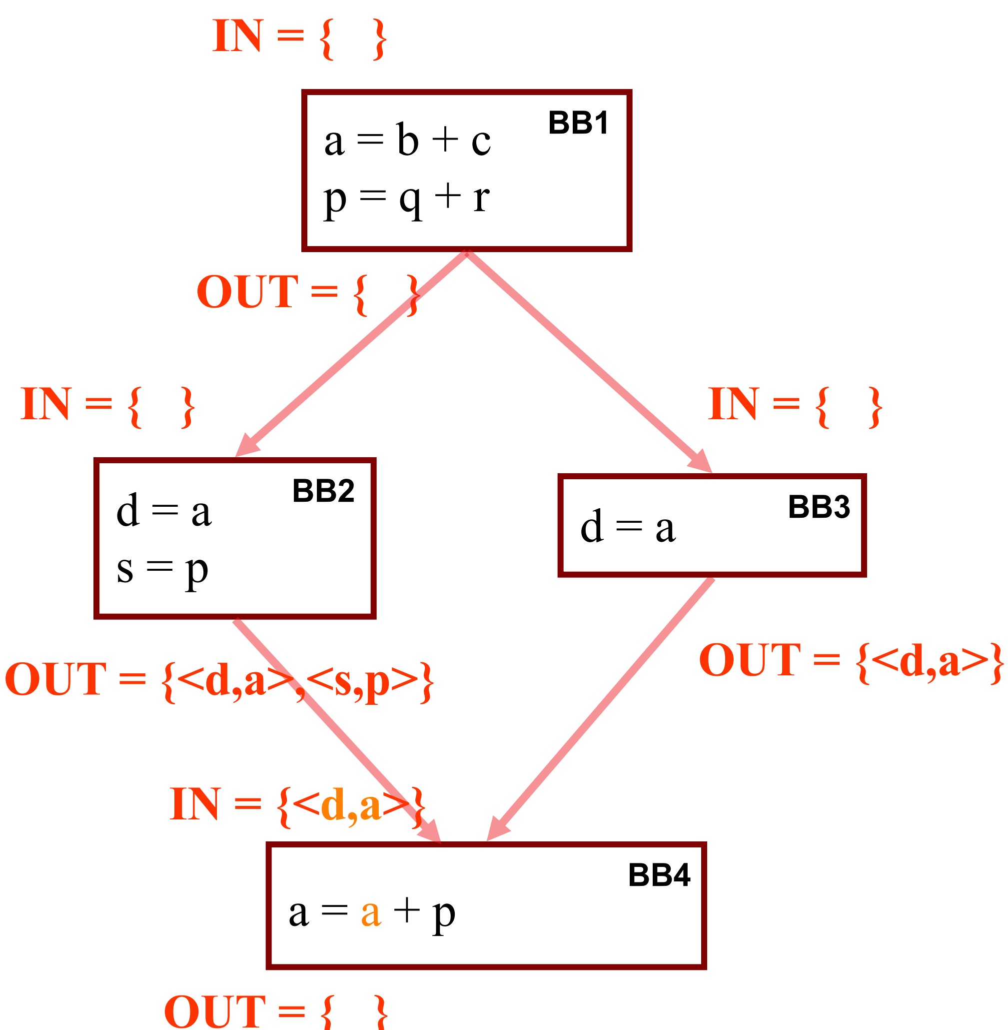
Another Example



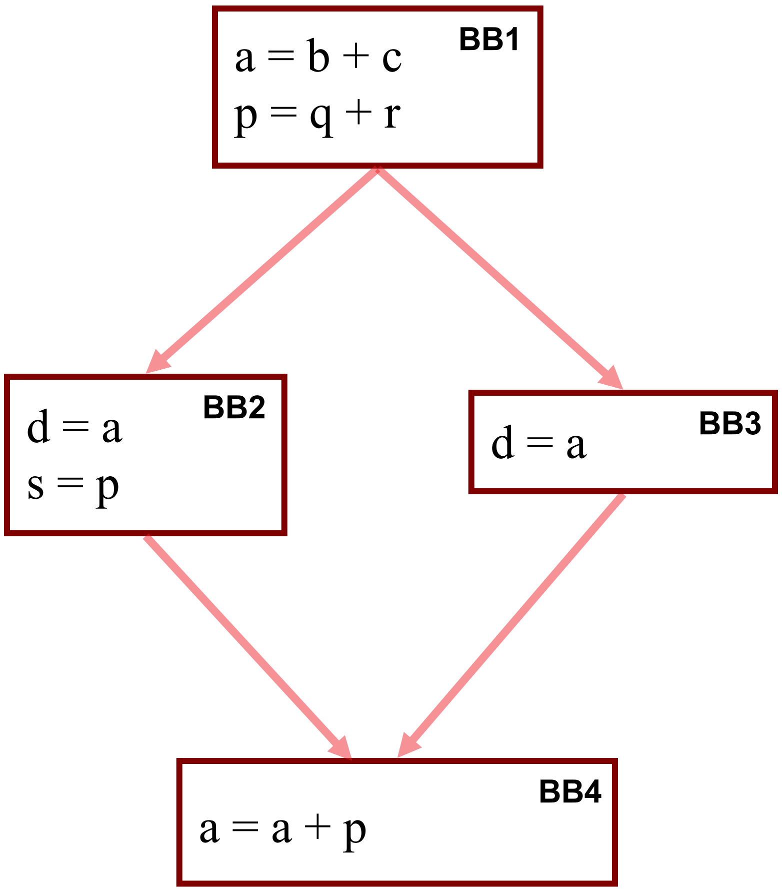
Another Example



Another Example



Another Example



Outline

- Overview of Control-Flow Analysis
- Algebraic Simplification
- Copy Propagation
- Constant Propagation

Constant Propagation

- Use Constant Values
  - Use the known constant of a variable

Constant Propagation

- Use Constant Values
  - Use the known constant of a variable
- Example

```
a = 43
b = 4
d = a + 2*b + c
```

Constant Propagation

- Use Constant Values
  - Use the known constant of a variable
- Example

```
a = 43
b = 4
d = a + 2*b + c
```

Constant Propagation

- Use Constant Values
  - Use the known constant of a variable
- Example

```
a = 43
b = 4
d = 43 + 2*b + c
```

Constant Propagation

- Use Constant Values
  - Use the known constant of a variable
- Example

```
a = 43
b = 4
d = a + 2*b + c
```

Constant Propagation

- Use Constant Values
  - Use the known constant of a variable
- Example

```
a = 43
b = 4
d = 43 + 2*4 + c
```

Constant Propagation Opportunities

- User-Defined Constants
  - Same Constants propagating from many different paths
  - Symbolic Constants defined as variables
- Constants Known to the Compiler
  - data sizes, stack offsets
- Constants Available after Other Optimizations
  - Algebraic Simplification
  - Copy propagation

Advantages of Constant Propagation

- Simplification of the Program

Advantages of Constant Propagation

- Simplification of the Program

- Example

```
a = 43
b = 4
d = 43 + 2*4 + c
```

Advantages of Constant Propagation

- Simplification of the Program

- Example

```
a = 43
b = 4
d = 43 + 2*4 + c
```

Advantages of Constant Propagation

- Simplification of the Program

- Example

```
a = 43
b = 4
d = 51 + c
```

Advantages of Constant Propagation

- Enabling further Optimizations

Advantages of Constant Propagation

- Enabling further Optimizations

- Example

```
a = 4
b = 8
d = 2*a - b + c
e = c + d
```

Advantages of Constant Propagation

- Enabling further Optimizations

- Example

```
a = 4
b = 8
d = 2*a - b + c
e = c + d
```

Advantages of Constant Propagation

- Enabling further Optimizations

- Example

```
a = 4
b = 8
d = 2*4 - b + c
e = c + d
```

Advantages of Constant Propagation

- Enabling further Optimizations

- Example

```
a = 4
b = 8
d = 2*4 - 8 + c
e = c + d
```

Advantages of Constant Propagation

- Enabling further Optimizations

- Example

```
a = 4
b = 8
d = 8 - 8 + c
e = c + d
```

DEPARTAMENTO DE ENGENHARIA INFORMÁTICA DEI
U.PORTO FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

Advantages of Constant Propagation

- Enabling further Optimizations

- Example

```
a = 4
b = 8
d = 8 - 8 + c
e = c + d
```

Advantages of Constant Propagation

- Enabling further Optimizations

- Example

```
a = 4
b = 8
d = 0 + c
e = c + d
```

Advantages of Constant Propagation

- Enabling further Optimizations

- Example

```
a = 4
b = 8
d = 0 + c
e = c + d
```

Advantages of Constant Propagation

- Enabling further Optimizations

- Example

```
a = 4
b = 8
d = c
e = c + d
```

Advantages of Constant Propagation

- Enabling further Optimizations

- Example

```
a = 4
b = 8
d = c
e = c + d
```

Advantages of Constant Propagation

- Enabling further Optimizations

- Example

```
a = 4
b = 8
d = c
e = c + d
```

Advantages of Constant Propagation

- Enabling further Optimizations

- Example

```
a = 4
b = 8
d = c
e = c + c
```

Advantages of Constant Propagation

- Enabling further Optimizations

- Example

```
a = 4
b = 8
d = c
e = c + c
```

Advantages of Constant Propagation

- Enabling further Optimizations

- Example

```
a = 4
b = 8
d = c
e = c + c
```

Advantages of Constant Propagation

- Enabling further Optimizations

- Example

```
a = 4
b = 8
d = c
e = 2*c
```

Advantages of Constant Propagation

- Enabling further Optimizations

- Example

```
a = 4
b = 8
d = c
e = 2*c
```


DEI DEPARTAMENTO DE ENGENHARIA INFORMÁTICA
U.PORTO FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

Advantages of Constant Propagation

- Enabling further Optimizations

- Example

```
a = 4
b = 8

e = 2*c
```

How to Perform Constant Propagation

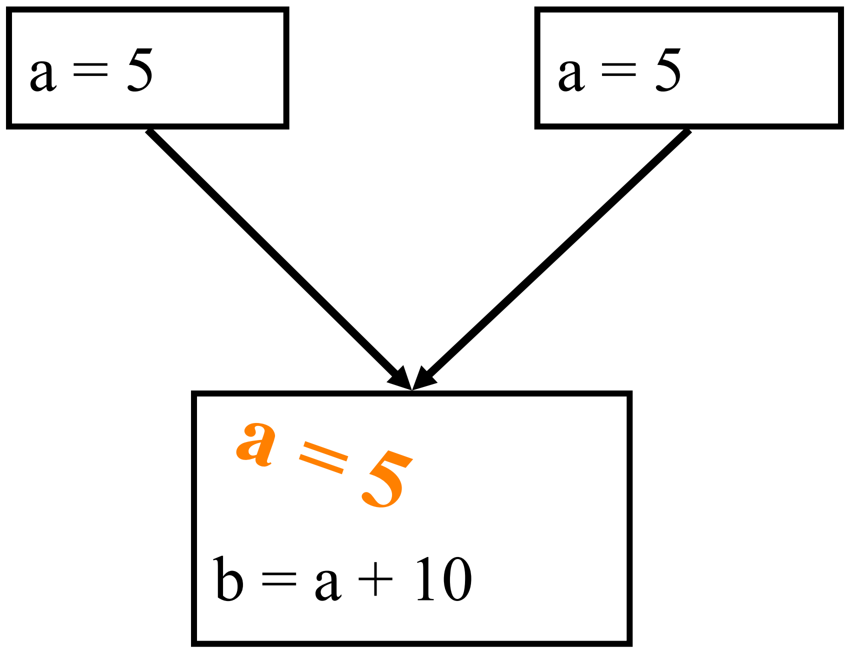
- At each RHS Expression
  - For each variable v used in the RHS
    - If the variable v is a known constant k
    - Replace the variable v by k
- At Each Point of the Program need to know:
  - For each variable v, if v is a constant,
  - If so, what the specific constant value is

How to Perform Constant Propagation

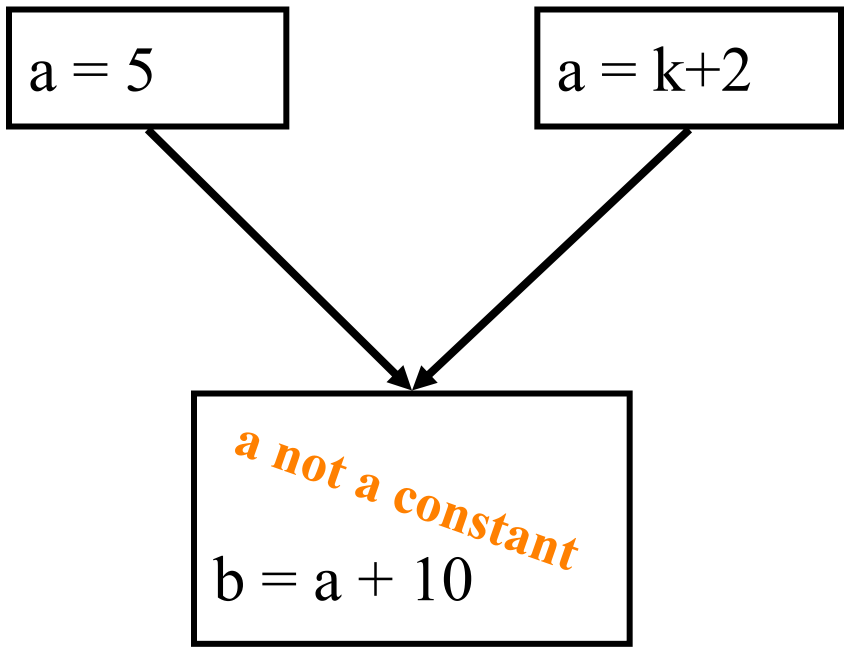
- A variable v is the constant k at a point of the execution if and only if
  - The current statement is v = k
    or
  - *Every path reaching the current point* has k assigned to v

- A Data-Flow Problem !!!

