

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. ©Flemming Nielson & Hanne Riis Nielson & Chris Hankin.

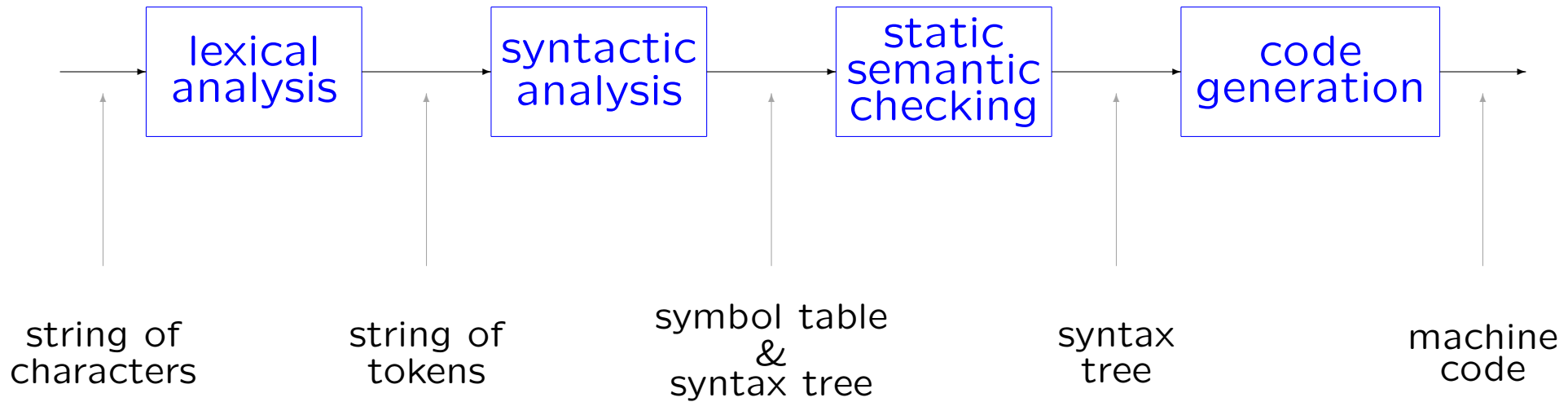
Compiler Optimisation

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.

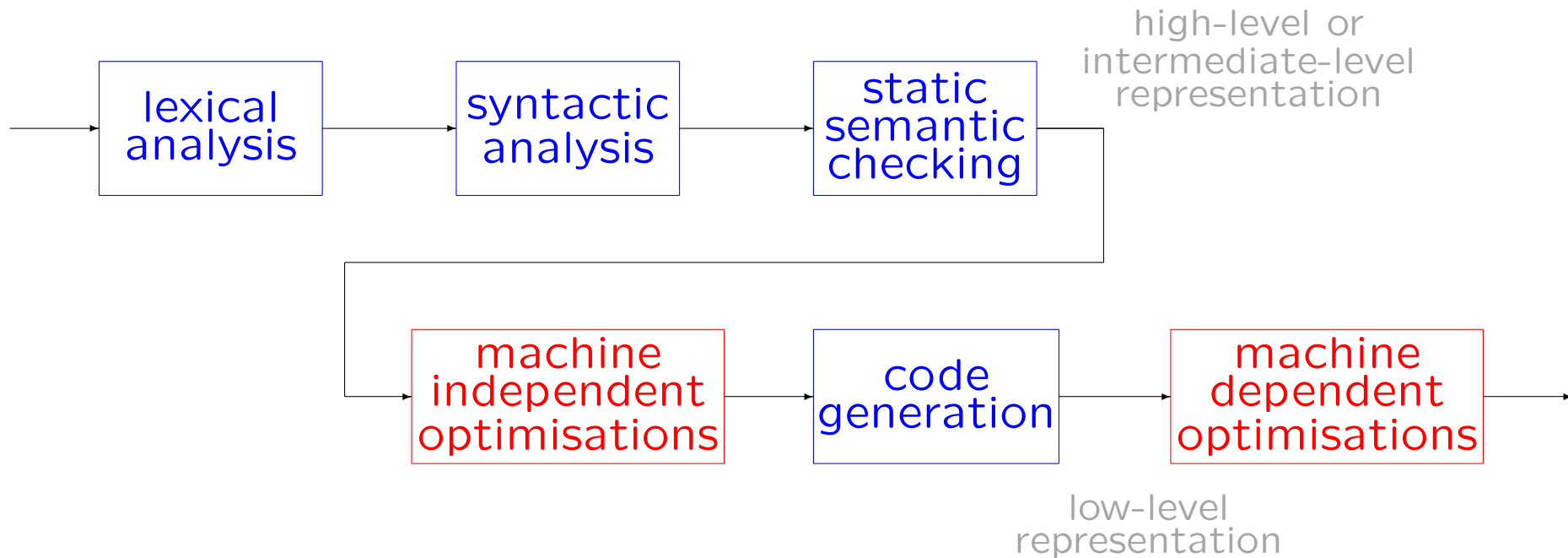
The structure of a simple compiler



Characteristics of a simple compiler:

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

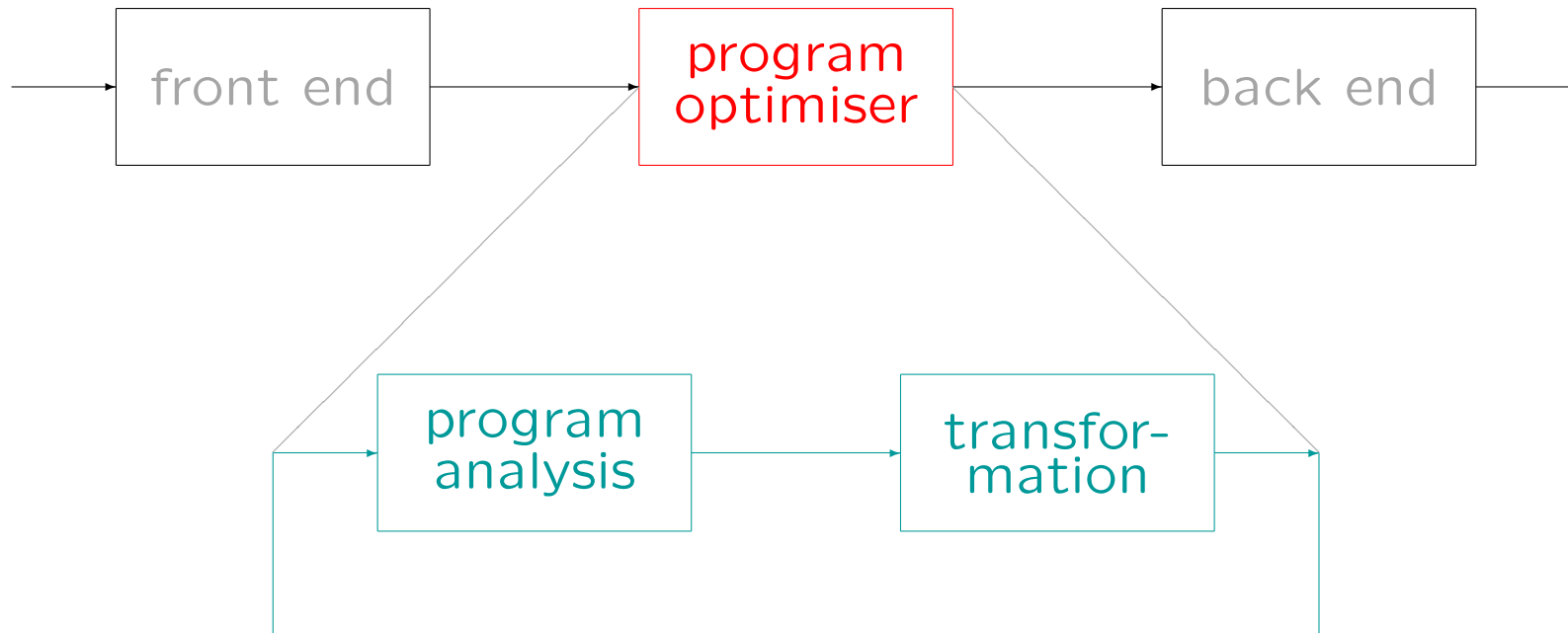
The structure of an optimising compiler



Characteristics of the optimising compiler:

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

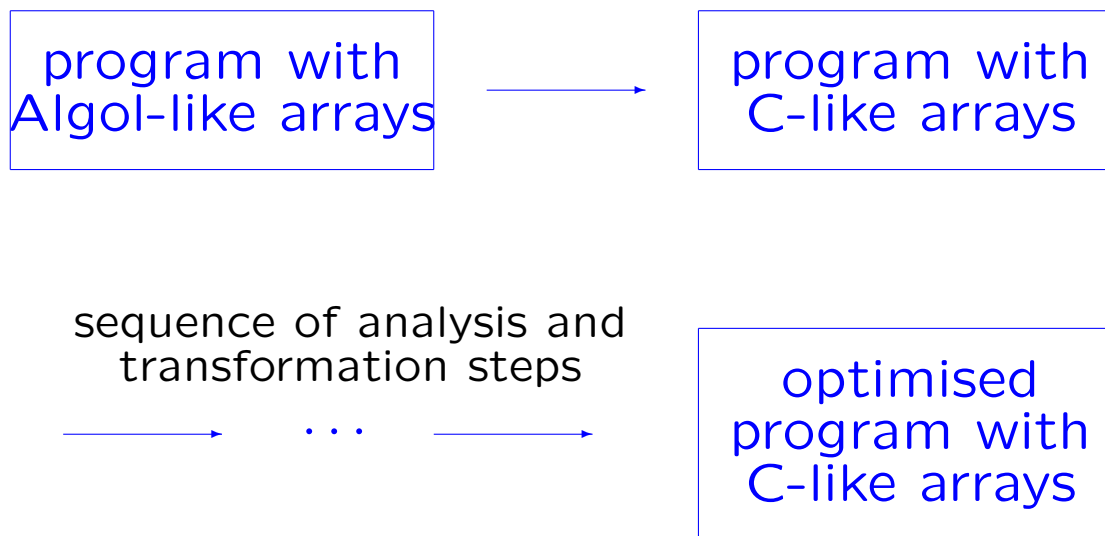
The structure of the optimisation phase



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

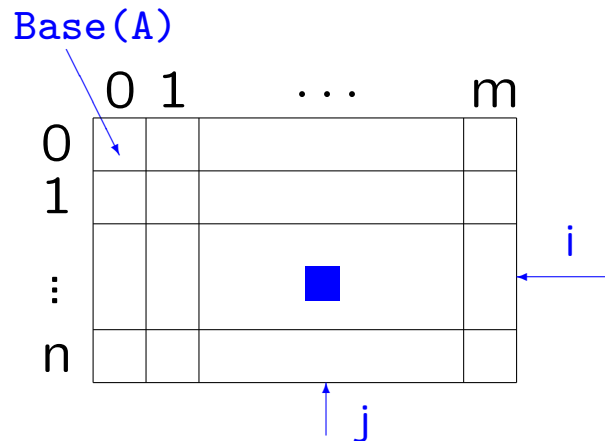
Avoid superfluous computations: results known not to be needed, results known already at compile time, ...

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:

