# Principles of Program Analysis:

# A Sampler of Approaches

Transparencies based on Chapter 1 of the book: Flemming Nielson, Hanne Riis Nielson and Chris Hankin: Principles of Program Analysis. Springer Verlag 2005. 

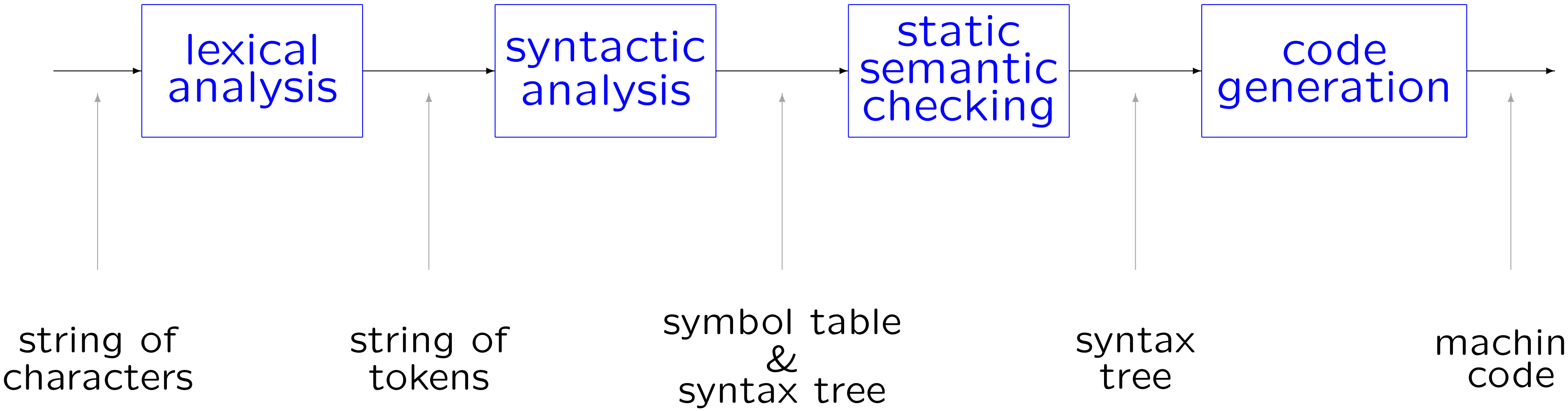
# Compiler Optimisation

The classical use of program analysis is to facilitate the construction of compilers generating "optimal" code.

We begin by outlining the structure of optimising compilers.

We then prepare the setting for a worked example where we "optimise" a naive implementation of Algol-like arrays in a C-like language by performing a series of analyses and transformations.

# The structure of a simple compiler

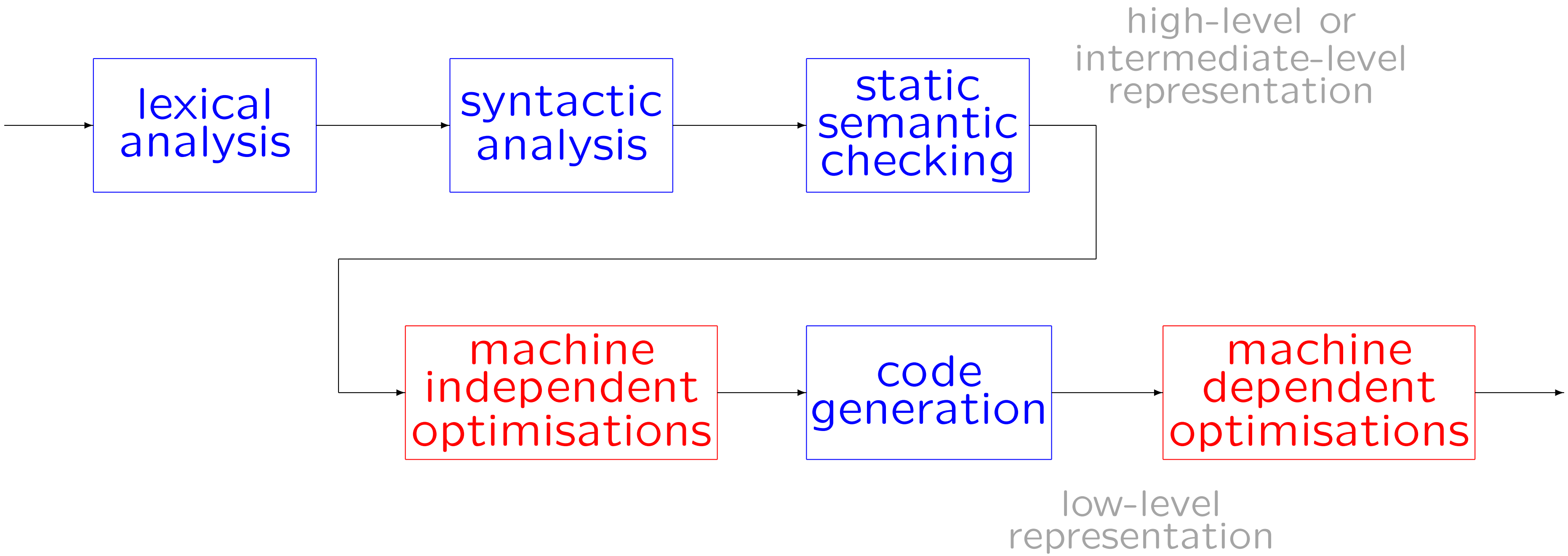



## Characteristics of a simple compiler:

- many phases – one or more passes
- the compiler is fast – but the code is not very efficient