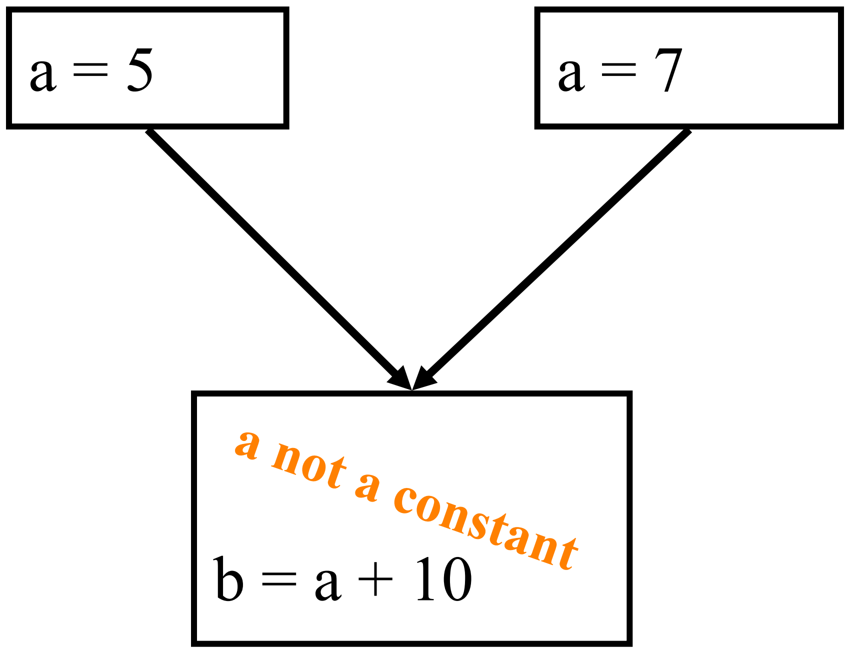
Values from Two Paths



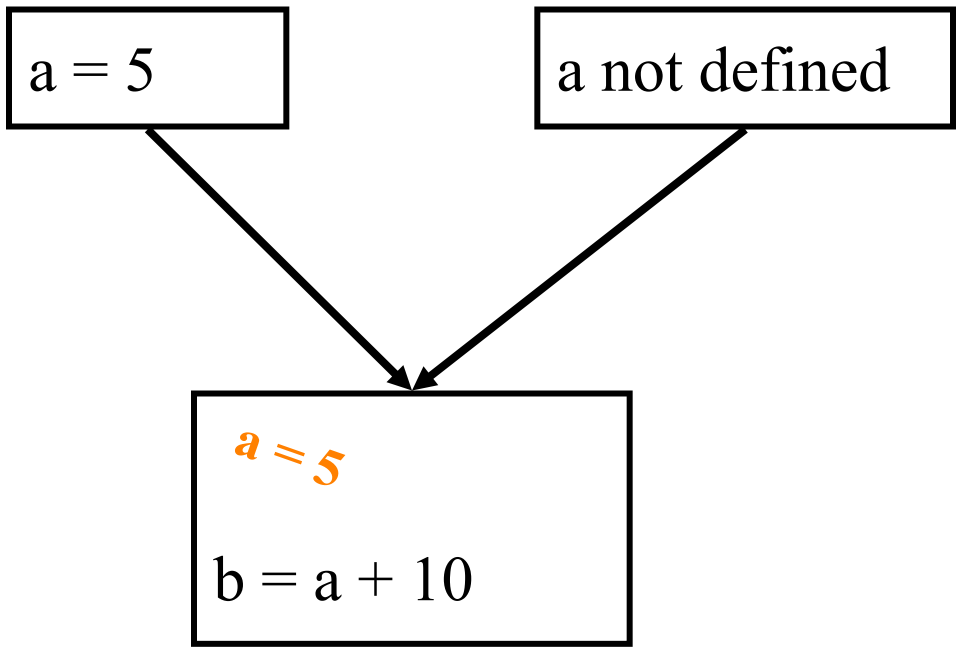
Values from Two Paths



Values from Two Paths

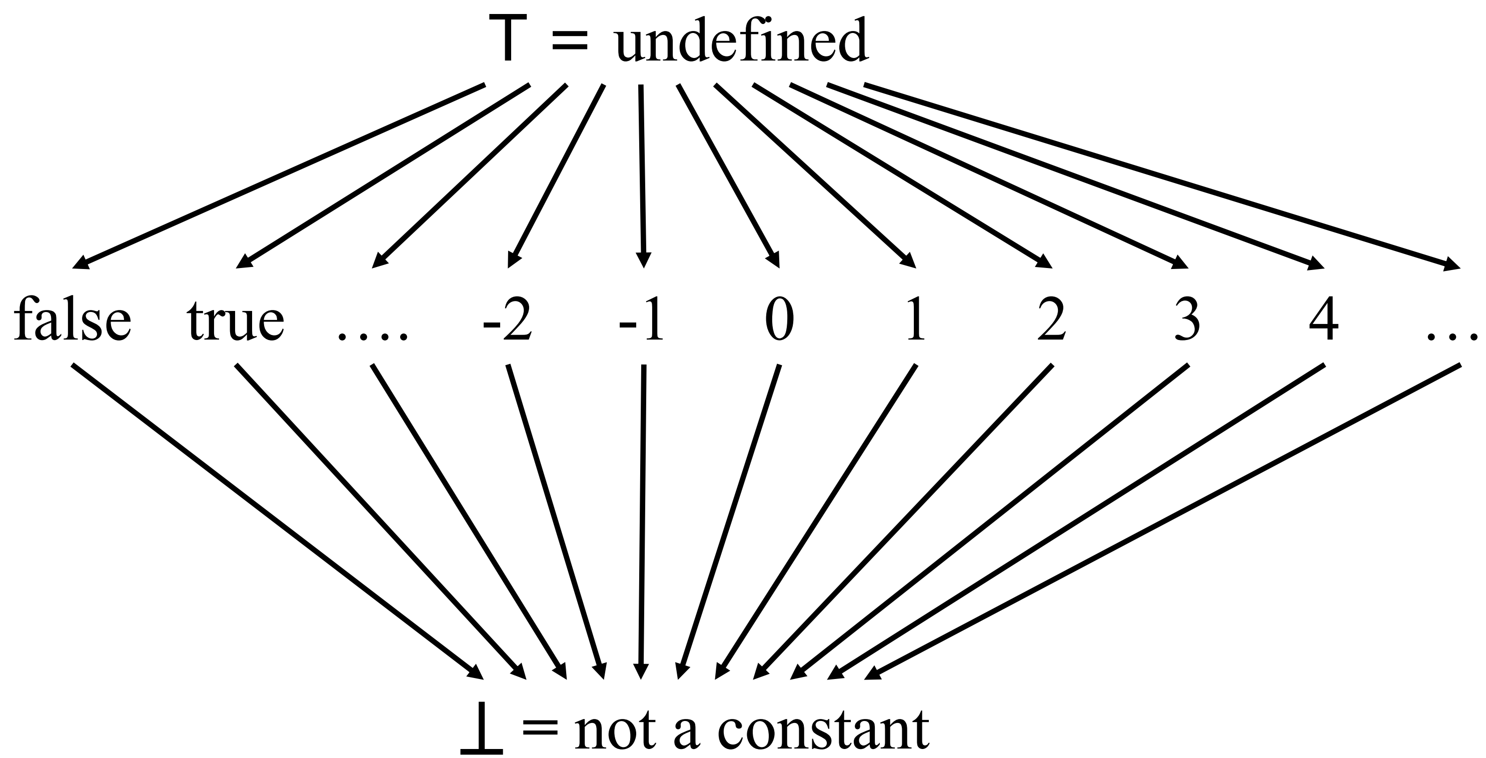


Values from Two Paths

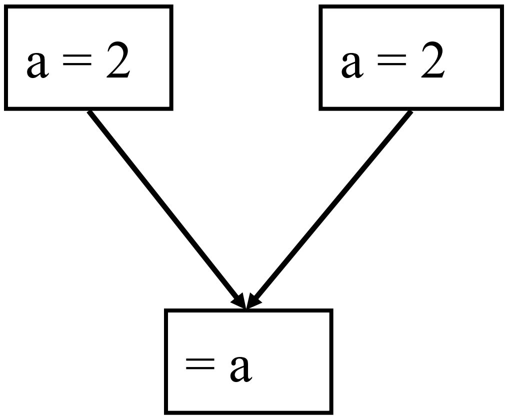


- Brain-dead program, uses an uninitialized value
- High level semantics the compiler don't understand makes this a correct program.