$\text{Cont}(\text{Base}(A) + i * (m+1) + j)$

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

```
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
```

```
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!

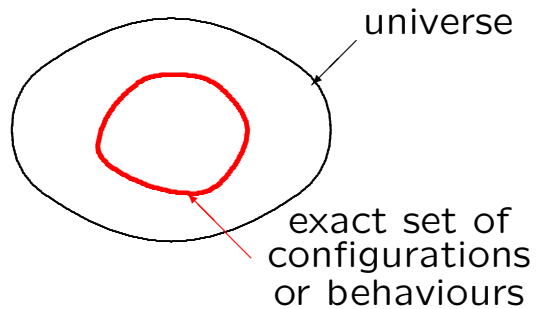
Safe: faithful to the semantics

Efficient: implementation with

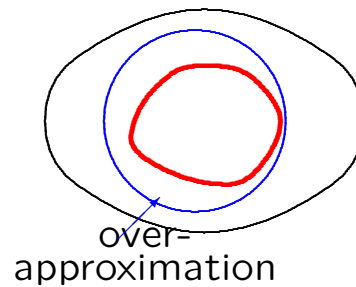
- good time performance and
- low space consumption

The Nature of Approximation

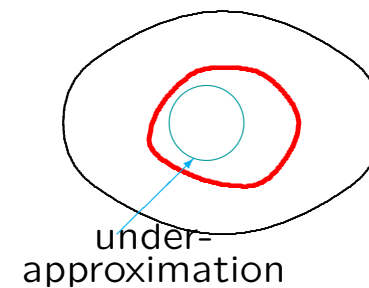
The exact world



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

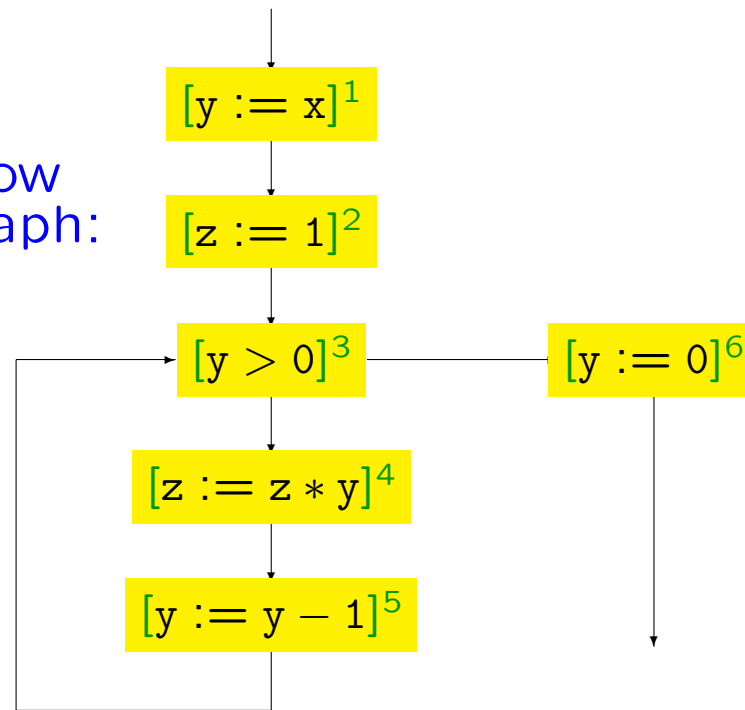
- **Technique:** Data Flow Analysis
- **Example:** Reaching Definitions analysis
 - idea
 - constructing an equation system
 - solving the equations
 - theoretical underpinnings

Example program

Program with labels for elementary blocks:

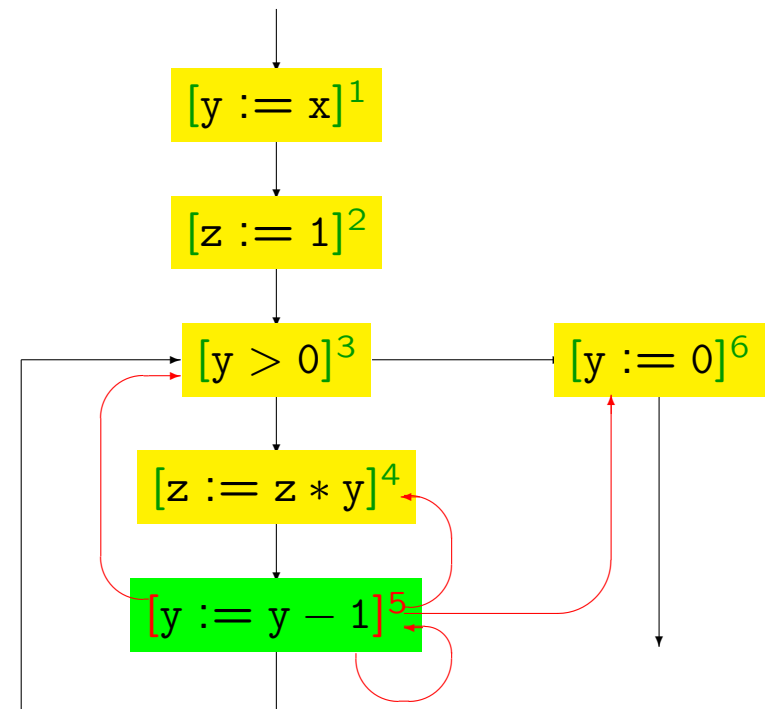
```
[y := x]1;  
[z := 1]2;  
while [y > 0]3 do  
  [z := z * y]4;  
  [y := y - 1]5  
od;  
[y := 0]6
```

Flow graph:

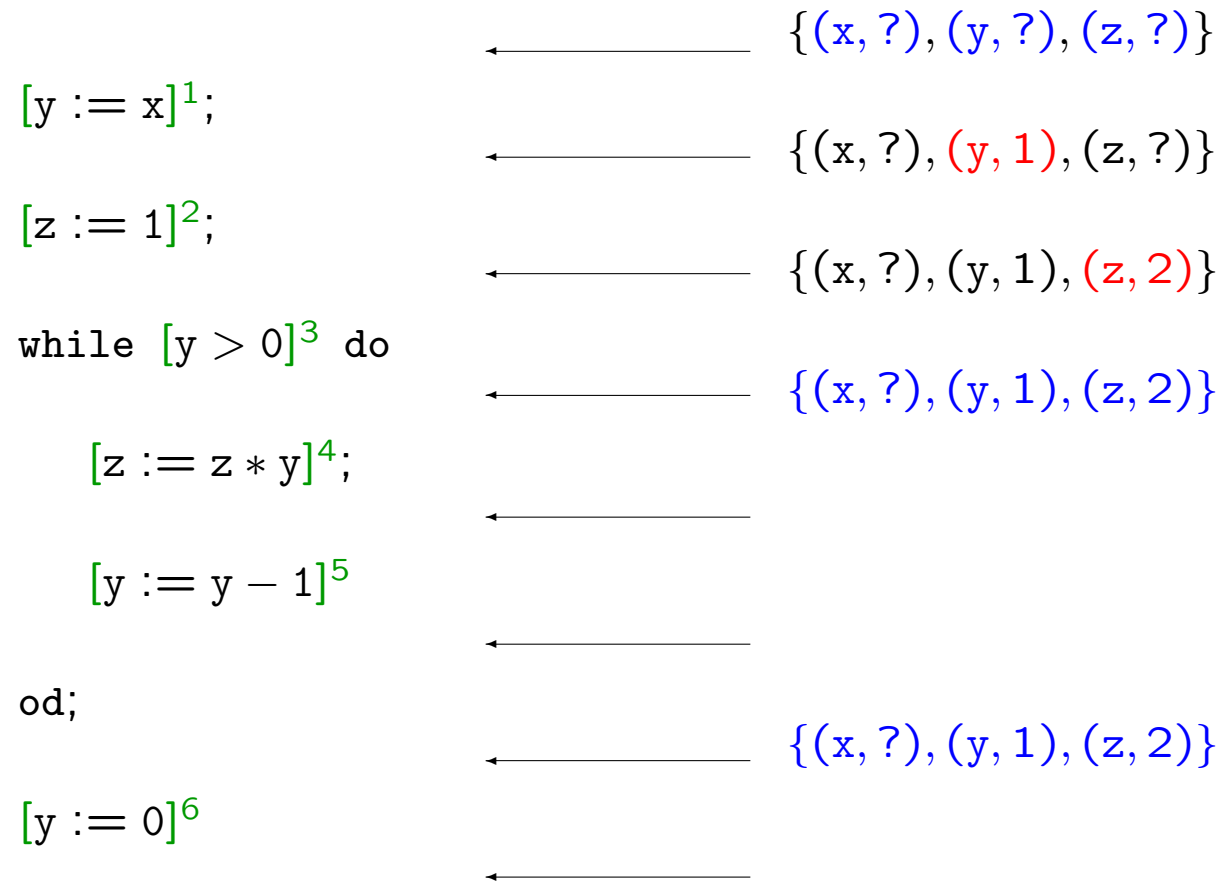


Example: Reaching Definitions

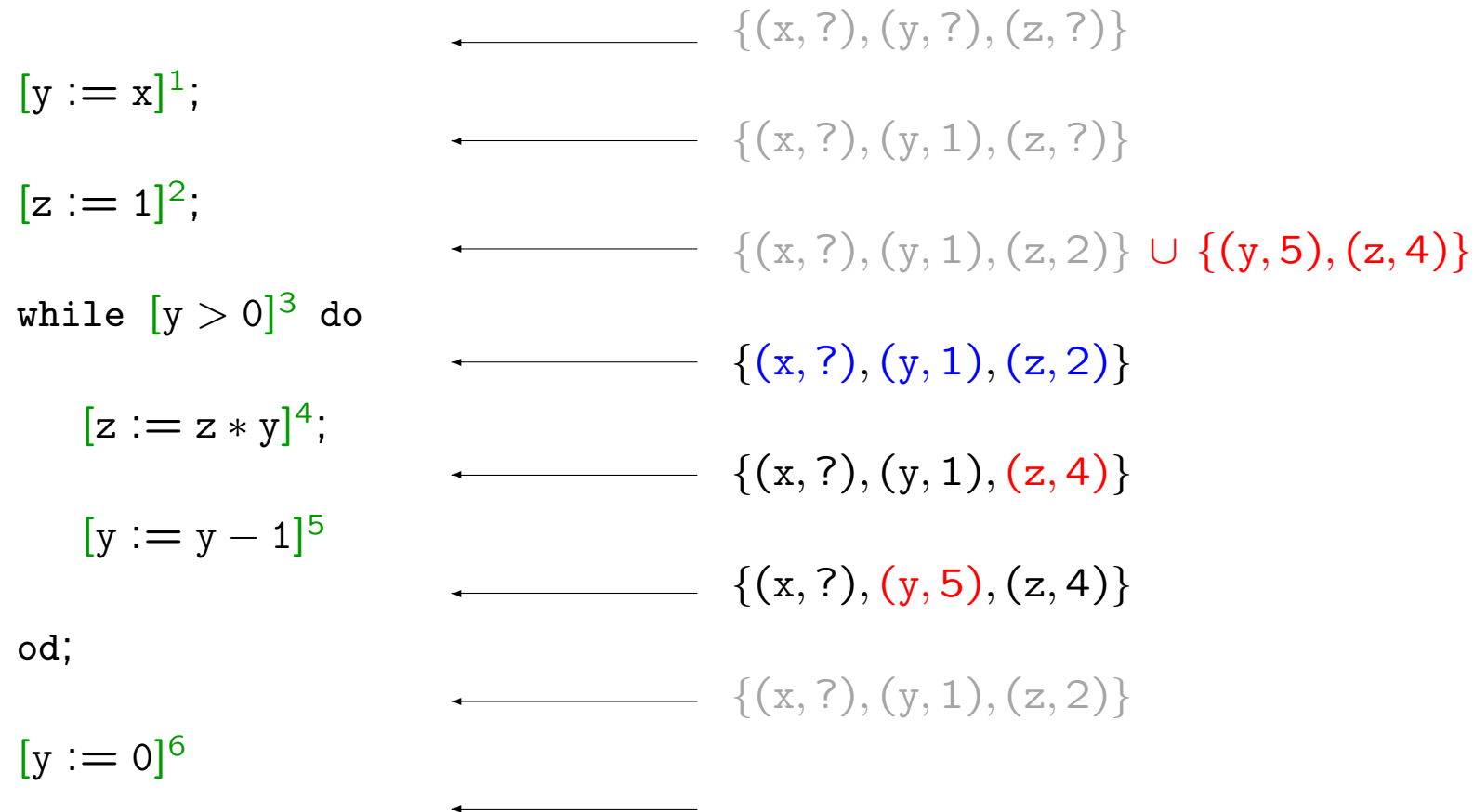
The assignment $[x := a]^{\ell}$ **reaches** ℓ' if there is an execution where x was last assigned at ℓ



Reaching Definitions analysis (1)



Reaching Definitions analysis (2)



Reaching Definitions analysis (3)

	←	$\{(x, ?), (y, ?), (z, ?)\}$
$[y := x]^1;$	←	$\{(x, ?), (y, 1), (z, ?)\}$
$[z := 1]^2;$	←	$\{(x, ?), (y, 1), (y, 5), (z, 2), (z, 4)\} \cup \{(y, 5), (z, 4)\}$
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

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

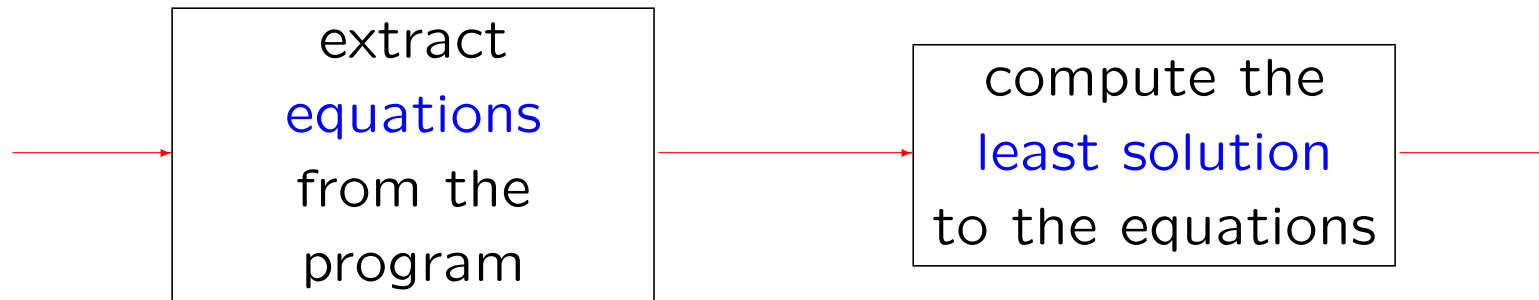
A safe solution — but not the best

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

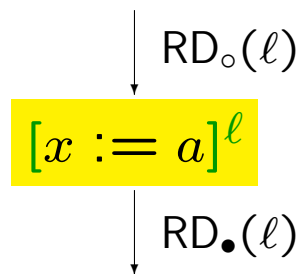
How to automate the analysis



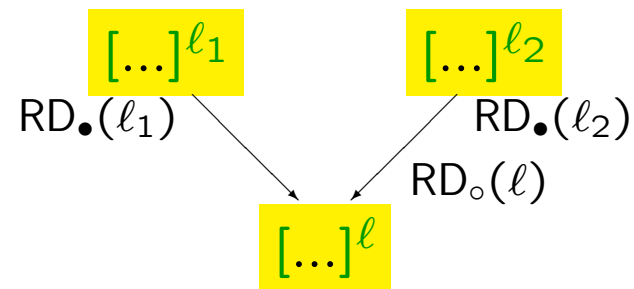
Analysis information:

- $RD_{\circ}(\ell)$: information available at the **entry** of block ℓ
- $RD_{\bullet}(\ell)$: information available at the **exit** of block ℓ

Two kinds of equations



$$\text{RD}_o(l) \setminus \{(x, l') \mid l' \in \mathbf{Lab}\} \cup \{(x, l)\} = \text{RD}_\bullet(l)$$



$$\text{RD}_\bullet(l_1) \cup \text{RD}_\bullet(l_2) = \text{RD}_o(l)$$

Flow through assignments and tests

$[y := x]^1;$ \longleftarrow $RD_{\bullet}(1) = RD_{\circ}(1) \setminus \{(y, \ell) \mid \ell \in \mathbf{Lab}\} \cup \{(y, 1)\}$

$[z := 1]^2;$ \longleftarrow $RD_{\bullet}(2) = RD_{\circ}(2) \setminus \{(z, \ell) \mid \ell \in \mathbf{Lab}\} \cup \{(z, 2)\}$

while $[y > 0]^3$ do \longleftarrow $RD_{\bullet}(3) = RD_{\circ}(3)$

$[z := z * y]^4;$ \longleftarrow $RD_{\bullet}(4) = RD_{\circ}(4) \setminus \{(z, \ell) \mid \ell \in \mathbf{Lab}\} \cup \{(z, 4)\}$

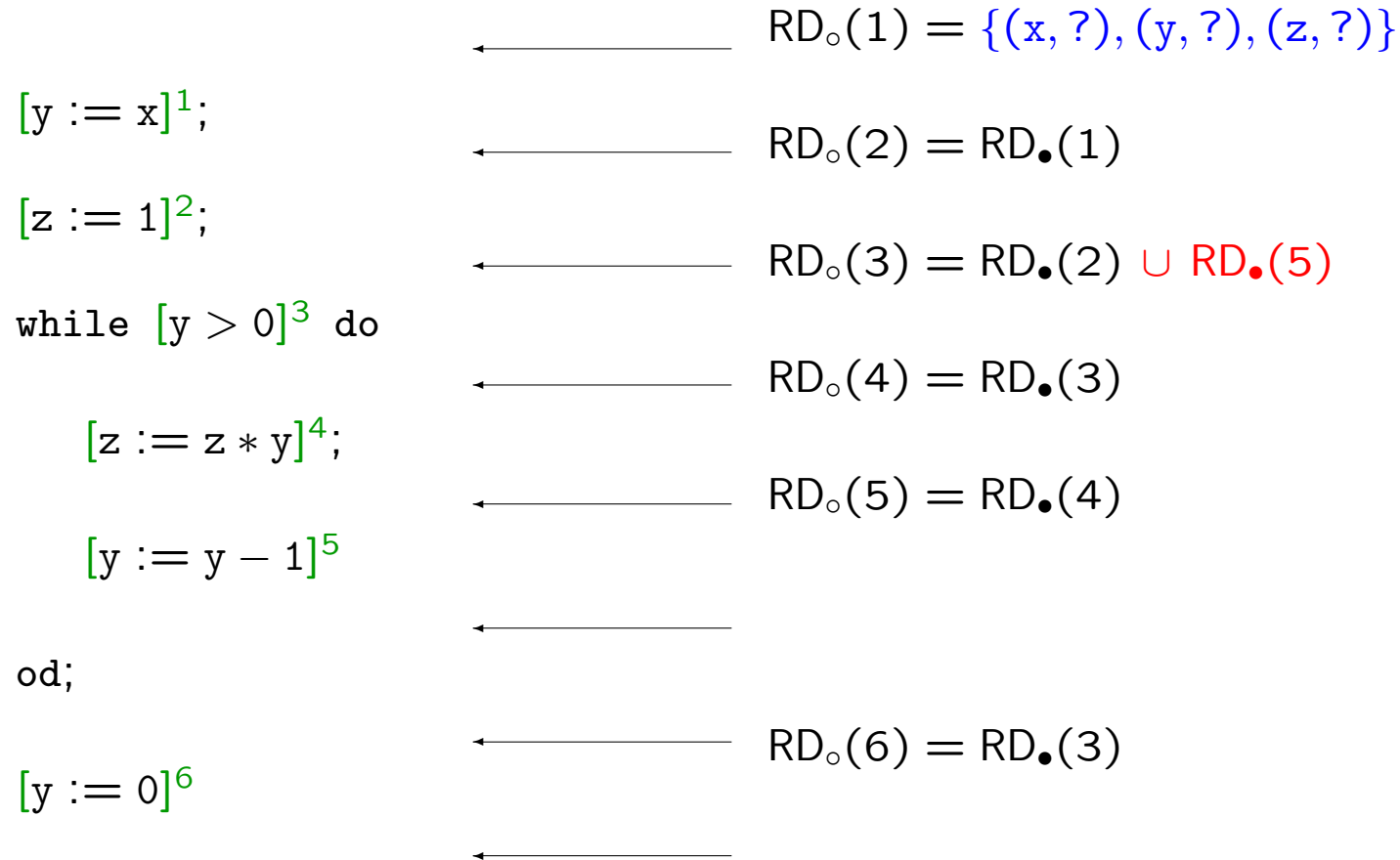
$[y := y - 1]^5$ \longleftarrow $RD_{\bullet}(5) = RD_{\circ}(5) \setminus \{(y, \ell) \mid \ell \in \mathbf{Lab}\} \cup \{(y, 5)\}$

od; \longleftarrow

$[y := 0]^6$ \longleftarrow $RD_{\bullet}(6) = RD_{\circ}(6) \setminus \{(y, \ell) \mid \ell \in \mathbf{Lab}\} \cup \{(y, 6)\}$

$\mathbf{Lab} = \{1, 2, 3, 4, 5, 6\}$ \longleftarrow 6 equations in $RD_{\circ}(1), \dots, RD_{\bullet}(6)$

Flow along the control



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

6 equations in
 $RD_o(1), \dots, RD_\bullet(6)$

Summary of equation system

$$RD_{\bullet}(1) = RD_{\circ}(1) \setminus \{(y, \ell) \mid \ell \in \mathbf{Lab}\} \cup \{(y, 1)\}$$

$$RD_{\bullet}(2) = RD_{\circ}(2) \setminus \{(z, \ell) \mid \ell \in \mathbf{Lab}\} \cup \{(z, 2)\}$$

$$RD_{\bullet}(3) = RD_{\circ}(3)$$

$$RD_{\bullet}(4) = RD_{\circ}(4) \setminus \{(z, \ell) \mid \ell \in \mathbf{Lab}\} \cup \{(z, 4)\}$$

$$RD_{\bullet}(5) = RD_{\circ}(5) \setminus \{(y, \ell) \mid \ell \in \mathbf{Lab}\} \cup \{(y, 5)\}$$

