CONTROL FLOW ANALYSES

PROGRAM ANALYSIS AND OPTIMIZATION – DCC888

Fernando Magno Quintão Pereira

fernando@dcc.ufmg.br

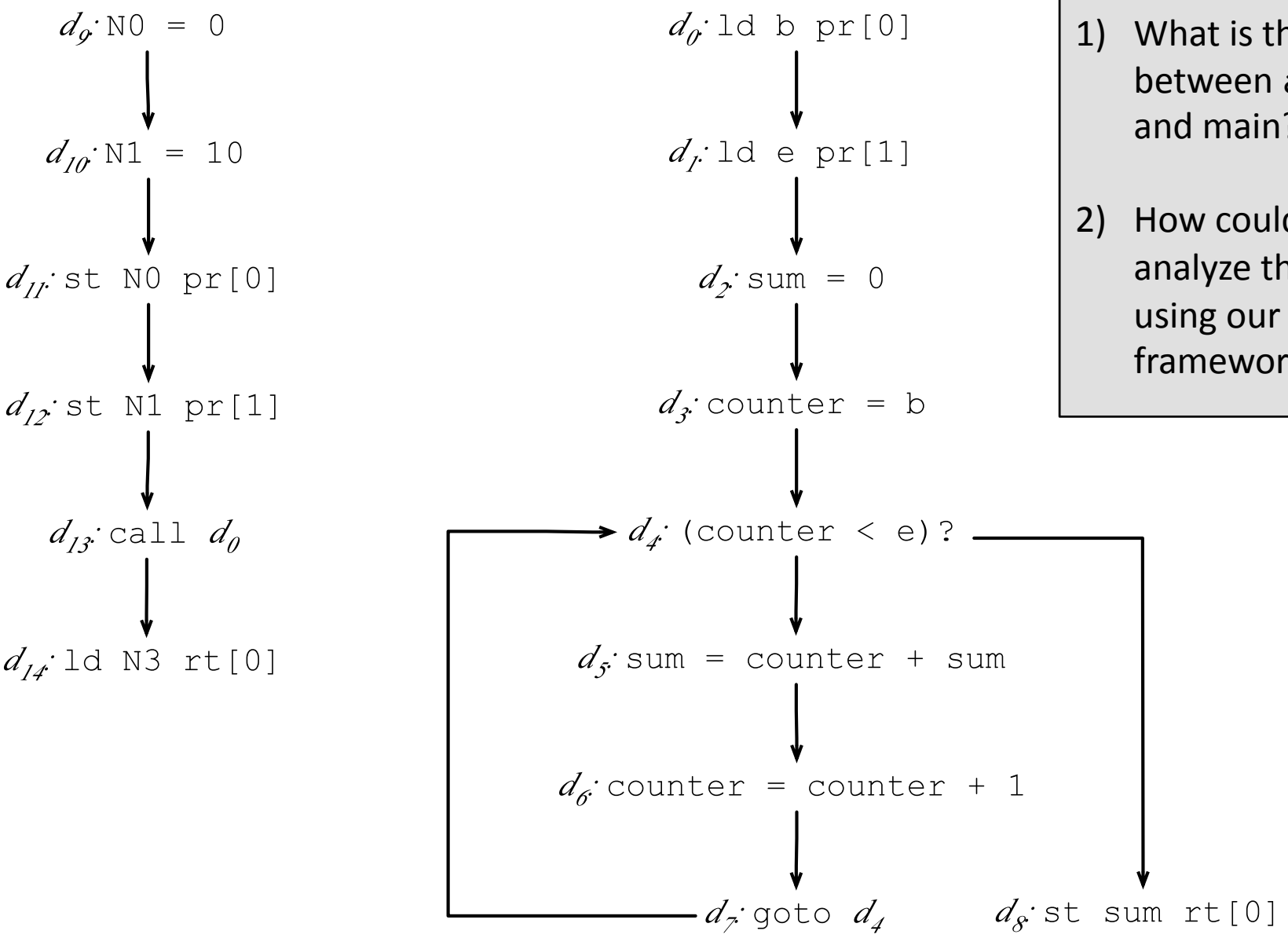
The material in these slides have been taken from Chapter 3 of the book "Principles of Program Analysis", by Nielson *et al*.

Warm up Example

```
int arith(int b, int e) {
  int sum = 0;
  int counter = b;
  while (counter < e) {
    sum += counter;
    counter++;
  }
  return sum;
}

int main() {
  int N0 = 0, N1 = 10;
  int N3 = arith(N0, N1);
}
```

1) How could we optimize this program?
2) Which knowledge would we need to optimize this program?
3) How can we obtain this knowledge?
4) How would be the CFG of this example, considering that we have two functions?

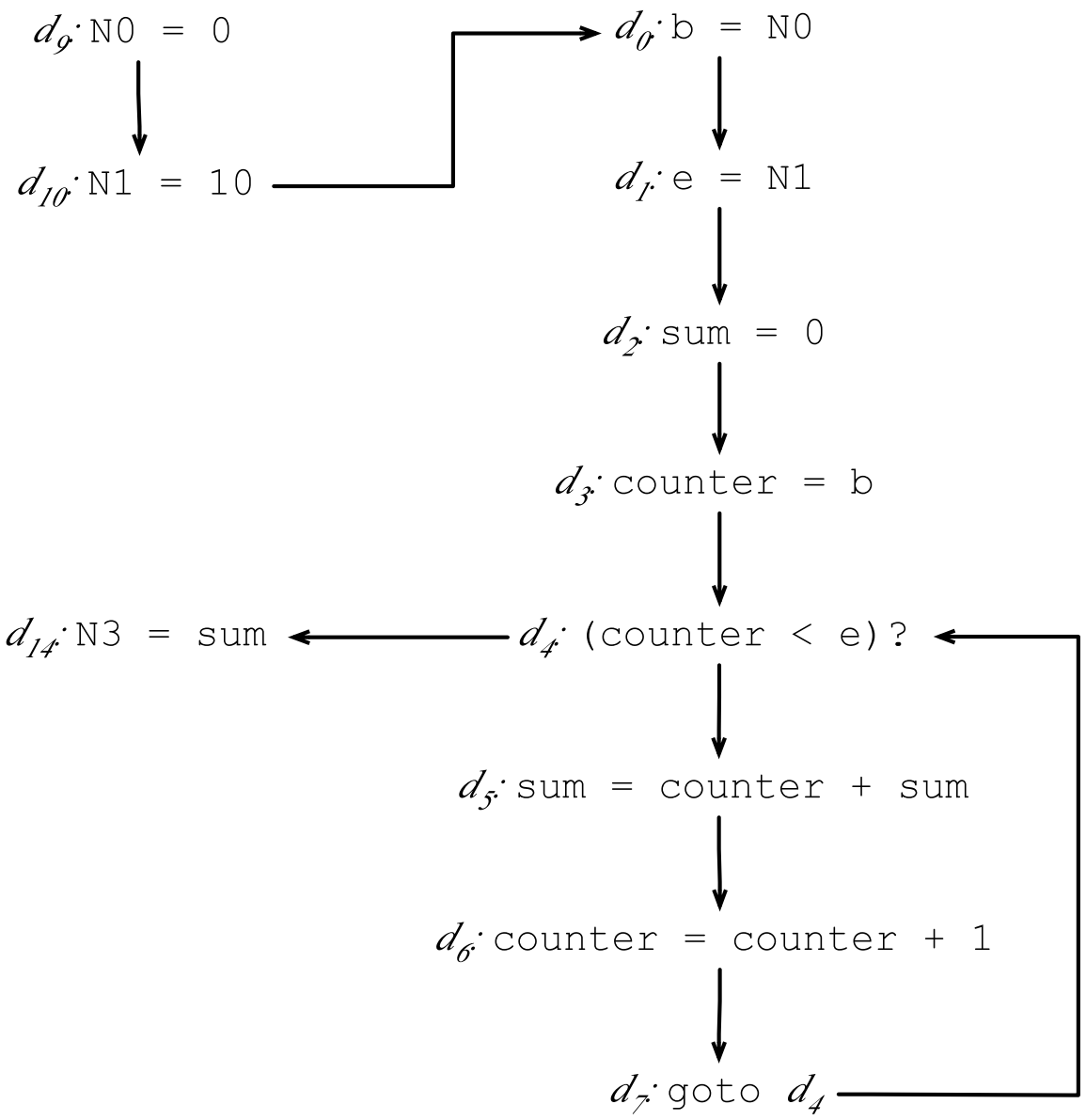
Inter-Procedural CFG



1) What is the glue between `arith` and main?

2) How could we analyze this code, using our data-flow framework?

Inter-Procedural Analysis



Once we have built the inter-procedural CFG, we can just run our ordinary data-flow analysis on it.

```
int arith(int b, int e) {
  int sum = 0;
  int counter = b;
  while (counter < e) {
    sum += counter;
    counter++;
  }
  return sum;
}

int main() {
  int N0 = 0, N1 = 10;
  int N3 = arith(N0, N1);
}
```

Making it a bit Harder

```
int arith(int b, int e) {
  int sum = 0;
  int counter = b;
  while (counter < e) {
    sum += counter;
    counter++;
  }
  return sum;
}

int (*f)(int, int) = &arith;

int main() {
  int N0 = 0, N1 = 10;
  int N3 = (*f)(N0, N1);
}
```

1) Which optimization would speed up the program on the left?

2) But this case is trickier: how can we find out the inter-procedural control flow graph of this program?

Control Flow Analysis

with regard to flow analysis in Lisp, we are faced with the following unfortunate situation:

- In order to do flow analysis, we need a control flow graph.
- In order to determine control flow graphs, we need to do flow analysis.

Oops. ‡

‡: Control Flow Analysis in Scheme, Olin Shivers, PLDI, 1988

Programs with Dynamic Control Flow

- The analyses that we have seen so far assume that the program has a well-defined shape.
  - Each instruction is succeeded by a small number of next instructions.
- Sometimes we do not know the shape of the control flow graph of a program.
  - This shortcoming is typical in programming languages with dynamic dispatch.

1) What is dynamic dispatch?

2) Can you give examples of programming languages with this capacity?

Programs with Dynamic Control Flow

- Dynamic dispatch is typical in functional and object oriented languages.
- As an example, consider the SML program below:

```
let
  val f = fn x => x 1;
  val g = fn y => y + 2;
  val h = fn z => z + 3
in
  (f g) + (f h)
end
```

1) What is the value of this program?
2) What is the control flow graph of this program?
3) Could similar difficulties surface in object oriented languages?