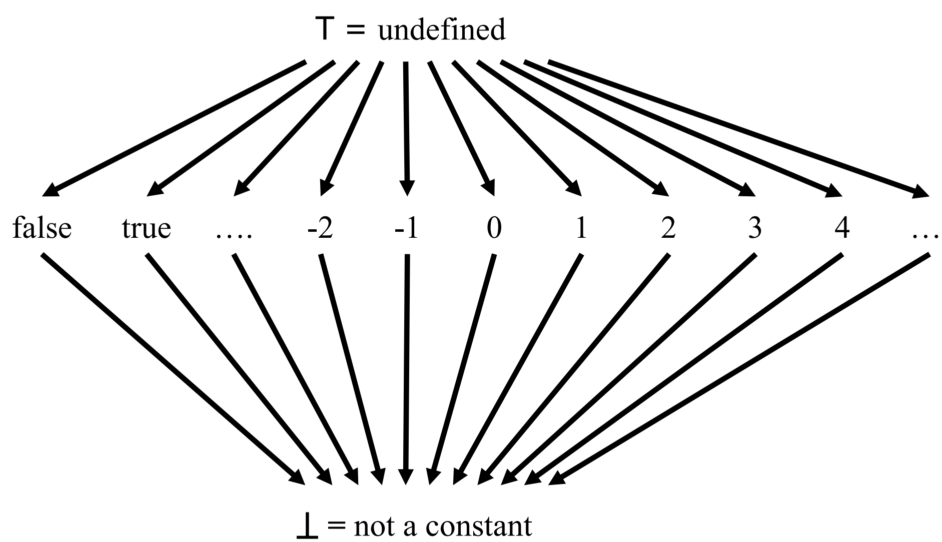
Lattice for Constant Propagation



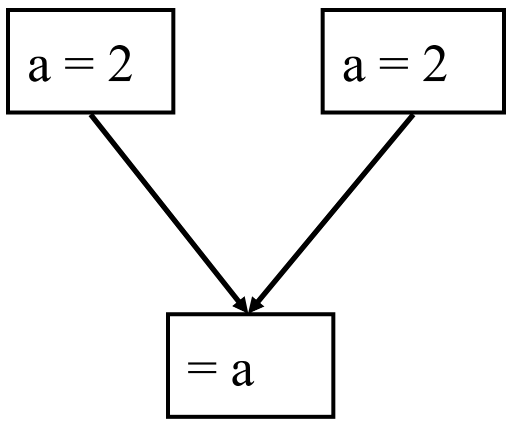
Meet Operations on the Lattice



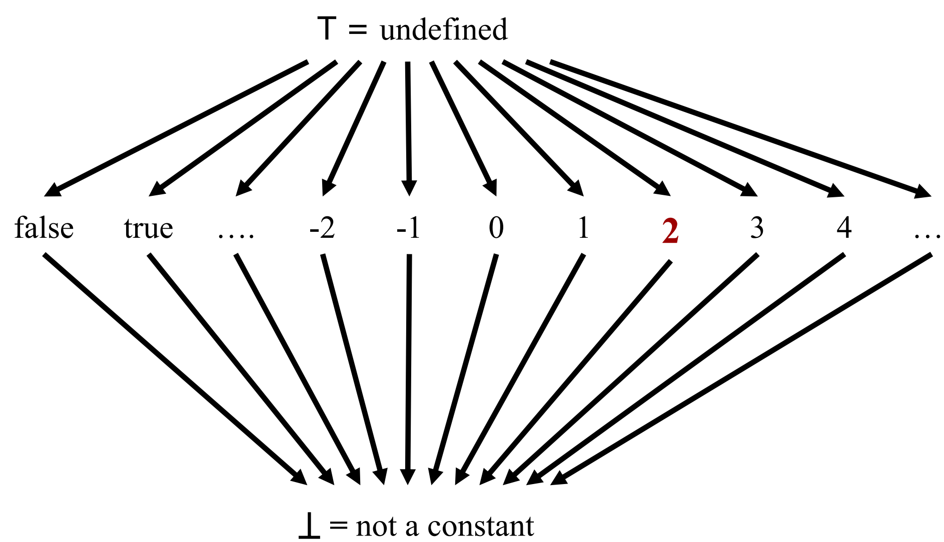
$2 \wedge 2 =$



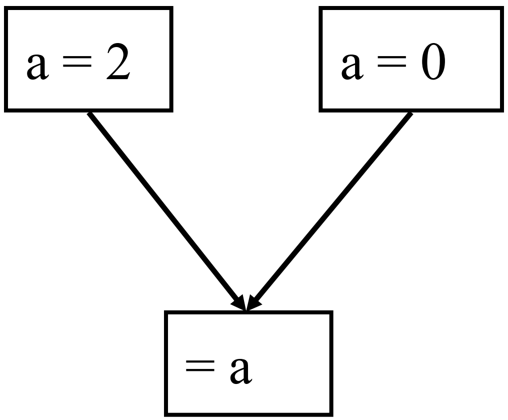
Meet Operations on the Lattice



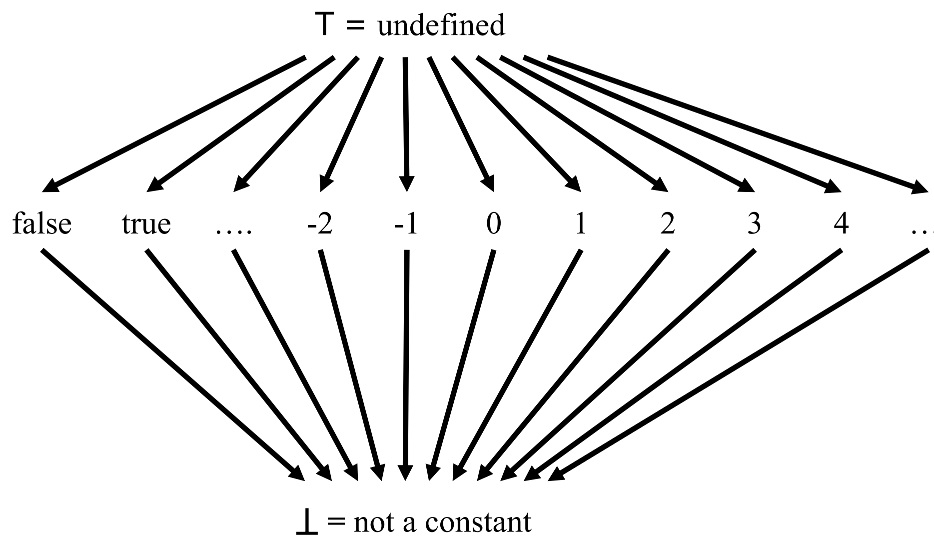
$$2 \wedge 2 = 2$$



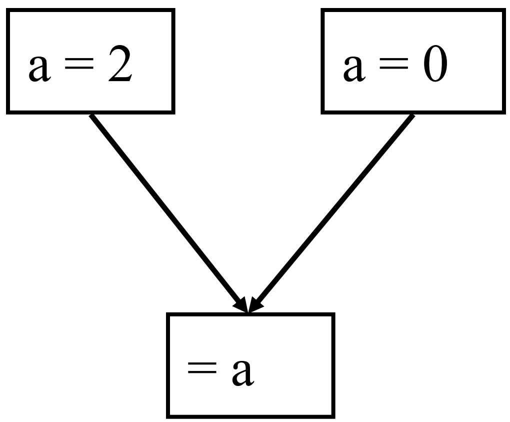
Meet Operations on the Lattice



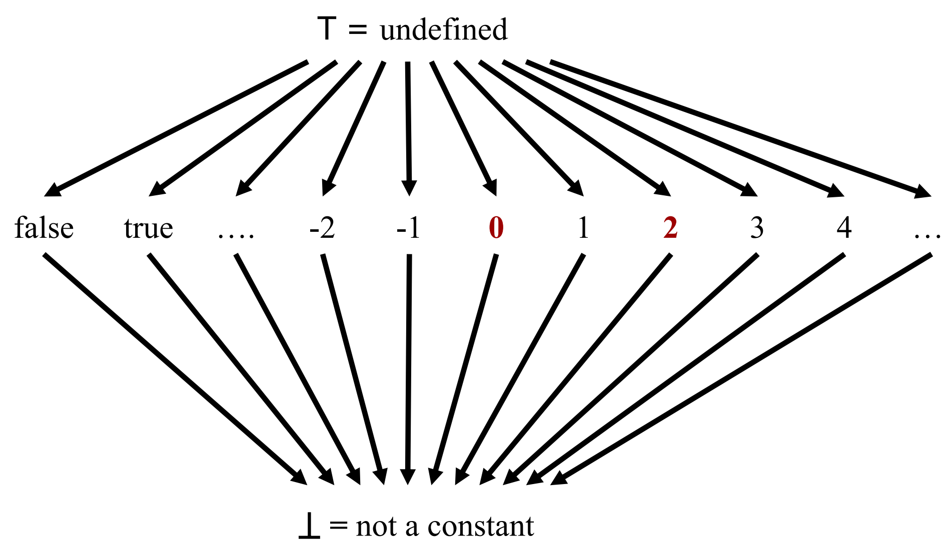
$$2 \wedge 0 =$$



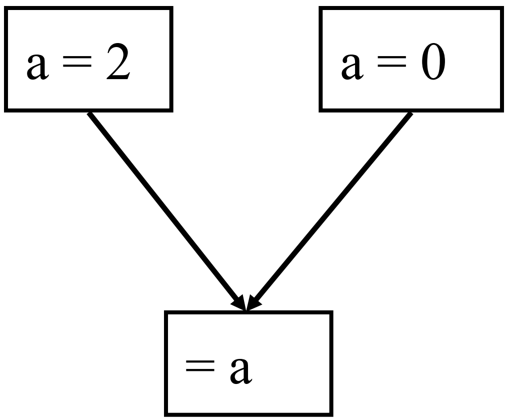
Meet Operations on the Lattice



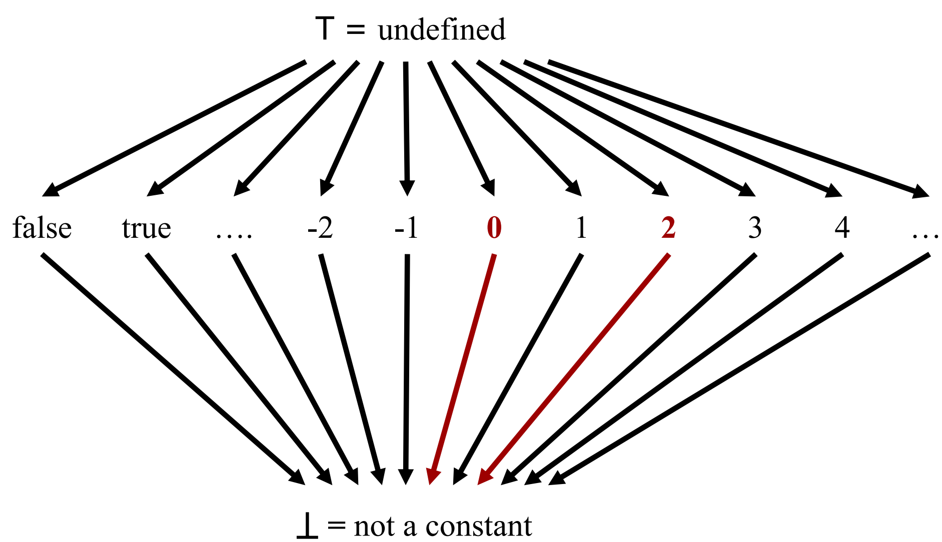
$2 \wedge 0 =$



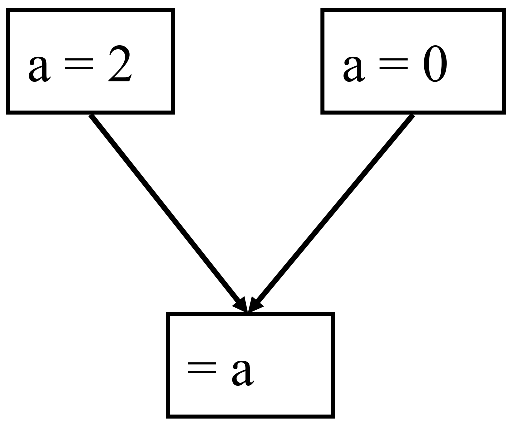
Meet Operations on the Lattice



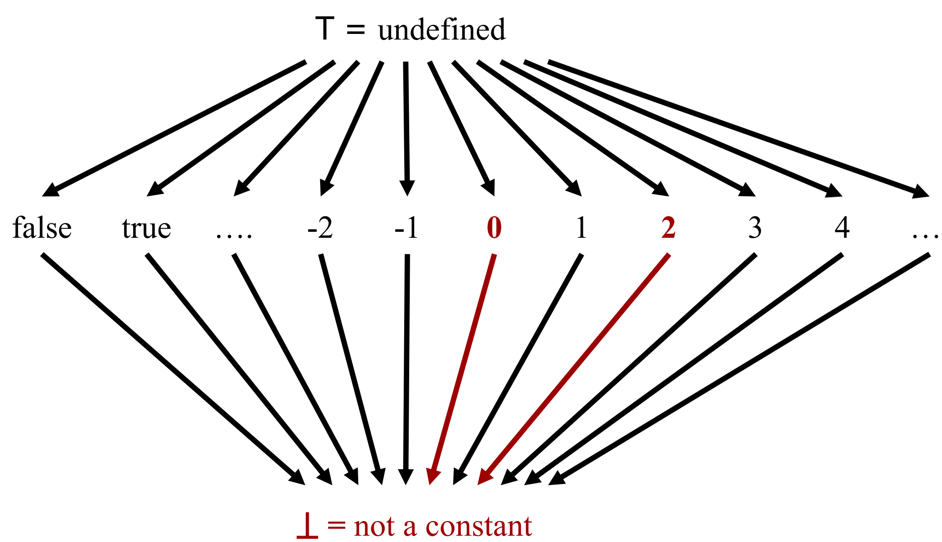
$$2 \wedge 0 =$$



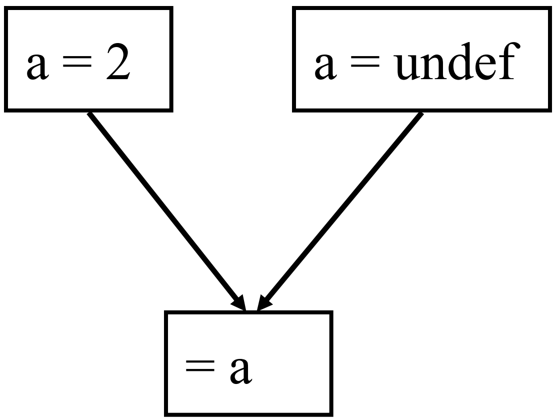
Meet Operations on the Lattice



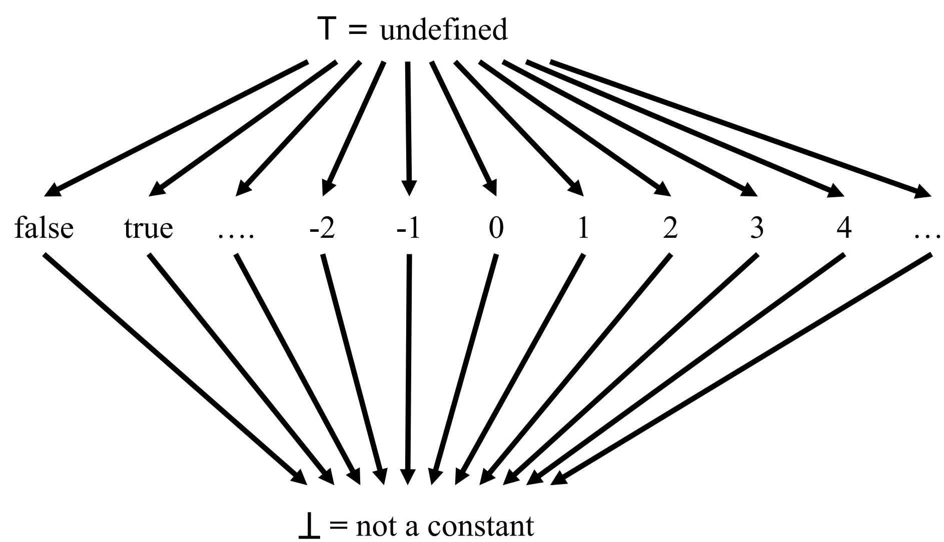
$$2 \wedge 0 \ = \text{not a constant}$$



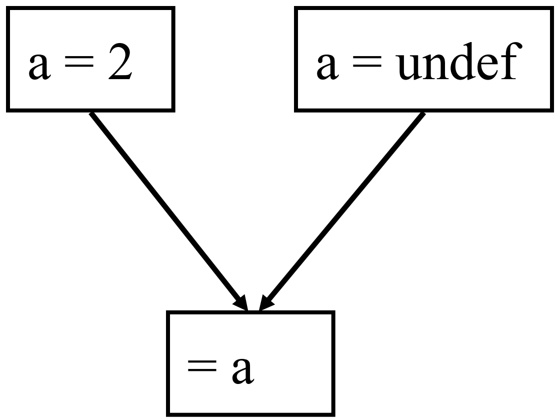
Meet Operations on the Lattice



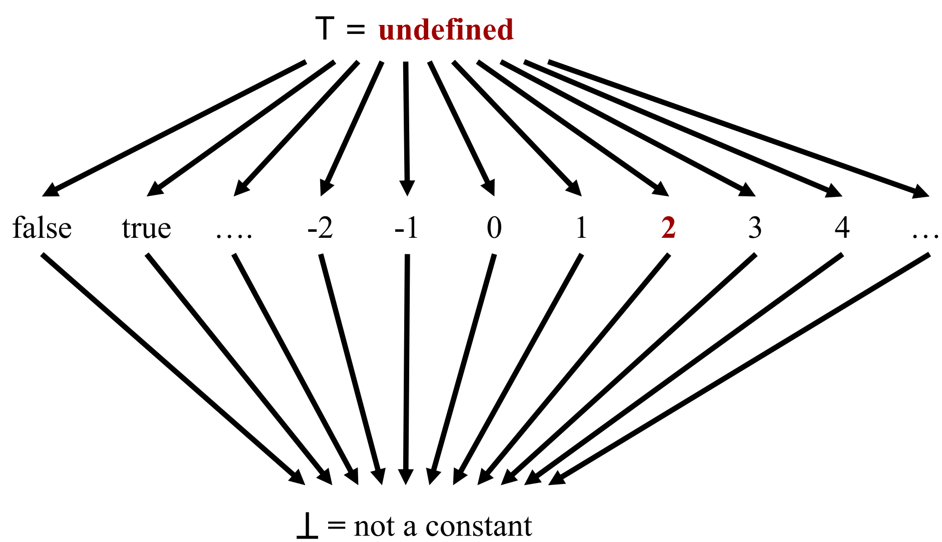
$2 \wedge \text{undef} =$



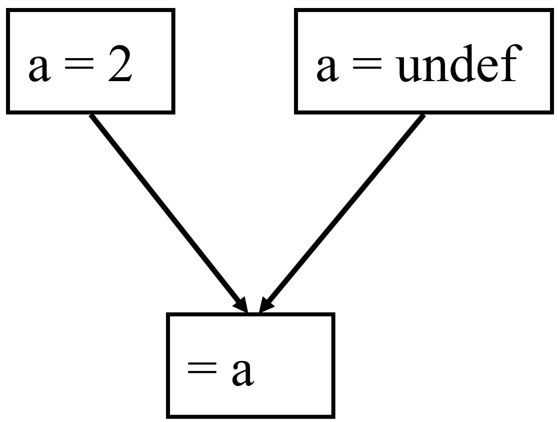
Meet Operations on the Lattice



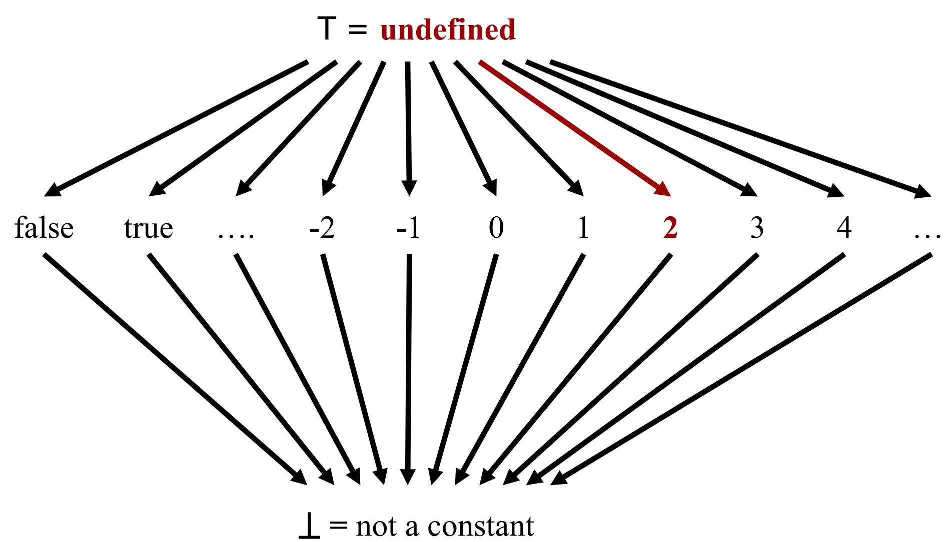
$2 \wedge \text{undef} =$



Meet Operations on the Lattice



$$2 \wedge \text{undef} = 2$$



Constant Propagation Data-Flow Problem

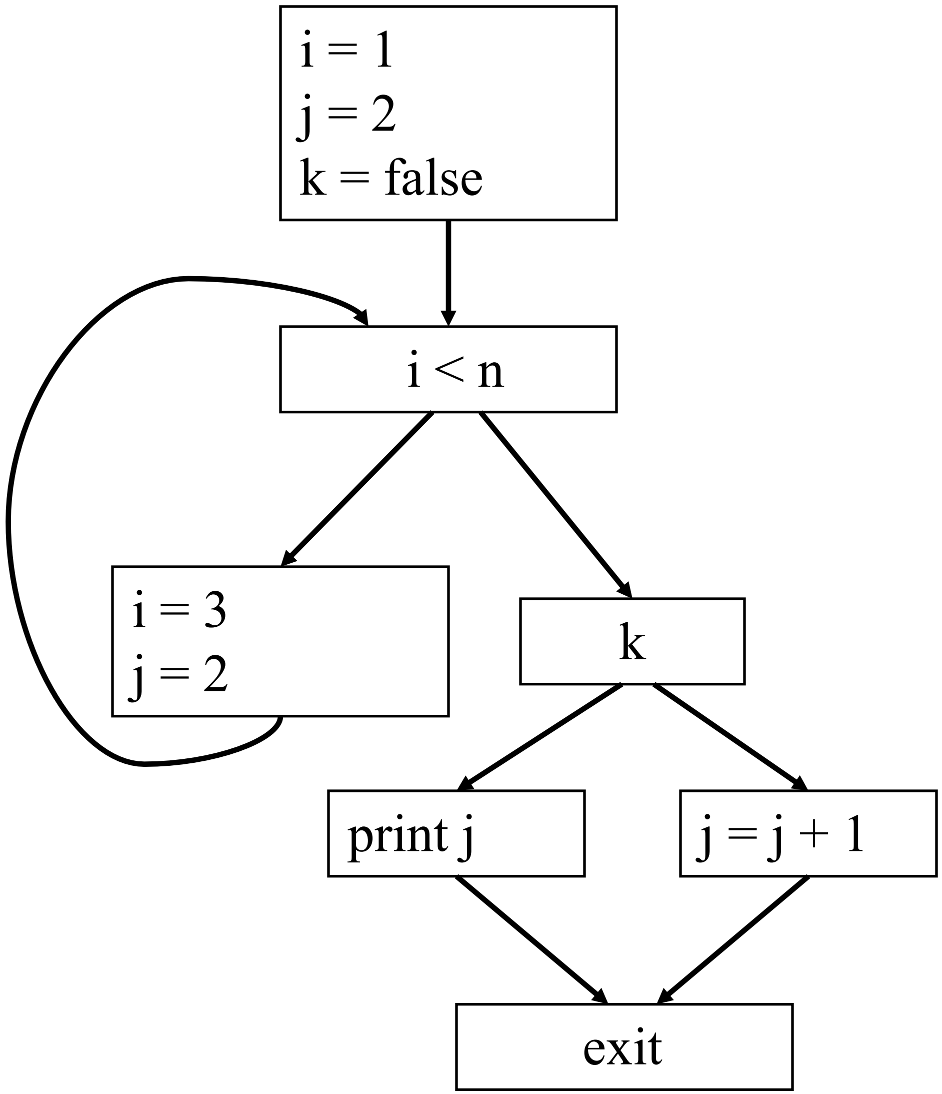
- Domain
  - For each variable a lattice Lv
- Data-Flow Direction
  - Forward
- Data-Flow Function
  - $OUT = gen \wedge (IN \vee prsv)$

Constant Propagation Data-Flow Problem

- Domain
  - For each variable a lattice Lv
- Data-Flow Direction
  - Forward
- Data-Flow Function
  - OUT = gen $\vee$ (IN $\wedge$ prsv)
  - gen $= \left\{ x_v \;\middle|\; x_v = \begin{cases} \top & \text{if v is not LHS} \\ \text{value} & \text{if v is the LHS \& RHS value is a const.} \\ \bot & \text{otherwise} \end{cases} \right\}$
  - prsv $= \left\{ x_v \;\middle|\; x_v = \begin{cases} \top & \text{if v is the LHS} \\ \bot & \text{if v is not the LHS} \end{cases} \right\}$

- **Obs:** For the prsv set, if a variable is not the LHS, we get $\bot$ and the value of Lv is $\bot$; For the gen set, if a variable is not the LHS, we get $\top$ and the value of Lv does not change.