$$RD_{\bullet}(6) = RD_{\circ}(6) \setminus \{(y, \ell) \mid \ell \in \mathbf{Lab}\} \cup \{(y, 6)\}$$

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

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

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

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

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

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

- **12 sets:** $RD_{\circ}(1), \dots, RD_{\bullet}(6)$
all being subsets of $\mathbf{Var} \times \mathbf{Lab}$

- **12 equations:**
 $RD_j = F_j(RD_{\circ}(1), \dots, RD_{\bullet}(6))$

- **one function:**

$$F : \mathcal{P}(\mathbf{Var} \times \mathbf{Lab})^{12} \rightarrow$$

$$\mathcal{P}(\mathbf{Var} \times \mathbf{Lab})^{12}$$

- 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

$RD_1 := \emptyset; \dots; RD_{12} := \emptyset;$

- Iteration

while $RD_j \neq F_j(RD_1, \dots, RD_{12})$ for some j

do

$RD_j := F_j(RD_1, \dots, RD_{12})$

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

The example equations

RD _o	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
⋮	⋮	⋮	⋮	⋮	⋮	⋮

RD _•	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
⋮	⋮	⋮	⋮	⋮	⋮	⋮

The equations:

$$RD_{\bullet}(1) = RD_o(1) \setminus \{(y, \ell) \mid \dots\} \cup \{(y, 1)\}$$

$$RD_{\bullet}(2) = RD_o(2) \setminus \{(z, \ell) \mid \dots\} \cup \{(z, 2)\}$$

$$RD_{\bullet}(3) = RD_o(3)$$

$$RD_{\bullet}(4) = RD_o(4) \setminus \{(z, \ell) \mid \dots\} \cup \{(z, 4)\}$$

$$RD_{\bullet}(5) = RD_o(5) \setminus \{(y, \ell) \mid \dots\} \cup \{(y, 5)\}$$

$$RD_{\bullet}(6) = RD_o(6) \setminus \{(y, \ell) \mid \dots\} \cup \{(y, 6)\}$$

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

$$RD_o(2) = RD_{\bullet}(1)$$

$$RD_o(3) = RD_{\bullet}(2) \cup RD_{\bullet}(5)$$

$$RD_o(4) = RD_{\bullet}(3)$$

$$RD_o(5) = RD_{\bullet}(4)$$

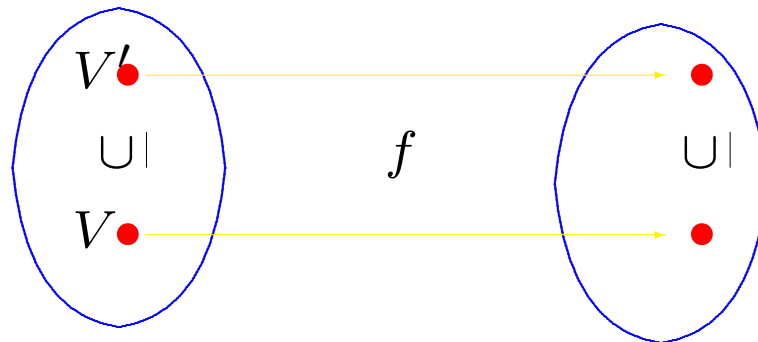
$$RD_o(6) = RD_{\bullet}(3)$$

Why does it work? (1)

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

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

(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

$$V_0 \subseteq V_1 \subseteq V_2 \subseteq V_3 \subseteq \dots$$

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

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.

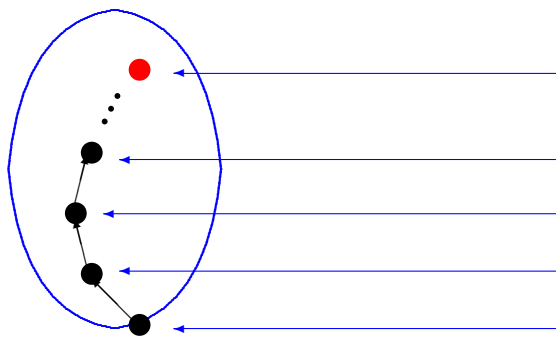
Why does it work? (3)

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

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

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 $f^n(\emptyset) = f^{n+1}(\emptyset) = \dots$

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



$\text{lfp}(f) = f^n(\emptyset) = f^{n+1}(\emptyset)$ for some n

$f^3(\emptyset)$

$f^2(\emptyset)$

$f^1(\emptyset)$

\emptyset

Correctness of the algorithm

- Initialisation

$RD_1 := \emptyset; \dots; RD_{12} := \emptyset;$

Invariant: $\vec{RD} \subseteq F^n(\vec{\emptyset})$ since $\vec{RD} = \vec{\emptyset}$ is the least element

- Iteration

while $RD_j \neq F_j(RD_1, \dots, RD_{12})$ for some j

do assume \vec{RD} is \vec{RD}' and $\vec{RD}' \subseteq F^n(\vec{\emptyset})$

$RD_j := F_j(RD_1, \dots, RD_{12})$

 then $\vec{RD} \subseteq F(\vec{RD}') \subseteq F^{n+1}(\vec{\emptyset}) = F^n(\vec{\emptyset})$ when $\text{lfp}(F) = F^n(\vec{\emptyset})$

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

The algorithm terminates because $RD_j \subset F_j(RD_1, \dots, RD_{12})$ is only possible finitely many times since $\mathcal{P}(\text{Var} \times \text{Lab})^{12}$ satisfies the Ascending Chain Condition.

Constraint Based Analysis

- **Technique:** Constraint Based Analysis
- **Example:** Control Flow Analysis
 - idea
 - constructing a constraint system
 - solving the constraints
 - theoretical underpinnings

Example: Control Flow Analysis