# The structure of an optimising compiler

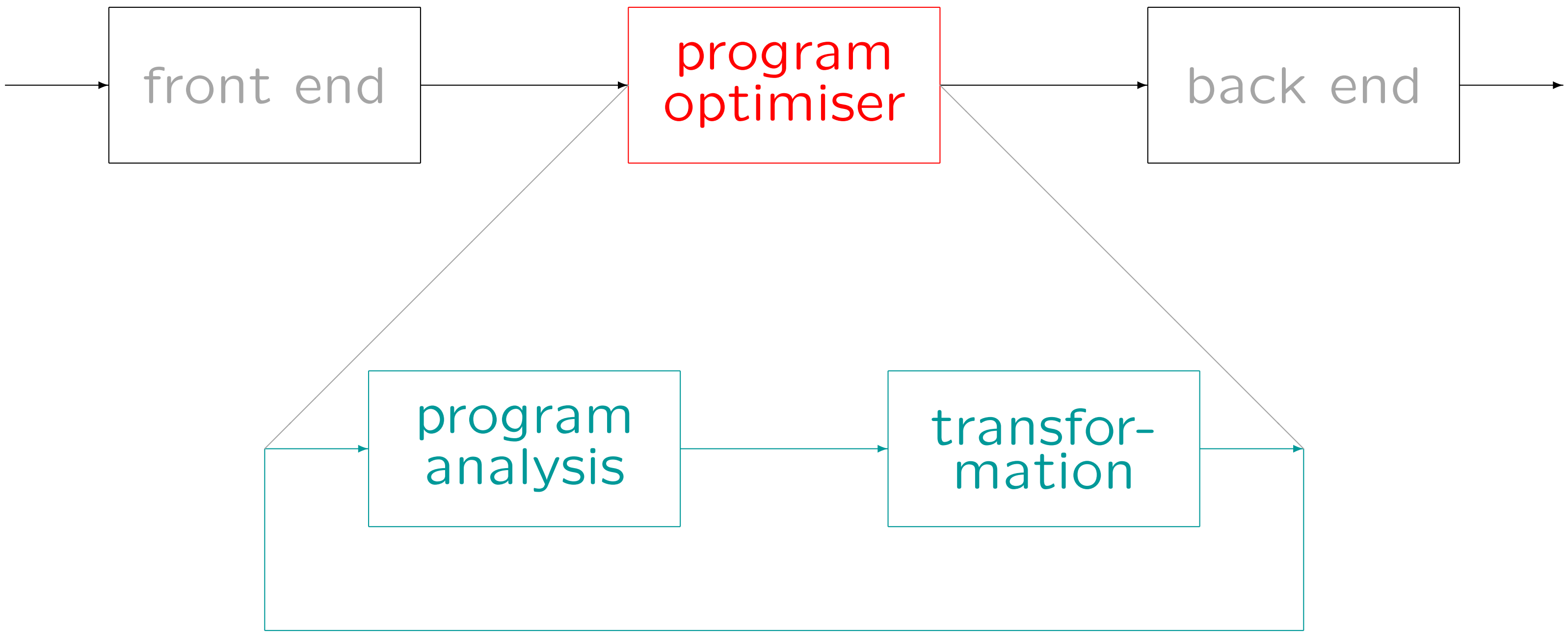



Characteristics of the optimising compiler:

- high-level optimisations: easy to adapt to new architectures
- low-level optimisations: less likely to port to new architectures

# The structure of the optimisation phase

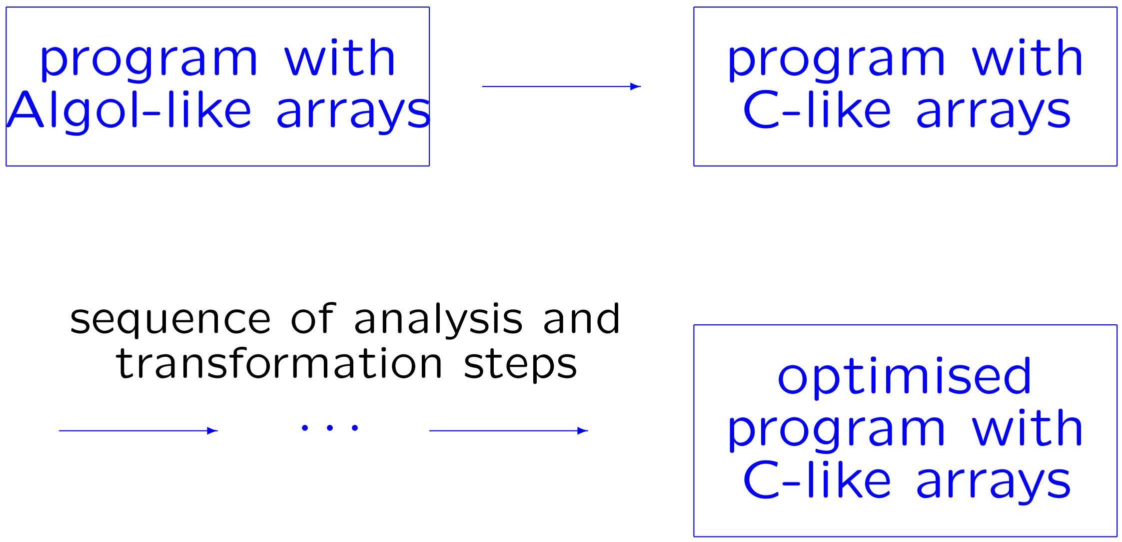



Avoid redundant computations: reuse available results, move loop invariant computations out of loops, ...

Avoid superfluous computations: results known not to be needed, results known already at compile time, ...

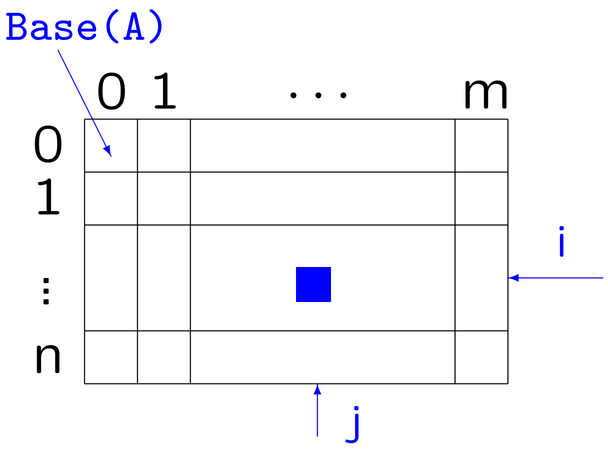
  

# Example: Array Optimisation



# Array representation: Algol vs. C

`A: array [0:n, 0:m] of integer`




Accessing the (i,j)'th element of `A`:

in Algol:

`A[i,j]`

in C:

`Cont(Base(A) + i * (m+1) + j)`

# An example program and its naive realisation

Algol-like arrays:

```
i := 0;
while i <= n do
   j := 0;
   while j <= m do
      A[i,j] := B[i,j] + C[i,j];
      j := j+1
   od;
   i := i+1
od
```

C-like arrays:

```
i := 0;
while i <= n do
   j := 0;
   while j <= m do
      temp := Base(A) + i * (m+1) + j;
      Cont(temp) := Cont(Base(B) + i * (m+1) + j)
                  + Cont(Base(C) + i * (m+1) + j);
      j := j+1
   od;
   i := i+1
od
```

# Available Expressions analysis and Common Subexpression Elimination

```
i := 0;
while i <= n do
   j := 0;
   while j <= m do
      temp := Base(A) + i*(m+1) + j;
      Cont(temp) := Cont(Base(B) + i*(m+1) + j)
                  + Cont(Base(C) + i*(m+1) + j);
      j := j+1
   od;
   i := i+1
od
```

first computation

re-computations

```
t1 := i * (m+1) + j;
temp := Base(A) + t1;
Cont(temp) := Cont(Base(B)+t1)
            + Cont(Base(C)+t1);
```

# Detection of Loop Invariants and Invariant Code Motion

```
i := 0;
while i <= n do
   j := 0;
   while j <= m do
      t1 := i * (m+1) + j;
      temp := Base(A) + t1;
      Cont(temp) := Cont(Base(B) + t1)
                    + Cont(Base(C) + t1);
      j := j+1
   od;
   i := i+1
od
```

loop invariant: `i * (m+1)`

```
t2 := i * (m+1);
while j <= m do
   t1 := t2 + j;
   temp := ...
   Cont(temp) := ...
   j := ...
od
```

# Detection of Induction Variables and Reduction of Strength

```
i := 0;
while i <= n do
   j := 0;
   t2 := i * (m+1);
   while j <= m do
      t1 := t2 + j;
      temp := Base(A) + t1;
      Cont(temp) := Cont(Base(B) + t1)
                    + Cont(Base(C) + t1);
      j := j+1
   od;
   i := i+1
od
```

induction variable

```
i := 0;
t3 := 0;
while i <= n do
   j := 0;
   t2 := t3;
   while j <= m do ...  od
   i := i + 1;
   t3 := t3 + (m+1)
od
```

# Equivalent Expressions analysis and Copy Propagation

```
i := 0;
t3 := 0;
while i <= n do
   j := 0;
   t2 := t3;
   while j <= m do
      t1 := t2 + j;
      temp := Base(A) + t1;
      Cont(temp) := Cont(Base(B) + t1)
                  + Cont(Base(C) + t1);
      j := j+1
   od;
   i := i+1;
   t3 := t3 + (m+1)
od
```

t2 = t3

```
while j <= m do
   t1 := t3 + j;
   temp := ...;
   Cont(temp) := ...;
   j := ...
od
```