Programs with Dynamic Control Flow

- The program on the right suffers from similar problems as our previous example, in SML
- We cannot tell precisely, at compilation time, what is the control flow graph of this program.
  - The last method invocation, e.g., `o.add(p)` will be defined only at execution time.

```
class A:
  def __init__(self):
    self.list = []
  def add(self, n):
    self.list.append(n)
  def __str__(self):
    return str(self.list)

class B:
  def __init__(self, n):
    self.n = n
  def add(self, n):
    self.n += n
  def __str__(self):
    return str(self.n)

o = A();
p = int(raw_input("p = "))
if p % 2 == 1:
  o = B(p)
o.add(p)
print o
```

Function Inlining

- Once we know the target of a function call, we can **inline** it.
- Inlining is a whole program optimization, in which the call to a function is replaced by the function body itself.

What is the main benefit of inlining?

Function Inlining

- Once we know the target of a function call, we can **inline** it.
- Inlining is a whole program optimization, in which the call to a function is replaced by the function body itself.
- The key benefit of inlining is speed: we save the cost of passing parameters and reading return values.
- But there is another benefit: we can optimize the program more aggressively!

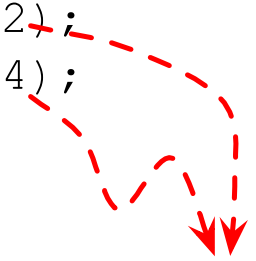
Can we do any constant propagation in the program on the right?

```
int arith(int b, int e) {
  int sum = 0;
  int counter = b;
  while (counter < e) {
    sum += counter;
    counter++;
  }
  return sum;
}

int main() {
  int N1 = 0, N2 = 10;
  int N3 = 11, N4 = 20;
  arith(N1, N2);
  arith(N3, N4);
}
```

Context

- The context of a function call is the sequence of calls that have been called before it.
- In this example, we have two contexts:
  - `main`, and `arith` at line 4
  - `main`, and `arith` at line 5
- Each context passes different information to arith; thus, we cannot assume that any parameter is constant.

```
int main() {
  int N1 = 0, N2 = 10;
  int N3 = 11, N4 = 20;
  arith(N1, N2);
  arith(N3, N4);
}

int arith(int b, int e) {
  int sum = 0;
  int counter = b;
  while (counter < e) {
    sum += counter;
    counter++;
  }
  return sum;
}
```



How can inlining fix this problem for us?

Function Inlining

Inlining creates new opportunities for optimization, because it makes them *context sensitive*. By context sensitive, we mean that the optimizer might disambiguate different calling sites.

```
int main() {
  int N1 = 0, N2 = 10;
  int N3 = 11, N4 = 20;
  int sum1 = 0, sum2 = 0;
  int counter1 = N1;
  int counter2 = N3;

  while (counter1 < N2) {
    sum1 += counter1;
    counter1++;
  }
  while (counter2 < N4) {
    sum2 += counter2;
    counter2++;
  }
}
```

```
int arith(int b, int e) {
  int sum = 0;
  int counter = b;
  while (counter < e) {
    sum += counter;
    counter++;
  }
  return sum;
}

int main() {
  int N1 = 0, N2 = 10;
  int N3 = 11, N4 = 20;
  arith(N1, N2);
  arith(N3, N4);
}
```

Which Function Can I inline?

Discovering which function we can inline is not always an easy problem, depending on the programming language.

```
fun end_point s = s ^ "."

fun inc x = x + 1

fun square y = y * y

fun map _ nil = nil
  | map f (h::t) = f h :: map f t

val l0 = [0, 1, 2]

fun imap f = map f l0

val points = map end_point ["hi", "there"]

val y = if true then Imap inc else Imap square
```

1. Which inlining can we perform in the program on the left?
2. Which information do we need to perform inlining in this kind of situation?
3. Any idea on how we can discover this kind of information?

PPGCC
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO DCC-UFMG

Fernando Magno Quintão Pereira

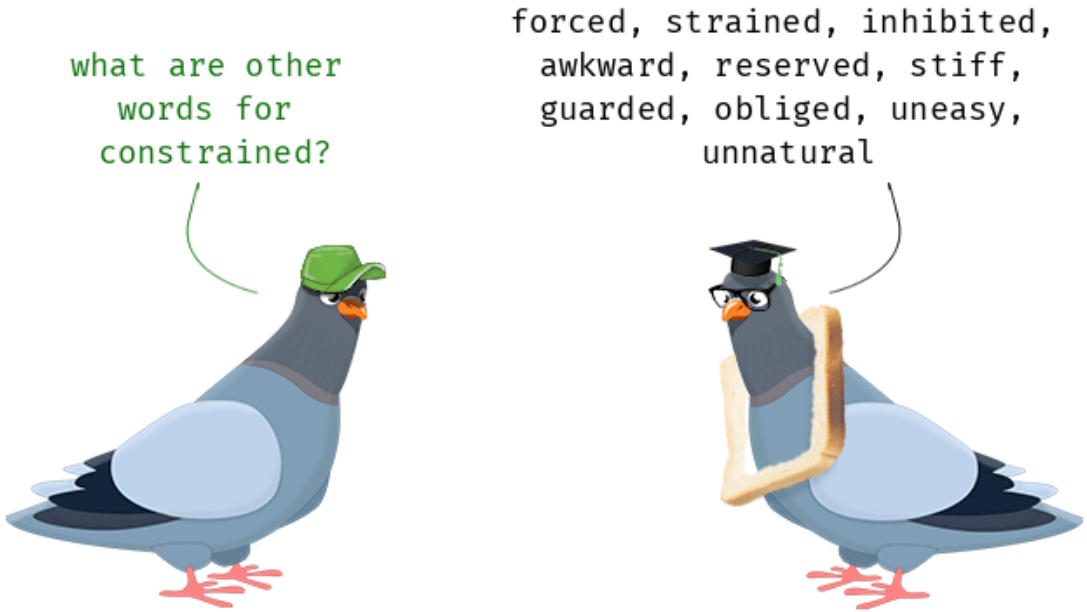
fernando@dcc.ufmg.br

lac.dcc.ufmg.br

Compilers Laboratory

Department of Computer Science
Universidade Federal de Minas Gerais
Federal University of Minas Gerais, Brazil

Constraint Based Analysis



fernando@dcc.ufmg.br

The Constraint Based Approach

- The algorithm that we shall see was originally proposed by Olin Shivers in 1988.

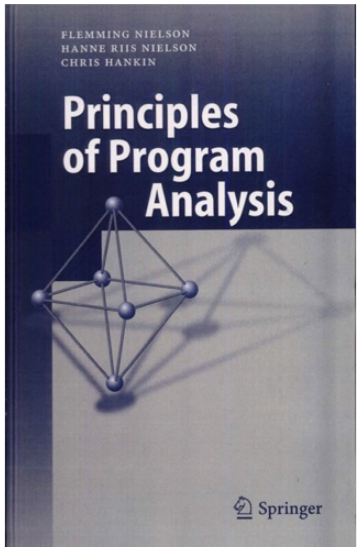
Control Flow Analysis in Scheme

Olin Shivers
Carnegie Mellon University
shivers@cs.cmu.edu

Abstract

1 The Task

- We shall adopt the presentation of Nielson *et al*.
  - See notes at the end of this presentation.



PPGCC
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO DCC-UFMG

Control Flow Analysis

- There exists a general technique to determine the control flow of programs with dynamic dispatch.
- This technique is called *control flow analysis*.
- We shall explain it in three steps:
  1. We will specify the problem formally, providing a definition of a correct solution to control flow analysis.
  2. We will extract constraints from this specification. A solution to these constraints is a solution to the formal specification of the problem.
  3. We will provide an efficient, graph-based, algorithm to solve these constraints.

The Subject Language

- We will be working with a subset of SML to illustrate the Control Flow Analysis.
  - Other languages, e.g., object oriented, would be analyzed in the same way.

$$e ::= t^{\ell}$$

$$t ::= c \mid x \mid \texttt{fn}\ x \Rightarrow e_0 \mid e_1\ e_2 \mid \texttt{if}\ e_0\ \texttt{then}\ e_1\ \texttt{else}\ e_2 \mid \texttt{let}\ x = e_1\ \texttt{in}\ e_2 \mid e_1\ \mathrm{op}\ e_2$$

Could you explain what each one of these terms is all about?

An Example in SML

$$e ::= t^{\ell}$$

$$t ::= c \mid x \mid \texttt{fn}\ x \Rightarrow e_0 \mid e_1\ e_2 \mid \texttt{if}\ e_0\ \texttt{then}\ e_1\ \texttt{else}\ e_2 \mid \texttt{let}\ x = e_1\ \texttt{in}\ e_2 \mid e_1\ \text{op}\ e_2$$

```
let
  val f = fn x => x + 1
  val g = fn y => y * y
  val max = fn a => fn b => if a > b then a else b
in
  max (f 1) (g 1)
end
```

What does this program above do?

Variables and Labels

- Our goal is to find a valid set of expressions bound to each variable and each syntactic term of a program.
  - Variables are determined by their names.
    - Let's assume that each variable name is unique!
  - Terms are determined by labels.

$$e ::= t^{\ell}$$

$$t ::= c \mid x \mid \texttt{fn}\ x \Rightarrow e_0 \mid e_1\ e_2 \mid \texttt{if}\ e_0\ \texttt{then}\ e_1\ \texttt{else}\ e_2 \mid \texttt{let}\ x = e_1\ \texttt{in}\ e_2 \mid e_1\ \text{op}\ e_2$$

- Below we have an example of input program:

$$((\texttt{fn}\ x \Rightarrow x^{1})^{2}\ (\texttt{fn}\ y \Rightarrow y^{3})^{4})^{5}$$

Which functions can be bound to variables x and y? What about the terms labeled 1, 2, 3, 4 and 5?

PPGCC
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO DCC-UFMG

Cache and Environment

- In our context, a value is the syntactical representation of a function, e.g., `fn x => x + 1`
- The *cache* is a map of labels to values.
- The *environment* is a map of variable names to values.
- A solution to the control flow analysis is a tuple (C, R), in which C is a cache, and R is an environment.

Cache and Environment

- We can accept conservative solutions, but we cannot accept incomplete solutions.

$$((\texttt{fn x => x}^1)^2\ (\texttt{fn y => y}^3)^4)^5$$

	(C_0, R_0)	(C_1, R_1)	(C_2, R_2)
1	{fn y => y^3}	{fn y => y^3}	{fn x => x^1, fn y => y^3}
2	{fn x => x^1}	{fn x => x^1}	{fn x => x^1, fn y => y^3}
3	{}	{}	{fn x => x^1, fn y => y^3}
4	{fn y => y^3}	{fn y => y^3}	{fn x => x^1, fn y => y^3}
5	{fn y => y^3}	{fn y => y^3}	{fn x => x^1, fn y => y^3}
x	{fn y => y^3}	{}	{fn x => x^1, fn y => y^3}
y	{}	{}	{fn x => x^1, fn y => y^3}

Which solution is too conservative, which solution is wrong, and which solution is just about right?