```
let f = fn x => x 7
    g = fn y => y
    h = fn z => 3
in f g + f (g h)
```

```
  ↓      ↓
g 7     f h
        ↓
        h 7
```

Aim: For each function application, which function abstractions may be applied?

function applications	function abstractions that may be applied
x 7	g, h
f g	f
g h	g
f (g h)	f

Solutions

```
let f = fn x => x 7
    g = fn y => y
    h = fn z => 3
in f g + f (g h)
```

The best solution:

function applications	function abstractions that may be applied
x 7	g, h
f g	f
g h	g
f (g h)	f

A safe solution – but not the best:

function applications	function abstractions that may be applied
x 7	g, h, f
f g	f
g h	g
f (g h)	f

An unsafe solution:

function applications	function abstractions that may be applied
x 7	g, h
f g	f
g h	g
f (g h)	f

An application of control flow analysis

```
let f = fn x => x 7  
    g = fn y => y  
    h = fn z => 3  
in f g + f (g h)
```

Aim: For each function application, which function abstractions may be applied?

Partial evaluation of function call:

```
let f = fn x => case x of  
                    g: 7  
                    h: 3  
    g = fn y => y  
    h = fn z => 3  
in f g + f (g h)
```

function applications	function abstractions that may be applied
x 7	g, h
f g	f
g h	g
f (g h)	f

The underlying analysis problem

```
let f = fn x => x 7
    g = fn y => y
    h = fn z => 3
in f g + f (g h)
```

Aim: for each **function application**, which function abstractions may be applied?

The analysis will compute:

- for each **subexpression**, which function abstractions may it denote?
— e.g. **(g h)** may evaluate to **h**
introduce abstract cache **C**
- for each **variable**, which function abstractions may it denote?
— e.g. **x** may be **g** or **h**
introduce abstract environment **R**

The best solution to the analysis problem

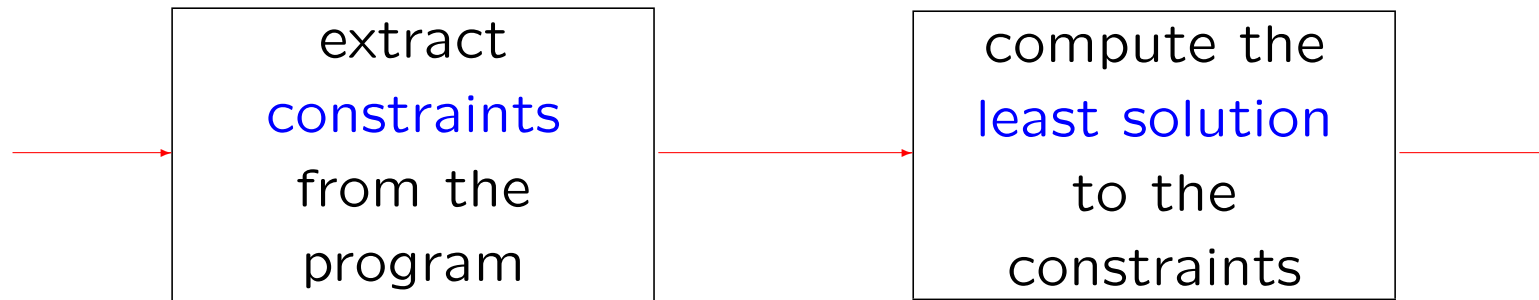
Add labels to subexpressions:

```
let f = fn x => (x1 72)3
    g = fn y => y4
    h = fn z => 35
in (f6 g7)8 + (f9 (g10 h11)12)13
```

R	variable may be bound to
x	{fn y => y ⁴ , fn z => 3 ⁵ }
y	{fn z => 3 ⁵ }
z	∅
f	{fn x => (x ¹ 7 ²) ³ }
g	{fn y => y ⁴ }
h	{fn z => 3 ⁵ }

C	subexpression may evaluate to
1	{fn y => y ⁴ , fn z => 3 ⁵ }
2	∅
3	∅
4	{fn z => 3 ⁵ }
5	∅
6	{fn x => (x ¹ 7 ²) ³ }
7	{fn y => y ⁴ }
8	∅
9	{fn x => (x ¹ 7 ²) ³ }
10	{fn y => y ⁴ }
11	{fn z => 3 ⁵ }
12	{fn z => 3 ⁵ }
13	∅

How to automate the analysis



Analysis information:

- $R(x)$: information available for the variable x
- $C(\ell)$: information available at the subexpression labelled ℓ

Three kinds of constraints

- let-bound variables evaluate to their abstraction
- variables evaluate to their (abstract) values
- if a function abstraction is applied to an argument then
 - the argument is a possible value of the formal parameter
 - the value of the body of the abstraction is a possible value of the application

let-bound variables

let-bound variables evaluate to their abstractions

`let $f = \text{fn } x \Rightarrow e$` gives rise to the constraint $\{\text{fn } x \Rightarrow e\} \subseteq R(f)$

<code>let $f = \text{fn } x \Rightarrow (x^1 - 7^2)^3$</code>	\longleftarrow	$\{\text{fn } x \Rightarrow (x^1 - 7^2)^3\} \subseteq R(f)$
<code> $g = \text{fn } y \Rightarrow y^4$</code>	\longleftarrow	$\{\text{fn } y \Rightarrow y^4\} \subseteq R(g)$
<code>in $(f^5 - g^6)^7$</code>		

Variables

Variables evaluate to their abstract values

x^ℓ gives rise to the constraint $R(x) \subseteq C(\ell)$

let f = fn x => (x ¹ 7 ²) ³	←	R(x) ⊆ C(1)
g = fn y => y ⁴	←	R(y) ⊆ C(4)
in (f ⁵ g ⁶) ⁷	←	{ R(f) ⊆ C(5) R(g) ⊆ C(6)

Function application (1)

if a function abstraction is applied to an argument then

- the argument is a possible value of the formal parameter
- the value of the body of the abstraction is a possible value of the application

```
let f = fn x => (x1 72)3
```

```
  g = fn y => y4
```

```
in (f5 g6)7
```

← if (fn y => y⁴) ∈ C(1)
then C(2) ⊆ R(y) and C(4) ⊆ C(3)

if (fn x => (x¹ 7²)³) ∈ C(1)
then C(2) ⊆ R(x) and C(3) ⊆ C(3)

Conditional constraints

Function application (2)

if a function abstraction is applied to an argument then

- the argument is a possible value of the formal parameter
- the value of the body of the abstraction is a possible value of the application