Example



Example

```
i = 1

j = 2

k = false
```

gen ={ i:1, j:2, k:false, n:⊤ }

prsv = { i:⊤, j:⊤, k:⊤, n: ⊥ }

.

$$\text{gen} = \left\{ x_v \;\middle|\; x_v = \begin{cases} \top & \text{if v is not LHS} \\ \text{value} & \text{if v is the LHS \& RHS value is a const.} \\ \bot & \text{otherwise} \end{cases} \right\}$$

$$\text{prsv} = \left\{ x_v \;\middle|\; x_v = \begin{cases} \top & \text{if v is the LHS} \\ \bot & \text{if v is not the LHS} \end{cases} \right\}$$

Example

$j = j + 1$

gen ={ i: ⊤, j: ⊥, k:⊤, n:⊤ }

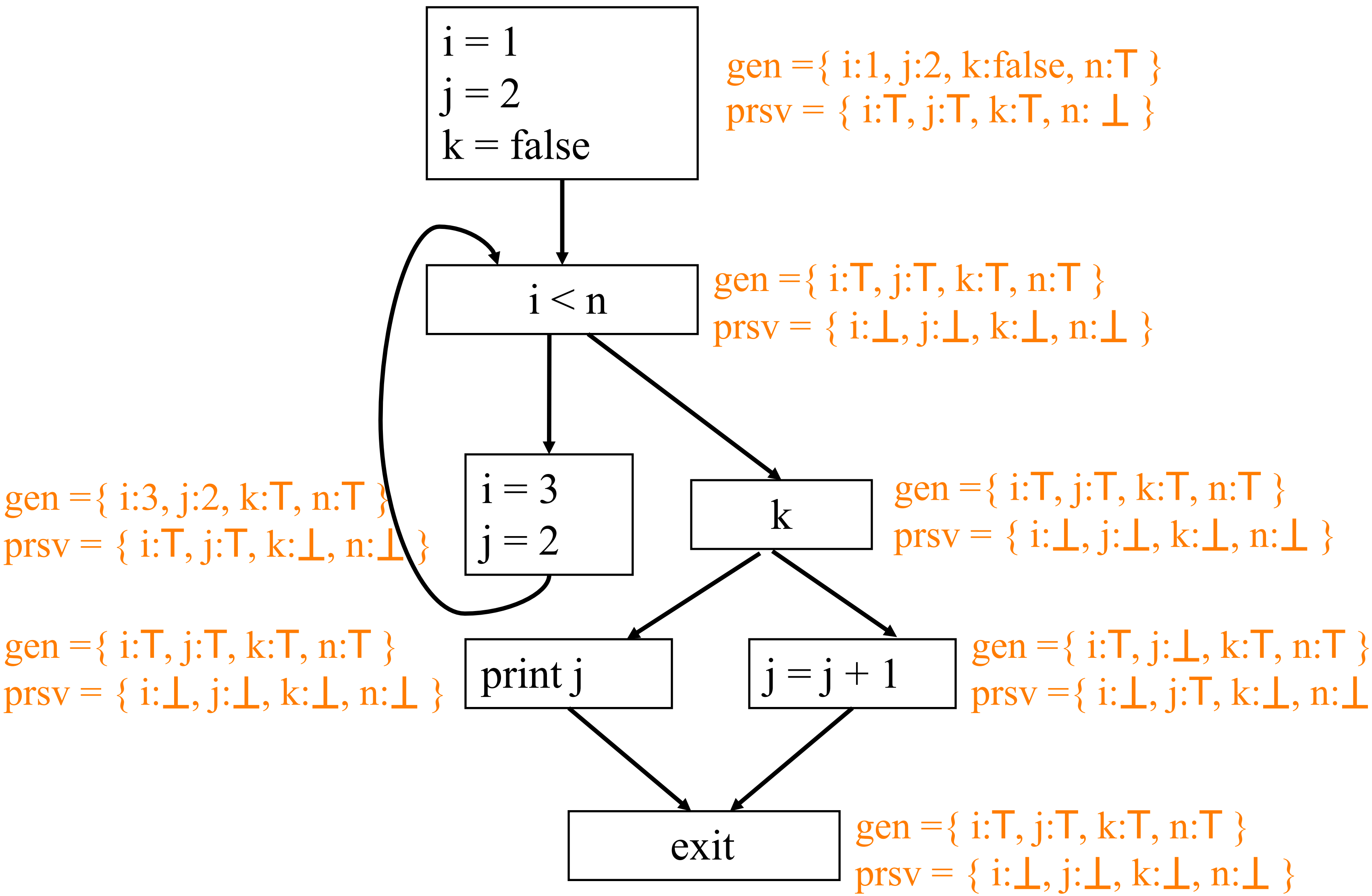
prsv = { i: ⊥, j:⊤, k: ⊥, n: ⊥ }

.

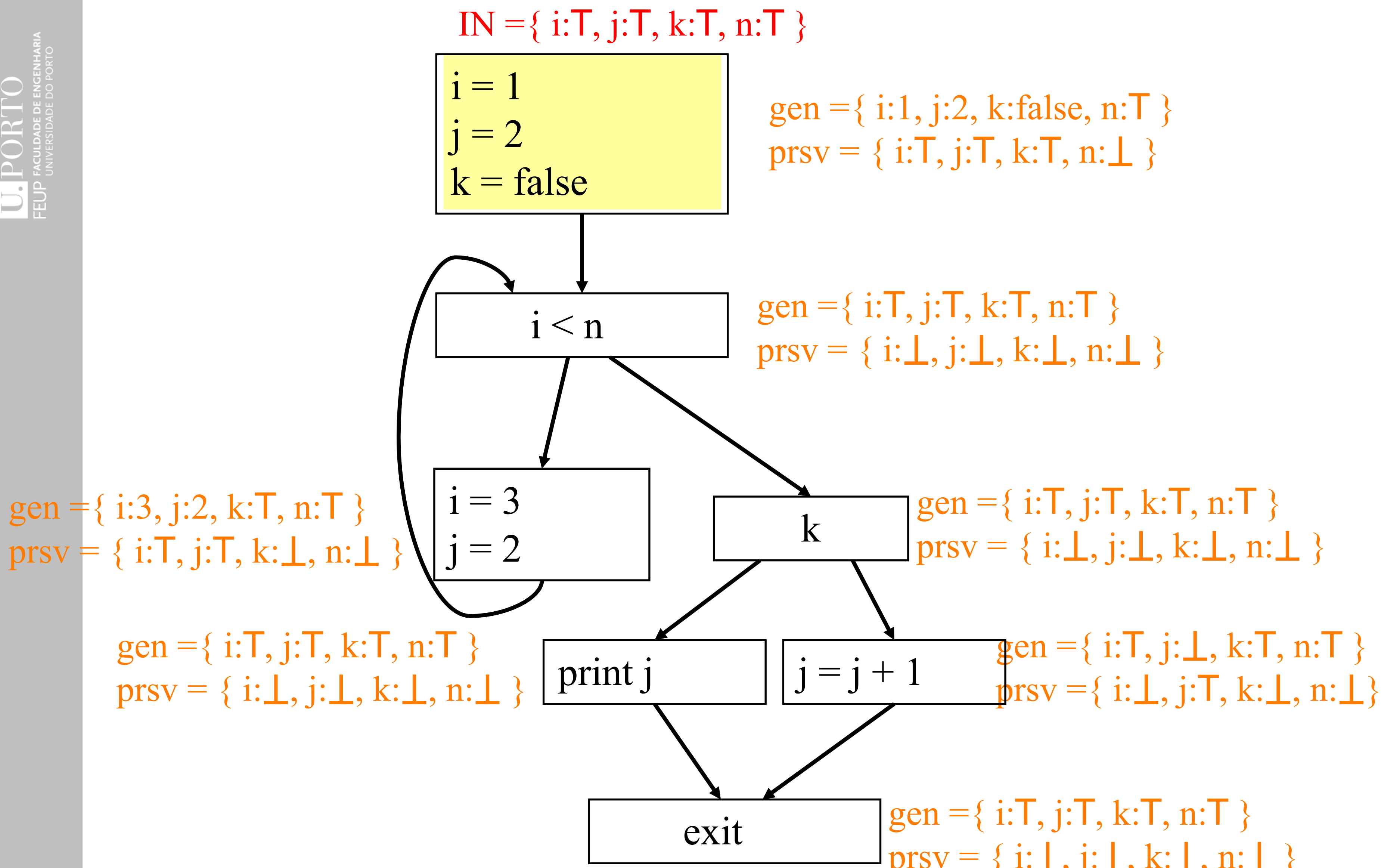
$$\text{gen} = \left\{ x_v \;\middle|\; x_v = \begin{cases} \top & \text{if } v \text{ is not LHS} \\ \text{value} & \text{if } v \text{ is the LHS \& RHS value is a const.} \\ \bot & \text{otherwise} \end{cases} \right\}$$

$$\text{prsv} = \left\{ x_v \;\middle|\; x_v = \begin{cases} \top & \text{if } v \text{ is the LHS} \\ \bot & \text{if } v \text{ is not the LHS} \end{cases} \right\}$$

Example


i = 1
j = 2
k = false
gen ={ i:1, j:2, k:false, n:T }
prsv = { i:T, j:T, k:T, n: ⊥ }
i < n
gen ={ i:T, j:T, k:T, n:T }
prsv = { i:⊥, j:⊥, k:⊥, n:⊥ }
gen ={ i:3, j:2, k:T, n:T }
prsv = { i:T, j:T, k:⊥, n:⊥ }
i = 3
j = 2
k
gen ={ i:T, j:T, k:T, n:T }
prsv = { i:⊥, j:⊥, k:⊥, n:⊥ }
gen ={ i:T, j:T, k:T, n:T }
prsv = { i:⊥, j:⊥, k:⊥, n:⊥ }
print j
j = j + 1
gen ={ i:T, j:⊥, k:T, n:T }
prsv ={ i:⊥, j:T, k:⊥, n:⊥}
exit
gen ={ i:T, j:T, k:T, n:T }
prsv = { i:⊥, j:⊥, k:⊥, n:⊥ }

Example

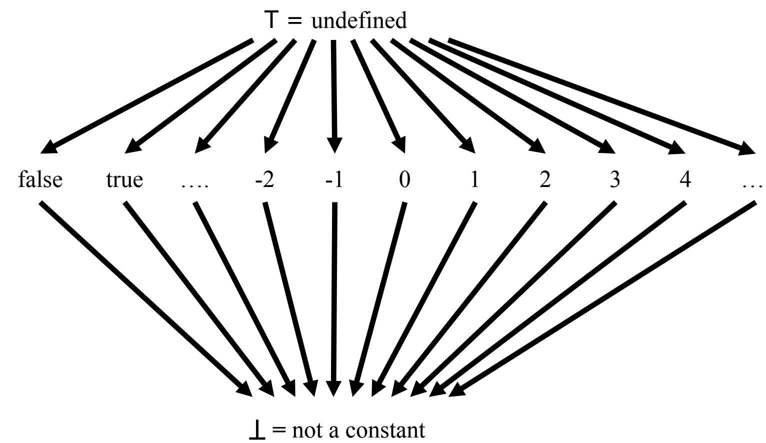

IN ={ i:T, j:T, k:T, n:T }
i = 1
j = 2
k = false
gen ={ i:1, j:2, k:false, n:T }
prsv = { i:T, j:T, k:T, n:⊥ }
i < n
gen ={ i:T, j:T, k:T, n:T }
prsv = { i:⊥, j:⊥, k:⊥, n:⊥ }
gen ={ i:3, j:2, k:T, n:T }
prsv = { i:T, j:T, k:⊥, n:⊥ }
i = 3
j = 2
k
gen ={ i:T, j:T, k:T, n:T }
prsv = { i:⊥, j:⊥, k:⊥, n:⊥ }
gen ={ i:T, j:T, k:T, n:T }
prsv = { i:⊥, j:⊥, k:⊥, n:⊥ }
print j
j = j + 1
gen ={ i:T, j:⊥, k:T, n:T }
prsv ={ i:⊥, j:T, k:⊥, n:⊥}
exit
gen ={ i:T, j:T, k:T, n:T }
prsv = { i:⊥, j:⊥, k:⊥, n:⊥ }

Example

i = 1

j = 2

k = false



gen = { i:1, j:2, k:false, n:$\top$ }

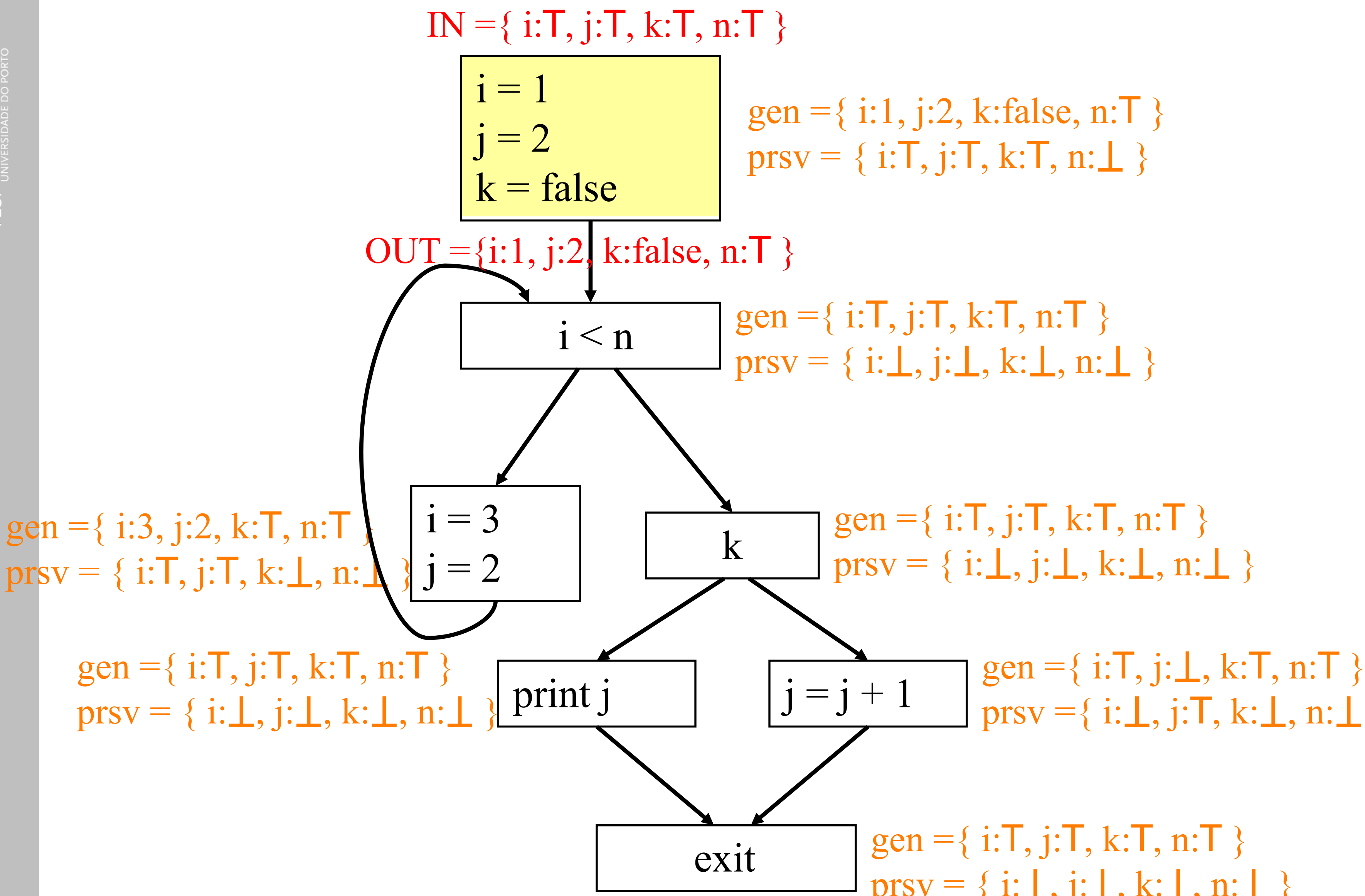
prsv = { i:$\top$, j:$\top$, k:$\top$, n:$\perp$ }