Conservative Solutions

$$((\texttt{fn x => x}^{1})^{2}\ (\texttt{fn y => y}^{3})^{4})^{5}$$

	(C_0, R_0)	(C_1, R_1)	(C_2, R_2)
1	{fn y => y^3}	{fn y => y^3}	{fn x => x^1, fn y => y^3}
2	{fn x => x^1}	{fn x => x^1}	{fn x => x^1, fn y => y^3}
3	{}	{}	{fn x => x^1, fn y => y^3}
4	{fn y => y^3}	{fn y => y^3}	{fn x => x^1, fn y => y^3}
5	{fn y => y^3}	{fn y => y^3}	{fn x => x^1, fn y => y^3}
x	{fn y => y^3}	{}	{fn x => x^1, fn y => y^3}
y	{}	{}	{fn x => x^1, fn y => y^3}

We want to know which functions *might* be bound to variables or terms. So, if we say that any function is bound to anything, we are not wrong (but this is the solution of a coward!). However, (C_1, R_1) is wrong, because clearly the parameter x can assume fn y => y. This is a false negative, which we must avoid.

The Specification of a Solution

- We say that (C, R) $\vDash e$ if the bindings in the cache C and in the environment R correctly represent the program *e*.
- We must specify correctness rules for each syntactic category in our programming language.

$$e ::= t^l$$

$$t ::= c \mid x \mid \texttt{fn}\ x \Rightarrow e_0 \mid e_1\ e_2 \mid \texttt{if}\ e_0\ \texttt{then}\ e_1\ \texttt{else}\ e_2 \mid \texttt{let}\ x = e_1\ \texttt{in}\ e_2 \mid e_1\ \mathrm{op}\ e_2$$

- Some rules are easy, e.g.:
  - (C, R) $\vDash c^l$ always
- When does (C, R) $\vDash x^l$?

PPGCC
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO DCC-UFMG

Variables and Functions

- We say that (C, R) $\vDash x^l$, if, and only if, R(x) $\subseteq$ C(l)
- When does (C, R) $\vDash$ (`fn` x => e_0) ^{l} ?

PPGCC
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO DCC-UFMG

Functions and Applications

- When does (C, R) $\models x^l$?
  - (C, R) $\models x^l$, if, and only if, R(x) $\subseteq$ C(l)

- When does (C, R) $\models$ (`fn` x => e_0)^l ?
  - (C, R) $\models$ (`fn` x => e_0)^l, if, and only if, {`fn` x => e_0} $\subseteq$ C(l)

- When does (C, R) $\models (t_1^{l1}\ t_2^{l2})^l$?

Stop and Breathe

- When does (C, R) $\vDash (t_1^{l1}\ t_2^{l2})^{l}$?
  - (C, R) $\vDash (t_1^{l1}\ t_2^{l2})^{l}$, if, and only if, (C, R) $\vDash t_1^{l1}$ and (C, R) $\vDash t_2^{l2}$ and ($\forall (\texttt{fn}\ x \Rightarrow t_0^{l0}) \in C(l_1) : (C, R) \vDash t_0^{l0}$ and $C(l_2) \subseteq R(x)$ and $C(l_0) \subseteq C(l)$)

Ok, now is when you start banging your head against the first solid thing around, decide to become a hippie, let the beard grow and give up… But not so quickly, if you look at these constraints, they **do** make some sense… If we have something like $t_1^{l1}\ t_2^{l2}$, then we must analyze each label recursively. It is also true that any $\texttt{fn}\ x \Rightarrow t_0^{l0}$ that makes its way into t_1^{l1} will receive t_2^{l2} into x. That is why we have that second part. And, by the way:

When does (C, R) $\vDash (\texttt{if}\ t_0^{l0}\ \texttt{then}\ t_1^{l1}\ \texttt{else}\ t_2^{l2})^{l}$?

Let Bindings and Binary Operators

- When does (C, R) $\models$ (`if` t_0^{l0} `then` t_1^{l1} `else` t_2^{l2}) ^{l}?
  - (C, R) $\models$ (`if` t_0^{l0} `then` t_1^{l1} `else` t_2^{l2}) ^{l}, if, and only if, (C, R) $\models$ t_0^{l0} and (C, R) $\models$ t_1^{l1} and (C, R) $\models$ t_2^{l2} and C(l_1) $\subseteq$ C(l) and C(l_2) $\subseteq$ C(l)

- When does (C, R) $\models$ (`let` x = t_1^{l1} `in` t_2^{l2}) ^{l}?
  - (C, R) $\models$ (`let` x = t_1^{l1} `in` t_2^{l2}) ^{l}, if, and only if, (C, R) $\models$ t_1^{l1} and (C, R) $\models$ t_2^{l2} and C(l_1) $\subseteq$ R(x) and C(l_2) $\subseteq$ C(l)

- When does (C, R) $\models$ (t_1^{l1} op t_2^{l2}) ^{l}?

Binary Operators and we are done!

- When does (C, R) $\vDash$ (`if` t_0^{l0} `then` t_1^{l1} `else` t_2^{l2}) ^{l}?
  - (C, R) $\vDash$ (`if` t_0^{l0} `then` t_1^{l1} `else` t_2^{l2}) ^{l}, if, and only if, (C, R) $\vDash$ t_0^{l0} and (C, R) $\vDash$ t_1^{l1} and (C, R) $\vDash$ t_2^{l2} and $C(l_1) \subseteq C(l)$ and $C(l_2) \subseteq C(l)$

- When does (C, R) $\vDash$ (`let` x = t_1^{l1} `in` t_2^{l2}) ^{l}?
  - (C, R) $\vDash$ (`let` x = t_1^{l1} `in` t_2^{l2}) ^{l}, if, and only if, ((C, R) $\vDash$ t_2^{l2} and $C(l_1) \subseteq R(x)$ and $C(l_2) \subseteq C(l)$

Why don't we have that (t_1^{l1} op t_2^{l2}) $\subseteq C(l)$ **here**?

- When does (C, R) $\vDash$ (t_1^{l1} op t_2^{l2}) ^{l}?
  - (C, R) $\vDash$ (t_1^{l1} op t_2^{l2}) ^{l}, if, and only if, (C, R) $\vDash$ t_1^{l1} and (C, R) $\vDash$ t_2^{l2}

Why all this notation?

- Notation is not bad: if we are fluent, then it is easier to read than natural language.
- A good notation may be a key success factor.
  - Today, we use Leibnitz version of the calculus, because he had a very good notation.
- In computer science, many programming languages, such as Prolog and SML, already resemble formal notation.
  - Thus, mapping from executable code to formal latex specification is not too hard.

0-CFA Analysis

- We will be working only with control flow analyses that do not distinguish between different calling context.
  - These analyses are less precise
  - But they are more efficient (much more indeed!)
- Let's illustrate this imprecision with an example:

```
(let f = (fn x => x^1)^2
in ((f^3 f^4)^5(fn y => y^6)^7)^8)^9
```

1) Which term(s) can be bound to the variable x?

2) Which term(s) can be bound to the whole thing, e.g., label 9?

0-CFA Analysis

- Our analysis will give the result below to the program:

```
(let f = (fn x => x¹)²
in ((f³ f⁴)⁵(fn y => y⁶)⁷)⁸)⁹
```

C(1) = {fn x => x¹, fn y => y⁶}

C(2) = {fn x => x¹}

C(3) = {fn x => x¹}

C(4) = {fn x => x¹}

C(5) = {fn x => x¹, fn y => y⁶}

C(6) = {fn y => y⁶}

C(7) = {fn y => y⁶}

C(8) = {fn x => x¹, fn y => y⁶}

C(9) = {fn x => x¹, fn y => y⁶}

R(f) = {fn x => x¹}

R(x) = {fn x => x¹, fn y => y⁶}

R(y) = {fn y => y⁶}

So, in the end we conclude that the entire program can be either `fn x => x` or `fn y => y`, yet, it is easy to see that this program can only be `fn y => y`

0-CFA Analysis

- We are imprecise because we are not *context sensitive*.

```
(let f = (fn x => x^1)^2
in ((f^3 f^4)^5(fn y => y^6)^7)^8)^9
```

- In other words, we cannot distinguish between the call sites of function `f` at label 3 and 4.
  - Indeed, in this program we have two different realizations of variable x, the first is only bound to fn x => x. The second is only bound to fn y => y.
- The equivalent program below would be precisely analyzed:

```
let f1 = (fn x1 => x1) in
  let f2 = (fn x2 => x2) in
    (f1 f2)(fn y => y)
```

What is the value of this program?

PPGCC
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO DCC-UFMG

Fernando Magno Quintão Pereira

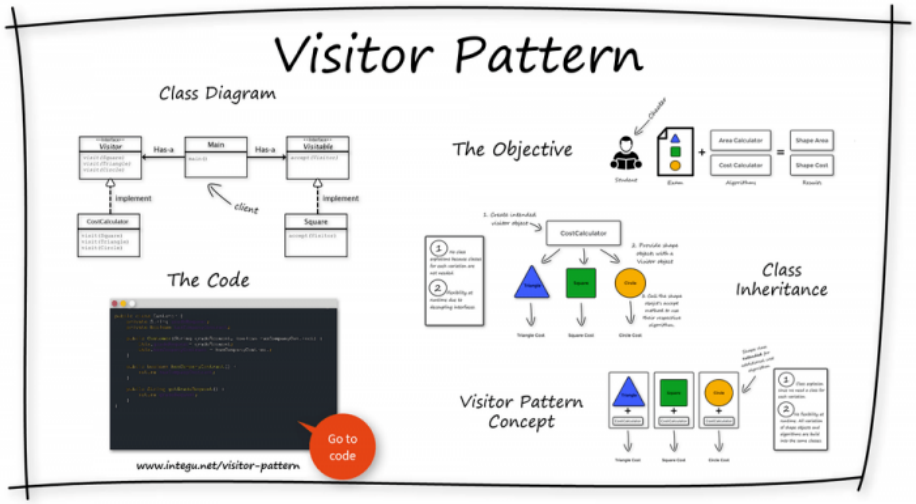
fernando@dcc.ufmg.br

lac.dcc.ufmg.br

Compilers Laboratory

Department of Computer Science
Universidade Federal de Minas Gerais
Federal University of Minas Gerais, Brazil

Constraint Generation



fernando@dcc.ufmg.br

DCC 888

Constraint Based 0-CFA Analysis

- We will extract constraints from a program.
- These constraints have one of two forms:
  - lhs $\subseteq$ rhs
  - $\{t\} \subseteq$ rhs' $\Rightarrow$ lhs $\subseteq$ rhs
- rhs can be either C(l) or R(x)
- lhs can be either C(l), R(x) or $\{t\}$
- We will see efficient algorithms to solve these constraints.
- And it is not too hard to show that they satisfy the correctness relations that we have defined before♣.
- Given a term t, we will denote the constraints that we extract from t by X[t]

♣: see "Principles of Program Analysis", by Nielson et al, section 3.4.1

Constants and Variables

- What are the constraints in X[c^l] ?
  - X[c^l] = {}

- What are the constraints in X[x^l] ?