```
let f = fn x => (x1 72)3
```

```
  g = fn y => y4
```

```
in (f5 g6)7
```

```
if (fn y => y4) ∈ C(5)
```

```
then C(6) ⊆ R(y) and C(4) ⊆ C(7)
```

```
if (fn x => (x1 72)3) ∈ C(5)
```

```
then C(6) ⊆ R(x) and C(3) ⊆ C(7)
```

Summary of constraint system

$$\{\text{fn } x \Rightarrow (x^1 \ 7^2)^3\} \subseteq R(\text{f})$$

$$\{\text{fn } y \Rightarrow y^4\} \subseteq R(\text{g})$$

$$R(x) \subseteq C(1)$$

$$R(y) \subseteq C(4)$$

$$R(\text{f}) \subseteq C(5)$$

$$R(\text{g}) \subseteq C(6)$$

$$(\text{fn } y \Rightarrow y^4) \in C(1) \Rightarrow C(2) \subseteq R(y)$$

$$(\text{fn } y \Rightarrow y^4) \in C(1) \Rightarrow C(4) \subseteq C(3)$$

$$(\text{fn } x \Rightarrow (x^1 \ 7^2)^3) \in C(1) \Rightarrow C(2) \subseteq R(x)$$

$$(\text{fn } x \Rightarrow (x^1 \ 7^2)^3) \in C(1) \Rightarrow C(3) \subseteq C(3)$$

$$(\text{fn } y \Rightarrow y^4) \in C(5) \Rightarrow C(6) \subseteq R(y)$$

$$(\text{fn } y \Rightarrow y^4) \in C(5) \Rightarrow C(4) \subseteq C(7)$$

$$(\text{fn } x \Rightarrow (x^1 \ 7^2)^3) \in C(5) \Rightarrow C(6) \subseteq R(x)$$

$$(\text{fn } x \Rightarrow (x^1 \ 7^2)^3) \in C(5) \Rightarrow C(3) \subseteq C(7)$$

- **11 sets:** $R(x), R(y), R(\text{f}), R(\text{g}), C(1), \dots, C(7)$; all being subsets of the set **Abstr** of function abstractions
- the constraints can be reformulated as **a function:**
 $F : \mathcal{P}(\text{Abstr})^{11} \rightarrow \mathcal{P}(\text{Abstr})^{11}$
- we want the **least fixed point** of F — this is the **best solution** to the constraint system

$\mathcal{P}(S)$ is the set of all subsets of the set S ; e.g. $\mathcal{P}(\{0, 1\}) = \{\emptyset, \{0\}, \{1\}, \{0, 1\}\}$.

The constraints define a function

$$\begin{aligned}
 &\{\text{fn } x \Rightarrow (x^1 \ 7^2)^3\} \subseteq R(f) \\
 &\{\text{fn } y \Rightarrow y^4\} \subseteq R(g) \\
 &R(x) \subseteq C(1) \\
 &R(y) \subseteq C(4) \\
 &R(f) \subseteq C(5) \\
 &R(g) \subseteq C(6) \\
 &(\text{fn } y \Rightarrow y^4) \in C(1) \Rightarrow C(2) \subseteq R(y) \\
 &(\text{fn } y \Rightarrow y^4) \in C(1) \Rightarrow C(4) \subseteq C(3) \\
 &(\text{fn } x \Rightarrow (x^1 \ 7^2)^3) \in C(1) \Rightarrow C(2) \subseteq R(x) \\
 &(\text{fn } x \Rightarrow (x^1 \ 7^2)^3) \in C(1) \Rightarrow C(3) \subseteq C(3) \\
 &(\text{fn } y \Rightarrow y^4) \in C(5) \Rightarrow C(6) \subseteq R(y) \\
 &(\text{fn } y \Rightarrow y^4) \in C(5) \Rightarrow C(4) \subseteq C(7) \\
 &(\text{fn } x \Rightarrow (x^1 \ 7^2)^3) \in C(5) \Rightarrow C(6) \subseteq R(x) \\
 &(\text{fn } x \Rightarrow (x^1 \ 7^2)^3) \in C(5) \Rightarrow C(3) \subseteq C(7)
 \end{aligned}$$

$F : \mathcal{P}(\text{Abstr})^{11} \rightarrow \mathcal{P}(\text{Abstr})^{11}$
 is defined by

$$\begin{aligned}
 &F_{R(f)}(\dots, R_f, \dots) = \\
 &\quad \vdots \quad R_f \cup \{\text{fn } x \Rightarrow (x^1 \ 7^2)^3\} \\
 &F_{C(1)}(R_x, \dots, C_1, \dots) = C_1 \cup R_x \\
 &\quad \vdots \\
 &F_{R(y)}(\dots, R_y, \dots, C_1, C_2, \dots, C_5, C_6, \dots) = \\
 &\quad R_y \cup \{a \in C_2 \mid \text{fn } y \Rightarrow y^4 \in C_1\} \\
 &\quad \cup \{a \in C_6 \mid \text{fn } y \Rightarrow y^4 \in C_5\}
 \end{aligned}$$

How to solve the constraints

- Initialisation

$X_1 := \emptyset; \dots; X_{11} := \emptyset;$

- Iteration

while $X_j \neq F_{X_j}(X_1, \dots, X_{11})$ for some j

do

$X_j := F_{X_j}(X_1, \dots, X_{11})$

writing

X_1, \dots, X_{11} for
 $R(x), \dots, R(g), C(1), \dots, C(7)$

The algorithm terminates and computes the least fixed point of F

In practice we propagate smaller contributions than F_{X_j} , e.g. a constraint at a time.

The example constraint system

R	x	y	f	g
0	\emptyset	\emptyset	\emptyset	\emptyset
1	\emptyset	\emptyset	$\text{fn } x \Rightarrow .3$	\emptyset
2	\emptyset	\emptyset	$\text{fn } x \Rightarrow .3$	$\text{fn } y \Rightarrow .4$
3	\emptyset	\emptyset	$\text{fn } x \Rightarrow .3$	$\text{fn } y \Rightarrow .4$
4	\emptyset	\emptyset	$\text{fn } x \Rightarrow .3$	$\text{fn } y \Rightarrow .4$
5	$\text{fn } y \Rightarrow .4$	\emptyset	$\text{fn } x \Rightarrow .3$	$\text{fn } y \Rightarrow .4$
6	$\text{fn } y \Rightarrow .4$	\emptyset	$\text{fn } x \Rightarrow .3$	$\text{fn } y \Rightarrow .4$

C	1	2	3	4	5	6	7
0	\emptyset	\emptyset	\emptyset	\emptyset	\emptyset	\emptyset	\emptyset
1	\emptyset	\emptyset	\emptyset	\emptyset	\emptyset	\emptyset	\emptyset
2	\emptyset	\emptyset	\emptyset	\emptyset	\emptyset	\emptyset	\emptyset
3	\emptyset	\emptyset	\emptyset	\emptyset	$\text{fn } x \Rightarrow .3$	\emptyset	\emptyset
4	\emptyset	\emptyset	\emptyset	\emptyset	$\text{fn } x \Rightarrow .3$	$\text{fn } y \Rightarrow .4$	\emptyset
5	\emptyset	\emptyset	\emptyset	\emptyset	$\text{fn } x \Rightarrow .3$	$\text{fn } y \Rightarrow .4$	\emptyset
6	$\text{fn } y \Rightarrow .4$	\emptyset	\emptyset	\emptyset	$\text{fn } x \Rightarrow .3$	$\text{fn } y \Rightarrow .4$	\emptyset

The constraints:

$$\{\text{fn } x \Rightarrow .3\} \subseteq R(f) \quad (1)$$

$$\{\text{fn } y \Rightarrow .4\} \subseteq R(g) \quad (2)$$

$$R(x) \subseteq C(1) \quad (6)$$

$$R(y) \subseteq C(4) \quad (3)$$

$$R(f) \subseteq C(5) \quad (3)$$

$$R(g) \subseteq C(6) \quad (4)$$

$$(\text{fn } y \Rightarrow .4) \in C(1) \Rightarrow C(2) \subseteq R(y)$$

$$(\text{fn } y \Rightarrow .4) \in C(1) \Rightarrow C(4) \subseteq C(3)$$

$$(\text{fn } x \Rightarrow .3) \in C(1) \Rightarrow C(2) \subseteq R(x)$$

$$(\text{fn } x \Rightarrow .3) \in C(1) \Rightarrow C(3) \subseteq C(3)$$

$$(\text{fn } y \Rightarrow .4) \in C(5) \Rightarrow C(6) \subseteq R(y)$$

$$(\text{fn } y \Rightarrow .4) \in C(5) \Rightarrow C(4) \subseteq C(7)$$

$$(\text{fn } x \Rightarrow .3) \in C(5) \Rightarrow C(6) \subseteq R(x) \quad (5)$$

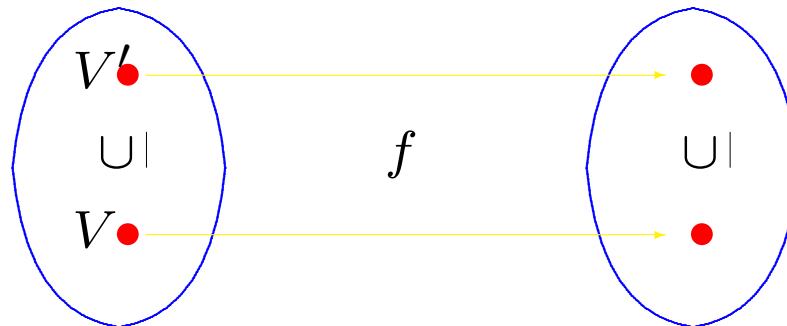
$$(\text{fn } x \Rightarrow .3) \in C(5) \Rightarrow C(3) \subseteq C(7)$$

Why does it work? (1)

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

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

(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

$$V_0 \subseteq V_1 \subseteq V_2 \subseteq V_3 \subseteq \dots$$

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