IN = { i:$\top$, j:$\top$, k:$\top$, n:$\top$ }

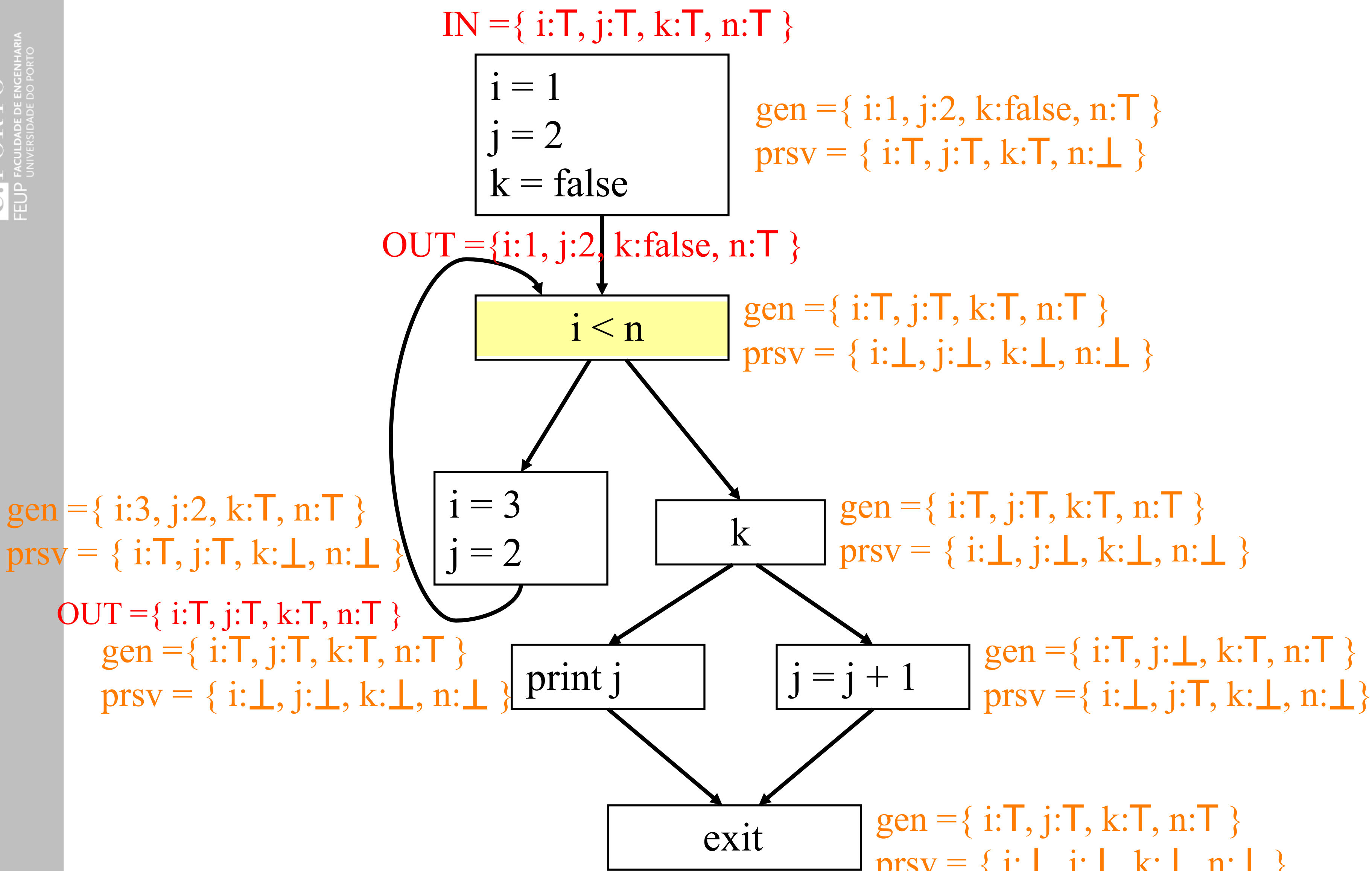
OUT = gen $\wedge$ (IN $\vee$ prsv)

OUT ={ i:1, j:2, k:false, n:$\top$ }

Example

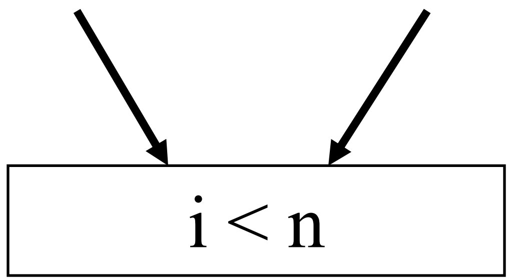

IN ={ i:T, j:T, k:T, n:T }
i = 1
j = 2
k = false
gen ={ i:1, j:2, k:false, n:T }
prsv = { i:T, j:T, k:T, n:⊥ }
OUT ={i:1, j:2, k:false, n:T }
i < n
gen ={ i:T, j:T, k:T, n:T }
prsv = { i:⊥, j:⊥, k:⊥, n:⊥ }
gen ={ i:3, j:2, k:T, n:T }
prsv = { i:T, j:T, k:⊥, n:⊥ }
i = 3
j = 2
k
gen ={ i:T, j:T, k:T, n:T }
prsv = { i:⊥, j:⊥, k:⊥, n:⊥ }
gen ={ i:T, j:T, k:T, n:T }
prsv = { i:⊥, j:⊥, k:⊥, n:⊥ }
print j
j = j + 1
gen ={ i:T, j:⊥, k:T, n:T }
prsv ={ i:⊥, j:T, k:⊥, n:⊥}
exit
gen ={ i:T, j:T, k:T, n:T }
prsv = { i:⊥, j:⊥, k:⊥, n:⊥ }

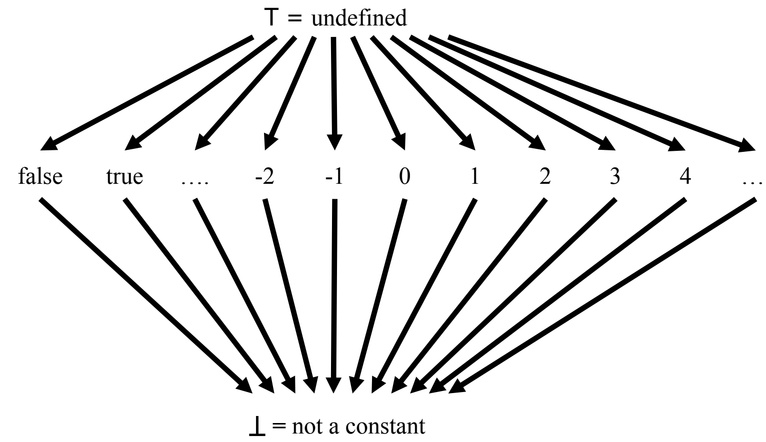
Example



IN ={ i:T, j:T, k:T, n:T }
i = 1
j = 2
k = false
gen ={ i:1, j:2, k:false, n:T }
prsv = { i:T, j:T, k:T, n:⊥ }
OUT ={i:1, j:2, k:false, n:T }
i < n
gen ={ i:T, j:T, k:T, n:T }
prsv = { i:⊥, j:⊥, k:⊥, n:⊥ }
gen ={ i:3, j:2, k:T, n:T }
prsv = { i:T, j:T, k:⊥, n:⊥ }
i = 3
j = 2
k
gen ={ i:T, j:T, k:T, n:T }
prsv = { i:⊥, j:⊥, k:⊥, n:⊥ }
OUT ={ i:T, j:T, k:T, n:T }
gen ={ i:T, j:T, k:T, n:T }
prsv = { i:⊥, j:⊥, k:⊥, n:⊥ }
print j
j = j + 1
gen ={ i:T, j:⊥, k:T, n:T }
prsv ={ i:⊥, j:T, k:⊥, n:⊥}
exit
gen ={ i:T, j:T, k:T, n:T }
prsv = { i:⊥, j:⊥, k:⊥, n:⊥ }

Example





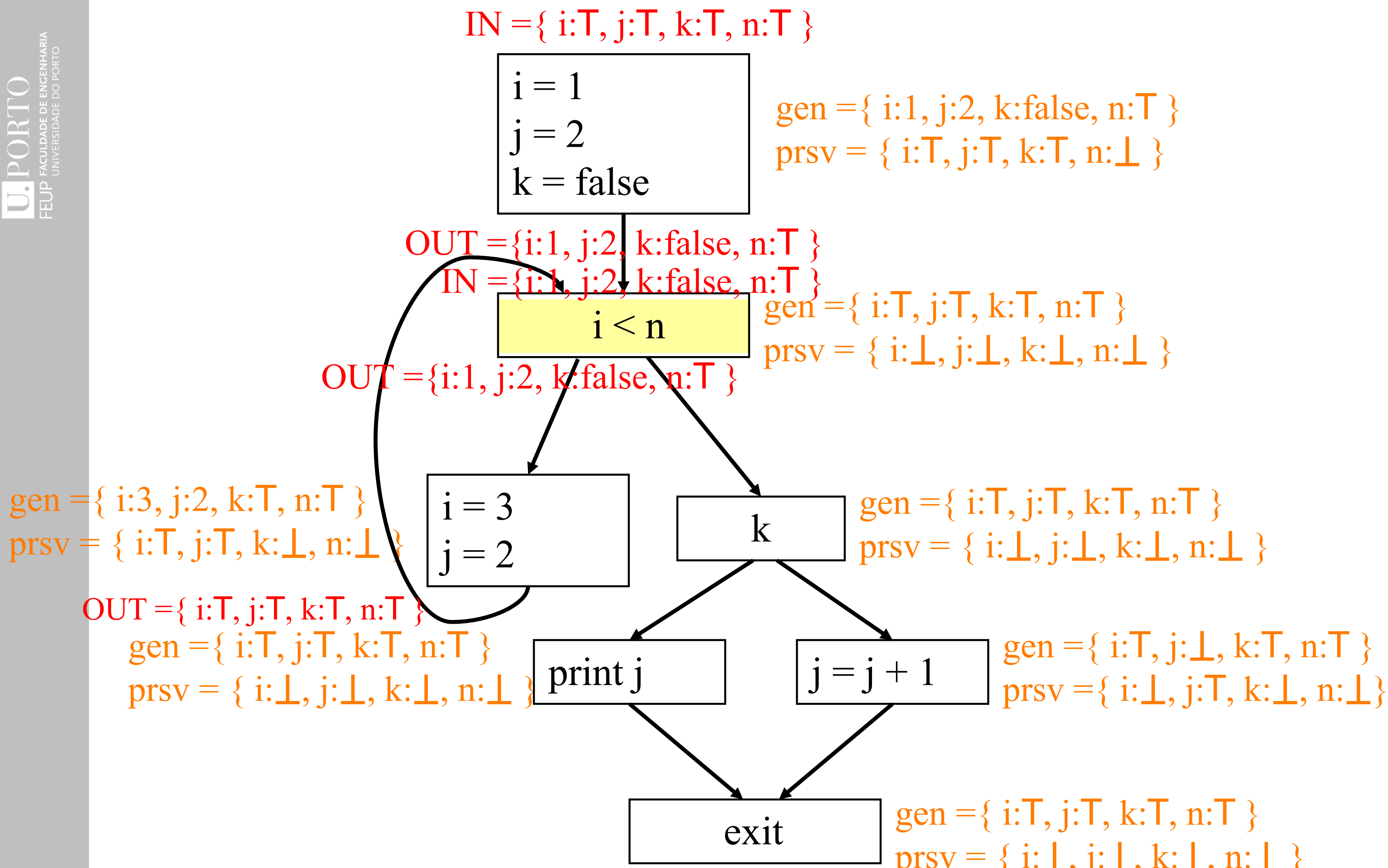
out1 = { i:$\top$, j:$\top$, k:$\top$, n:$\top$ }