# Live Variables analysis and Dead Code Elimination

```
i := 0;
t3 := 0;
while i <= n do
   j := 0;
   t2 := t3;
   while j <= m do
      t1 := t3 + j;
      temp := Base(A) + t1;
      Cont(temp) := Cont(Base(B) + t1)
                  + Cont(Base(C) + t1);
      j := j+1
   od;
   i := i+1;
   t3 := t3 + (m+1)
od
```

dead variable

```
i := 0;
t3 := 0;
while i <= n do
   j := 0;
   while j <= m do
      t1 := t3 + j;
      temp := Base(A) + t1;
      Cont(temp) := Cont(Base(B) + t1)
                  + Cont(Base(C) + t1);
      j := j+1
   od;
   i := i+1;
   t3 := t3 + (m+1)
od
```

## Summary of analyses and transformations

| Analysis | Transformation |
|---|---|
| Available expressions analysis | Common subexpression elimination |
| Detection of loop invariants | Invariant code motion |
| Detection of induction variables | Strength reduction |
| Equivalent expression analysis | Copy propagation |
| Live variables analysis | Dead code elimination |

# The Essence of Program Analysis

Program analysis offers techniques for predicting
statically at compile-time
safe & efficient approximations
to the set of configurations or behaviours arising
dynamically at run-time

we cannot expect exact answers!

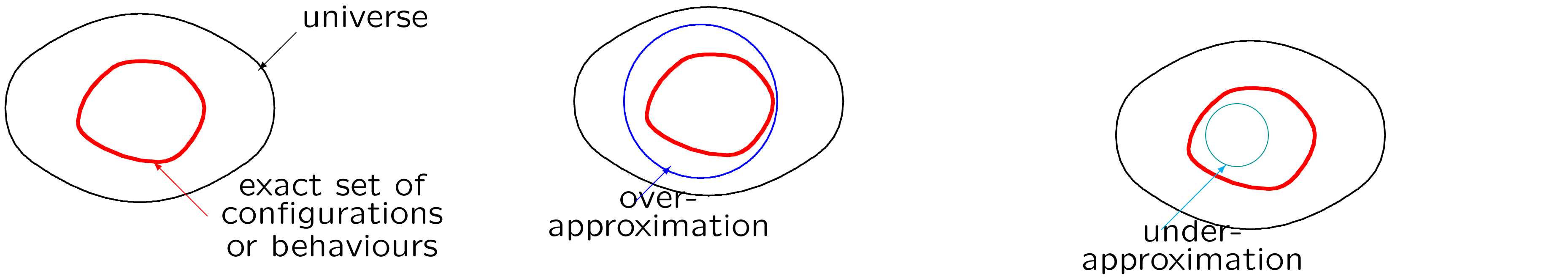
Safe: faithful to the semantics

Efficient: implementation with

– good time performance and
– low space consumption

# The Nature of Approximation

The exact world | Over-approximation | Under-approximation




Slogans: Err on the safe side!
Trade precision for efficiency!

# Approaches to Program Analysis

A family of techniques . . .

- data flow analysis
- constraint based analysis
- abstract interpretation
- type and effect systems
- . . .
- flow logic:
  a unifying framework

. . . that differ in their focus:

- algorithmic methods
- semantic foundations
- language paradigms
  - imperative/procedural
  - object oriented
  - logical
  - functional
  - concurrent/distributive
  - mobile
  - . . .

# Data Flow Analysis

- **Technique:** Data Flow Analysis
- **Example:** Reaching Definitions analysis
  - idea
  - constructing an equation system
  - solving the equations
  - theoretical underpinnings

# Example program

Program with labels for elementary blocks:

$[y := x]^1$;

$[z := 1]^2$;

**while** $[y > 0]^3$ **do**

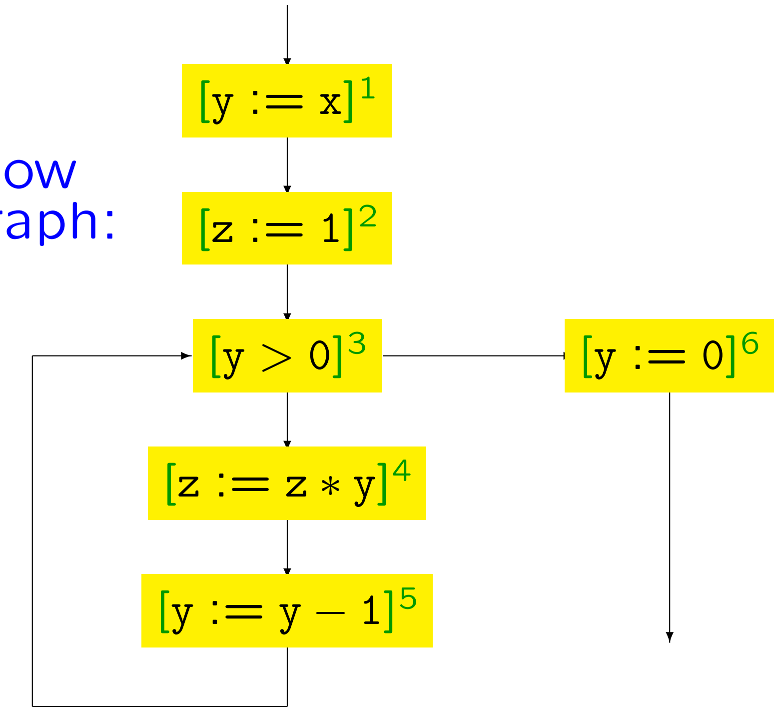
  $[z := z * y]^4$;

  $[y := y - 1]^5$

**od**;

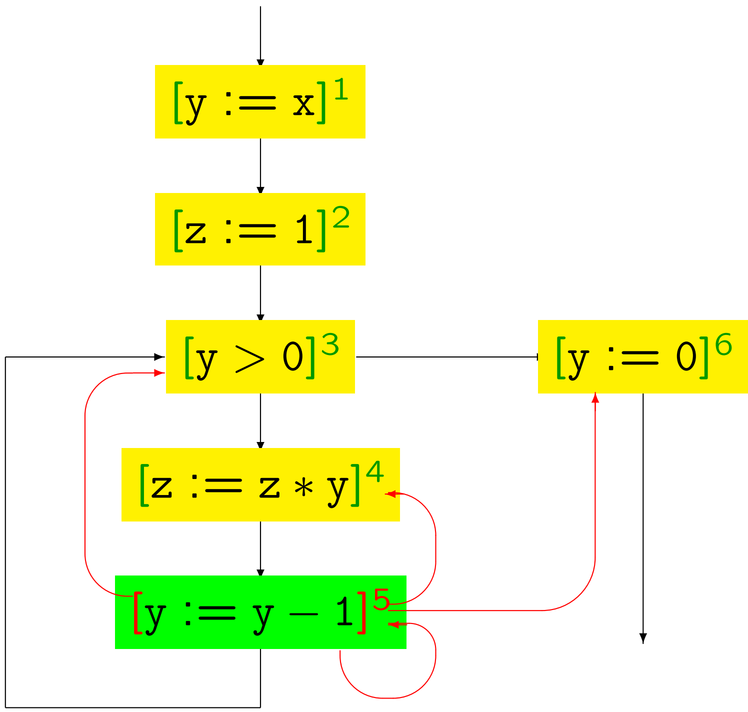
$[y := 0]^6$

Flow graph:



# Example: Reaching Definitions

The assignment $[x := a]^\ell$ reaches $\ell'$ if there is an execution where $x$ was last assigned at $\ell$