Variables and Functions

- What are the constraints in X[c^l] ?
  - X[c^l] = {}
- What are the constraints in X[x^l] ?
  - X[x^l] = {R(x) $\subseteq$ C(l)}
- What are the constraints in X[(`fn` x => e_0)^l] ?

Functions and Applications

- What are the constraints in X[c^l] ?
  - X[c^l] = {}
- What are the constraints in X[x^l] ?
  - X[x^l] = {R(x) $\subseteq$ C(l)}
- What are the constraints in X[(`fn` x => e_0)^l] ?
  - X[(`fn` x => e_0)^l] = {{`fn` x => e_0} $\subseteq$ C(l)} $\cup$ X[e_0]
- What are the constraints in X[(t_1^{l1} t_2^{l2})^l] ?

Applications and Conditionals

- What are the constraints in X[$(t_1^{l1}\ t_2^{l2})^l$] ?
  - X[$(t_1^{l1}\ t_2^{l2})^l$] = X[t_1^{l1}] ∪ X[t_2^{l2}] ∪
    {{t} ⊆ C(l_1) ⇒ C(l_2) ⊆ R(x) | t = (`fn` x => t_0^{l0}) ∈ Prog} ∪
    {{t} ⊆ C(l_1) ⇒ C(l_0) ⊆ C(l) | t = (`fn` x => t_0^{l0}) ∈ Prog}

- What are the constraints in X[(`if` t_0^{l0} `then` t_1^{l1} `else` t_2^{l2}) ^{l}] ?

We are traversing the program, extracting constraints. Upon finding an application, we need to determine this "t". How do we find all the elements that t can be?

Ok, ok… no complains… let's all behave.

Conditionals and Let Bindings

- What are the constraints in X[$(t_1^{l1}\ t_2^{l2})^l$] ?
  - X[$(t_1^{l1}\ t_2^{l2})^l$] = X[t_1^{l1}] ∪ X[t_2^{l2}] ∪
    {{t} ⊆ C(l_1) ⇒ C(l_2) ⊆ R(x) | t = (`fn` x => t_0^{l0}) in Prog} ∪
    {{t} ⊆ C(l_1) ⇒ C(l_0) ⊆ C(l) | t = (`fn` x => t_0^{l0}) in Prog}

- What are the constraints in X[(`if` t_0^{l0} `then` t_1^{l1} `else` t_2^{l2}) ^{l}] ?
  - X[(`if` t_0^{l0} `then` t_1^{l1} `else` t_2^{l2}) ^{l}] = X[t_0^{l0}] ∪ X[t_1^{l1}] ∪ X[t_2^{l2}]
    ∪ {C(l_1) ⊆ C(l)} ∪ {C(l_2) ⊆ C(l)}

- What are the constraints in X[(`let` x = t_1^{l1} `in` t_2^{l2}) ^{l}]?

Let Bindings and Binary Operators

- What are the constraints in X[(`let` $x = t_1^{l1}$ `in` t_2^{l2}) ^{l}]?
  - X[(`let` $x = t_1^{l1}$ `in` t_2^{l2}) ^{l}] = X[t_1^{l1}] ∪ X[t_2^{l2}] ∪ {C(l_1) ⊆ R(x)} ∪ {C(l_2) ⊆ C(l)}

- What are the constraints in X[(t_1^{l1} op t_2^{l2})^{l}]?

Let Bindings and Binary Operators

- What are the constraints in X[(`let` x = t_1^{l1} `in` t_2^{l2})^{l}]?
  - X[(`let` x = t_1^{l1} `in` t_2^{l2})^{l}] = X[t_1^{l1}] ∪ X[t_2^{l2}] ∪ {C(l_1) ⊆ R(x)} ∪ {C(l_2) ⊆ C(l)}

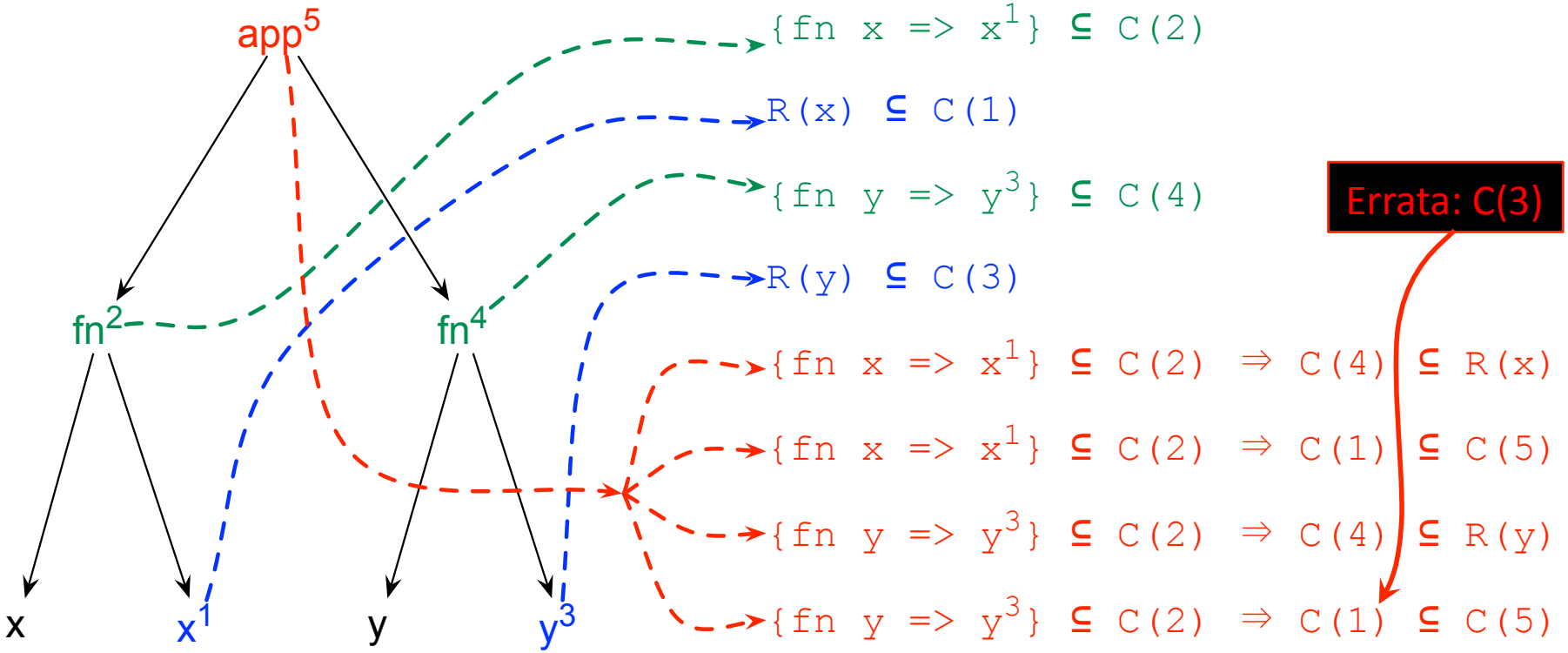
- What are the constraints in X[(t_1^{l1} op t_2^{l2})^{l}]?
  - X[(t_1^{l1} op t_2^{l2})^{l}] = X[t_1^{l1}] ∪ X[t_2^{l2}]

Which constraints do we have for
X[((`fn x => x`^{1})^{2} (`fn y => y`^{3})^{4})^{5}]?

Walking Down the Tree

- Function 'X' can be implemented as a visitor
  - It traverses the Abstract Syntax Tree of a program, producing constraints on the way. Example:

$$X[((\texttt{fn x => } x^1)^2\ (\texttt{fn y => } y^3)^4)^5]$$



Example of Constraints

$$X[((\texttt{fn x => } x^1)^2\ (\texttt{fn y => } y^3)^4)^5] =$$
$$\{$$
$$\{\texttt{fn x => } x^1\} \subseteq C(2),$$
$$R(x) \subseteq C(1),$$
$$\{\texttt{fn y => } y^3\} \subseteq C(4),$$
$$R(y) \subseteq C(3),$$
$$\{\texttt{fn x => } x^1\} \subseteq C(2) \Rightarrow C(4) \subseteq R(x),$$
$$\{\texttt{fn x => } x^1\} \subseteq C(2) \Rightarrow C(1) \subseteq C(5),$$
$$\{\texttt{fn y => } y^3\} \subseteq C(2) \Rightarrow C(4) \subseteq R(y),$$
$$\{\texttt{fn y => } y^3\} \subseteq C(2) \Rightarrow C(3) \subseteq C(5),$$
$$\}$$

Can you put bounds on the number of constraints that we can extract from a program?

On the Number of Constraints

- If we assume a program of size n, then we could have, in principle:
  - $O(n^2)$ constraints like lhs $\subseteq$ rhs
  - $O(n^4)$ constraints like $\{t\} \subseteq$ rhs' $\Rightarrow$ lhs $\subseteq$ rhs

Can you explain these bounds?

- However, if we look into the constructs that we may have in our programming language, then we can conclude that we have much less constraints.

Can you revise the bounds found above?

On the Number of Constraints

- If we assume a program of size n, then we could have, in principle:
  - $O(n^2)$ constraints like lhs $\subseteq$ rhs
  - $O(n^4)$ constraints like $\{t\} \subseteq$ rhs' $\Rightarrow$ lhs $\subseteq$ rhs

- However, if we look into the constructs that we may have in our programming language, then we can conclude that we have much less constraints.
  - Every construct, but applications, e.g, $(e_1\ e_2)$, gives us only one constraint *per se*. Applications may gives us a linear number of constraints.

In the end, how many constraints can we have, considering a program of size n?

