Meng Wu

2nd Year Ph.D. student at CESCA, ECE department.

Research Interest: software verification, program synthesis

Fast Static Analysis of C++ Virtual Function Call

DAVID F. BACON

PETER F. SWEENEY

OOPSLA 1996

Virtual Function

Pros:

- Identical Interface
- Code Reuse
- Flexibility…

Cons:

- Execution Overhead
- Code Size Overhead
- Analogous Problem

Static Analysis

- Unique Name (UN)
- Class Hierarchy Analysis (CHA)
- Rapid Type Analysis (RTA)

Unique Name

```
class A{
        public:
                virtual int foo() { return 1; };
};

class B: public A{
        public:
                virtual int foo() { return 2;};
                virtual int foo(int i) { return i+1;};
};

void main(){
        B* p = new B;
        int result1 = p->foo(1);
        int result2 = p->foo();
        A* q = p;
        int result3 = q->foo();
}
```

Unique Name

```
class A{
        public:
                virtual int foo() { return 1; };
};

class B: public A{
        public:
                virtual int foo() { return 2;};
                virtual int foo(int i) { return i+1;};
};

void main(){
        B* p = new B;
        int result1 = p->foo(1);
        int result2 = p->foo();
        A* q = p;
        int result3 = q->foo();
}
```

foo(int i) has Unique Name!

Unique Name

```
class A{
        public:
                virtual int foo() { return 1; };
};

class B: public A{
        public:
                virtual int foo() { return 2;};
                virtual int foo(int i) { return i+1;};
};

void main(){
        B* p = new B;
        int result1 = p->foo(1);
        int result2 = p->foo();
        A* q = p;
        int result3 = q->foo();
}
```

foo(int i) has Unique Name!

Class Hierarchy Analysis

```
class A{
        public:
                virtual int foo() { return 1; };
};

class B: public A{
        public:
                virtual int foo() { return 2;};
                virtual int foo(int i) { return i+1;};
};

void main(){
        B* p = new B;
        int result1 = p->foo(1);
        int result2 = p->foo();
        A* q = p;
        int result3 = q->foo();
}
```

Class Hierarchy Analysis

```
class A{
        public:
                virtual int foo() { return 1; };
};

class B: public A{
        public:
                virtual int foo() { return 2;};
                virtual int foo(int i) { return i+1;};
};

void main(){
        B* p = new B;
        int result1 = p->foo(1);
        int result2 = p->foo();
        A* q = p;
        int result3 = q->foo();
}
```

p can only point to class B or Derived(B)
And there are no derived classes of B

Class Hierarchy Analysis

```
class A{
        public:
                virtual int foo() { return 1; };
};

class B: public A{
        public:
                virtual int foo() { return 2;};
                virtual int foo(int i) { return i+1;};
};

void main(){
        B* p = new B;
        int result1 = p->foo(1);
        int result2 = p->foo();
        A* q = p;
        int result3 = q->foo();
}
```

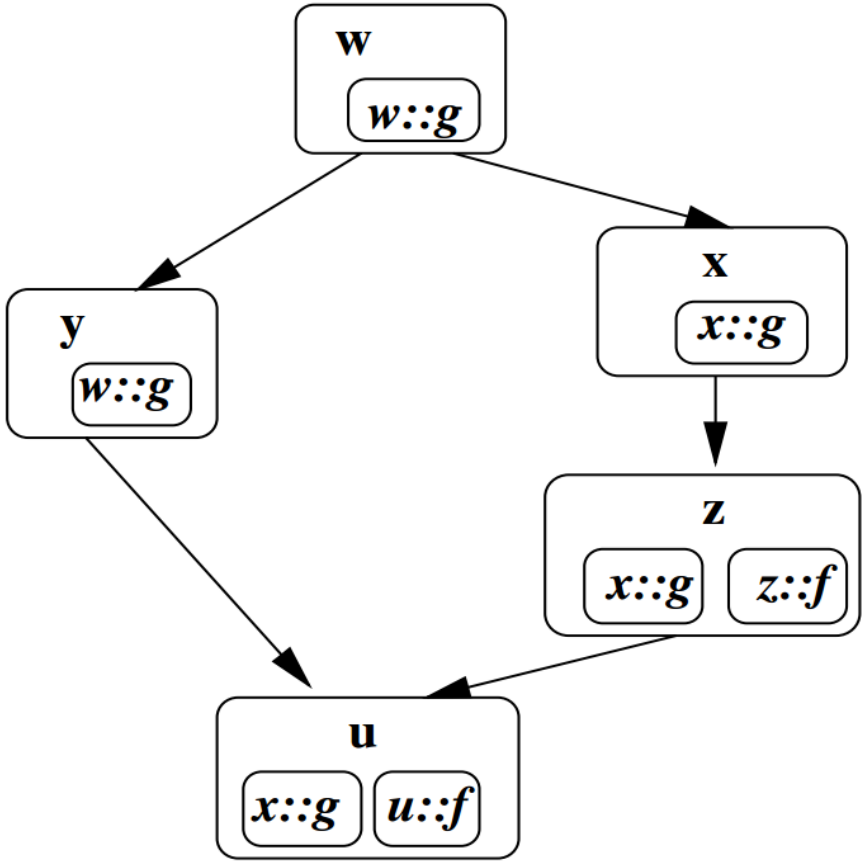
p can only point to class B or Derived(B)
And there are no derived classes of B

