CS6848 - Principles of Programming Languages

Flow analysis

V. Krishna Nandivada

IIT Madras

Recap

- Idea of CPS
- Step by step approach to convert scheme to cps.
- Algorithm to convert Scheme programs to Tail form.
- Algorithm to convert programs in tail form to first-order form.
- Algorithm to convert programs in first-order form to imperative form.

What you should be able to answer (necessary not sufficient)

- Given a scheme program convert it to imperative form.

Flow analysis - What, Why and How

What

- Tell what "flows" into a variable/expression. (instances - values, type of values, properties of values ...)
- One instance of flow analysis information - finite set of classes.
- Say the flow set of an expression e is {A, B, C}.
- $\Longrightarrow$ e can evaluate to `null` or an instance of a class mentioned in {A, B, C}.

Utility of such flow information:

- We can inline a message send (method call), if the flow set for the receiver is a singleton set.
- If the method of a class is not called at all, then we can discard thus "dead code".

How

- To compute flow sets for each expression, we will do flow analysis.

Our focus

- We will study flow analysis that will help in inlining.
- Our assumptions:
  - closed-world assumption:
    All parts of the program are known at the time of the analysis and will not change.
  - open-world assumption:
    Some parts of the program are not known or may change. Recall the principal type inference.

Method inlining example

- Method inlining is a popular optimization in OO languages: Java, C++. (Why?)

```
class A{
  void m (Q arg) {
    arg.p();
  }
}
class B extends A{
  void m (Q arg)
    { ... }
}
```

```
class Q {
  void p()
    { ... }
}
class S extends Q {
  void p()
    { ... }
}
```

```
A x = new A();
B y = new B();

x.m(new Q());
y.m(new S());
```

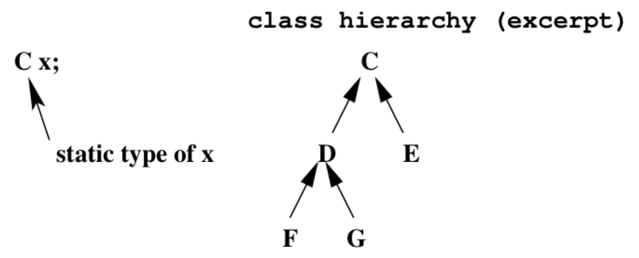
- Flow sets for `x`: `A`
- Flow sets for `y`: `B` – can also be inlined
- What if there is some code in between?

Flow analysis

- **Goal**: Find the call sites for each caller; unique callees are of interest.
- We will use a set based analysis.
  - The flow set for an expression is a set of class names.
  - Say the <u>flow set</u> of an expression e is {A, B, C}.
  - $\Longrightarrow$ e can evaluate to `null` or an instance of a class mentioned in {A, B, C}.
- Note: Any flow analysis must be an approximation.
- Tradeoff - precision and speed.
- CHA $\Longrightarrow$ 0-CFA
  $\Longrightarrow$ improved precision and cost
- CHA - Class hierarchy analysis, CFA - Control Flow Analysis

CHA - Class hierarchy analysis

- Relies on the type system; and hence the type information.
- It is a type based analysis.
- Flow set of any expression e:
  - Say the static type of the expression e is A.
  - Flow set = all subtypes of the static type.
  - Example:



Flow set for x = { C, D, E, F, G }

Revisit the example with CHA

```
class A{
  void m (Q arg) {
    arg.p();
  }
}
class B extends A{
  void m (Q arg)
    { ... }
}
```

```
class Q {
  void p()
    { ... }
}
class S extends Q {
  void p()
    { ... }
}
```

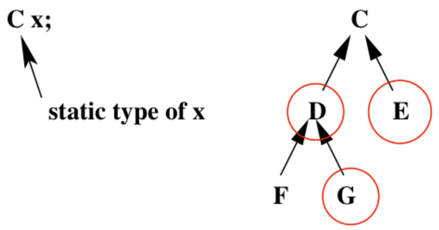
```
A x = new A();
B y = new B();

//FS(x)={A, B}
x.m(new Q());
//FS(y)={B}
y.m(new S());
```

0-CFA

- Flow insensitive, context insensitive flow analysis.
- Can be done $O(n^3)$ time.
- Does not rely on type system, but is type preserving.
- Example: (Say `x` gets values of type `D, E,` and G.