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 **Abstr** will be a finite set so $\mathcal{P}(\mathbf{Abstr})$ with the subset ordering satisfies the Ascending Chain Condition.

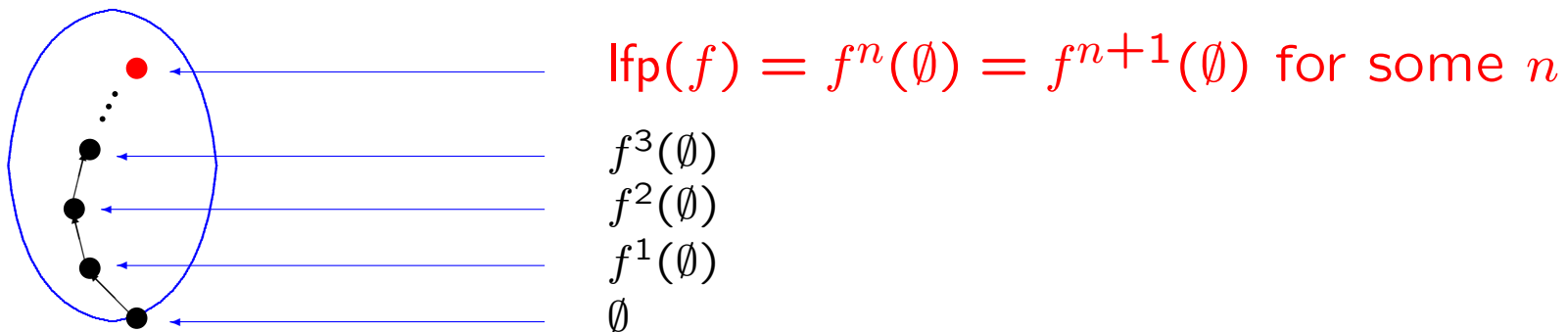
Why does it work? (3)

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

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

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

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



Correctness of the algorithm

- Initialisation

$X_1 := \emptyset; \dots; X_{11} := \emptyset;$

Invariant: $\vec{X} \subseteq F^n(\vec{\emptyset})$ since $\vec{X} = \vec{\emptyset}$ is the least element

- Iteration

while $X_j \neq F_{X_j}(X_1, \dots, X_{11})$ for some j

do assume \vec{X} is \vec{X}' and $\vec{X}' \subseteq F^n(\vec{\emptyset})$

$X_j := F_{X_j}(X_1, \dots, X_{11})$

 then $\vec{X} \subseteq F(\vec{X}') \subseteq F^{n+1}(\vec{\emptyset}) = F^n(\vec{\emptyset})$ when $\text{lfp}(F) = F^n(\vec{\emptyset})$

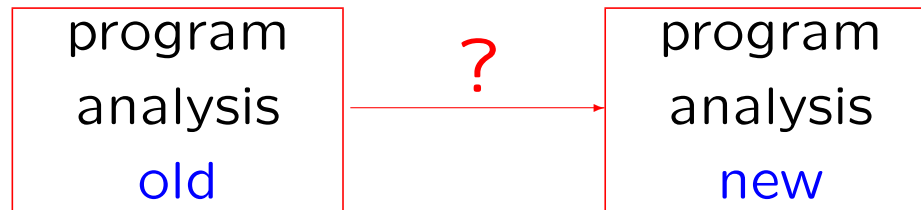
If the algorithm terminates then it computes the least fixed point of F

The algorithm terminates because $X_j \subset F_{X_j}(X_1, \dots, X_{11})$ is only possible finitely many times since $\mathcal{P}(\text{AbsExp})^{11}$ satisfies the Ascending Chain Condition

Abstract Interpretation

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

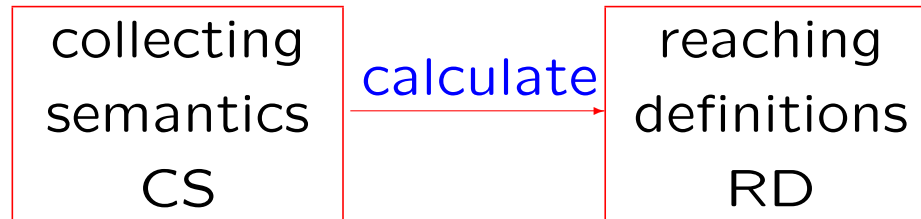
Abstract Interpretation



- 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



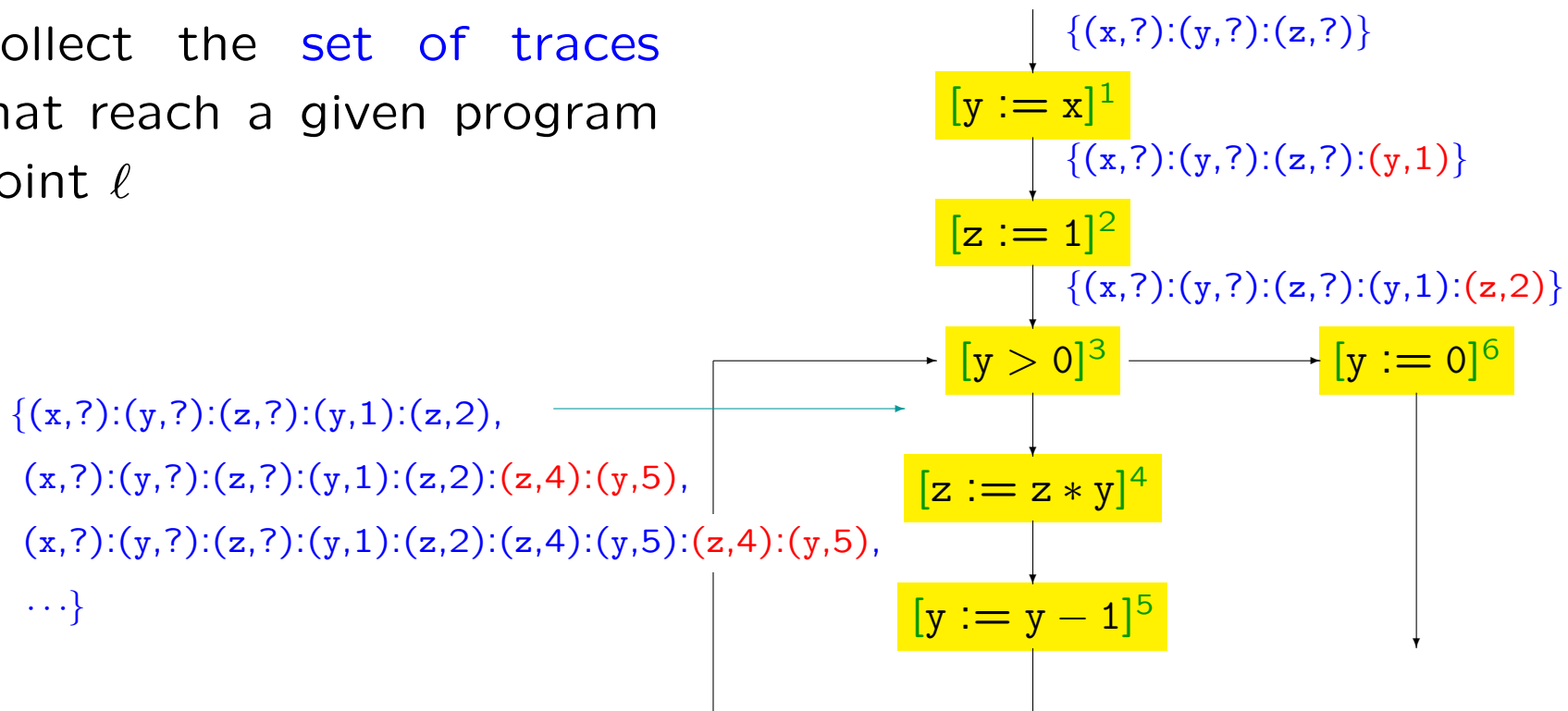
The **collecting semantics** CS

- 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 ℓ



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:

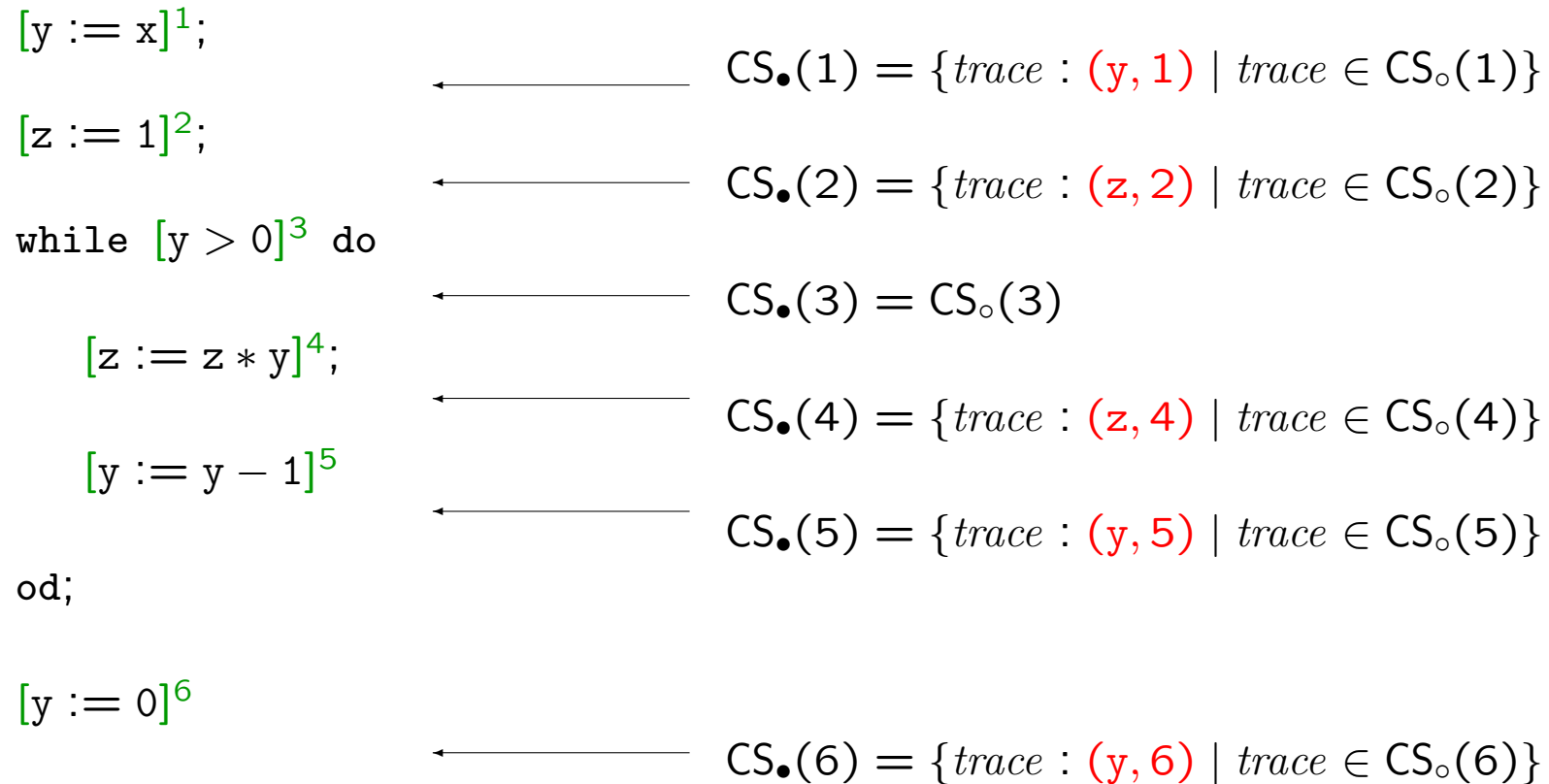
- prove the **correctness**: the set of traces computed by the analysis is a superset of the possible traces

Two kinds of equations

$$\begin{array}{c}
 \downarrow \text{CS}_\circ(\ell) \\
 [x := a]^\ell \\
 \downarrow \text{CS}_\bullet(\ell) \\
 \{ \text{trace} : (x, \ell) \mid \text{trace} \in \text{CS}_\circ(\ell) \} \\
 = \text{CS}_\bullet(\ell)
 \end{array}$$

$$\begin{array}{ccc}
 \begin{array}{c} \text{CS}_\bullet(\ell_1) \\ \downarrow \\ [\dots]^{\ell_1} \end{array} & & \begin{array}{c} \text{CS}_\bullet(\ell_2) \\ \downarrow \\ [\dots]^{\ell_2} \end{array} \\
 & \searrow \quad \swarrow & \\
 & \text{CS}_\circ(\ell) & \\
 & \downarrow & \\
 & [\dots]^\ell & \\
 \text{CS}_\bullet(\ell_1) \cup \text{CS}_\bullet(\ell_2) & = & \text{CS}_\circ(\ell)
 \end{array}$$

Flow through assignments and tests



6 equations in
 $CS_{\circ}(1), \dots, CS_{\bullet}(6)$

Flow along the control

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

6 equations in
 $CS_o(1), \dots, CS_\bullet(6)$

Summary of Collecting Semantics

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

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

$$CS_{\bullet}(3) = CS_{\circ}(3)$$

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

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

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

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

$$CS_{\circ}(2) = CS_{\bullet}(1)$$

$$CS_{\circ}(3) = CS_{\bullet}(2) \cup CS_{\bullet}(5)$$

$$CS_{\circ}(4) = CS_{\bullet}(3)$$

$$CS_{\circ}(5) = CS_{\bullet}(4)$$

$$CS_{\circ}(6) = CS_{\bullet}(3)$$

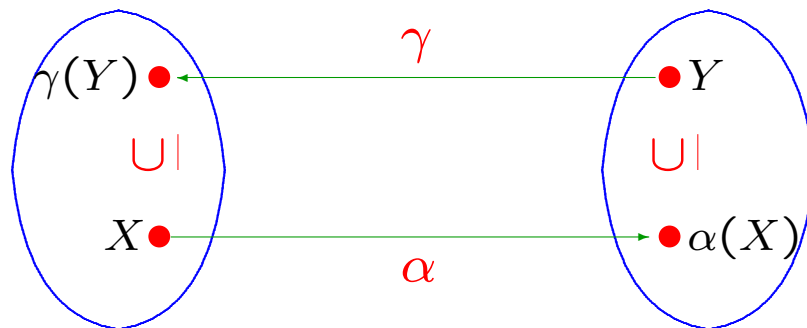
- **12 sets:** $CS_{\circ}(1), \dots, CS_{\bullet}(6)$
all being subsets of **Trace**
- **12 equations:**
 $CS_j = G_j(CS_{\circ}(1), \dots, CS_{\bullet}(6))$
- **one function:**
 $G : \mathcal{P}(\mathbf{Trace})^{12} \rightarrow \mathcal{P}(\mathbf{Trace})^{12}$
- we want the **least fixed point** of G — **but it is uncomputable!**

Example: Inducing an analysis

Galois Connections

A Galois connection between two sets is a pair of (α, γ) of functions between the sets satisfying

$$X \subseteq \gamma(Y) \iff \alpha(X) \subseteq Y$$



$\mathcal{P}(\text{Trace})$

$\mathcal{P}(\text{Var} \times \text{Lab})$

collecting semantics reaching definitions

α : abstraction function
 γ : concretisation function

Semantically Reaching Definitions

For a single trace:

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

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

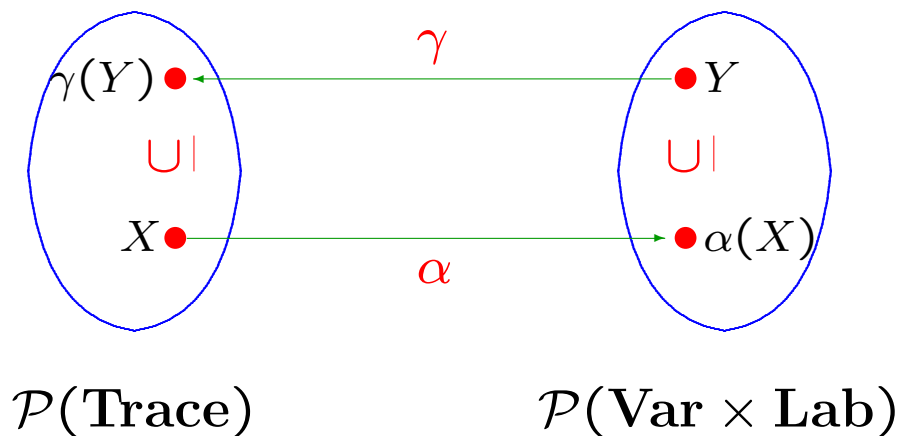
For a set of traces:

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

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

Galois connection for Reaching Definitions analysis

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



Galois connection:

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

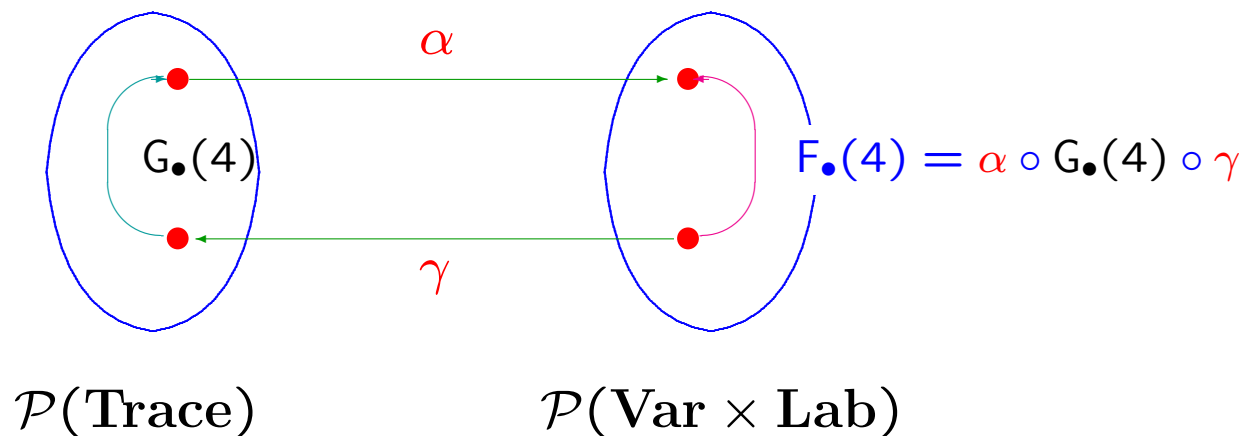
Inducing the Reaching Definitions analysis (1)

Known:

- $G_{\bullet}(4)$ defined on $\mathcal{P}(\mathbf{Trace})$
- the Galois connection (α, γ)

Calculate:

- $F_{\bullet}(4)$ defined on $\mathcal{P}(\mathbf{Var} \times \mathbf{Lab})$
as $F_{\bullet}(4) = \alpha \circ G_{\bullet}(4) \circ \gamma$



Inducing the Reaching Definitions analysis (2)

$$\begin{aligned}
 \text{RD}_\bullet(4) &= \text{F}_\bullet(4)(\dots, \text{RD}_\circ(4), \dots) \\
 &= \alpha(\text{G}_\bullet(4)(\gamma(\dots, \text{RD}_\circ(4), \dots))) \quad \text{using } \text{F}_\bullet(4) = \alpha \circ \text{G}_\bullet(4) \circ \gamma \\
 &= \alpha(\{tr : (z, 4) \mid tr \in \gamma(\text{RD}_\circ(4))\}) \\
 &\quad \text{using } \text{G}_\bullet(4)(\dots, \text{CS}_\circ(4), \dots) = \{tr : (z, 4) \mid tr \in \text{CS}_\circ(4)\} \\
 &= \text{SRD}(\{tr : (z, 4) \mid tr \in \gamma(\text{RD}_\circ(4))\}) \quad \text{using } \alpha = \text{SRD} \\
 &= (\text{SRD}(\{tr \mid tr \in \gamma(\text{RD}_\circ(4))\}) \setminus \{(z, \ell) \mid \ell \in \mathbf{Lab}\}) \cup \{(z, 4)\} \\
 &= (\alpha(\gamma(\text{RD}_\circ(4))) \setminus \{(z, \ell) \mid \ell \in \mathbf{Lab}\}) \cup \{(z, 4)\} \quad \text{using } \alpha = \text{SRD} \\
 &= (\text{RD}_\circ(4) \setminus \{(z, \ell) \mid \ell \in \mathbf{Lab}\}) \cup \{(z, 4)\} \quad \text{using } \alpha \circ \gamma = id
 \end{aligned}$$

just as before!

Type and Effect Systems

- **Technique:** Annotated Type Systems
- **Example:** Reaching Definitions analysis
 - idea
 - annotated base types
 - annotated type constructors