Comparison with Data-Flow Constraints

- We had seen constraint systems before, as a part of the data-flow monotone framework:

$IN[x_1] = \{\}$

$IN[x_2] = OUT[x_1] \cup OUT[x_3]$

$IN[x_3] = OUT[x_2]$

$IN[x_4] = OUT[x_1] \cup OUT[x_5]$

$IN[x_5] = OUT[x_4]$

$IN[x_6] = OUT[x_2] \cup OUT[x_4]$

$OUT[x_1] = IN[x_1]$

$OUT[x_2] = IN[x_2]$

$OUT[x_3] = (IN[x_3] \setminus \{3,5,6\}) \cup \{3\}$

$OUT[x_4] = IN[x_4]$

$OUT[x_5] = (IN[x_5] \setminus \{3,5,6\}) \cup \{5\}$

$OUT[x_6] = (IN[x_6] \setminus \{3,5,6\}) \cup \{6\}$

Data-Flow Analysis

$\{fn\ x => x^1\} \subseteq C(2),$

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

$\{fn\ y => y^3\} \subseteq C(4),$

$R(y) \subseteq C(3),$

$\{fn\ x => x^1\} \subseteq C(2) \Rightarrow C(4) \subseteq R(x),$

$\{fn\ x => x^1\} \subseteq C(2) \Rightarrow C(1) \subseteq C(5),$

$\{fn\ y => y^3\} \subseteq C(2) \Rightarrow C(4) \subseteq R(y),$

$\{fn\ y => y^3\} \subseteq C(2) \Rightarrow C(3) \subseteq C(5)$

Control Flow Analysis

What are the main differences between these two constraint systems?

PPGCC
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO DCC-UFMG

Fernando Magno Quintão Pereira

fernando@dcc.ufmg.br

lac.dcc.ufmg.br

Compilers Laboratory

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSIDADE FEDERAL DE MINAS GERAIS
FEDERAL UNIVERSITY OF MINAS GERAIS, BRAZIL

SOLVING CONSTRAINTS

fernando@dcc.ufmg.br

DCC 888

Solving the Constraints

- We can visualize a constraint system as a *constraint graph*:
  - The constraint graph has a node C(l) for each label l, and a node R(x) for each variable x.
  - Each constraint $p_1 \subseteq p_2$ adds an initial edge (p_1, p_2) to the graph.
  - Each constraint $\{t\} \subseteq p \Rightarrow p_1 \subseteq p_2$ adds a *candidate edge* (p_1, p_2), with *trigger* $\{t\} \subseteq p$.

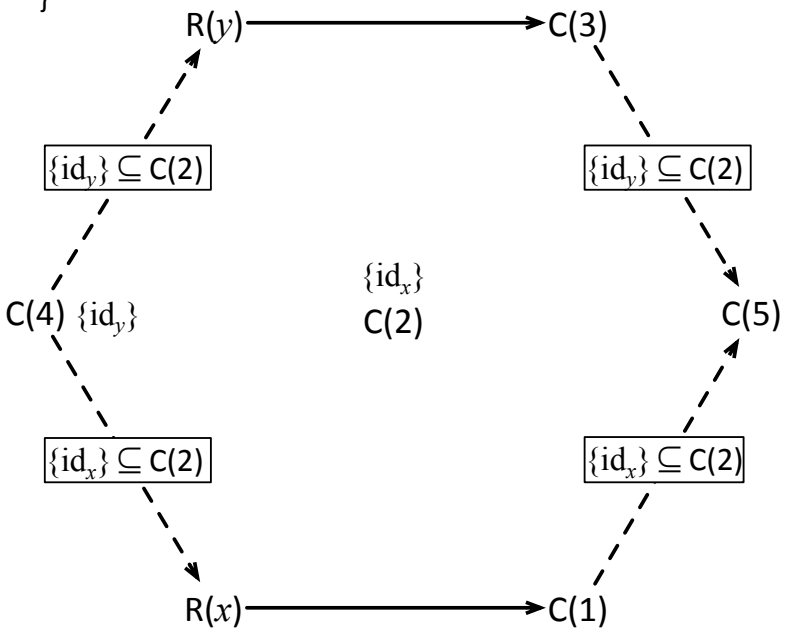
$$X[((\texttt{fn x => x}^1)^2\ (\texttt{fn y => y}^3)^4)^5] =$$
$$\{$$
$$\{\texttt{fn x => x}^1\} \subseteq C(2),$$
$$R(\texttt{x}) \subseteq C(1),$$
$$\{\texttt{fn y => y}^3\} \subseteq C(4),$$
$$R(\texttt{y}) \subseteq C(3),$$
$$\{\texttt{fn x => x}^1\} \subseteq C(2) \Rightarrow C(4) \subseteq R(\texttt{x}),$$
$$\{\texttt{fn x => x}^1\} \subseteq C(2) \Rightarrow C(1) \subseteq C(5),$$
$$\{\texttt{fn y => y}^3\} \subseteq C(2) \Rightarrow C(4) \subseteq R(\texttt{y}),$$
$$\{\texttt{fn y => y}^3\} \subseteq C(2) \Rightarrow C(3) \subseteq C(5),$$
$$\}$$

How would be the constraint graph for the system above?

Solving the Constraints with the Constraint Graph

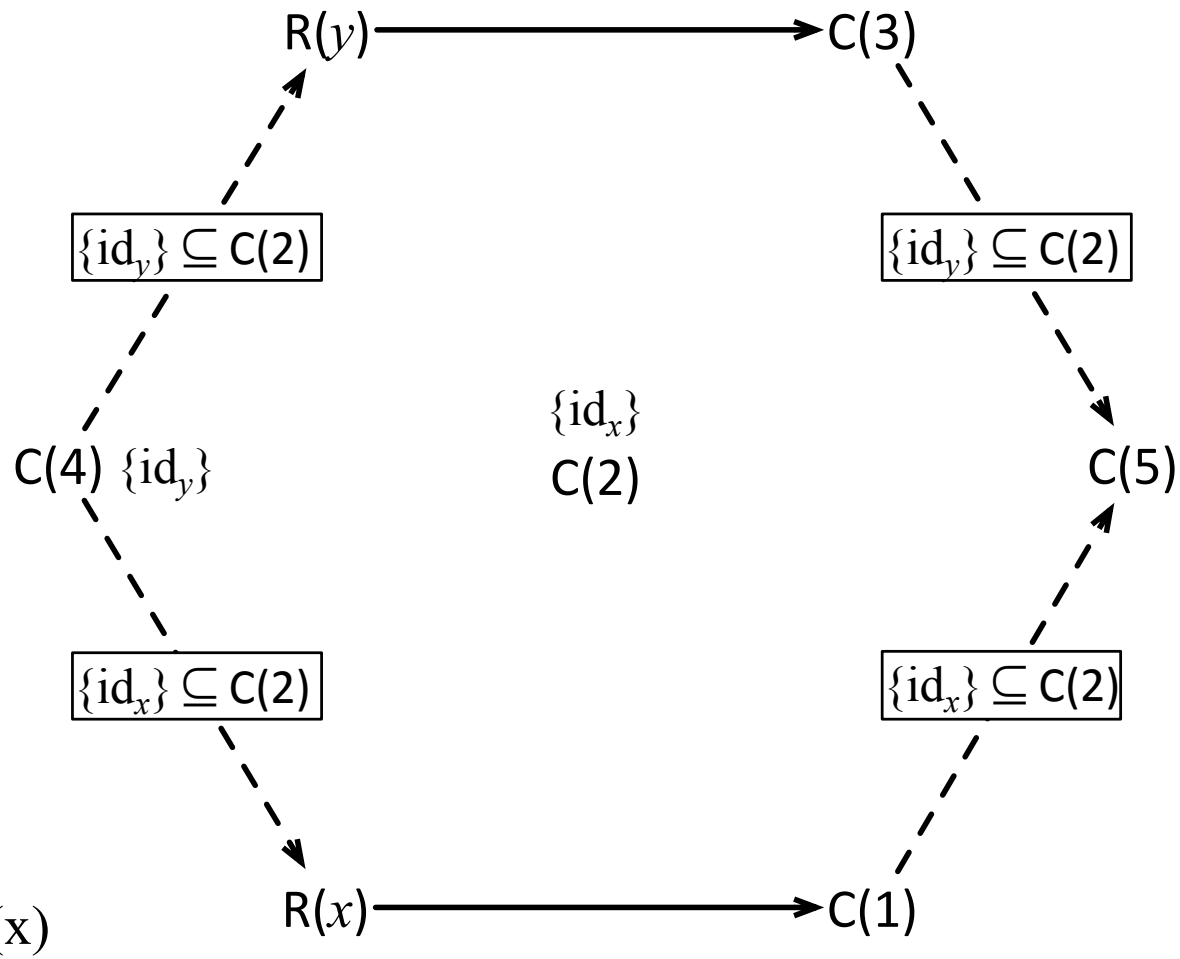
- We can visualize a constraint system as a *constraint graph*:
  - The constraint graph has a node C(l) for each label l, and a node R(x) for each variable x.
  - Each constraint $p_1 \subseteq p_2$ adds an initial edge (p_1, p_2) to the graph.
  - Candidate edges are dashed
  - Triggers are in squared boxes

$X[((\texttt{fn x => x}^1)^2\ (\texttt{fn y => y}^3)^4)^5] =$
{
  $\{\texttt{fn x => x}^1\} \subseteq C(2)$,
  $R(\texttt{x}) \subseteq C(1)$,
  $\{\texttt{fn y => y}^3\} \subseteq C(4)$,
  $R(\texttt{y}) \subseteq C(3)$,
  $\{\texttt{fn x => x}^1\} \subseteq C(2) \Rightarrow C(4) \subseteq R(\texttt{x})$,
  $\{\texttt{fn x => x}^1\} \subseteq C(2) \Rightarrow C(1) \subseteq C(5)$,
  $\{\texttt{fn y => y}^3\} \subseteq C(2) \Rightarrow C(4) \subseteq R(\texttt{y})$,
  $\{\texttt{fn y => y}^3\} \subseteq C(2) \Rightarrow C(3) \subseteq C(5)$,
}



Zooming In

Can you make sure that you understand how each constraint edge, candidate edge and trigger has been created?