# Reading Definitions analysis (1)

$\{(x,?),(y,?),(z,?)\}$

$[y := x]^1;$

$\{(x,?),(y,1),(z,?)\}$

$[z := 1]^2;$

$\{(x,?),(y,1),(z,2)\}$

$\texttt{while}\ [y > 0]^3\ \texttt{do}$

$\{(x,?),(y,1),(z,2)\}$

$\quad [z := z * y]^4;$

$\quad [y := y - 1]^5$

$\texttt{od};$

$\{(x,?),(y,1),(z,2)\}$

$[y := 0]^6$

# Reaching Definitions analysis (2)

| Program | | Reaching definitions |
|---|---|---|
| | ← | $\{(x,?),(y,?),(z,?)\}$ |
| $[y := x]^1$; | | |
| | ← | $\{(x,?),(y,1),(z,?)\}$ |
| $[z := 1]^2$; | | |
| | ← | $\{(x,?),(y,1),(z,2)\} \cup \{(y,5),(z,4)\}$ |
| `while` $[y > 0]^3$ `do` | | |
| | ← | $\{(x,?),(y,1),(z,2)\}$ |
| $[z := z * y]^4$; | | |
| | ← | $\{(x,?),(y,1),(z,4)\}$ |
| $[y := y - 1]^5$ | | |
| | ← | $\{(x,?),(y,5),(z,4)\}$ |
| `od`; | | |
| | ← | $\{(x,?),(y,1),(z,2)\}$ |
| $[y := 0]^6$ | | |
| | ← | |

# Reaching Definitions analysis (3)

| Program | | Reaching definitions |
|---|---|---|
| | ← | $\{(x,?),(y,?),(z,?)\}$ |
| $[y := x]^1$; | | |
| | ← | $\{(x,?),(y,1),(z,?)\}$ |
| $[z := 1]^2$; | | |
| | ← | $\{(x,?),(y,1),(y,5),(z,2),(z,4)\} \cup \{(y,5),(z,4)\}$ |
| `while` $[y > 0]^3$ `do` | | |
| | ← | $\{(x,?),(y,1),(y,5),(z,2),(z,4)\}$ |
| $[z := z * y]^4$; | | |
| | ← | $\{(x,?),(y,1),(y,5),(z,4)\}$ |
| $[y := y - 1]^5$ | | |
| | ← | $\{(x,?),(y,5),(z,4)\}$ |
| `od`; | | |
| | ← | $\{(x,?),(y,1),(y,5),(z,2),(z,4)\}$ |
| $[y := 0]^6$ | | |
| | ← | |

# The best solution

$\longleftarrow \quad \{(x,?),(y,?),(z,?)\}$

$[y := x]^1;$

$\longleftarrow \quad \{(x,?),(y,1),(z,?)\}$

$[z := 1]^2;$

$\longleftarrow \quad \{(x,?),(y,1),(y,5),(z,2),(z,4)\}$

$\texttt{while}\ [y > 0]^3\ \texttt{do}$

$\longleftarrow \quad \{(x,?),(y,1),(y,5),(z,2),(z,4)\}$

$\quad [z := z * y]^4;$

$\longleftarrow \quad \{(x,?),(y,1),(y,5),(z,4)\}$

$\quad [y := y - 1]^5$

$\longleftarrow \quad \{(x,?),(y,5),(z,4)\}$

$\texttt{od};$

$\longleftarrow \quad \{(x,?),(y,1),(y,5),(z,2),(z,4)\}$

$[y := 0]^6$

$\longleftarrow \quad \{(x,?),(y,6),(z,2),(z,4)\}$

# A safe solution — but not the best

$\{(x,?),(y,?),(z,?)\}$

$[y := x]^1$;

$\{(x,?),(y,1),(z,?)\}$

$[z := 1]^2$;

$\{(x,?),(y,1),(y,5),(z,2),(z,4)\}$

**while** $[y > 0]^3$ **do**

$\{(x,?),(y,1),(y,5),(z,2),(z,4)\}$

  $[z := z * y]^4$;

$\{(x,?),(y,1),(y,5),\boxed{(z,2)},(z,4)\}$

  $[y := y - 1]^5$

$\{(x,?),\boxed{(y,1)},(y,5),\boxed{(z,2)},(z,4)\}$

**od**;

$\{(x,?),(y,1),(y,5),(z,2),(z,4)\}$

$[y := 0]^6$

$\{(x,?),(y,6),(z,2),(z,4)\}$

# An unsafe solution

$\{(x,?),(y,?),(z,?)\}$

$[y := x]^1$;

$\{(x,?),(y,1),(z,?)\}$

$[z := 1]^2$;

$\{(x,?),(y,1),(z,2),(y,5),(z,4)\}$

**while** $[y > 0]^3$ **do**

$\{(x,?),\boxed{(y,1)},(y,5),\boxed{(z,2)},(z,4)\}$

$[z := z * y]^4$;

$\{(x,?),\boxed{(y,1)},(y,5),(z,4)\}$

$[y := y - 1]^5$

$\{(x,?),(y,5),(z,4)\}$

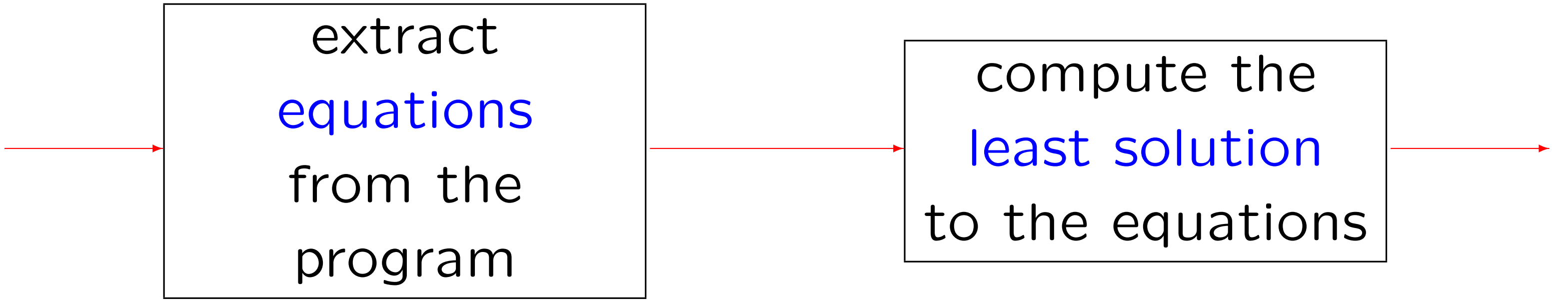
**od**;

$\{(x,?),\boxed{(y,1)},(y,5),\boxed{(z,2)},(z,4)\}$

$[y := 0]^6$

$\{(x,?),(y,6),(z,2),(z,4)\}$

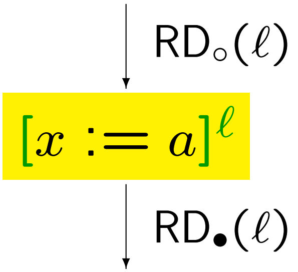
# How to automate the analysis



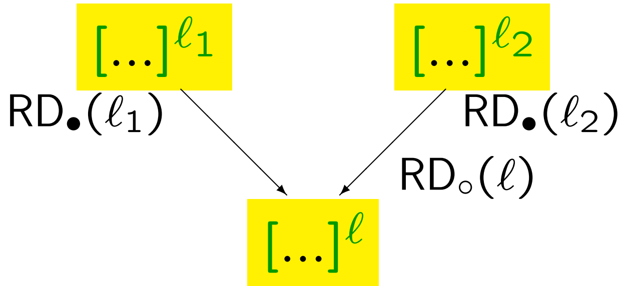

# Analysis information:

- $\mathsf{RD}_\circ(\ell)$: information available at the entry of block $\ell$
- $\mathsf{RD}_\bullet(\ell)$: information available at the exit of block $\ell$