out2 = { i:1, j:2, k:false, n:$\top$ }

IN = out1 $\wedge$ out2

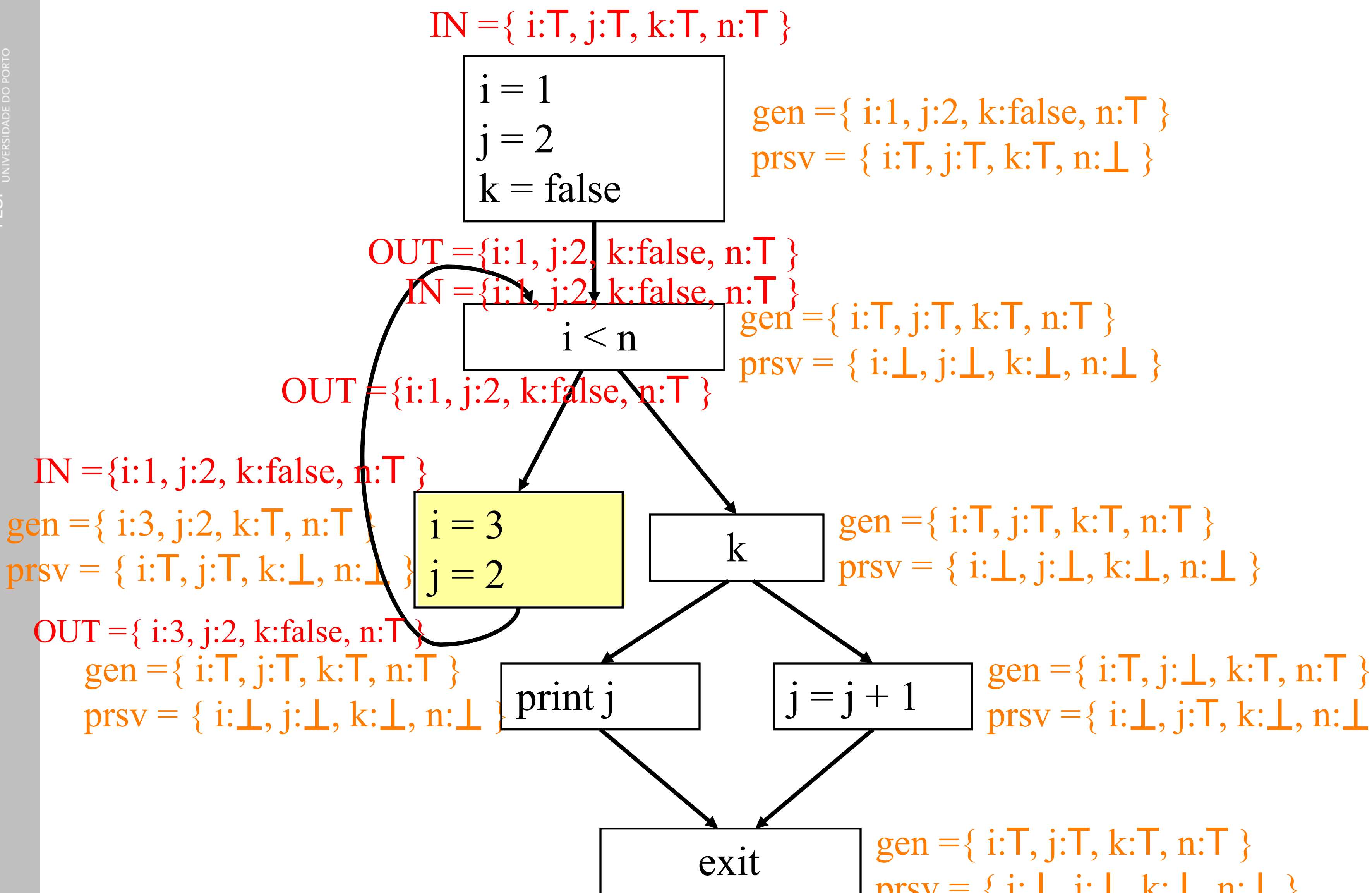
IN = { i:1, j:2, k:false, n:$\top$ }

Example



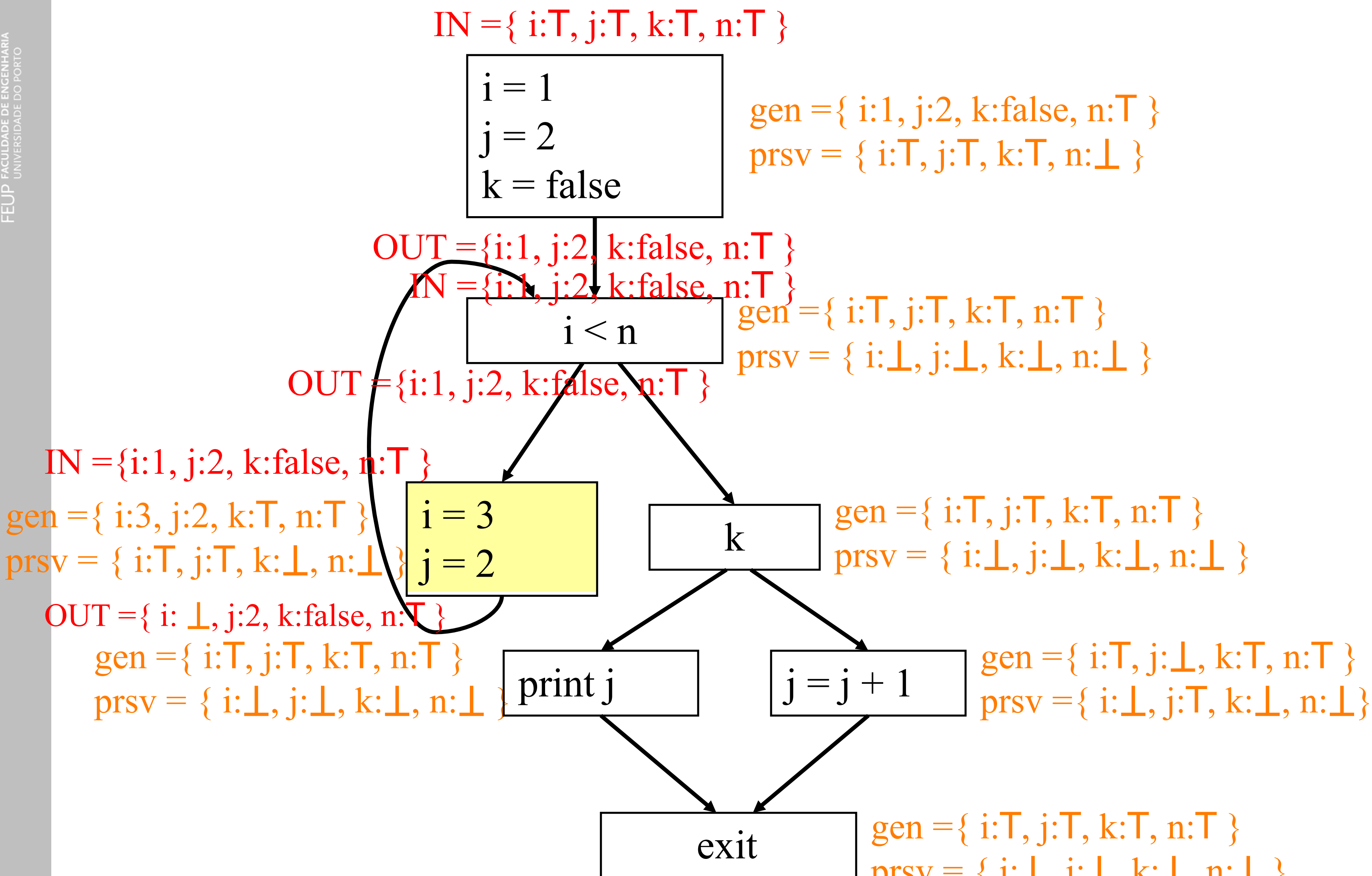
IN ={ i:T, j:T, k:T, n:T }
i = 1
j = 2
k = false
gen ={ i:1, j:2, k:false, n:T }
prsv = { i:T, j:T, k:T, n:⊥ }
OUT ={i:1, j:2, k:false, n:T }
IN ={i:1, j:2, k:false, n:T }
i < n
gen ={ i:T, j:T, k:T, n:T }
prsv = { i:⊥, j:⊥, k:⊥, n:⊥ }
OUT ={i:1, j:2, k:false, n:T }
gen ={ i:3, j:2, k:T, n:T }
prsv = { i:T, j:T, k:⊥, n:⊥ }
i = 3
j = 2
k
gen ={ i:T, j:T, k:T, n:T }
prsv = { i:⊥, j:⊥, k:⊥, n:⊥ }
OUT ={ i:T, j:T, k:T, n:T }
gen ={ i:T, j:T, k:T, n:T }
prsv = { i:⊥, j:⊥, k:⊥, n:⊥ }
print j
j = j + 1
gen ={ i:T, j:⊥, k:T, n:T }
prsv ={ i:⊥, j:T, k:⊥, n:⊥}
exit
gen ={ i:T, j:T, k:T, n:T }
prsv = { i:⊥, j:⊥, k:⊥, n:⊥ }

Example



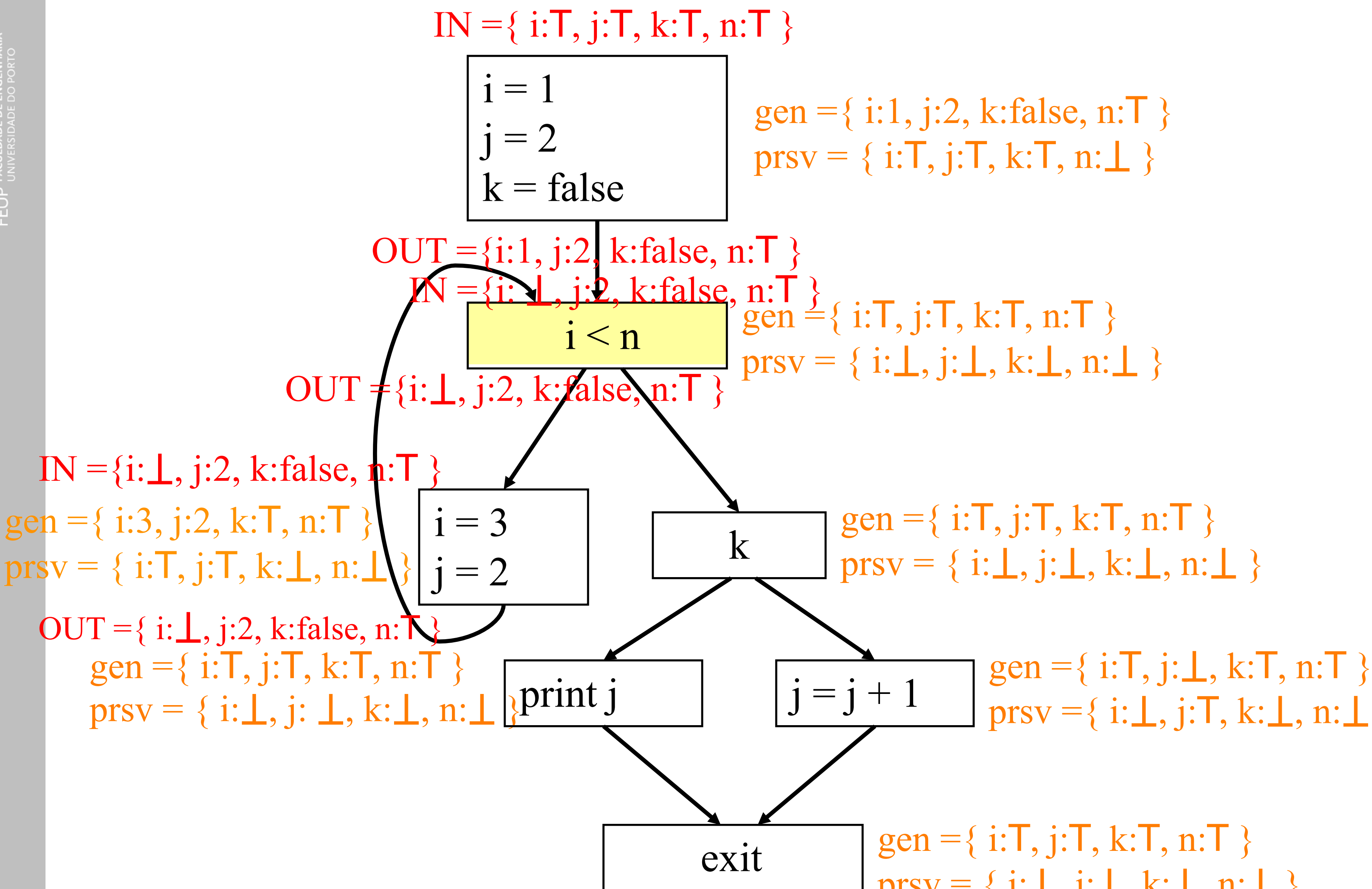
IN ={ i:T, j:T, k:T, n:T }
i = 1
j = 2
k = false
gen ={ i:1, j:2, k:false, n:T }
prsv = { i:T, j:T, k:T, n:⊥ }
OUT ={i:1, j:2, k:false, n:T }
IN ={i:1, j:2, k:false, n:T }
i < n
gen ={ i:T, j:T, k:T, n:T }
prsv = { i:⊥, j:⊥, k:⊥, n:⊥ }
OUT ={i:1, j:2, k:false, n:T }
IN ={i:1, j:2, k:false, n:T }
gen ={ i:3, j:2, k:T, n:T }
prsv = { i:T, j:T, k:⊥, n:⊥ }
i = 3
j = 2
k
gen ={ i:T, j:T, k:T, n:T }
prsv = { i:⊥, j:⊥, k:⊥, n:⊥ }
OUT ={ i:3, j:2, k:false, n:T }
gen ={ i:T, j:T, k:T, n:T }
prsv = { i:⊥, j:⊥, k:⊥, n:⊥ }
print j
j = j + 1
gen ={ i:T, j:⊥, k:T, n:T }
prsv ={ i:⊥, j:T, k:⊥, n:⊥}
exit
gen ={ i:T, j:T, k:T, n:T }
prsv = { i:⊥, j:⊥, k:⊥, n:⊥ }

Example

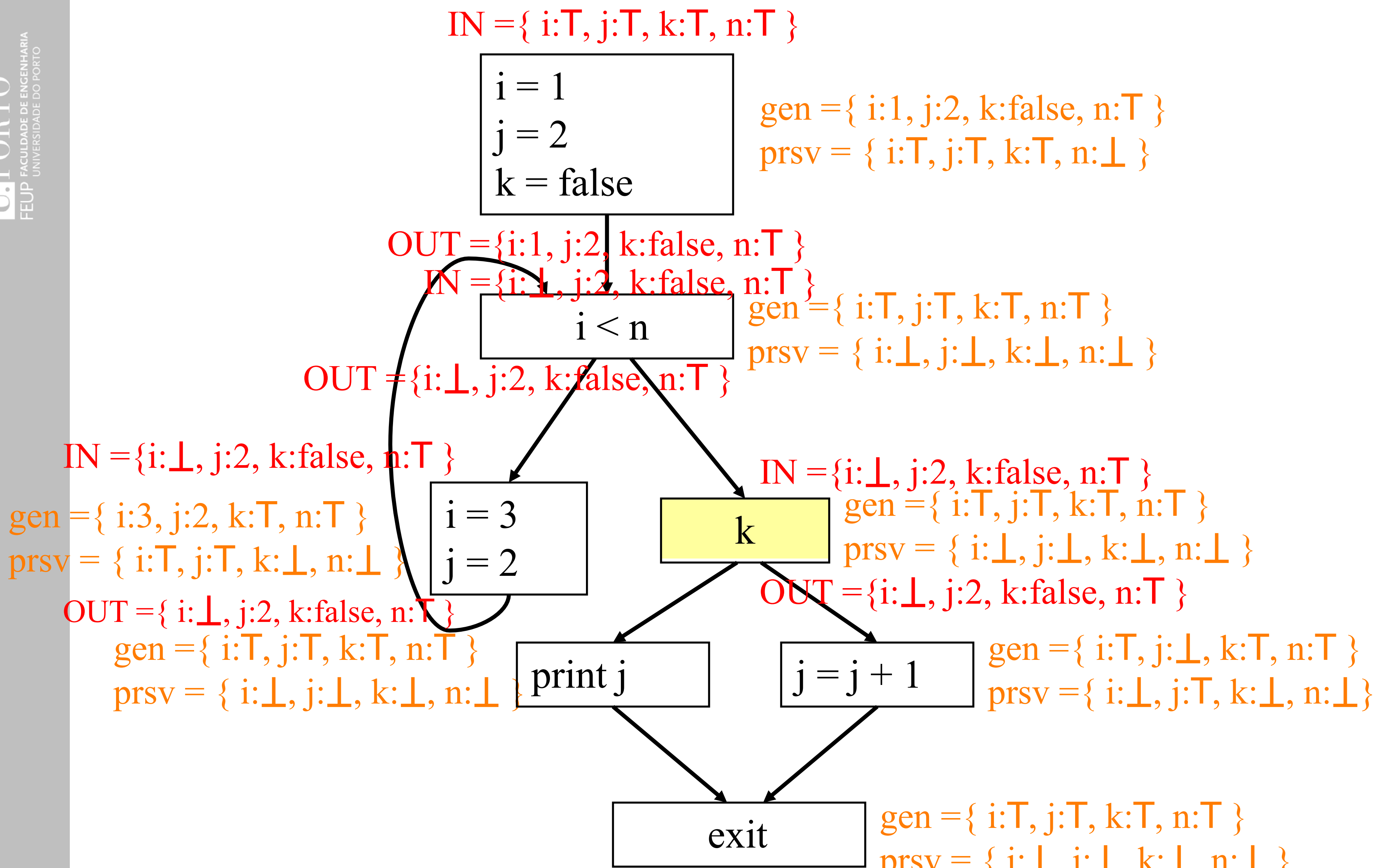


IN ={ i:T, j:T, k:T, n:T }
i = 1
j = 2
k = false
gen ={ i:1, j:2, k:false, n:T }
prsv = { i:T, j:T, k:T, n:⊥ }
OUT ={i:1, j:2, k:false, n:T }
IN ={i:1, j:2, k:false, n:T }
i < n
gen ={ i:T, j:T, k:T, n:T }
prsv = { i:⊥, j:⊥, k:⊥, n:⊥ }
OUT ={i:1, j:2, k:false, n:T }
IN ={i:1, j:2, k:false, n:T }
gen ={ i:3, j:2, k:T, n:T }
prsv = { i:T, j:T, k:⊥, n:⊥ }
i = 3
j = 2
k
gen ={ i:T, j:T, k:T, n:T }
prsv = { i:⊥, j:⊥, k:⊥, n:⊥ }
OUT ={ i: ⊥, j:2, k:false, n:T }
gen ={ i:T, j:T, k:T, n:T }
prsv = { i:⊥, j:⊥, k:⊥, n:⊥ }
print j
j = j + 1
gen ={ i:T, j:⊥, k:T, n:T }
prsv ={ i:⊥, j:T, k:⊥, n:⊥}
exit
gen ={ i:T, j:T, k:T, n:T }
prsv = { i:⊥, j:⊥, k:⊥, n:⊥ }