The while language

- syntax of statements:

$$\begin{aligned} S & ::= [x:=a]^\ell \mid S_1; S_2 \\ & \quad \mid \text{if } [b]^\ell \text{ then } S_1 \text{ else } S_2 \text{ fi} \\ & \quad \mid \text{while } [b]^\ell \text{ do } S \text{ od} \end{aligned}$$

- semantics:

statements map states to states

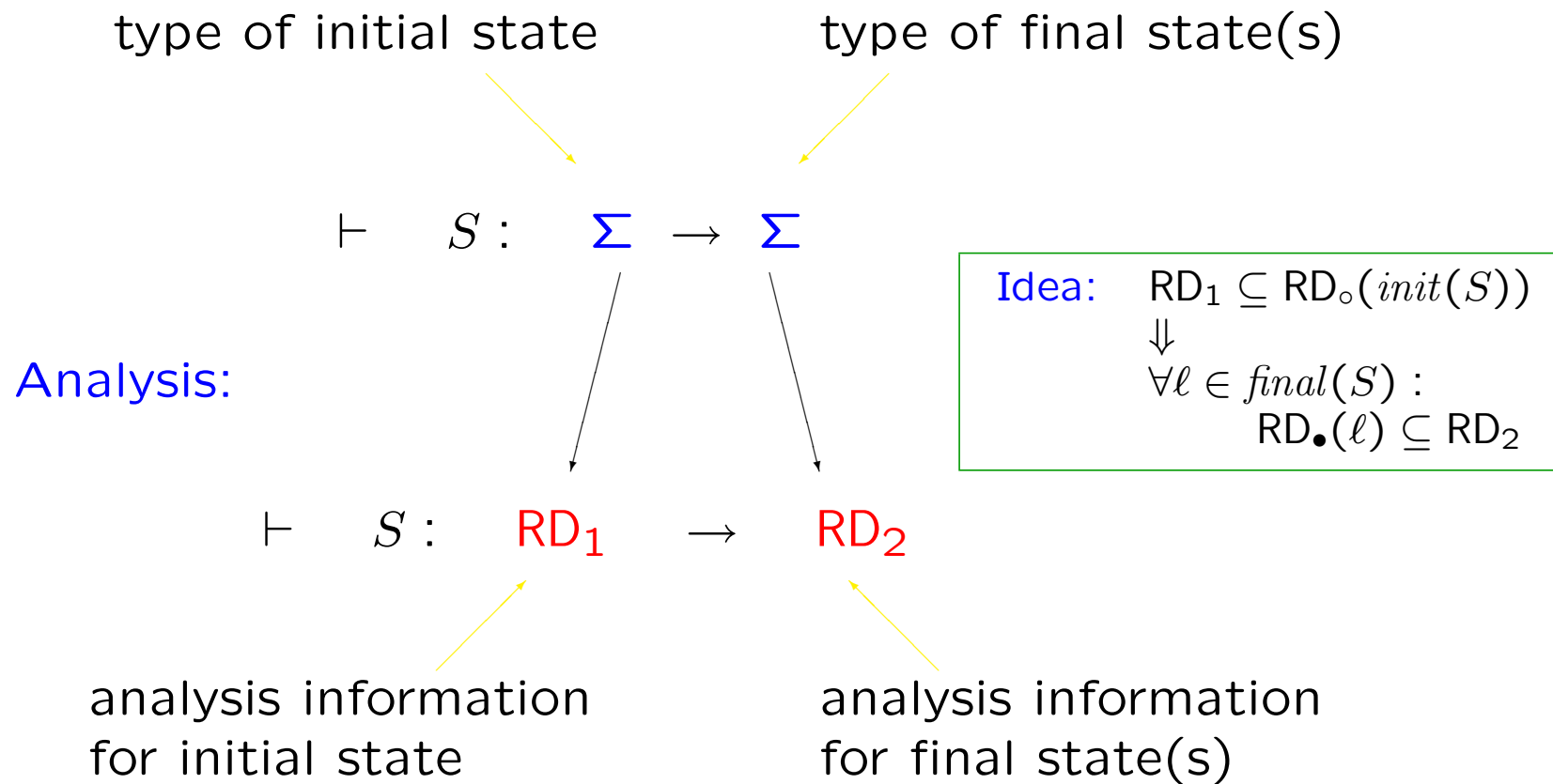
- types:

Σ is the type of states;

all statements S have type $\Sigma \rightarrow \Sigma$

written $\vdash S : \Sigma \rightarrow \Sigma$

Annotated base types



Annotated type system (1)

$$\vdash [x:=a]^\ell : \underbrace{\text{RD}}_{\text{before}} \rightarrow \underbrace{(\text{RD} \setminus \{(x, \ell') \mid \ell' \in \mathbf{Lab}\}) \cup \{(x, \ell)\}}_{\text{after}}$$

$$\frac{\vdash S_1 : \text{RD}_1 \rightarrow \text{RD}_2 \quad \vdash S_2 : \text{RD}_2 \rightarrow \text{RD}_3}{\vdash S_1; S_2 : \underbrace{\text{RD}_1}_{\text{before}} \rightarrow \underbrace{\text{RD}_3}_{\text{after}}} \quad \frac{\text{assumptions}}{\text{conclusion}}$$

Implicit: the analysis information at the **exit** of S_1
 equals the analysis information at the **entry** of S_2

Annotated type system (2)

$$\frac{\vdash S_1 : RD_1 \rightarrow RD_2 \quad \vdash S_2 : RD_1 \rightarrow RD_2}{\vdash \text{if } [b]^\ell \text{ then } S_1 \text{ else } S_2 \text{ fi} : RD_1 \rightarrow RD_2}$$

Implicit: the two branches have the same analysis information at their respective **entry** and **exit** points

$$\frac{\vdash S : RD \rightarrow RD}{\vdash \text{while } [b]^\ell \text{ do } S \text{ od} : RD \rightarrow RD}$$

Implicit: the occurrences of **RD** express an invariance i.e. a fixed point property!

Annotated type system (3)

The subsumption rule:

$$\frac{\vdash S : RD'_1 \rightarrow RD'_2}{\vdash S : RD_1 \rightarrow RD_2} \quad \text{if } RD_1 \subseteq RD'_1 \text{ and } RD'_2 \subseteq RD_2$$

The rule is essential for the rules for conditional and iteration to work

- $RD_1 \subseteq RD'_1$: strengthen the analysis information for the initial state
- $RD'_2 \subseteq RD_2$: weaken the analysis information for the final states

Example inference in the annotated type system

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

$$\begin{array}{l} \vdash [z:=z*y]^4: RD \rightarrow \{(x, ?), (y, 1), (y, 5), (z, 4)\} \\ \vdash [y:=y-1]^5: \{(x, ?), (y, 1), (y, 5), (z, 4)\} \rightarrow \{(x, ?), (y, 5), (z, 4)\} \end{array}$$

$$\vdash [z:=z*y]^4; [y:=y-1]^5: RD \rightarrow \{(x, ?), (y, 5), (z, 4)\}$$

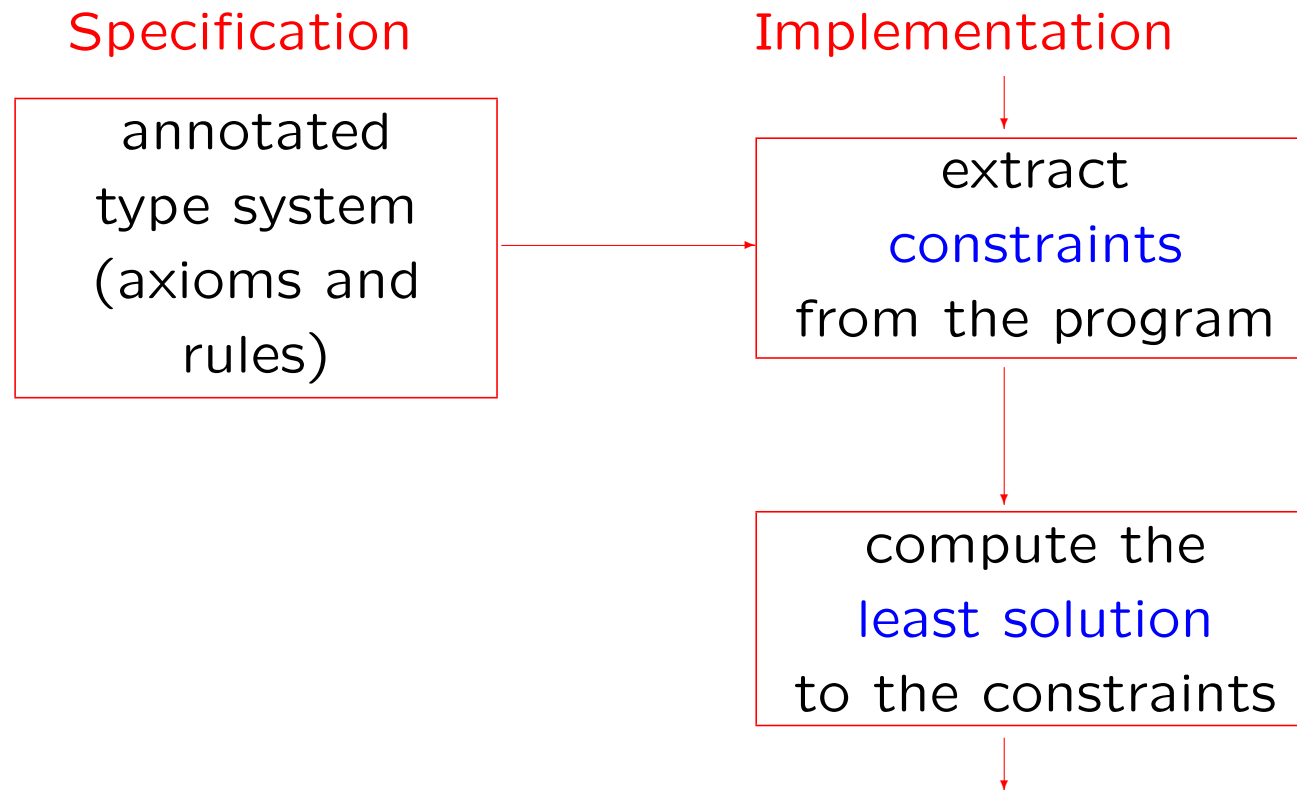
$$\begin{array}{l} \vdash [z:=z*y]^4; [y:=y-1]^5: RD \rightarrow RD \\ \text{using } \{(x, ?), (y, 5), (z, 4)\} \subseteq RD \end{array}$$

$$\vdash \text{while } [y>1]^3 \text{ do } [z:=z*y]^4; [y:=y-1]^5 \text{ od: } RD \rightarrow RD$$

⋮

$$\vdash [y:=x]^1; [z:=1]^2; \text{while } [y>1]^3 \text{ do } [z:=z*y]^4; [y:=y-1]^5 \text{ od; } [y:=0]^6: \\ \{(x, ?), (y, ?), (z, ?)\} \rightarrow \{(x, ?), (y, 6), (z, 2), (z, 4)\}$$

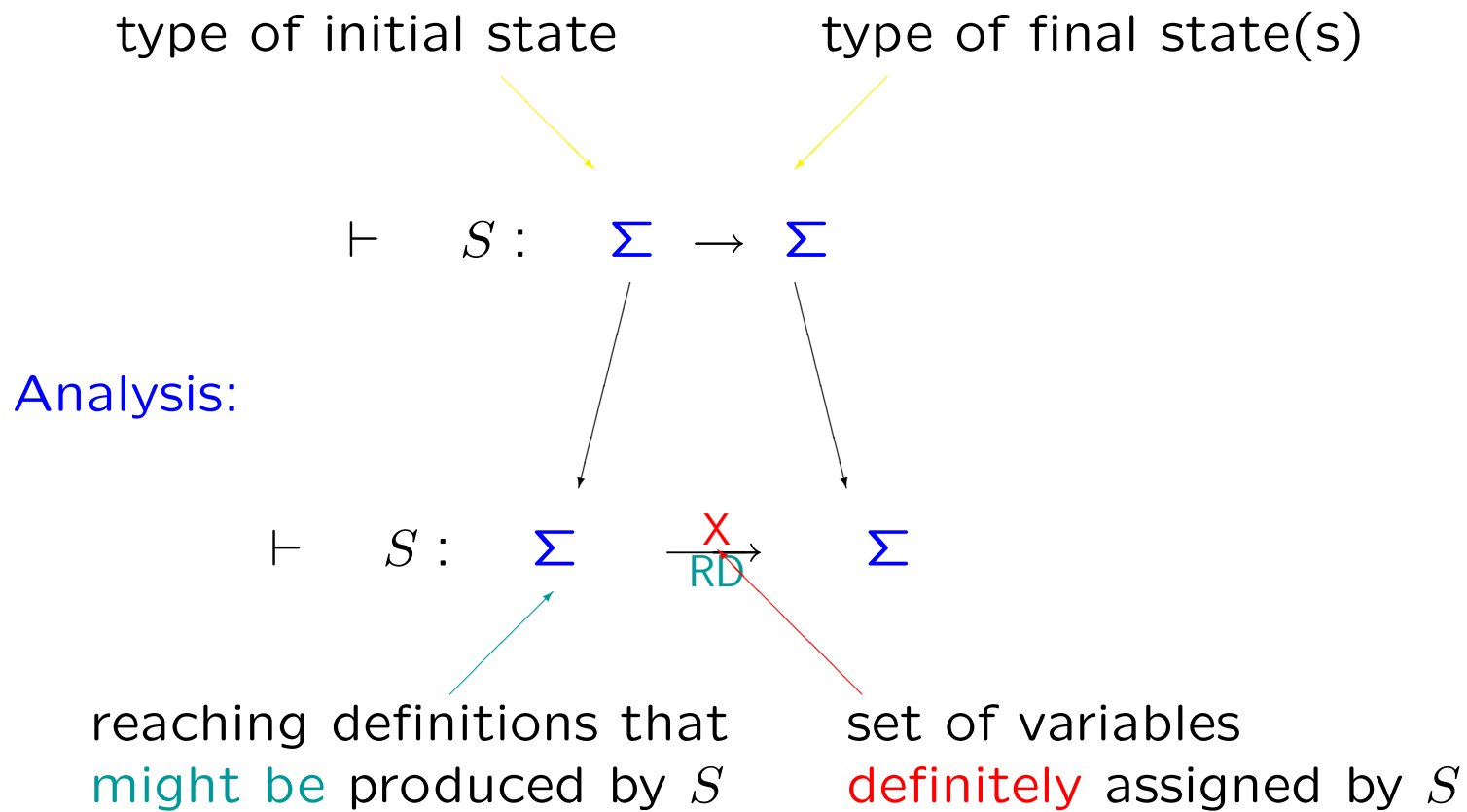
How to automate the analysis



Change of abstraction level: annotated type constructors

- **Until now:**
given a statement and a **specific** entry information RD_{\circ} we determine the **specific** exit information RD_{\bullet} .
- **Now:**
given a statement we determine how entry information is **transformed** into exit information

Annotated type constructors



Annotated type constructors (1)

$$\vdash [x := a]^\ell : \Sigma \xrightarrow[\{(x, \ell)\}]{{x}} \Sigma$$

$\{x\}$: variables definitely assigned

$\{(x, \ell)\}$: potential reaching definitions

$$\frac{\vdash S_1 : \Sigma \xrightarrow[RD_1]{X_1} \Sigma \quad \vdash S_2 : \Sigma \xrightarrow[RD_2]{X_2} \Sigma}{\vdash S_1; S_2 : \Sigma \xrightarrow[(RD_1 \setminus X_2) \cup RD_2]{X_1 \cup X_2} \Sigma}$$

$X_1 \cup X_2$: variables definitely assigned

$(RD_1 \setminus X_2) \cup RD_2$:
potential reaching definitions

Annotated type constructors (2)

$$\frac{\vdash S_1 : \Sigma \xrightarrow[\text{RD}_1]{X_1} \Sigma \quad \vdash S_2 : \Sigma \xrightarrow[\text{RD}_2]{X_2} \Sigma}{\vdash \text{if } [b]^\ell \text{ then } S_1 \text{ else } S_2 \text{ fi} : \Sigma \xrightarrow[\text{RD}_1 \cup \text{RD}_2]{X_1 \cap X_2} \Sigma}$$

$X_1 \cap X_2$:

variables definitely assigned

$\text{RD}_1 \cup \text{RD}_2$:

potential reaching definitions

$$\frac{\vdash S : \Sigma \xrightarrow[\text{RD}]{X} \Sigma}{\vdash \text{while } [b]^\ell \text{ do } S \text{ od} : \Sigma \xrightarrow[\text{RD}]{\emptyset} \Sigma}$$

\emptyset : variables definitely assigned

RD : potential reaching definitions

Annotated type constructors (3)

Subsumption rule:

$$\frac{\vdash S : \Sigma \xrightarrow[\text{RD}]{X} \Sigma}{\vdash S : \Sigma \xrightarrow[\text{RD}']{X'} \Sigma} \quad \begin{array}{l} \text{if } X' \subseteq X \quad (\text{variables definite assigned}) \\ \text{and } \text{RD} \subseteq \text{RD}' \quad (\text{potential reaching definitions}) \end{array}$$

the rule can be omitted!

Example inference in the annotated type system

$$\vdash [z:=z*y]^4: \Sigma \frac{\{z\}}{\{(z,4)\}} \rightarrow \Sigma$$

$$\vdash [y:=y-1]^5: \Sigma \frac{\{y\}}{\{(y,5)\}} \rightarrow \Sigma$$

$$\vdash [z:=z*y]^4; [y:=y-1]^5: \Sigma \frac{\{y,z\}}{\{(y,5),(z,4)\}} \rightarrow \Sigma$$

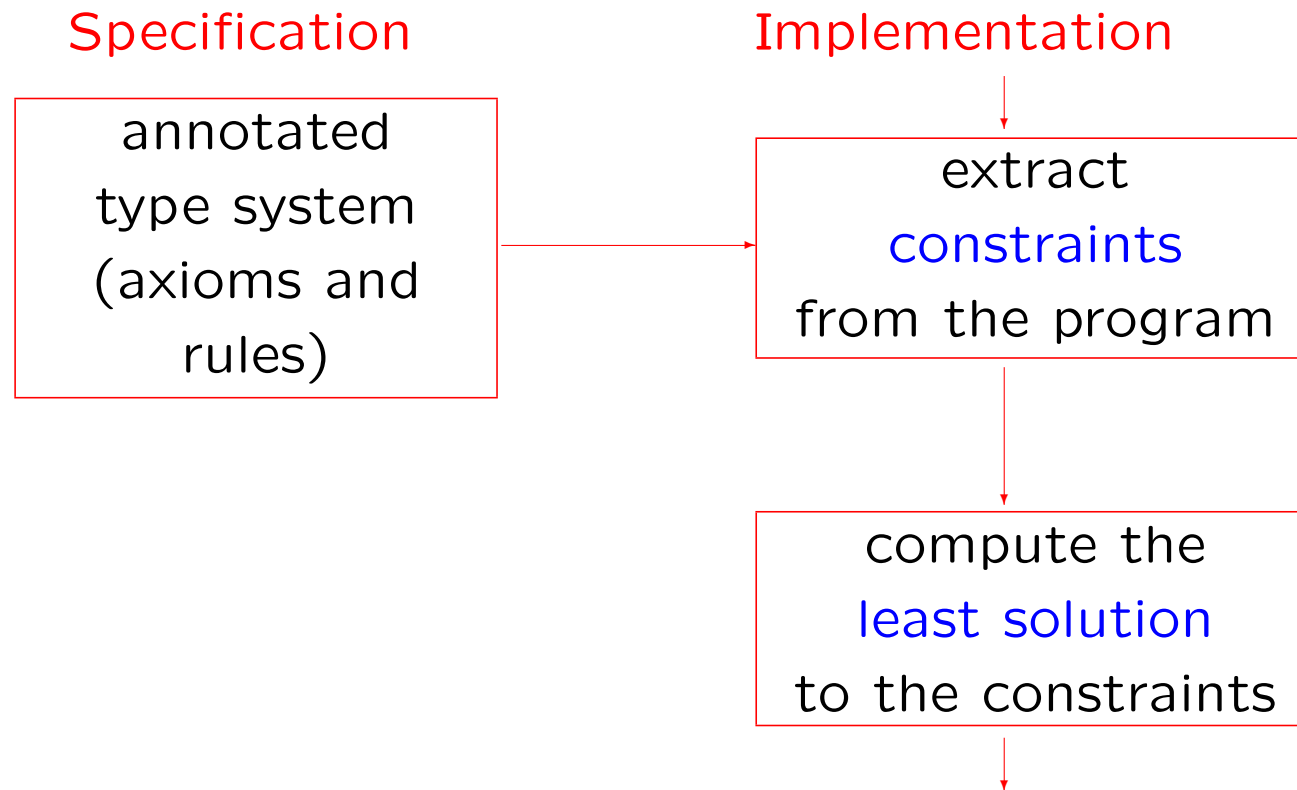
$$\vdash \text{while } [y>1]^3 \text{ do } [z:=z*y]^4; [y:=y-1]^5 \text{ od: } \Sigma \frac{\emptyset}{\{(y,5),(z,4)\}} \rightarrow \Sigma$$

⋮

$$\vdash [y:=x]^1; [z:=1]^2; \text{while } [y>1]^3 \text{ do } [z:=z*y]^4; [y:=y-1]^5 \text{ od; } [y:=0]^6:$$

$$\Sigma \frac{\{y,z\}}{\{(y,6),(z,2),(z,4)\}} \rightarrow \Sigma$$

How to automate the analysis



Type and Effect Systems

- **Technique:** Effect systems
- **Example:** Call Tracking analysis
 - idea
 - simple type system
 - effect system

The fun language

- syntax of expressions

$$e ::= x \mid \text{fn}_{\pi} x \Rightarrow e \mid e_1 e_2 \mid \dots$$

π names the function abstraction

- types

$$\tau ::= \text{int} \mid \text{bool} \mid \tau_1 \rightarrow \tau_2$$

f has type $\tau_1 \rightarrow \tau_2$ means that

- f expects a parameter of type τ_1
- f returns a value of type τ_2

Call Tracking analysis