# Two kinds of equations




$$\mathsf{RD}_\circ(\ell) \setminus \{(x,\ell') \mid \ell' \in \mathbf{Lab}\} \cup \{(x,\ell)\} = \mathsf{RD}_\bullet(\ell)$$




$$\mathsf{RD}_\bullet(\ell_1) \cup \mathsf{RD}_\bullet(\ell_2) = \mathsf{RD}_\circ(\ell)$$

## Flow through assignments and tests

| | |
|---|---|
| $[\mathtt{y} := \mathtt{x}]^1;$ | $\mathsf{RD}_\bullet(1) = \mathsf{RD}_\circ(1) \setminus \{(\mathtt{y},\ell) \mid \ell \in \mathbf{Lab}\} \cup \{(\mathtt{y},1)\}$ |
| $[\mathtt{z} := 1]^2;$ | $\mathsf{RD}_\bullet(2) = \mathsf{RD}_\circ(2) \setminus \{(\mathtt{z},\ell) \mid \ell \in \mathbf{Lab}\} \cup \{(\mathtt{z},2)\}$ |
| `while` $[\mathtt{y} > 0]^3$ `do` | $\mathsf{RD}_\bullet(3) = \mathsf{RD}_\circ(3)$ |
| $[\mathtt{z} := \mathtt{z} * \mathtt{y}]^4;$ | $\mathsf{RD}_\bullet(4) = \mathsf{RD}_\circ(4) \setminus \{(\mathtt{z},\ell) \mid \ell \in \mathbf{Lab}\} \cup \{(\mathtt{z},4)\}$ |
| $[\mathtt{y} := \mathtt{y} - 1]^5$ | $\mathsf{RD}_\bullet(5) = \mathsf{RD}_\circ(5) \setminus \{(\mathtt{y},\ell) \mid \ell \in \mathbf{Lab}\} \cup \{(\mathtt{y},5)\}$ |
| `od`; | |
| $[\mathtt{y} := 0]^6$ | $\mathsf{RD}_\bullet(6) = \mathsf{RD}_\circ(6) \setminus \{(\mathtt{y},\ell) \mid \ell \in \mathbf{Lab}\} \cup \{(\mathtt{y},6)\}$ |

**Lab** = {1,2,3,4,5,6}

6 equations in $\mathsf{RD}_\circ(1), \cdots, \mathsf{RD}_\bullet(6)$

# Flow along the control

$RD_\circ(1) = \{(x,?),(y,?),(z,?)\}$

$[y := x]^1$;

$RD_\circ(2) = RD_\bullet(1)$

$[z := 1]^2$;

$RD_\circ(3) = RD_\bullet(2) \cup RD_\bullet(5)$

**while** $[y > 0]^3$ **do**

$RD_\circ(4) = RD_\bullet(3)$

$[z := z * y]^4$;

$RD_\circ(5) = RD_\bullet(4)$

$[y := y - 1]^5$

**od**;

$RD_\circ(6) = RD_\bullet(3)$

$[y := 0]^6$

**Lab** $= \{1,2,3,4,5,6\}$

6 equations in $RD_\circ(1), \cdots, RD_\bullet(6)$

# Summary of equation system

$$\mathsf{RD}_\bullet(1) = \mathsf{RD}_\circ(1) \setminus \{(y,\ell) \mid \ell \in \mathbf{Lab}\} \cup \{(y,1)\}$$
$$\mathsf{RD}_\bullet(2) = \mathsf{RD}_\circ(2) \setminus \{(z,\ell) \mid \ell \in \mathbf{Lab}\} \cup \{(z,2)\}$$
$$\mathsf{RD}_\bullet(3) = \mathsf{RD}_\circ(3)$$
$$\mathsf{RD}_\bullet(4) = \mathsf{RD}_\circ(4) \setminus \{(z,\ell) \mid \ell \in \mathbf{Lab}\} \cup \{(z,4)\}$$
$$\mathsf{RD}_\bullet(5) = \mathsf{RD}_\circ(5) \setminus \{(y,\ell) \mid \ell \in \mathbf{Lab}\} \cup \{(y,5)\}$$
$$\mathsf{RD}_\bullet(6) = \mathsf{RD}_\circ(6) \setminus \{(y,\ell) \mid \ell \in \mathbf{Lab}\} \cup \{(y,6)\}$$

$$\mathsf{RD}_\circ(1) = \{(x,?),(y,?),(z,?)\}$$
$$\mathsf{RD}_\circ(2) = \mathsf{RD}_\bullet(1)$$
$$\mathsf{RD}_\circ(3) = \mathsf{RD}_\bullet(2) \cup \mathsf{RD}_\bullet(5)$$
$$\mathsf{RD}_\circ(4) = \mathsf{RD}_\bullet(3)$$
$$\mathsf{RD}_\circ(5) = \mathsf{RD}_\bullet(4)$$
$$\mathsf{RD}_\circ(6) = \mathsf{RD}_\bullet(3)$$

- 12 sets: $\mathsf{RD}_\circ(1), \cdots, \mathsf{RD}_\bullet(6)$
  all being subsets of $\mathbf{Var} \times \mathbf{Lab}$
- 12 equations:
  $\mathsf{RD}_j = \mathsf{F}_j(\mathsf{RD}_\circ(1), \cdots, \mathsf{RD}_\bullet(6))$
- one function:
  $\mathsf{F} : \mathcal{P}(\mathbf{Var} \times \mathbf{Lab})^{12} \to \mathcal{P}(\mathbf{Var} \times \mathbf{Lab})^{12}$
- we want the least fixed point of F — this is the best solution to the equation system

# How to solve the equations

## A simple iterative algorithm

- Initialisation
  $\mathsf{RD}_1 := \emptyset; \cdots; \mathsf{RD}_{12} := \emptyset;$

- Iteration
  **while** $\mathsf{RD}_j \neq \mathsf{F}_j(\mathsf{RD}_1, \cdots, \mathsf{RD}_{12})$ for some $j$
  **do**
  $\quad \mathsf{RD}_j := \mathsf{F}_j(\mathsf{RD}_1, \cdots, \mathsf{RD}_{12})$

The algorithm terminates and computes the least fixed point of F.

# The example equations

| $RD_\circ$ | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 0 | $\emptyset$ | $\emptyset$ | $\emptyset$ | $\emptyset$ | $\emptyset$ | $\emptyset$ |
| 1 | $x_?, y_?, z_?$ | $\emptyset$ | $\emptyset$ | $\emptyset$ | $\emptyset$ | $\emptyset$ |
| 2 | $x_?, y_?, z_?$ | $\emptyset$ | $\emptyset$ | $\emptyset$ | $\emptyset$ | $\emptyset$ |
| 3 | $x_?, y_?, z_?$ | $x_?, y_1, z_?$ | $\emptyset$ | $\emptyset$ | $\emptyset$ | $\emptyset$ |
| 4 | $x_?, y_?, z_?$ | $x_?, y_1, z_?$ | $\emptyset$ | $\emptyset$ | $\emptyset$ | $\emptyset$ |
| 5 | $x_?, y_?, z_?$ | $x_?, y_1, z_?$ | $x_?, y_1, z_2$ | $\emptyset$ | $\emptyset$ | $\emptyset$ |
| 6 | $x_?, y_?, z_?$ | $x_?, y_1, z_?$ | $x_?, y_1, z_2$ | $\emptyset$ | $\emptyset$ | $\emptyset$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |

| $RD_\bullet$ | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 0 | $\emptyset$ | $\emptyset$ | $\emptyset$ | $\emptyset$ | $\emptyset$ | $\emptyset$ |
| 1 | $\emptyset$ | $\emptyset$ | $\emptyset$ | $\emptyset$ | $\emptyset$ | $\emptyset$ |
| 2 | $x_?, y_1, z_?$ | $\emptyset$ | $\emptyset$ | $\emptyset$ | $\emptyset$ | $\emptyset$ |
| 3 | $x_?, y_1, z_?$ | $\emptyset$ | $\emptyset$ | $\emptyset$ | $\emptyset$ | $\emptyset$ |
| 4 | $x_?, y_1, z_?$ | $x_?, y_1, z_2$ | $\emptyset$ | $\emptyset$ | $\emptyset$ | $\emptyset$ |
| 5 | $x_?, y_1, z_?$ | $x_?, y_1, z_2$ | $\emptyset$ | $\emptyset$ | $\emptyset$ | $\emptyset$ |
| 6 | $x_?, y_1, z_?$ | $x_?, y_1, z_2$ | $x_?, y_1, z_2$ | $\emptyset$ | $\emptyset$ | $\emptyset$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |