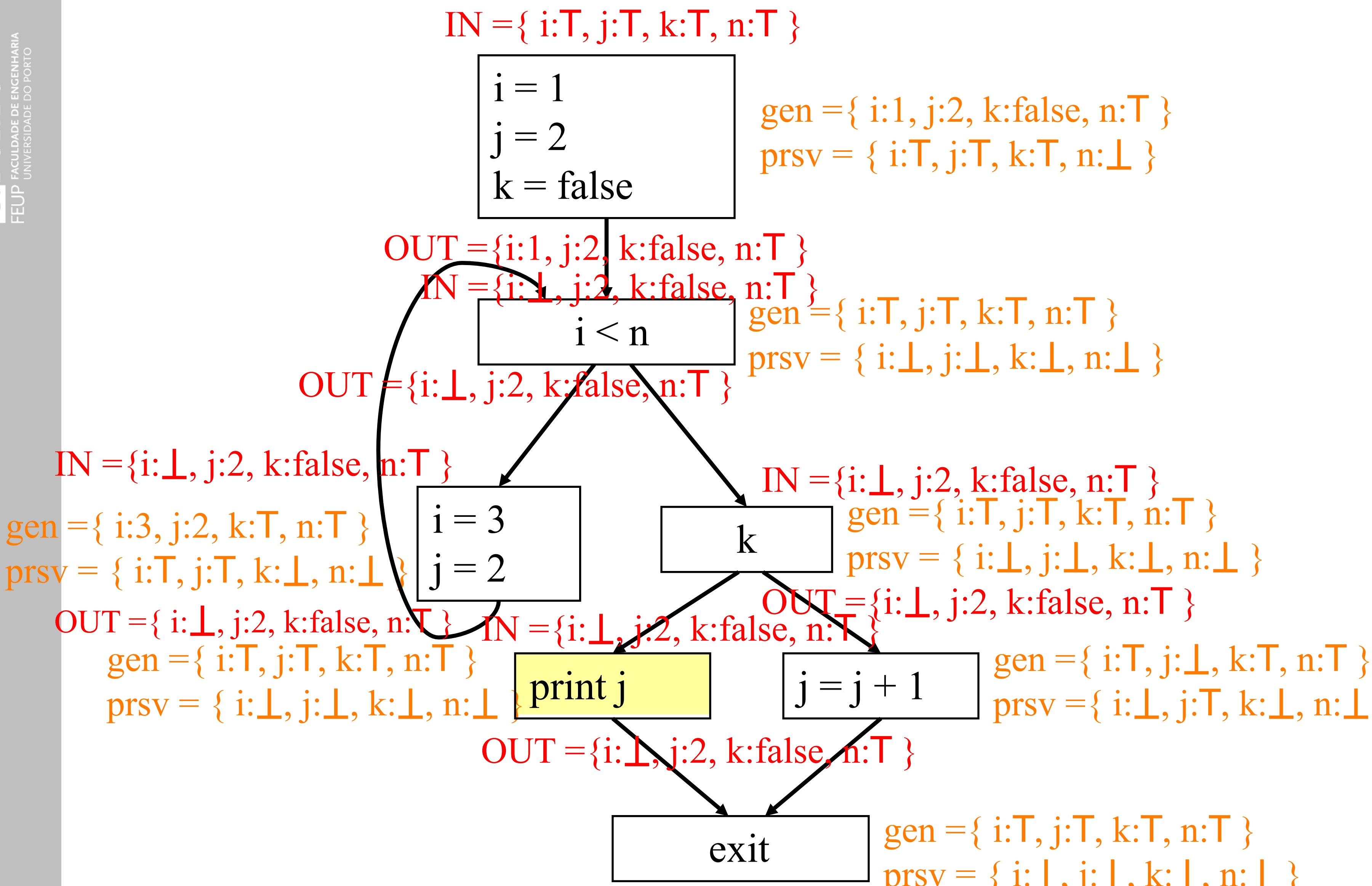
Example


IN ={ i:T, j:T, k:T, n:T }
i = 1
j = 2
k = false
gen ={ i:1, j:2, k:false, n:T }
prsv = { i:T, j:T, k:T, n:⊥ }
OUT ={i:1, j:2, k:false, n:T }
IN ={i: ⊥, j:2, k:false, n:T }
i < n
gen ={ i:T, j:T, k:T, n:T }
prsv = { i:⊥, j:⊥, k:⊥, n:⊥ }
OUT ={i:⊥, j:2, k:false, n:T }
IN ={i:⊥, j:2, k:false, n:T }
gen ={ i:3, j:2, k:T, n:T }
prsv = { i:T, j:T, k:⊥, n:⊥ }
i = 3
j = 2
k
gen ={ i:T, j:T, k:T, n:T }
prsv = { i:⊥, j:⊥, k:⊥, n:⊥ }
OUT ={ i:⊥, j:2, k:false, n:T }
gen ={ i:T, j:T, k:T, n:T }
prsv = { i:⊥, j: ⊥, k:⊥, n:⊥ }
print j
j = j + 1
gen ={ i:T, j:⊥, k:T, n:T }
prsv ={ i:⊥, j:T, k:⊥, n:⊥}
exit
gen ={ i:T, j:T, k:T, n:T }
prsv = { i:⊥, j:⊥, k:⊥, n:⊥ }

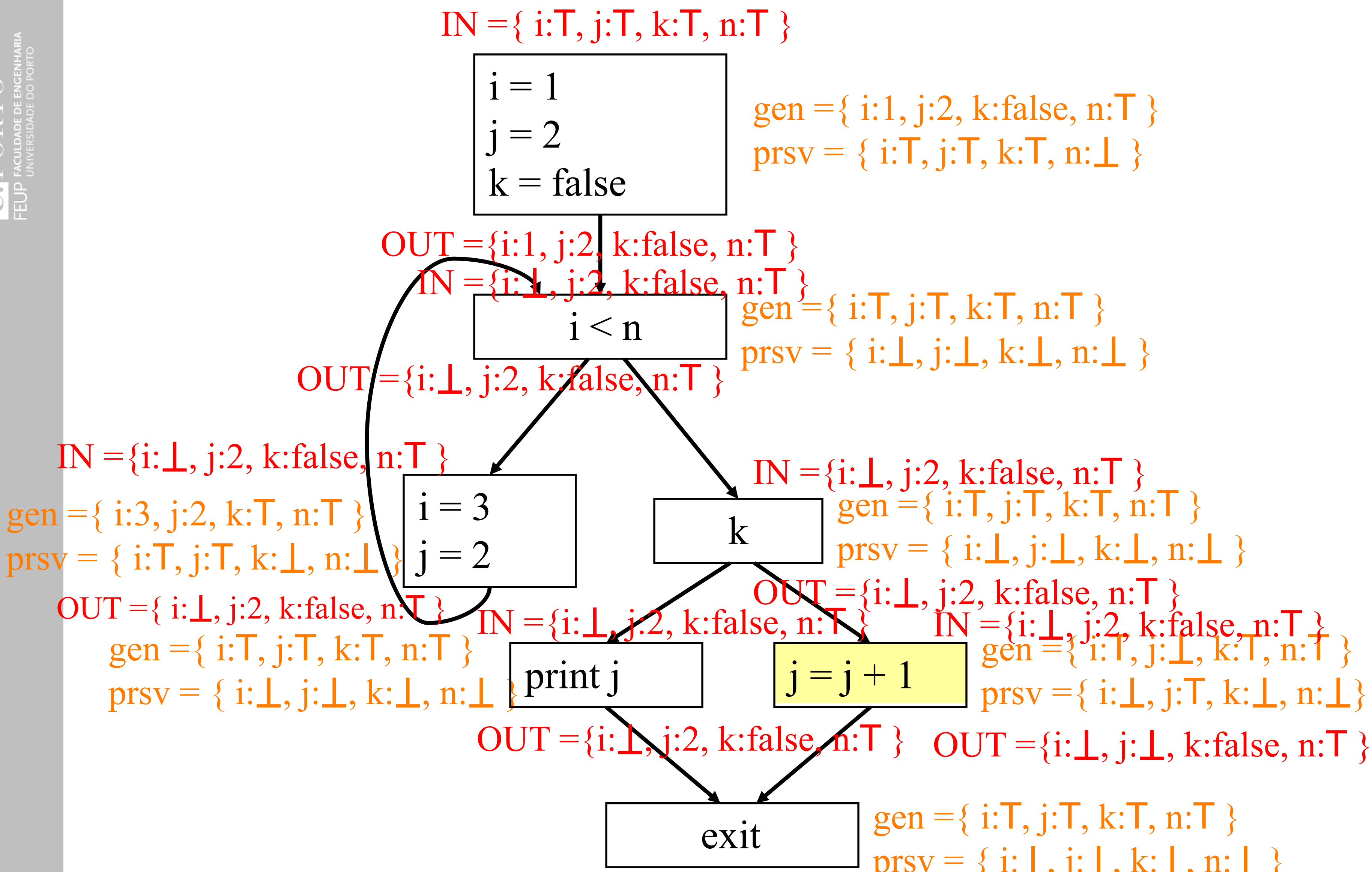
Example


IN ={ i:⊤, j:⊤, k:⊤, n:⊤ }
i = 1
j = 2
k = false
gen ={ i:1, j:2, k:false, n:⊤ }
prsv = { i:⊤, j:⊤, k:⊤, n:⊥ }
OUT ={i:1, j:2, k:false, n:⊤ }
IN ={i:⊥, j:2, k:false, n:⊤ }
i < n
gen ={ i:⊤, j:⊤, k:⊤, n:⊤ }
prsv = { i:⊥, j:⊥, k:⊥, n:⊥ }
OUT ={i:⊥, j:2, k:false, n:⊤ }
IN ={i:⊥, j:2, k:false, n:⊤ }
i = 3
j = 2
gen ={ i:3, j:2, k:⊤, n:⊤ }
prsv = { i:⊤, j:⊤, k:⊥, n:⊥ }
OUT ={ i:⊥, j:2, k:false, n:⊤ }
IN ={i:⊥, j:2, k:false, n:⊤ }
k
gen ={ i:⊤, j:⊤, k:⊤, n:⊤ }
prsv = { i:⊥, j:⊥, k:⊥, n:⊥ }
OUT ={i:⊥, j:2, k:false, n:⊤ }
print j
gen ={ i:⊤, j:⊤, k:⊤, n:⊤ }
prsv = { i:⊥, j:⊥, k:⊥, n:⊥ }
j = j + 1
gen ={ i:⊤, j:⊥, k:⊤, n:⊤ }
prsv ={ i:⊥, j:⊤, k:⊥, n:⊥}
exit
gen ={ i:⊤, j:⊤, k:⊤, n:⊤ }
prsv = { i:⊥, j:⊥, k:⊥, n:⊥ }

Example

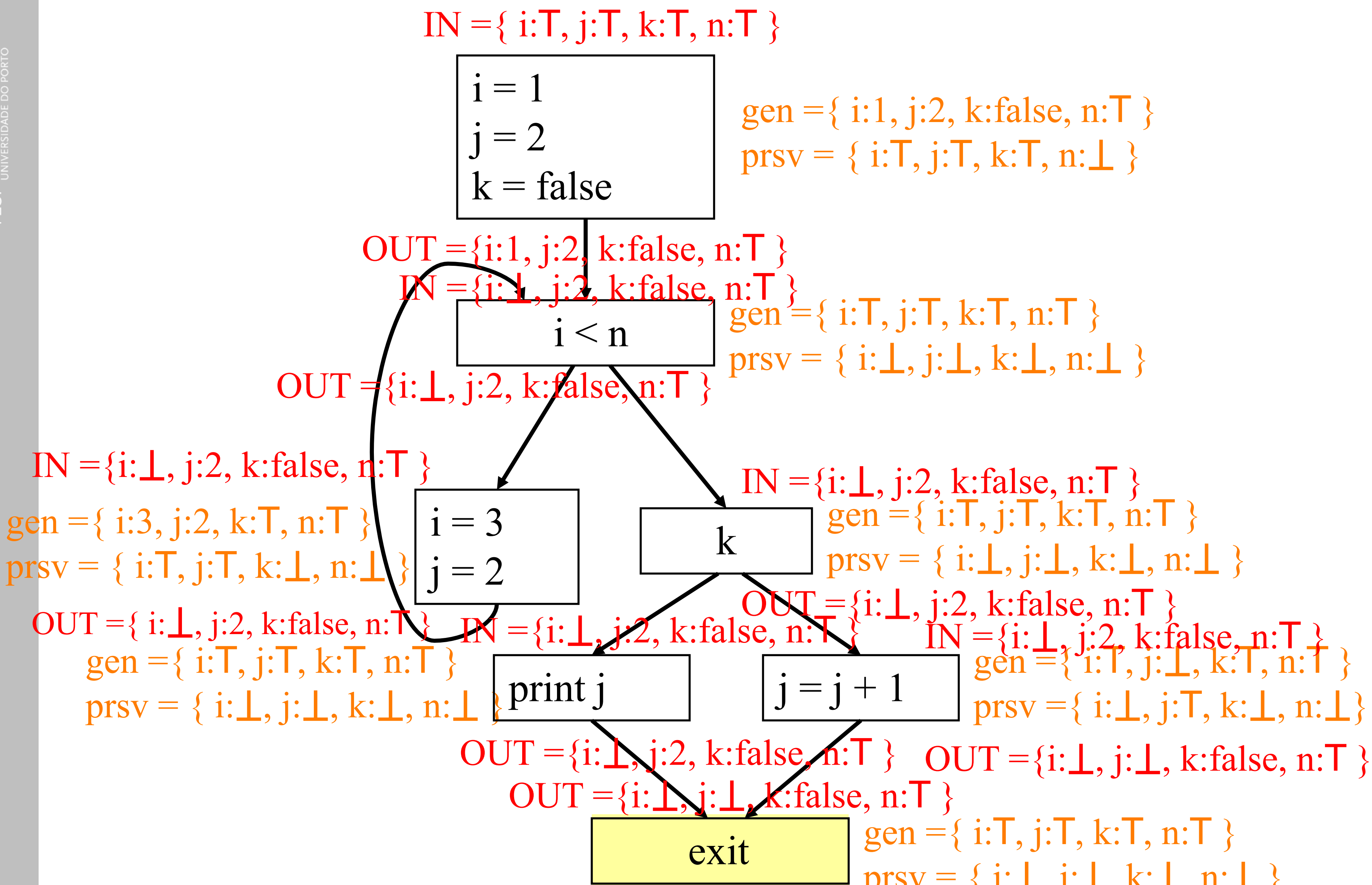

IN ={ i:⊤, j:⊤, k:⊤, n:⊤ }
i = 1
j = 2
k = false
gen ={ i:1, j:2, k:false, n:⊤ }
prsv = { i:⊤, j:⊤, k:⊤, n:⊥ }
OUT ={i:1, j:2, k:false, n:⊤ }
IN ={i:⊥, j:2, k:false, n:⊤ }
i < n
gen ={ i:⊤, j:⊤, k:⊤, n:⊤ }
prsv = { i:⊥, j:⊥, k:⊥, n:⊥ }
OUT ={i:⊥, j:2, k:false, n:⊤ }
IN ={i:⊥, j:2, k:false, n:⊤ }
gen ={ i:3, j:2, k:⊤, n:⊤ }
prsv = { i:⊤, j:⊤, k:⊥, n:⊥ }
i = 3
j = 2
OUT ={ i:⊥, j:2, k:false, n:⊤ }
IN ={i:⊥, j:2, k:false, n:⊤ }
k
gen ={ i:⊤, j:⊤, k:⊤, n:⊤ }
prsv = { i:⊥, j:⊥, k:⊥, n:⊥ }
OUT ={i:⊥, j:2, k:false, n:⊤ }
IN ={i:⊥, j:2, k:false, n:⊤ }
gen ={ i:⊤, j:⊤, k:⊤, n:⊤ }
prsv = { i:⊥, j:⊥, k:⊥, n:⊥ }
print j
j = j + 1
gen ={ i:⊤, j:⊥, k:⊤, n:⊤ }
prsv ={ i:⊥, j:⊤, k:⊥, n:⊥}
OUT ={i:⊥, j:2, k:false, n:⊤ }
exit
gen ={ i:⊤, j:⊤, k:⊤, n:⊤ }
prsv = { i:⊥, j:⊥, k:⊥, n:⊥ }