$\{\text{fn x => } x^1\} \subseteq C(2)$

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

$\{\text{fn y => } y^3\} \subseteq C(4)$

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

$\{\text{fn x => } x^1\} \subseteq C(2) \Rightarrow C(4) \subseteq R(x)$

$\{\text{fn x => } x^1\} \subseteq C(2) \Rightarrow C(1) \subseteq C(5)$

$\{\text{fn y => } y^3\} \subseteq C(2) \Rightarrow C(4) \subseteq R(y)$

$\{\text{fn y => } y^3\} \subseteq C(2) \Rightarrow C(3) \subseteq C(5)$

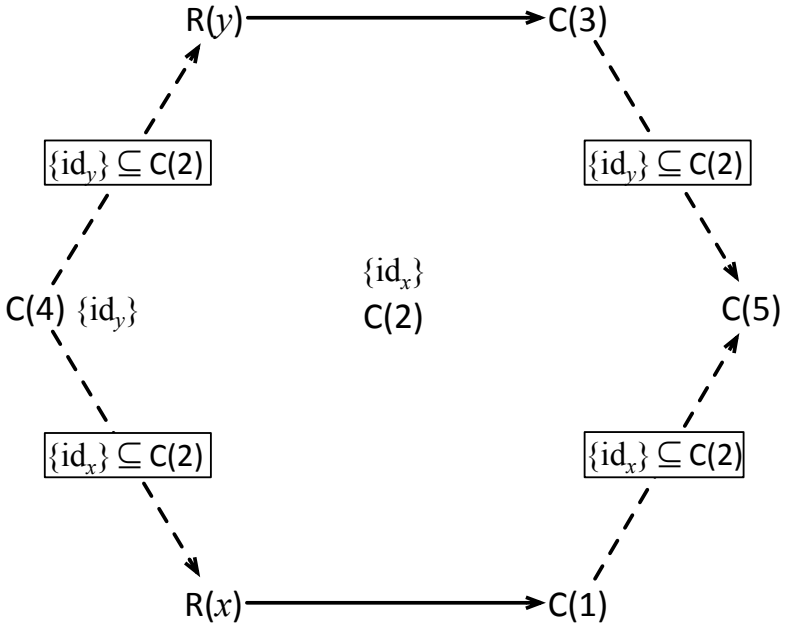
By the way: $\text{fn x => } x^1 = id_x$, and $\text{fn y => } y^3 = id_y$

Value Sets

- Each node p of the constraint graph is associated with a set D[p] of *values*.
  - Just to remember: here a value is the syntactical representation of a function, e.g., `fn x => x + 1`
- Our goal is to find the value set of every node
- Initially, we have that D[p] = $\{t \mid (\{t\} \subseteq p) \in X[e]\}$

X[((fn x => x¹)² (fn y => y³)⁴)⁵] =
{
  {fn x => x¹} ⊆ C(2),
  R(x) ⊆ C(1),
  {fn y => y³} ⊆ C(4),
  R(y) ⊆ C(3),
  {fn x => x¹} ⊆ C(2) ⇒ C(4) ⊆ R(x),
  {fn x => x¹} ⊆ C(2) ⇒ C(1) ⊆ C(5),
  {fn y => y³} ⊆ C(2) ⇒ C(4) ⊆ R(y),
  {fn y => y³} ⊆ C(2) ⇒ C(3) ⊆ C(5),
}

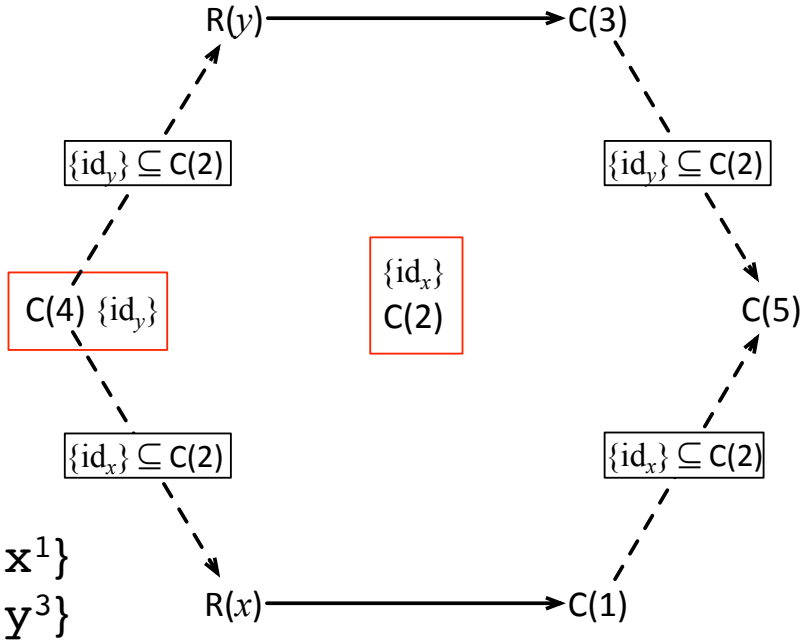
What are the initial value sets in our example?



Value Sets

- Each node p of the constraint graph is associated with a set D[p] of *values*.
  - Just to remember: here a value is the syntactical representation of a function, e.g., `fn x => x + 1`
- Our goal is to find the value set of every node
- Initially, we have that D[p] = $\{t \mid (\{t\} \subseteq p) \in X[e]\}$

X[((fn x => x^1)^2 (fn y => y^3)^4)^5] =
{
  {fn x => x^1} ⊆ C(2),
  R(x) ⊆ C(1),
  {fn y => y^3} ⊆ C(4),
  R(y) ⊆ C(3),
  {fn x => x^1} ⊆ C(2) ⇒ C(4) ⊆ R(x),
  {fn x => x^1} ⊆ C(2) ⇒ C(1) ⊆ C(5),
  {fn y => y^3} ⊆ C(2) ⇒ C(4) ⊆ R(y),
  {fn y => y^3} ⊆ C(2) ⇒ C(1) ⊆ C(5),
}



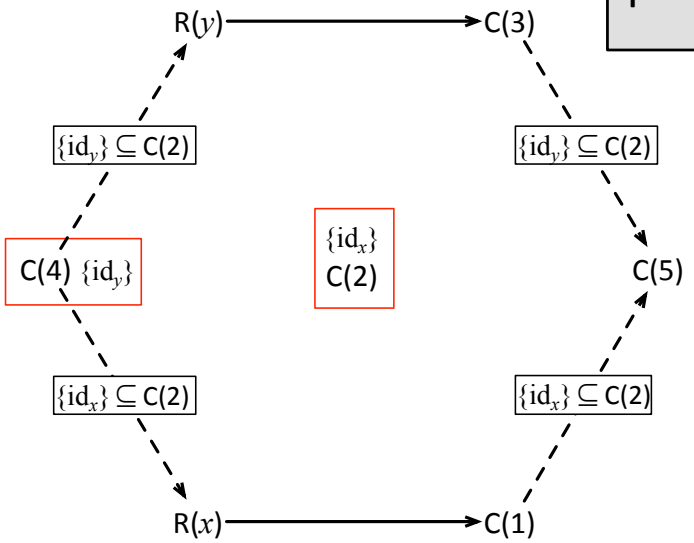
Again:
id_x = {fn x => x^1}
id_y = {fn y => y^3}

The Sliding Behavior

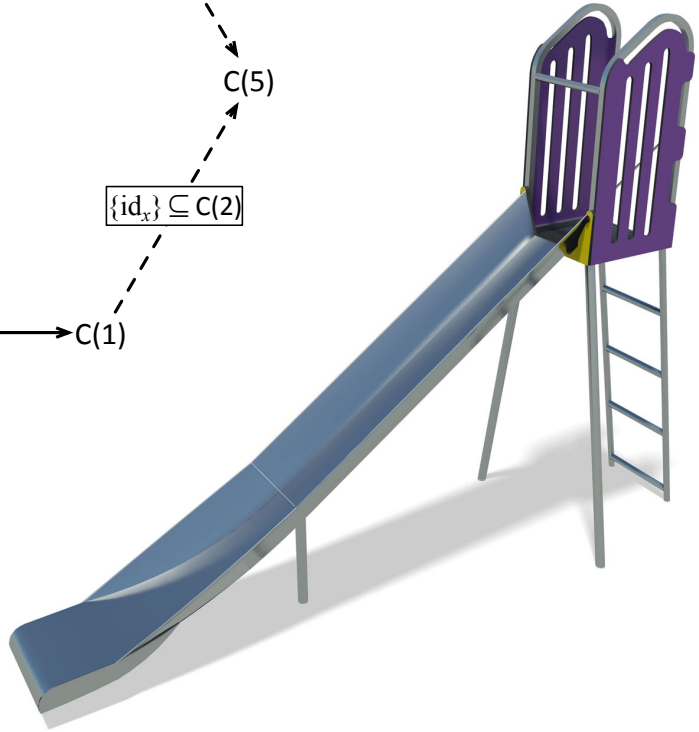
- If we have an edge (p_1, p_2) in our graph, then the value set of p_1 must flow into the value set of p_2.

Why is **this** property true?

$X[((\texttt{fn x => x}^1)^2\ (\texttt{fn y => y}^3)^4)^5] =$
$\{\texttt{fn x => x}^1\} \subseteq C(2)$,
$R(x) \subseteq C(1)$,
$\{\texttt{fn y => y}^3\} \subseteq C(4)$,
$R(y) \subseteq C(3)$,
$\{\texttt{fn x => x}^1\} \subseteq C(2) \Rightarrow C(4) \subseteq R(x)$,
$\{\texttt{fn x => x}^1\} \subseteq C(2) \Rightarrow C(1) \subseteq C(5)$,
$\{\texttt{fn y => y}^3\} \subseteq C(2) \Rightarrow C(4) \subseteq R(y)$,
$\{\texttt{fn y => y}^3\} \subseteq C(2) \Rightarrow C(3) \subseteq C(5)$,



We can think about these solid edges as an "slide". If our constraint graph contains an edge (i, j), then every value stored in i slides into j.

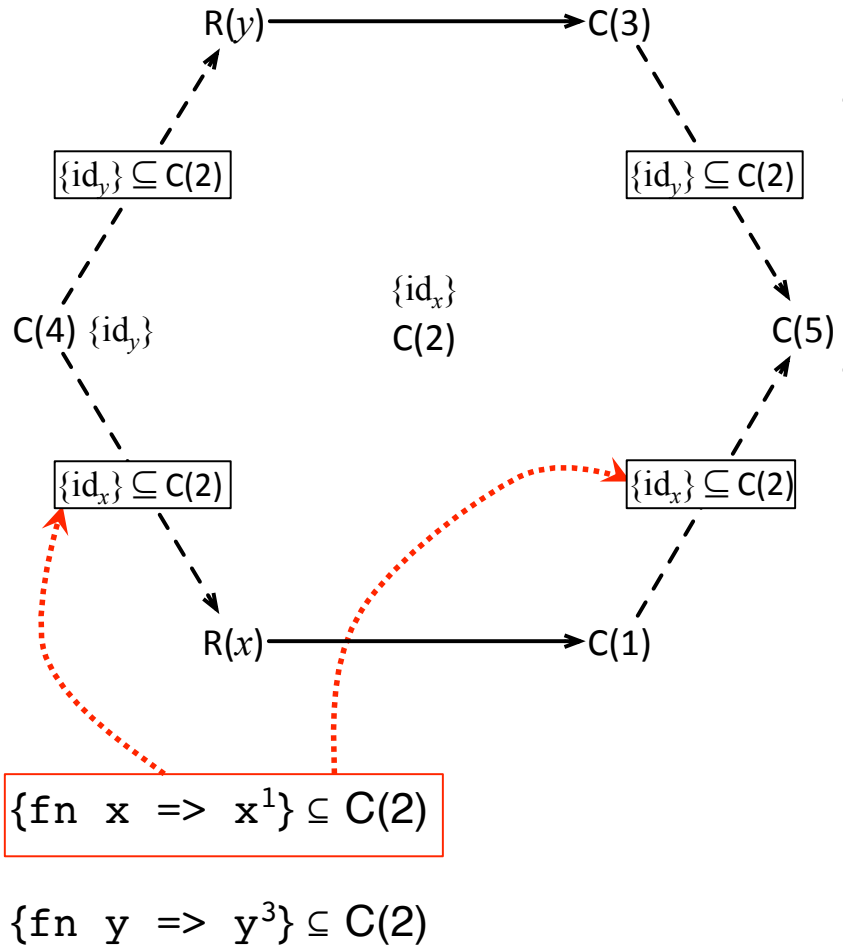


The Iterative Propagation Phase

- If we have an edge (p_1, p_2) in our graph, then the value set of p_1 must flow into the value set of p_2.