class hierarchy (excerpt)



Flow set for x = { D, E, G }

0-CFA process

1. Generate constraints.
2. Solve constraints.

- For each expression e, there is a flow variable $[\![e]\!]$.
- Example:

Program	Constraints
`new C()`	`C` $\in [\![$ `new C()` $]\!]$
`x = e;`	$[\![x]\!] \supseteq [\![e]\!]$
`e1.m (e2);` `and` `class C{` `B m(A a) {` `...` `return e; } }`	`C` $\in [\![e1]\!] \Rightarrow [\![e2]\!] \subseteq [\![a]\!]$ `C` $\in [\![e1]\!] \Rightarrow [\![e]\!] \subseteq [\![e1.m(e2)]\!]$

0-CFA Constraint generation

- Assume that all program variable and argument names are distinct (rename otherwise).
- We will use the notation $[\![$`this`$-$`C`$]\!]$ for the flow variable for the "this" in the class `C`.
- Generate constraints based on the syntax.
- We are looking at constraints in three forms:
  - $c \in X$ Beginning
  - $X \subseteq Y$ Propagation
  - $(c \in X) \Rightarrow (Y \subseteq Z)$ conditional
- A unique minimal solution is guaranteed.

Constraint generation (contd.)

- "`ID = EXP`" (assignment)
  $[\![\mathtt{EXP}]\!] \subseteq [\![\mathtt{ID}]\!]$
- "`this`". Say `this` occurs in a method in class `C`:
  $\mathtt{C} \in [\![\mathtt{this} - \mathtt{C}]\!]$
- "`new C`" (object creation)
  $\mathtt{C} \in [\![\mathtt{new\ C()}]\!]$
- "`EXP.METH (EXP1, ..., EXPn)`" (message send)
  Say `C` implements a method for the message `METH`:

```
retType METH (type1 ID1, ... type_n IDn){
   return EXP0 }
```

Generate constraints:

$$\mathtt{C} \in [\![\mathtt{EXP}]\!] \Rightarrow \begin{cases} [\![EXP1]\!] \subseteq [\![ID1]\!] \\ \ldots \\ [\![EXPn]\!] \subseteq [\![IDn]\!] \\ [\![EXP0]\!] \subseteq [\![EXP.METH(EXP1, \cdots EXPn)]\!] \end{cases}$$

Running on the example:

```
class A{
  void m (Q arg) {
    arg.p();
  }
}
class B extends A{
  void m (Q arg)
    { ... }
}
```

```
class Q {
  void p()
    { ... }
}
class S extends Q {
  void p()
    { ... }
}
```

```
A x = new A();
B y = new B();

x.m(new Q());
y.m(new S())
```

Generate constraints

Starting	Propagation	Conditional
$\mathtt{A} \in [\![\mathtt{new\ A()}]\!]$	$[\![\mathtt{new\ A()}]\!] \subseteq [\![\mathtt{x}]\!]$	$\mathtt{A} \in [\![\mathtt{x}]\!] \Rightarrow [\![\mathtt{new\ Q()}]\!] \subseteq [\![\mathtt{A.arg}]\!]$
$\mathtt{B} \in [\![\mathtt{new\ B()}]\!]$	$[\![\mathtt{new\ B()}]\!] \subseteq [\![\mathtt{y}]\!]$	$\mathtt{B} \in [\![\mathtt{x}]\!] \Rightarrow [\![\mathtt{new\ Q()}]\!] \subseteq [\![\mathtt{B.arg}]\!]$
$\mathtt{Q} \in [\![\mathtt{new\ Q()}]\!]$		$\mathtt{A} \in [\![\mathtt{y}]\!] \Rightarrow [\![\mathtt{new\ S()}]\!] \subseteq [\![\mathtt{A.arg}]\!]$
$\mathtt{S} \in [\![\mathtt{new\ S()}]\!]$		$\mathtt{B} \in [\![\mathtt{y}]\!] \Rightarrow [\![\mathtt{new\ S()}]\!] \subseteq [\![\mathtt{B.arg}]\!]$

Recap

- Introduction to flow analysis.
- CHA.
- Constraint generation for CFA.