Example

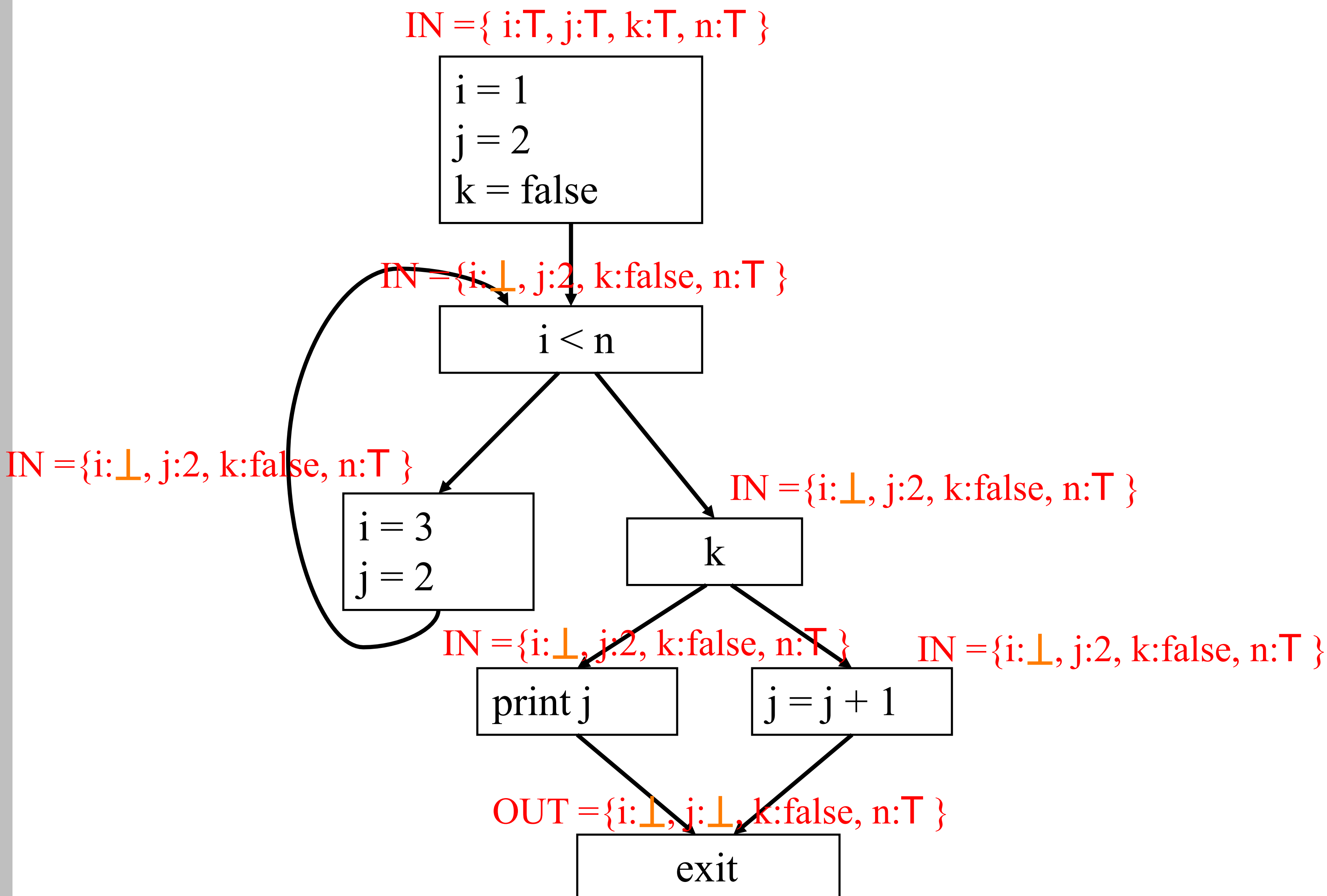


IN ={ i:T, j:T, k:T, n:T }
i = 1
j = 2
k = false
gen ={ i:1, j:2, k:false, n:T }
prsv = { i:T, j:T, k:T, n:⊥ }
OUT ={i:1, j:2, k:false, n:T }
IN ={i:⊥, j:2, k:false, n:T }
i < n
gen ={ i:T, j:T, k:T, n:T }
prsv = { i:⊥, j:⊥, k:⊥, n:⊥ }
OUT ={i:⊥, j:2, k:false, n:T }
IN ={i:⊥, j:2, k:false, n:T }
i = 3
j = 2
gen ={ i:3, j:2, k:T, n:T }
prsv = { i:T, j:T, k:⊥, n:⊥ }
OUT ={ i:⊥, j:2, k:false, n:T }
IN ={i:⊥, j:2, k:false, n:T }
k
gen ={ i:T, j:T, k:T, n:T }
prsv = { i:⊥, j:⊥, k:⊥, n:⊥ }
OUT ={i:⊥, j:2, k:false, n:T }
IN ={i:⊥, j:2, k:false, n:T }
print j
gen ={ i:T, j:T, k:T, n:T }
prsv = { i:⊥, j:⊥, k:⊥, n:⊥ }
OUT ={i:⊥, j:2, k:false, n:T }
IN ={i:⊥, j:2, k:false, n:T }
j = j + 1
gen ={ i:T, j:⊥, k:T, n:T }
prsv ={ i:⊥, j:T, k:⊥, n:⊥}
OUT ={i:⊥, j:⊥, k:false, n:T }
exit
gen ={ i:T, j:T, k:T, n:T }
prsv = { i:⊥, j:⊥, k:⊥, n:⊥ }

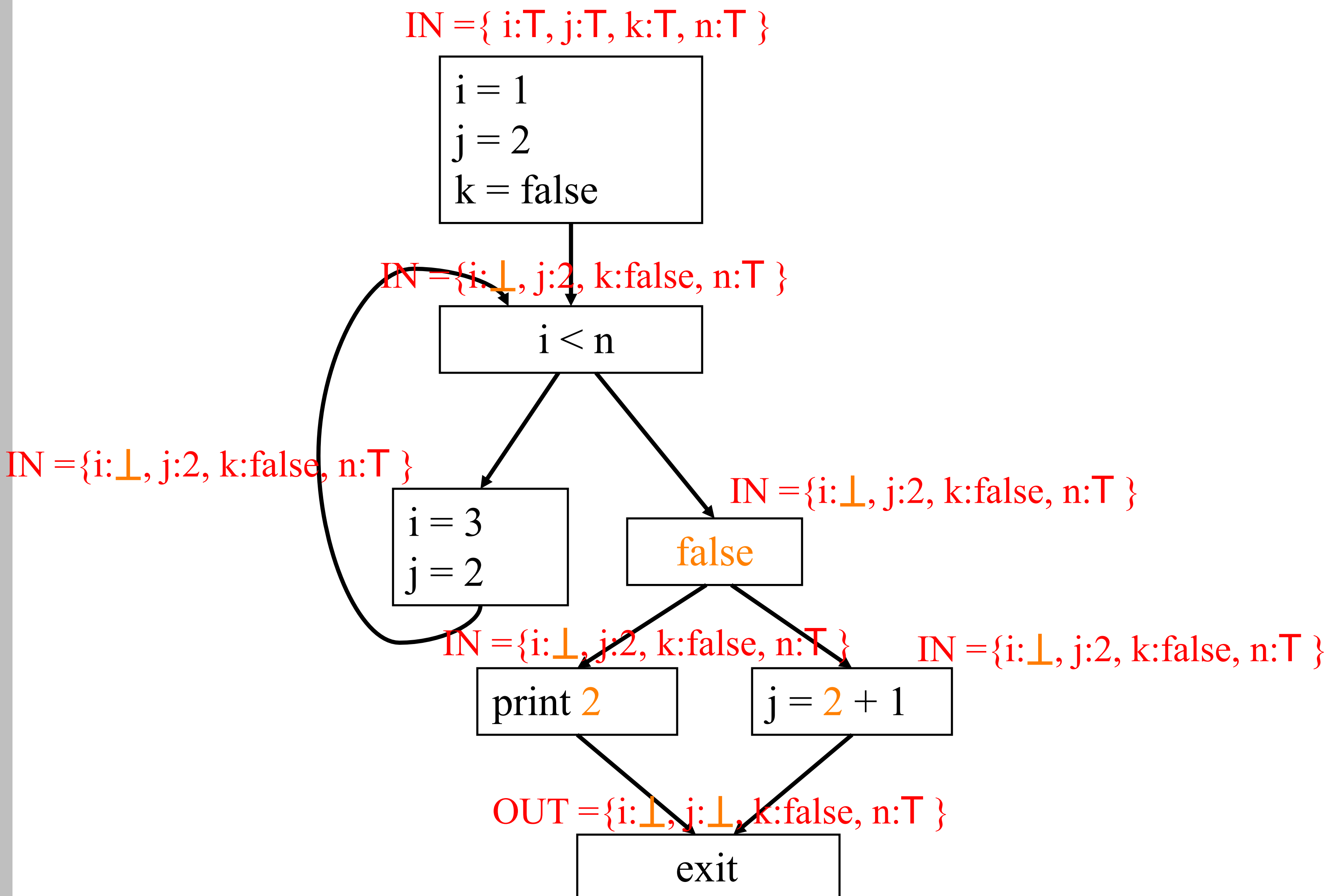
Example


IN ={ i:T, j:T, k:T, n:T }
i = 1
j = 2
k = false
gen ={ i:1, j:2, k:false, n:T }
prsv = { i:T, j:T, k:T, n:⊥ }
OUT ={i:1, j:2, k:false, n:T }
IN ={i:⊥, j:2, k:false, n:T }
i < n
gen ={ i:T, j:T, k:T, n:T }
prsv = { i:⊥, j:⊥, k:⊥, n:⊥ }
OUT ={i:⊥, j:2, k:false, n:T }
IN ={i:⊥, j:2, k:false, n:T }
gen ={ i:3, j:2, k:T, n:T }
prsv = { i:T, j:T, k:⊥, n:⊥ }
i = 3
j = 2
OUT ={ i:⊥, j:2, k:false, n:T }
IN ={i:⊥, j:2, k:false, n:T }
k
gen ={ i:T, j:T, k:T, n:T }
prsv = { i:⊥, j:⊥, k:⊥, n:⊥ }
OUT ={i:⊥, j:2, k:false, n:T }
IN ={i:⊥, j:2, k:false, n:T }
gen ={ i:T, j:T, k:T, n:T }
prsv = { i:⊥, j:⊥, k:⊥, n:⊥ }
print j
IN ={i:⊥, j:2, k:false, n:T }
j = j + 1
gen ={ i:T, j:⊥, k:T, n:T }
prsv ={ i:⊥, j:T, k:⊥, n:⊥}
OUT ={i:⊥, j:2, k:false, n:T }
OUT ={i:⊥, j:⊥, k:false, n:T }
OUT ={i:⊥, j:⊥, k:false, n:T }
exit
gen ={ i:T, j:T, k:T, n:T }
prsv = { i:⊥, j:⊥, k:⊥, n:⊥ }

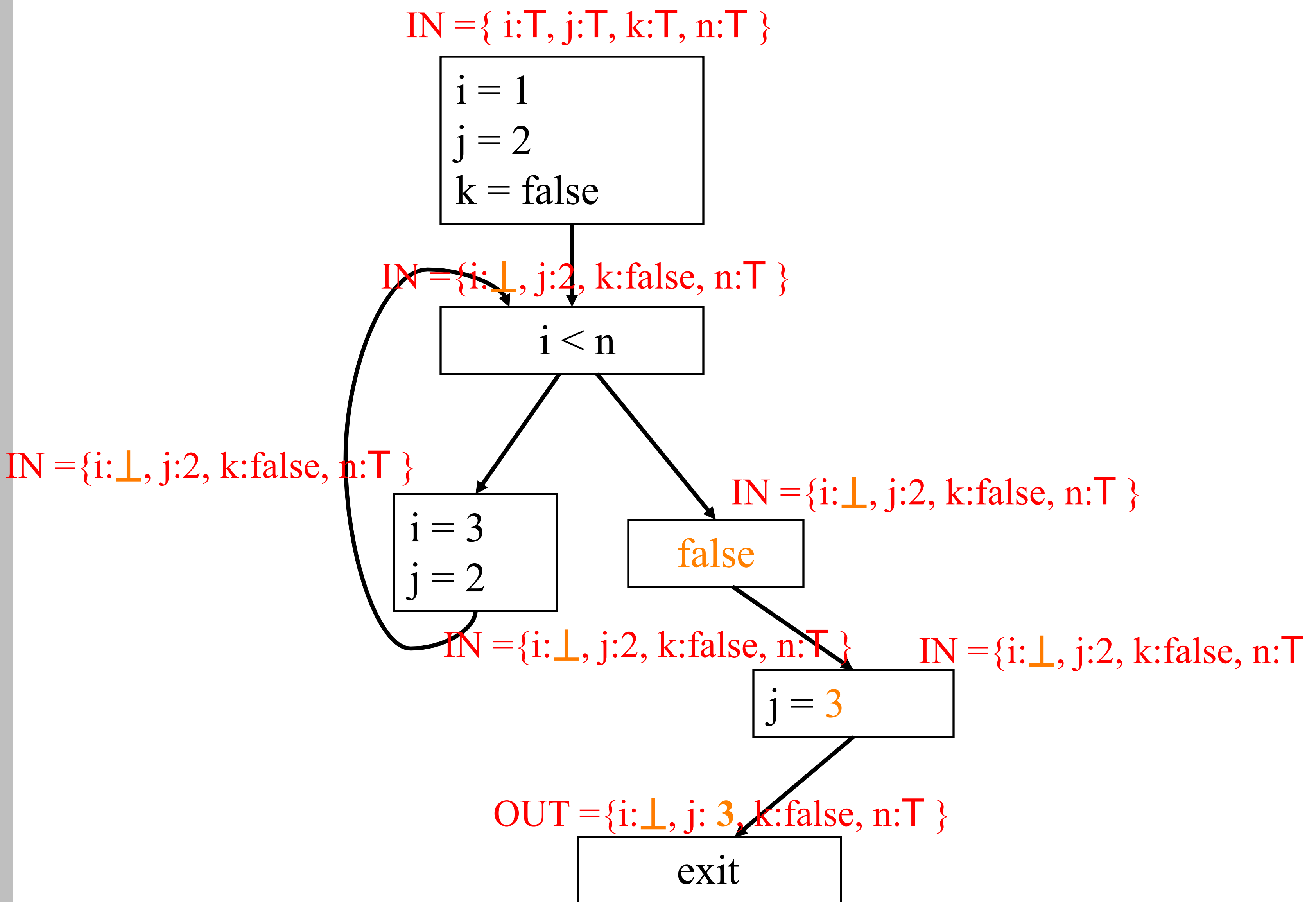
Example


IN ={ i:T, j:T, k:T, n:T }
i = 1
j = 2
k = false
IN ={i:⊥, j:2, k:false, n:T }
i < n
IN ={i:⊥, j:2, k:false, n:T }
i = 3
j = 2
IN ={i:⊥, j:2, k:false, n:T }
k
IN ={i:⊥, j:2, k:false, n:T }
print j
IN ={i:⊥, j:2, k:false, n:T }
j = j + 1
OUT ={i:⊥, j:⊥, k:false, n:T }
exit

Example


IN ={ i:T, j:T, k:T, n:T }
i = 1
j = 2
k = false
IN ={i:⊥, j:2, k:false, n:T }
i < n
IN ={i:⊥, j:2, k:false, n:T }
i = 3
j = 2
IN ={i:⊥, j:2, k:false, n:T }
false
IN ={i:⊥, j:2, k:false, n:T }
IN ={i:⊥, j:2, k:false, n:T }
print 2
j = 2 + 1
OUT ={i:⊥, j:⊥, k:false, n:T }
exit

Example


IN ={ i:T, j:T, k:T, n:T }
i = 1
j = 2
k = false
IN ={i:⊥, j:2, k:false, n:T }
i < n
IN ={i:⊥, j:2, k:false, n:T }
i = 3
j = 2
IN ={i:⊥, j:2, k:false, n:T }
false
IN ={i:⊥, j:2, k:false, n:T }
IN ={i:⊥, j:2, k:false, n:T }
j = 3
OUT ={i:⊥, j: 3, k:false, n:T }
exit

Summary

- Overview of Control-Flow Analysis
- Algebraic Simplification
- Copy Propagation
- Constant Propagation