## The equations:

$$RD_\bullet(1) = RD_\circ(1) \setminus \{(y, \ell) \mid \cdots\} \cup \{(y, 1)\}$$
$$RD_\bullet(2) = RD_\circ(2) \setminus \{(z, \ell) \mid \cdots\} \cup \{(z, 2)\}$$
$$RD_\bullet(3) = RD_\circ(3)$$
$$RD_\bullet(4) = RD_\circ(4) \setminus \{(z, \ell) \mid \cdots\} \cup \{(z, 4)\}$$
$$RD_\bullet(5) = RD_\circ(5) \setminus \{(y, \ell) \mid \cdots\} \cup \{(y, 5)\}$$
$$RD_\bullet(6) = RD_\circ(6) \setminus \{(y, \ell) \mid \cdots\} \cup \{(y, 6)\}$$

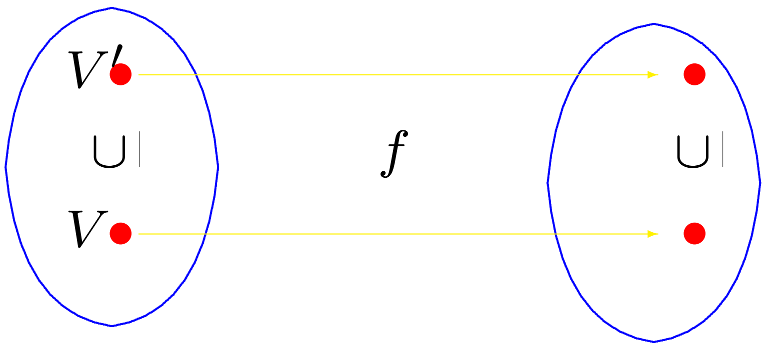
$$RD_\circ(1) = \{(x, ?), (y, ?), (z, ?)\}$$
$$RD_\circ(2) = RD_\bullet(1)$$
$$RD_\circ(3) = RD_\bullet(2) \cup RD_\bullet(5)$$
$$RD_\circ(4) = RD_\bullet(3)$$
$$RD_\circ(5) = RD_\bullet(4)$$
$$RD_\circ(6) = RD_\bullet(3)$$

# Why does it work? (1)

A function $f : \mathcal{P}(S) \rightarrow \mathcal{P}(S)$ is a monotone function if

$$V \subseteq V' \quad \Rightarrow \quad f(V) \subseteq f(V')$$

(the larger the argument – the larger the result)



## Why does it work? (2)

A set $L$ equipped with an ordering $\subseteq$ satisfies the Ascending Chain Condition if all chains

$$V_0 \subseteq V_1 \subseteq V_2 \subseteq V_3 \subseteq \cdots$$

stabilise, that is, if there exists some $n$ such that $V_n = V_{n+1} = V_{n+2} = \cdots$

If $S$ is a finite set then $\mathcal{P}(S)$ equipped with the subset ordering $\subseteq$ satisfies the Ascending Chain Condition — the chains cannot grow forever since each element is a subset of a finite set.

**Fact**

For a given program $\mathbf{Var} \times \mathbf{Lab}$ will be a finite set so $\mathcal{P}(\mathbf{Var} \times \mathbf{Lab})$ with the subset ordering satisfies the Ascending Chain Condition.

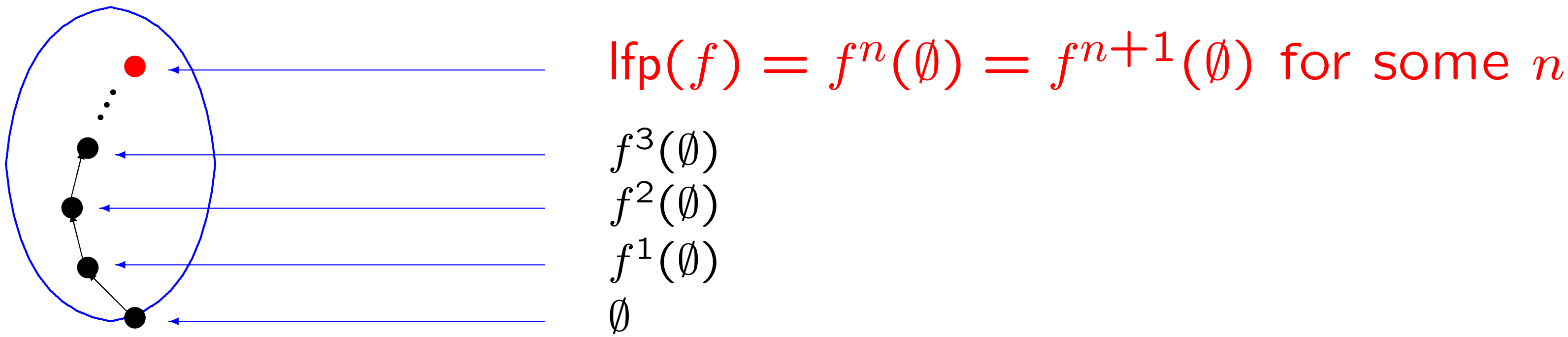
# Why does it work? (3)

Let $f : \mathcal{P}(S) \to \mathcal{P}(S)$ be a monotone function. Then

$$\emptyset \subseteq f(\emptyset) \subseteq f^2(\emptyset) \subseteq f^3(\emptyset) \subseteq \cdots$$

Assume that $S$ is a finite set; then the Ascending Chain Condition is satisfied. This means that the chain cannot be growing infinitely so there exists $n$ such that $f^n(\emptyset) = f^{n+1}(\emptyset) = \cdots$

$f^n(\emptyset)$ is the least fixed point of $f$



# Correctness of the algorithm

- Initialisation
  $\mathsf{RD}_1 := \emptyset; \cdots; \mathsf{RD}_{12} := \emptyset;$
  Invariant: $\vec{\mathsf{RD}} \subseteq \mathsf{F}^n(\vec{\emptyset})$ since $\vec{\mathsf{RD}} = \vec{\emptyset}$ is the least element

- Iteration
  **while** $\mathsf{RD}_j \neq \mathsf{F}_j(\mathsf{RD}_1, \cdots, \mathsf{RD}_{12})$ for some $j$
  **do** assume $\vec{\mathsf{RD}}$ is $\vec{\mathsf{RD}}'$ and $\vec{\mathsf{RD}}' \subseteq \mathsf{F}^n(\vec{\emptyset})$
  $\mathsf{RD}_j := \mathsf{F}_j(\mathsf{RD}_1, \cdots, \mathsf{RD}_{12})$
  then $\vec{\mathsf{RD}} \subseteq \mathsf{F}(\vec{\mathsf{RD}}') \subseteq \mathsf{F}^{n+1}(\vec{\emptyset}) = \mathsf{F}^n(\vec{\emptyset})$ when $\mathsf{lfp}(\mathsf{F}) = \mathsf{F}^n(\vec{\emptyset})$

If the algorithm terminates then it computes the least fixed point of F.