```

let f = fnF x => x 7
    g = fnG y => y
    h = fnH z => z
in f g + f (h g)

```

```

  ↓           ↓
g 7         f g
           ↓
           g 7

```

Aim: For each function application, which function abstractions might be applied during its execution?

function applications	function abstractions that might be applied during its execution
x 7	G, H
f g	F, G
h g	H, G
f (h g)	F, H, G

Simple types

```
let f = fnF x => x 7   ← f: (int → int) → int
    g = fnG y => y     ← g: int → int
    h = fnH z => z     ← h: (int → int) → (int → int)
in f g + f (h g)      ← int
```

Simple type system

- type environment: Γ gives types to the variables (like R)
- an expression e has type τ relative to type environment Γ (like C)

$\Gamma \vdash e : \tau$

A simple type system

$\Gamma \vdash x : \tau_x$ if $\Gamma(x) = \tau_x$

$$\frac{\Gamma[x \mapsto \tau_x] \vdash e : \tau}{\Gamma \vdash \text{fn}_\pi x \Rightarrow e : \tau_x \rightarrow \tau}$$

guess: τ_x is the type of the argument x

$$\frac{\Gamma \vdash e_1 : \tau_2 \rightarrow \tau, \quad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash e_1 e_2 : \tau}$$

the type of the formal parameter
equals that of the actual parameter

Effect systems

- Call Tracking analysis:

For each function application, which function abstractions might be applied during its execution?

- Idea:

Annotate the function arrows with **sets of names** of function abstractions that **might be applied** when calling the function.

$$\tau_1 \xrightarrow{\varphi} \tau_2$$

the type of functions from τ_1 to τ_2 that might call functions with names in φ .

Example: Types and Effects

let f = fn_F x => x 7 ← f: (int $\xrightarrow{\{G\}}$ int) $\xrightarrow{\{F,G\}}$ int
g = fn_G y => y ← g: int $\xrightarrow{\{G\}}$ int
h = fn_H z => z ← h: (int $\xrightarrow{\{G\}}$ int) $\xrightarrow{\{H\}}$ (int $\xrightarrow{\{G\}}$ int)
in f g + f (h g) ← int & {F, G, H}
the effect
of executing
f g + f (g h)

The effect system

$\Gamma \vdash x : \tau_x \ \& \ \emptyset$ if $\Gamma(x) = \tau_x$

variables have no effect

$$\frac{\Gamma[x \mapsto \tau_x] \vdash e : \tau \ \& \ \varphi}{\Gamma \vdash \text{fn}_{\pi} x \Rightarrow e : \tau_x \xrightarrow{\varphi \cup \{\pi\}} \tau \ \& \ \emptyset}$$

the latent effect consists of

- that of the function body
- the function itself

the function abstraction itself has no effect

$$\frac{\Gamma \vdash e_1 : \tau_2 \xrightarrow{\varphi} \tau \ \& \ \varphi_1 \quad \Gamma \vdash e_2 : \tau_2 \ \& \ \varphi_2}{\Gamma \vdash e_1 e_2 : \tau \ \& \ \varphi_1 \cup \varphi_2 \cup \varphi}$$

the overall effect comes from

- evaluating the function
- evaluating the argument
- evaluating the function application: **the latent effect!**

The effect system

The subsumption rule:

$$\frac{\Gamma \vdash e : \tau \ \& \ \varphi}{\Gamma \vdash e : \tau \ \& \ \varphi \ \cup \ \varphi'}$$

the names of functions that **may** be applied

Example (1)

```

let f = fnF x => x 7
    g = fnG y => y
    h = fnH z => z
in f g + f (h g)

```

$$\frac{\Gamma[x \mapsto \tau_x] \vdash e : \tau \ \& \ \varphi}{\Gamma \vdash \text{fn}_{\pi} x \Rightarrow e : \tau_x \xrightarrow{\varphi \cup \{\pi\}} \tau \ \& \ \emptyset}$$

$$\frac{\Gamma \vdash e_1 : \tau_2 \xrightarrow{\varphi} \tau \ \& \ \varphi_1 \quad \Gamma \vdash e_2 : \tau_2 \ \& \ \varphi_2}{\Gamma \vdash e_1 e_2 : \tau \ \& \ \varphi_1 \cup \varphi_2 \cup \varphi}$$

$$\frac{
 \begin{array}{c}
 [x \mapsto \text{int} \xrightarrow{\{G\}} \text{int}] \vdash x : \text{int} \xrightarrow{\{G\}} \text{int} \ \& \ \emptyset \\
 [x \mapsto \text{int} \xrightarrow{\{G\}} \text{int}] \vdash 7 : \text{int} \ \& \ \emptyset
 \end{array}
 }{
 [x \mapsto \text{int} \xrightarrow{\{G\}} \text{int}] \vdash x \ 7 : \text{int} \ \& \ \{G\}
 }$$

$$\frac{
 [x \mapsto \text{int} \xrightarrow{\{G\}} \text{int}] \vdash x \ 7 : \text{int} \ \& \ \{G\}
 }{
 [] \vdash \text{fn}_F x \Rightarrow x \ 7 : (\text{int} \xrightarrow{\{G\}} \text{int}) \xrightarrow{\{F,G\}} \text{int} \ \& \ \emptyset
 }$$

Example (2)

```

let f = fnF x => x 7
    g = fnG y => y
    h = fnH z => z
in f g + f (h g)

```

$$\frac{\Gamma[x \mapsto \tau_x] \vdash e : \tau \ \& \ \varphi}{\Gamma \vdash \text{fn}_{\pi} x \Rightarrow e : \tau_x \xrightarrow{\varphi \cup \{\pi\}} \tau \ \& \ \emptyset}$$

$$\frac{\Gamma \vdash e_1 : \tau_2 \xrightarrow{\varphi} \tau \ \& \ \varphi_1 \quad \Gamma \vdash e_2 : \tau_2 \ \& \ \varphi_2}{\Gamma \vdash e_1 e_2 : \tau \ \& \ \varphi_1 \cup \varphi_2 \cup \varphi}$$

$$\frac{\Gamma \vdash h : (\text{int} \xrightarrow{\{G\}} \text{int}) \xrightarrow{\{H\}} (\text{int} \xrightarrow{\{G\}} \text{int}) \ \& \ \emptyset \quad \Gamma \vdash g : \text{int} \xrightarrow{\{G\}} \text{int} \ \& \ \emptyset}{\Gamma \vdash h g : \text{int} \xrightarrow{\{G\}} \text{int} \ \& \ \{H\}}$$

$$\frac{\Gamma \vdash f : (\text{int} \xrightarrow{\{G\}} \text{int}) \xrightarrow{\{F,G\}} \text{int} \ \& \ \emptyset \quad \Gamma \vdash h g : \text{int} \xrightarrow{\{G\}} \text{int} \ \& \ \{H\}}{\Gamma \vdash f (h g) : \text{int} \ \& \ \{F, G, H\}}$$

How to automate the analysis