Constraint generation. Example 2

```
class A implements I {
  I x = new D();
  public I m(I f) {
    return f.m(x);
  }
}

class B implements I {
  public I m(I g) {
    return this;
  }
}

... new A().m(new B()).m(new C()) ...
```


Constraints for example 2

$\mathtt{D} \in [\![\mathtt{x}]\!]$

$\mathtt{B} \in [\![\mathtt{this\text{-}B}]\!]$

$\mathtt{A} \in [\![\mathtt{new\ A()}]\!] \quad \mathtt{B} \in [\![\mathtt{new\ B()}]\!]$

$\mathtt{C} \in [\![\mathtt{new\ C()}]\!] \quad \mathtt{D} \in [\![\mathtt{new\ D()}]\!]$

$$\mathtt{A} \in [\![\mathtt{f}]\!] \Rightarrow \begin{cases} [\![\mathtt{x}]\!] \subseteq [\![\mathtt{f}]\!] \\ [\![\mathtt{f.m(x)}]\!] \subseteq [\![\mathtt{f.m(x)}]\!] \end{cases}$$

$$\mathtt{B} \in [\![\mathtt{f}]\!] \Rightarrow \begin{cases} [\![\mathtt{x}]\!] \subseteq [\![\mathtt{g}]\!] \\ [\![\mathtt{this\text{-}B}]\!] \subseteq [\![\mathtt{f.m(x)}]\!] \end{cases}$$

$$\mathtt{A} \in [\![\mathtt{new\ A()}]\!] \Rightarrow \begin{cases} [\![\mathtt{new\ B()}]\!] \subseteq [\![\mathtt{f}]\!] \\ [\![\mathtt{f.m(x)}]\!] \subseteq [\![\mathtt{new\ A().m(new\ B())}]\!] \end{cases}$$

$$\mathtt{B} \in [\![\mathtt{new\ A()}]\!] \Rightarrow \begin{cases} [\![\mathtt{new\ B()}]\!] \subseteq [\![\mathtt{g}]\!] \\ [\![\mathtt{this\text{-}B}]\!] \subseteq [\![\mathtt{new\ A().m(new\ B())}]\!] \end{cases}$$

$$\mathtt{A} \in [\![\mathtt{new\ A().m(new\ B())}]\!] \Rightarrow \begin{cases} [\![\mathtt{new\ C()}]\!] \subseteq [\![\mathtt{f}]\!] \\ [\![\mathtt{f.m(x)}]\!] \subseteq [\![\mathtt{new\ A().m(new\ B()).m(new\ C())}]\!] \end{cases}$$

$$\mathtt{B} \in [\![\mathtt{new\ A().m(new\ B())}]\!] \Rightarrow \begin{cases} [\![\mathtt{new\ C()}]\!] \subseteq [\![\mathtt{g}]\!] \\ [\![\mathtt{this\text{-}B}]\!] \subseteq [\![\mathtt{new\ A().m(new\ B()).m(new\ C())}]\!] \end{cases}$$

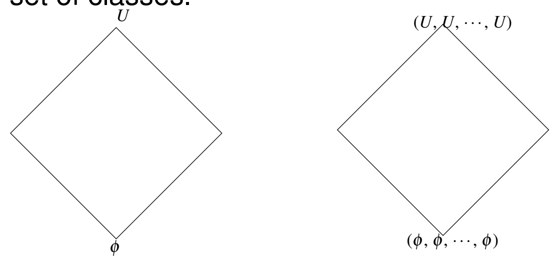
Recap

What you should be able to answer? (necessary not sufficient)

- Given a Java/C++ program inline methods using CHA.
- Given a Java/C++ program generate flow constraints for 0-CFA.

Computing Flow sets

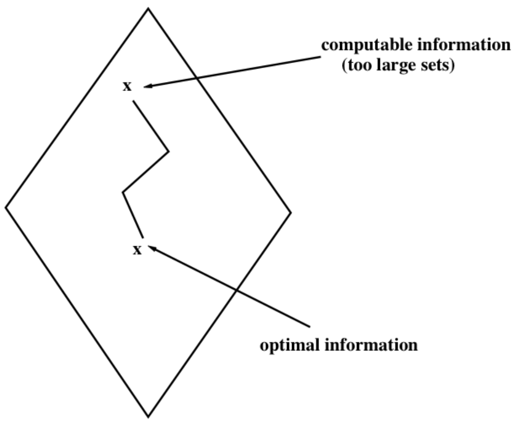
- For each flow variable, we want to compute the flow set.
- We go with the closed world assumption. $\Rightarrow$ maximal set of classes present in flow set is finite (the total number of classes).
- We use U to denote the maximal set of classes.
- The flow set for any expression $\in P(U)$.
- The set of flow sets for all the expressions $\subseteq$ powerset of a finite set of classes.