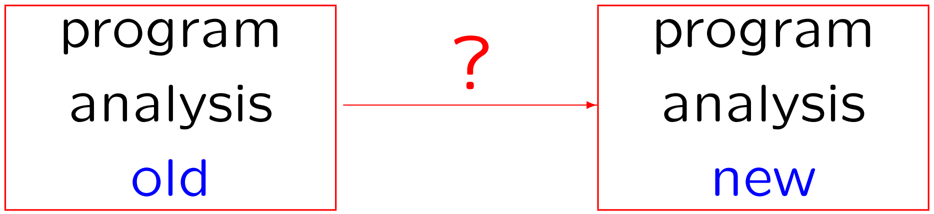
The algorithm terminates because $\mathsf{RD}_j \subset \mathsf{F}_j(\mathsf{RD}_1, \cdots, \mathsf{RD}_{12})$ is only possible finitely many times since $\mathcal{P}(\mathbf{Var} \times \mathbf{Lab})^{12}$ satisfies the Ascending Chain Condition.

# Abstract Interpretation

- **Technique:** Abstract Interpretation
- **Example:** Reaching Definitions analysis
  - idea
  - collecting semantics
  - Galois connections
  - Inducing the analysis

# Abstract Interpretation




- We have the analysis old: it has already been proved correct but it is inefficient, or maybe even uncomputable

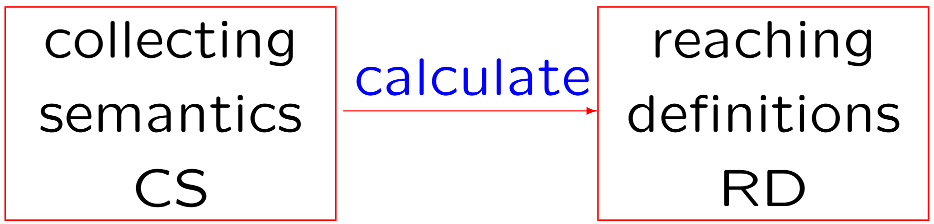
- We want the analysis new: it has to be correct as well as efficient!

- Can we develop new from old?

  abstract interpretation !

# Example: Collecting Semantics and Reaching Definitions




The collecting semantics CS

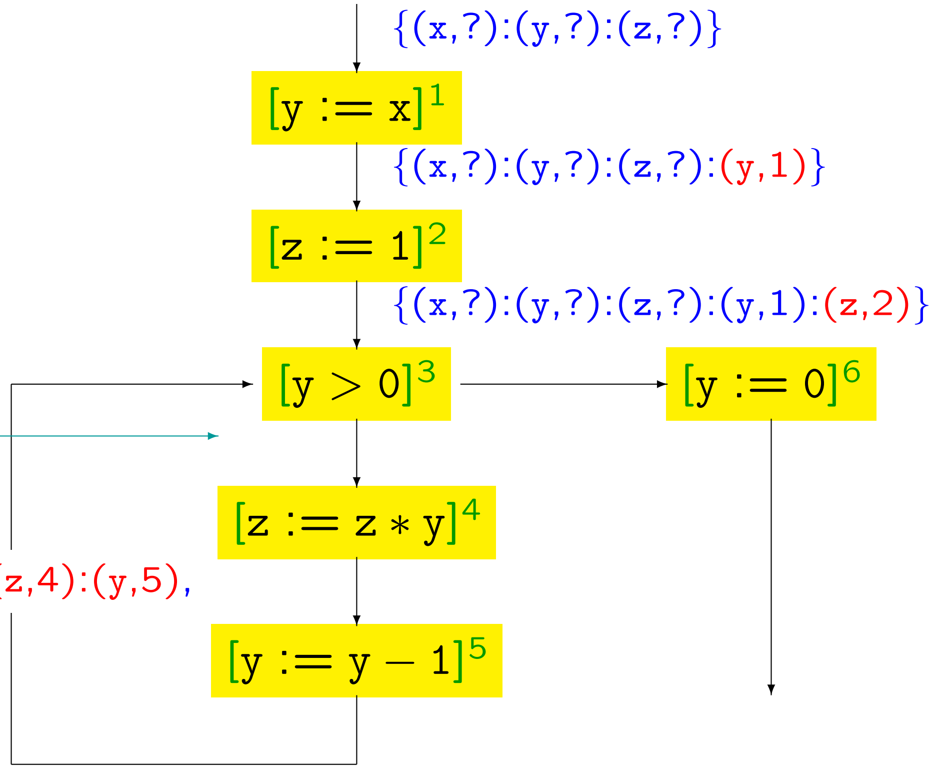
- collects the set of traces that can reach a given program point
- has an easy correctness proof
- is uncomputable

The reaching definitions analysis RD is as before

# Example: Collecting Semantics

Collect the set of traces that reach a given program point $\ell$




{(x,?):(y,?):(z,?):(y,1):(z,2),
(x,?):(y,?):(z,?):(y,1):(z,2):(z,4):(y,5),
(x,?):(y,?):(z,?):(y,1):(z,2):(z,4):(y,5):(z,4):(y,5),
$\cdots$}

# How to proceed

As before:

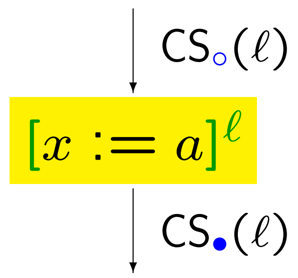
- extract a set of equations defining the possible sets of traces
- compute the least fixed point of the set of equations

And furthermore:

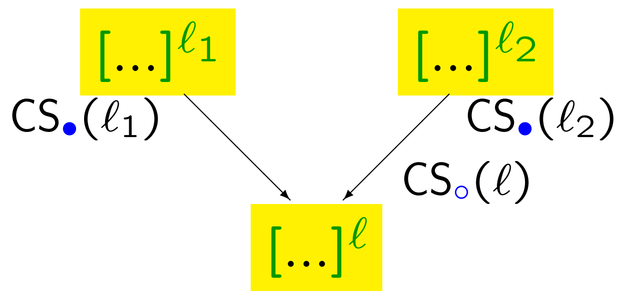
- prove the correctness: the set of traces computed by the analysis is a superset of the possible traces

# Two kinds of equations




$$\{trace : (x,\ell) \mid trace \in \mathsf{CS}_\circ(\ell)\} = \mathsf{CS}_\bullet(\ell)$$




$$\mathsf{CS}_\bullet(\ell_1) \cup \mathsf{CS}_\bullet(\ell_2) = \mathsf{CS}_\circ(\ell)$$

# Flow through assignments and tests

$[y := x]^1;$

$\mathsf{CS}_\bullet(1) = \{trace : (y, 1) \mid trace \in \mathsf{CS}_\circ(1)\}$

$[z := 1]^2;$

$\mathsf{CS}_\bullet(2) = \{trace : (z, 2) \mid trace \in \mathsf{CS}_\circ(2)\}$

`while` $[y > 0]^3$ `do`

$\mathsf{CS}_\bullet(3) = \mathsf{CS}_\circ(3)$

$[z := z * y]^4;$

$\mathsf{CS}_\bullet(4) = \{trace : (z, 4) \mid trace \in \mathsf{CS}_\circ(4)\}$