- We iterate over the triggers, until no change is observed. Each iteration checks the following property:
  - If a trigger T is true, then we transform every candidate edge that is guarded by T into a true edge.
  - If an edge (p_1, p_2) is transformed, then we propagate the value set of p_1 to all the other nodes that are reached through this new edge.

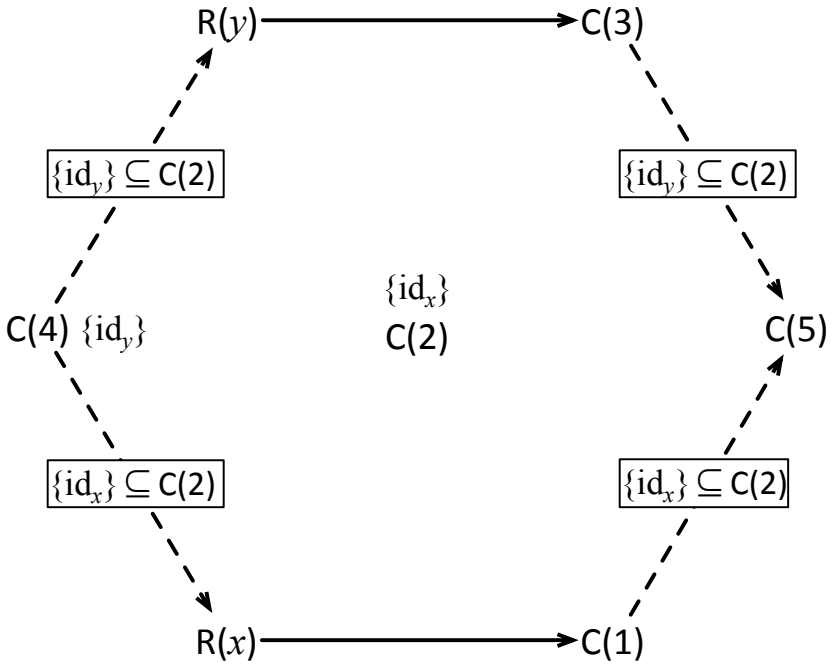
The Evaluation of a Trigger



- If we have an edge (p_1, p_2) in our graph, then the value set of p_1 must flow into the value set of p_2.
- We iterate over the triggers, until no change is observed. Each iteration checks the following property:
  - If a trigger T is true, then we transform every candidate edge that is guarded by T into a true edge.
  - If an edge (p_1, p_2) is transformed, then we propagate the value set of p_1 to all the other nodes that are reached through this new edge.

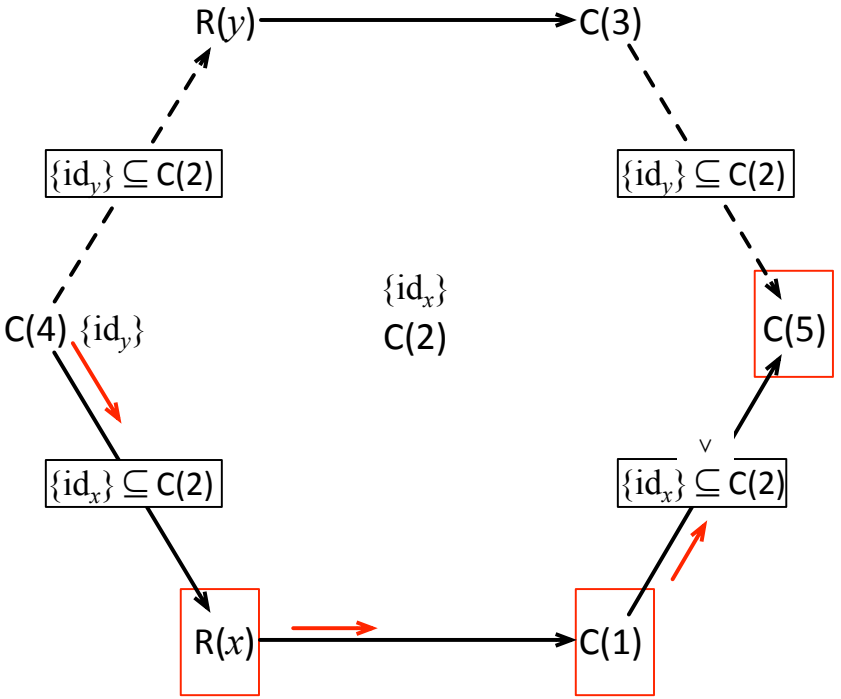
What is the result of evaluating this trigger?

The Evaluation of a Trigger



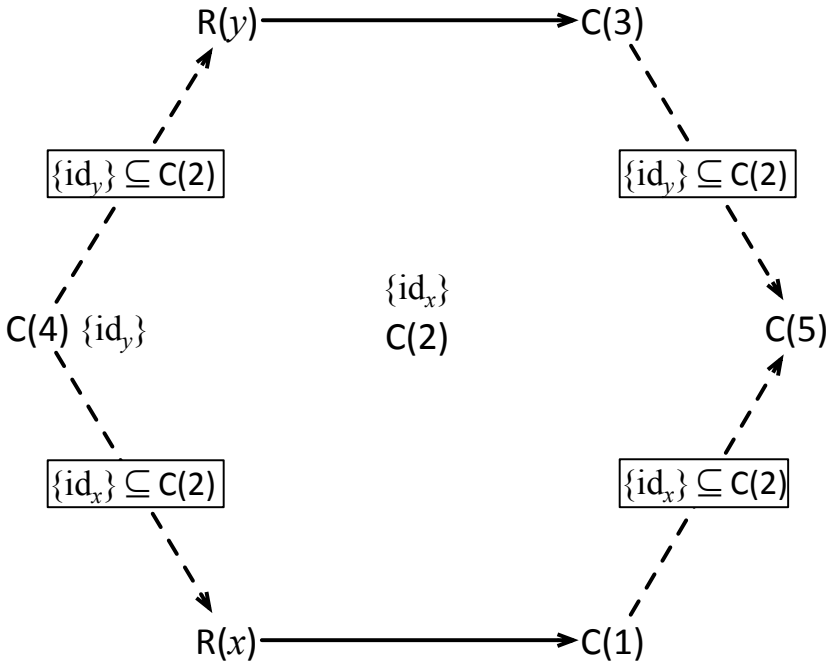
{fn x => x^1} ⊆ C(2)

{fn y => y^3} ⊆ C(2)



What is the result of evaluating the other trigger?

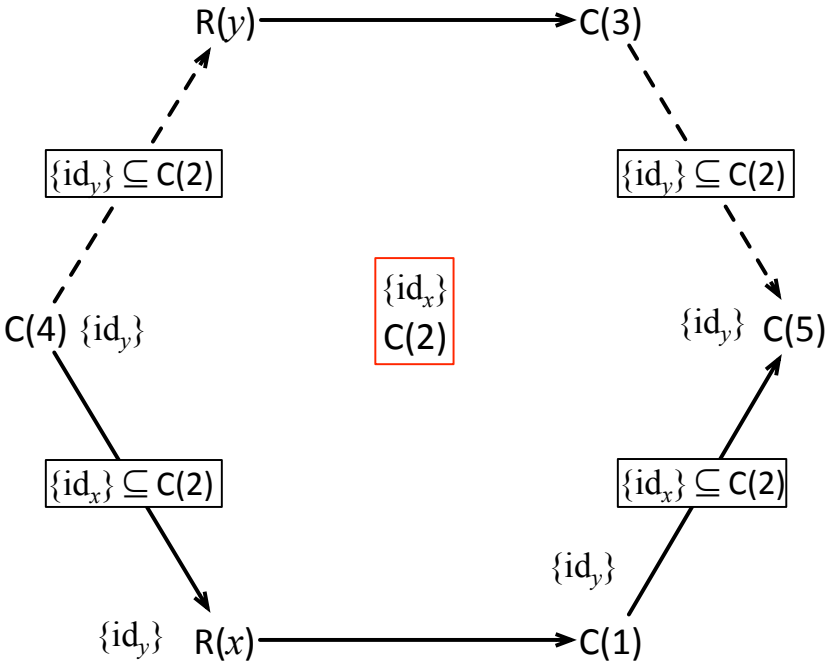
The Evaluation of a Trigger



This time, the trigger evaluates to false, and nothing happened. A second evaluation of $\{id_x\} \subseteq C(2)$ also results in false, and we are done.