- Power set is a lattice (read: properties of lattices).
- The top of the lattice corresponds to trivial flow information.

Property of conservative flow analysis

- The minimal solution is above the optimal information.



- We will start with the most trivial solution.
- Iteratively improve the solution.

Constraint solver

- Takes one constraint at a time.
- At any point of time it maintains the minimal solution.
- Internally constraints are represented as a graph (N,E).
  - N: set of flow variables.
  - E: $(v \to w \in E) \Rightarrow v \subseteq w$
    (Why is it one way?)
- The value of a flow variable X is stored in a bit vector $B(X)$
  - Initialized to all 0s.
- Each bit i (which corresponds to a class), has an associated set of pending constraints (may be empty) corresponding to the conditional constraints; given by $K(X,i)$
  - For example: $C \in X \Rightarrow Y \subseteq Z$: $Y \subseteq Z \in K(X,i)$, where i is the bit corresponding to class C.
  - Note: $B(i)$ will be 0.

Solver details

Function Insert($i \in X$)
begin
  Propagate(X, i);
end
Function Insert($X \subseteq Y$)
begin
  Add an edge $X \rightarrow Y$;
  foreach $i \in B(X)$ **do**
    Propagate(Y, i);
  end
end
Function Insert($c \in X \Rightarrow Y \subseteq Z$)
begin
  if $B(X,c)$ **then**
    Insert ($Y \subseteq Z$);
  end
  else
    $K(X,c) = K(X,c) \cup \{(Y \subseteq Z)\}$
  end
end

Function Propagate(v, i)
begin
  if $\neg B(v,i)$ **then**
    $B(v,i) = true$;
    foreach $(v \rightarrow w) \in Edges$ **do**
      Propagate (w, i);
    end
    foreach $k \in K(v,i)$ **do**
      Insert (k);
    end
    $K(v,i) = \{\}$
  end
end

Running on example

Generate constraints

Starting	Propagation	Conditional
$\texttt{A} \in [\![\texttt{new A()}]\!]$	$[\![\texttt{new A()}]\!] \subseteq [\![\texttt{x}]\!]$	$\texttt{A} \in [\![\texttt{x}]\!] \Rightarrow [\![\texttt{new Q()}]\!] \subseteq [\![\texttt{A.arg}]\!]$
$\texttt{B} \in [\![\texttt{new B()}]\!]$	$[\![\texttt{new B()}]\!] \subseteq [\![\texttt{y}]\!]$	$\texttt{B} \in [\![\texttt{x}]\!] \Rightarrow [\![\texttt{new Q()}]\!] \subseteq [\![\texttt{B.arg}]\!]$
$\texttt{Q} \in [\![\texttt{new Q()}]\!]$		$\texttt{A} \in [\![\texttt{y}]\!] \Rightarrow [\![\texttt{new S()}]\!] \subseteq [\![\texttt{A.arg}]\!]$
$\texttt{S} \in [\![\texttt{new S()}]\!]$		$\texttt{B} \in [\![\texttt{y}]\!] \Rightarrow [\![\texttt{new S()}]\!] \subseteq [\![\texttt{B.arg}]\!]$

Useless assignment

Run the 0CFA algorithm on the constraints generated in Example 2.

Complexity analysis of the algorithm

- Say the size of the program is n.
- Number of classes: $O(n)$.
- Number of nodes (flow variables): $O(n)$
- Number of edges: $O(n^2)$
- Number of constraints added:
  At each call site ($O(n)$), for each class $O(n)$, add $O(n)$ constraints. $O(n^3)$
- Max size of $K(v,i)$, for any given v, and i: $O(n)$.
- Work done:
  - Each bit (class) is propagated along a specific edge at most once – $O(n^2)$. And each propagate may process $O(n)$ Insert functions. = $O(n^3)$
  - Each of the constraint may
    - be inserted into and deleted from a list once
    - cause the creation of a single edge.
- Cost = $O(n^3)$.
- In practise – mostly linear.