$[y := y - 1]^5$

$\mathsf{CS}_\bullet(5) = \{trace : (y, 5) \mid trace \in \mathsf{CS}_\circ(5)\}$

`od;`

$[y := 0]^6$

$\mathsf{CS}_\bullet(6) = \{trace : (y, 6) \mid trace \in \mathsf{CS}_\circ(6)\}$

6 equations in $\mathsf{CS}_\circ(1), \cdots, \mathsf{CS}_\bullet(6)$

# Flow along the control

$[y := x]^1;$

$[z := 1]^2;$

$\texttt{while}\ [y > 0]^3\ \texttt{do}$

$\quad [z := z * y]^4;$

$\quad [y := y - 1]^5$

$\texttt{od};$

$[y := 0]^6$

$$\mathsf{CS}_\circ(1) = \{(x,?) : (y,?) : (z,?)\}$$

$$\mathsf{CS}_\circ(2) = \mathsf{CS}_\bullet(1)$$

$$\mathsf{CS}_\circ(3) = \mathsf{CS}_\bullet(2) \cup \mathsf{CS}_\bullet(5)$$

$$\mathsf{CS}_\circ(4) = \mathsf{CS}_\bullet(3)$$

$$\mathsf{CS}_\circ(5) = \mathsf{CS}_\bullet(4)$$

$$\mathsf{CS}_\circ(6) = \mathsf{CS}_\bullet(3)$$

6 equations in $\mathsf{CS}_\circ(1), \cdots, \mathsf{CS}_\bullet(6)$

# Summary of Collecting Semantics

$$\mathsf{CS}_\bullet(1) = \{\mathit{trace} : (\mathrm{y}, 1) \mid \mathit{trace} \in \mathsf{CS}_\circ(1)\}$$
$$\mathsf{CS}_\bullet(2) = \{\mathit{trace} : (\mathrm{z}, 2) \mid \mathit{trace} \in \mathsf{CS}_\circ(2)\}$$
$$\mathsf{CS}_\bullet(3) = \mathsf{CS}_\circ(3)$$
$$\mathsf{CS}_\bullet(4) = \{\mathit{trace} : (\mathrm{z}, 4) \mid \mathit{trace} \in \mathsf{CS}_\circ(4)\}$$
$$\mathsf{CS}_\bullet(5) = \{\mathit{trace} : (\mathrm{y}, 5) \mid \mathit{trace} \in \mathsf{CS}_\circ(5)\}$$
$$\mathsf{CS}_\bullet(6) = \{\mathit{trace} : (\mathrm{y}, 6) \mid \mathit{trace} \in \mathsf{CS}_\circ(6)\}$$

$$\mathsf{CS}_\circ(1) = \{(\mathrm{x}, ?) : (\mathrm{y}, ?) : (\mathrm{z}, ?)\}$$
$$\mathsf{CS}_\circ(2) = \mathsf{CS}_\bullet(1)$$
$$\mathsf{CS}_\circ(3) = \mathsf{CS}_\bullet(2) \cup \mathsf{CS}_\bullet(5)$$
$$\mathsf{CS}_\circ(4) = \mathsf{CS}_\bullet(3)$$
$$\mathsf{CS}_\circ(5) = \mathsf{CS}_\bullet(4)$$
$$\mathsf{CS}_\circ(6) = \mathsf{CS}_\bullet(3)$$

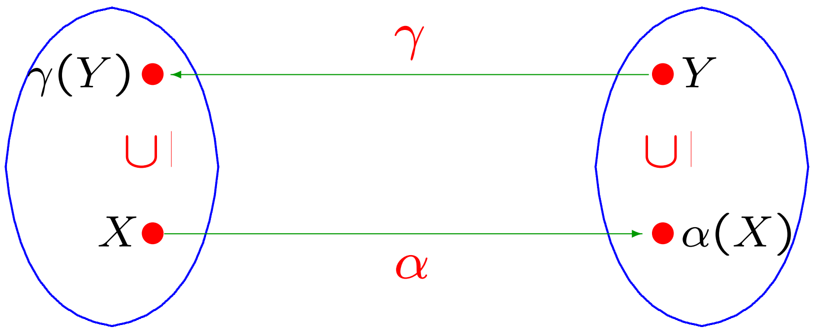
- 12 sets: $\mathsf{CS}_\circ(1), \cdots, \mathsf{CS}_\bullet(6)$
  all being subsets of $\mathbf{Trace}$
- 12 equations:
  $\mathsf{CS}_j = \mathsf{G}_j(\mathsf{CS}_\circ(1), \cdots, \mathsf{CS}_\bullet(6))$
- one function:
  $\mathsf{G} : \mathcal{P}(\mathbf{Trace})^{12} \to \mathcal{P}(\mathbf{Trace})^{12}$
- we want the least fixed point of G — but it is uncomputable!

# Example: Inducing an analysis

## Galois Connections

A Galois connection between two sets is a pair of $(\alpha, \gamma)$ of functions between the sets satisfying

$$X \subseteq \gamma(Y) \quad \Leftrightarrow \quad \alpha(X) \subseteq Y$$




$\mathcal{P}(\mathbf{Trace})$ collecting semantics

$\mathcal{P}(\mathbf{Var} \times \mathbf{Lab})$ reaching definitions

$\alpha$: abstraction function
$\gamma$: concretisation function

# Semantically Reaching Definitions

For a single trace:

*trace*: $(x,?):(y,?):(z,?):(y,1):(z,2)$

SRD(*trace*): $\{(x,?),(y,1),(z,2)\}$

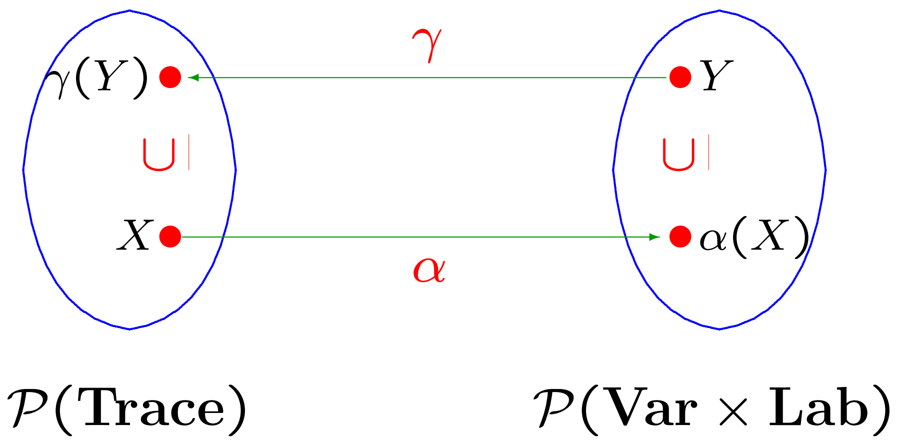
For a set of traces:

$X \in \mathcal{P}(\mathbf{Trace})$: $\{(x,?):(y,?):(z,?):(y,1):(z,2),$
$(x,?):(y,?):(z,?):(y,1):(z,2):(z,4):(y,5)\}$

SRD($X$): $\{(x,?),(y,1),(z,2),(z,4),(y,5)\}$

## Galois connection for Reaching Definitions analysis

$$\begin{aligned} \alpha(X) &= \mathsf{SRD}(X) \\ \gamma(Y) &= \{trace \mid \mathsf{SRD}(trace) \subseteq Y\} \end{aligned}$$




Galois connection:

$$X \subseteq \gamma(Y) \quad \Leftrightarrow \quad \alpha(X) \subseteq Y$$