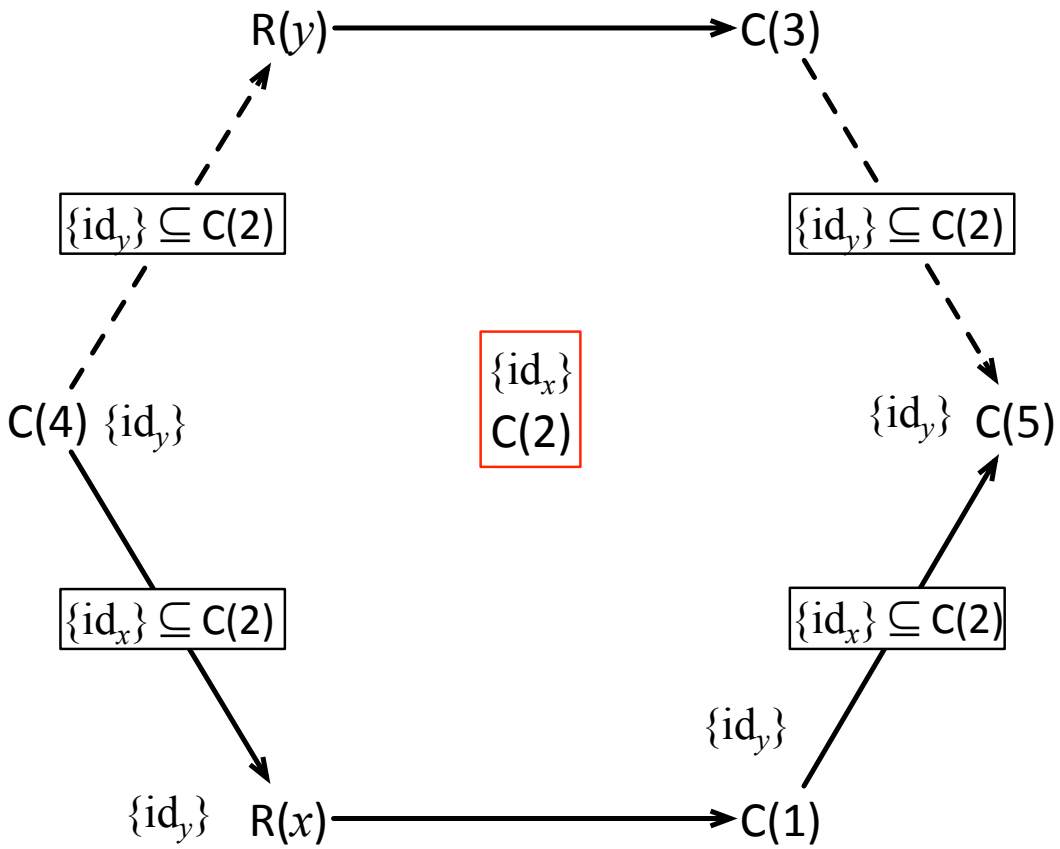
$\{\texttt{fn x => x}^1\} \subseteq C(2)$

$\{\texttt{fn y => y}^3\} \subseteq C(2)$



So, what is the final solution to the analysis?

The Final Result



	(C_0, R_0)
1	{fn y => y^3}
2	{fn x => x^3}
3	{}
4	{fn y => y^3}
5	{fn y => y^3}
x	{fn y => y^3}
y	{}

What is the complexity of this algorithm?

The value set of each node gives us the solution to the constraint based analysis. This solution is valid, and is relatively precise.

Complexity Bounds♠

- We may have to evaluate the triggers at most $O(n^2)$ times, because we may have $O(n^2)$ candidate edges.
- Each evaluation implies inspecting $O(n)$ triggers, as this is the number of triggers we have.
- Each trigger may cause the updating of up to $O(n)$ value sets, because we have one value set associated with each node.
- Each update is an $O(n)$ operation, assuming that we can implement the union of two sets in $O(n)$ time.

So, what is the final complexity bound?

♠: although our simple algorithm has high complexity, it is possible to shrink this complexity down to $O(n^3)$, using a worklist implementation. For a full description of this more efficient implementation, see "Principles of Program Analysis", pp 181

PPGCC
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO DCC-UFMG

Fernando Magno Quintão Pereira

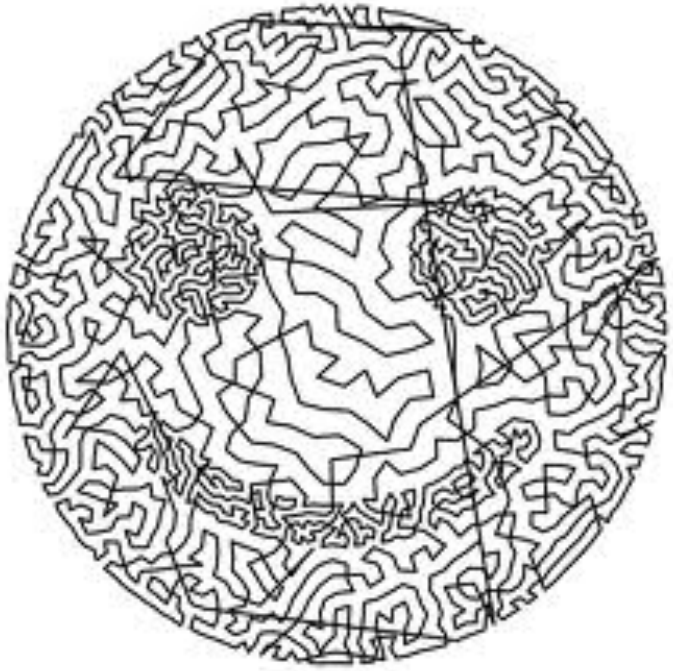
fernando@dcc.ufmg.br

lac.dcc.ufmg.br

Compilers Laboratory

Department of Computer Science
Universidade Federal de Minas Gerais
Federal University of Minas Gerais, Brazil

Heuristics on the Play



DCC 888

PPGCC
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO DCC-UFMG

Improving Runtime with Types

- So far we have assumed that each function may be assigned to each variable.
- In statically typed programming languages, this is usually not the case.
- We can use the type system to restrict the values-sets of each node, so that $\{t\} \subseteq \mathrm{R}(x)$ only if t has the same static type as x.

Improving Runtime with Types

```
val v0 = fn x => x + 1

val v1 = fn x => x * x

val v2 = fn x => x ^ "."

val v3 = fn x => hd (explode x)

fun apply n f = (fn x => x + 1) (f n)

val v4 = apply 1

val v5 = apply "oi"
```

1) What is the type of v4?
2) What is the type of v5?
3) Which functions, e.g., v0, v1, v2 or v3, can we pass to v4?
4) Which functions can we pass to v5?

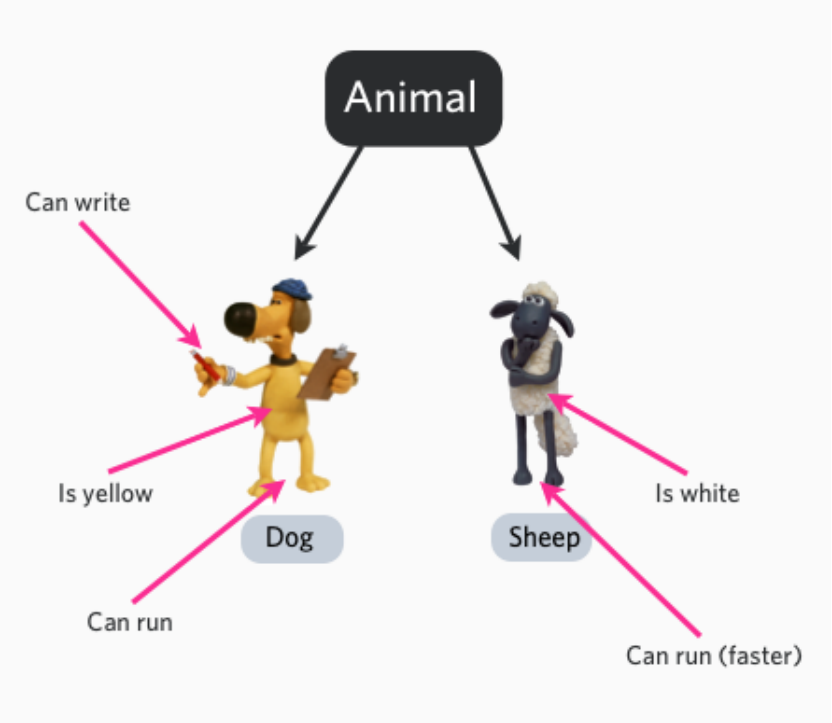
This approach is the key idea behind a well-known static analysis called *Type Hierarchy Analysis*, which has been used with great profit in object oriented programming languages.

Object Oriented Languages

- In the context of an object oriented programming language we want to know which implementations might be the target of a method call, e.g., o.m(…):

1) How is this problem similar to control flow analysis?

2) In case we do not want to run that expensive algorithm to find the possible implementations of m(), which cheaper strategy could we use?



Syntax Directed Solution

- The simplest solution to the problem is to scan the entire application's source code, finding all the methods that are named m in the code.
- We can improve this:
  - Use the method's signature, e.g., number of arguments, and argument types.

What are the possible targets of a.setX(1)? What about b.setX(2)? And b.setX(2, 3)?

```
class A {
  private int x;
  public void setX(int y) { x = y; }
}

class B {
  private int x;
  private int getX() { return x; }
  public void setX() { x = 0; }
  public void setX(int y) { x = y; }
  public void setX(int y, int z) { x = y + z; }
}

public class T {
  public static void main(String args[]) {
    A a = new A();
    a.setX(1);
    B b = new B();
    b.setX(2);
    b.setX(2, 3);
  }
}
```

How to improve this even more?

Class Hierarchy Analysis♤

- We are looking for the target of the call o.m(...). We can already scan the program looking for the methods with the same signature as m(...).
- We can restrict our syntactic search to only those methods that belong into classes that could instantiate the object o.

Any how can we improve even a bit further the precision of this analysis, without running any expensive algorithm?

```
class Animal {
  public void eat() {
    System.out.println(this);
  }
  public String toString () {
    return "Animal";
  }
}
class Mammal extends Animal {
  public void suckMilk() {
    System.out.println(this);
  }
  public String toString () {
    return "Mammal";
  }
  public void eat() {
    System.out.println(this);
  }
}
class Dog extends Mammal {
  public void bark() {
    System.out.println(this);
  }
  public String toString () {
    return "Dog";
  }
  public void eat() {
    System.out.println(this);
  }
}
```

♤: Optimization of Object-Oriented Programs Using Static Class (1995)

Rapid Type Analysis♧

- We want to find the target of o.m(…). We already restrict our quest to:
  - Methods that have the same signature as m(…)
  - Methods declared in the class hierarchy of o
- And we can restrict our search further, to only those classes that have static instantiation calls in the program, e.g., "new C(…)", where C is a subclass of o.

```
public void test () {
  Animal a1 = new Animal();
  Animal a2 = new Mammal();
  Animal a3 = new Dog();
  System.out.println(a1);
  System.out.println(a2);
  System.out.println(a3);
  a1.eat();
  a2.eat();
  Dog d1 = new Dog();
  Mammal m1 = d1;
  d1.bark();
  m1.suckMilk();
  d1.suckMilk();
  Dog d2 = (Dog)a3;
  d2.bark();
  Dog d3 = (Dog)a2;
}
```

♧: Fast Static Analysis of C++ Virtual Function Calls (1996)

A Bit of History

- The control flow analysis based on constraints was first proposed by Olin Shivers, in a well-known PLDI paper.
- Class Hierarchy Analysis is an idea due mostly to Dean *et al*., and was published in ECOOP, in 1995

- Shivers, O, "Control-Flow Analysis in Scheme", PLDI (1988), pp. 164-174
- Dean, J. and Grove, D. and Chambers, C. "Optimization of Object-Oriented Programs Using Static Class Hierarchy Analysis", ECOOP (1995) pp. 77-101