Recap

- Flow analysis using 0-CFA and some simple improvements.

What you should be able to answer? (necessary not sufficient)

- Given a set of flow constraints solve them to get the flow sets.

Reminder

- Assignment due in 5 days.

Challenges and issues

- The algorithm is not very precise.
- Several challenges:
  - huge class libraries.
  - polymorphic methods.
  - polymorphic container classes.
- Can improve by
  - dead code detection.
  - code duplication.
- We will study couple of ways.

Identify dead code

- Idea: Don't generate constraints for parts of program that is unreachable.
- Take the example of library code - most of the code in the libraries is "dead code" for any program.

Modified solver

$L = \phi$;
foreach $k \in$ constraints (main) **do**
    Insert(k);
end
`// Updates the reachable methods of main in Live`
while `Live` is not empty **do**
    m = Get a method from `Live`;
    foreach $k \in$ constraints (m) **do**
        Insert(k);
    end
    `// Updates the reachable methods of` m `in Live`
end

- Complexity? The algorithm is still $O(n^3)$.
- Will be efficient in practise.

Method duplication

- Say `A` and `B` implement the interface `I`.

```
class C {
  I id (I x) { // A polymorphic identity function
    return x;   // flow set for x = {A, B}
  } }
new C().id(new A()).m(5);
new C().id(new B()).m(5);
```

- Is there a way to get the flow set of `x` to singleton sets?
- Create a copy of each method implementation for each syntactic invocation.

```
class C2 {
  I id1 (I x) { // Convert to a monomorphic identity function?
    return x;    // flow set for x = {A}
  }
  I id1 (I x) { // Convert to a monomorphic identity function?
    return x;    // flow set for x = {B}
  } }
new C().id1(new A()).m(5);
new C().id2(new B()).m(5);
```

Class duplication

```
class C{
  I x;
  C put (I v) { x = v; return this; }
  I get() {return x; } // flow set for x = {A, B} }
new C().put(new A()).get().m(5);
new C().put(new B()).get().m(5);
```

- Is there a way to get the flow set of x to singleton sets?
- Create a copy of each class for each syntactic object creation (via `new`).

```
class C1{
  I x;
  C1 put (I v) { x = v; return this; }
  I get() {return x; } // flow set for x = {A} }
class C2{
  I x;
  C1 put (I v) { x = v; return this; }
  I get() {return x; } // flow set for x = {B} }
new C1().put(new A()).get().m(5);
new C2().put(new B()).get().m(5);
```


Closure conversion - a quick revisit

- Closure conversion - converting higher order functions to first order (has an environment that maps variables to values).

Translating Closures to C.

```
(define f (lambda (x)
    (let (g (lambda () x)) g)))

(set! a (f 10))
(a) ?
```

- Value should be 10.
- How to translate it to C?

Closure conversion to C

- Naive translation is problematic.

```
typedef int (* fp)() ; // function pointer
int g () {
  return x ; // Oops: which x is this?
}
fp f(int x) {
  return g ;
}
```


Closure conversion to C

- No nested functions in C. So use globals.

```
typedef int (*fp)() ; // function pointer
int globalX;
int g () {
  return globalX;
}
fp f(int x) {
  globalX = x ;
  return g ;
}
```

- Any problem? – a = f(10); b = f(20); a(); b();

Closure conversion - revisited

- Create an environment and pass free variables.
- At each application site - remmeber that it is a closure.

```
typedef int (*fp)() ; // function pointer
int g(e) {
  return e["x"];
}
(fp, env) f(int x) {
  return (g, {"x" = x}) ;
}
```

- `a = f(10)` → `(t1Fp, t2Env) = f(10); a[0] = t1Fp; a[1] = t2Env;`
- `b = f(20)` → `(t3Fp, t4Env) = f(20); b[0] = t3Fp; b[1] = t4Env;`
- `a()` → `a[0](a[1])`
- `b()` → `b[0](b[1])`

Recap

- Flow analysis using 0-CFA and some simple improvements.
- Closure conversion.

What you should be able to answer? (necessary not sufficient)

- Given a set of flow constraints solve them to get the flow sets.
- Translate closures in Scheme to C.

Reminder

- Assignment due in 4 days.