Class Hierarchy Graph

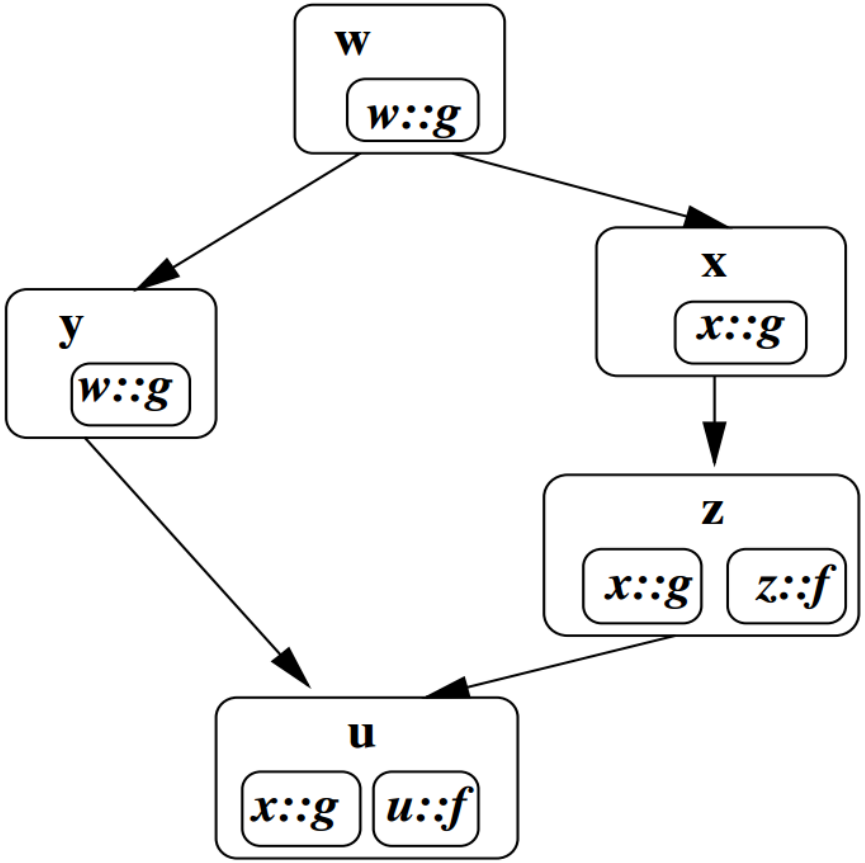
```
class w{
        virtual void g(){ .... };
};
class x : public virtual w{
        virtual void g(){ .... };
};
class y : public virtual w{
};
class z : public x{
        virtual void f(){ .... };
};
class u : public y, public z{
        void f(){ .... };
};
```

Class Hierarchy Graph

```
class w{
        virtual void g(){ .... };
};
class x : public virtual w{
        virtual void g(){ .... };
};
class y : public virtual w{
};
class z : public x{
        virtual void f(){ .... };
};
class u : public y, public z{
        void f(){ .... };
};
```



Class Hierarchy Graph



CHG is a tuple < C, D, V> where:

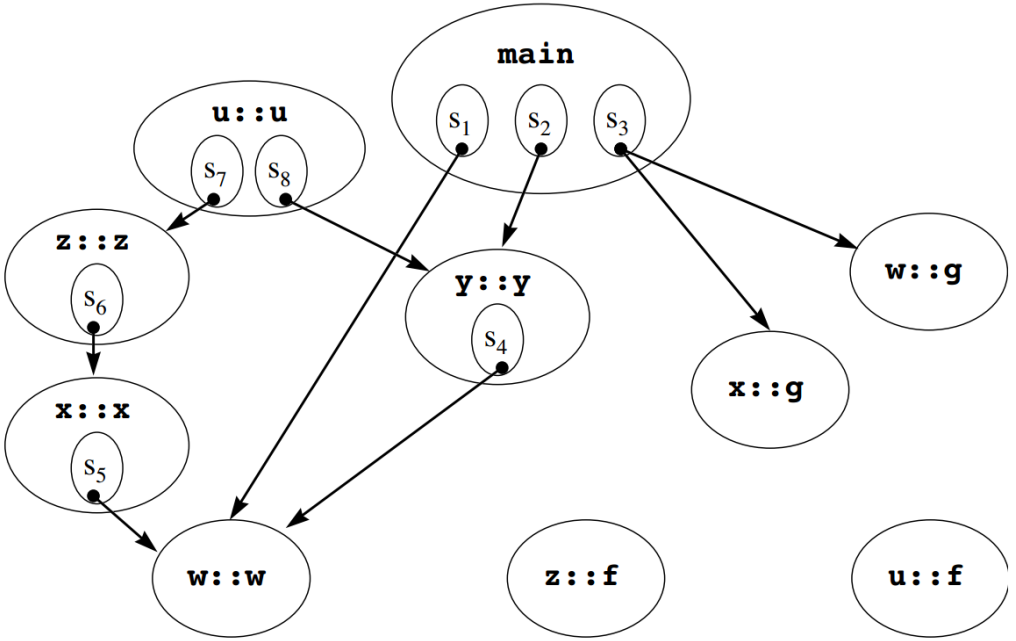
- C is the set of classes
- D is the set of derivations, which forming the edges of the graph
- V is the set of visible methods in particular class

Program Virtual Call Graph

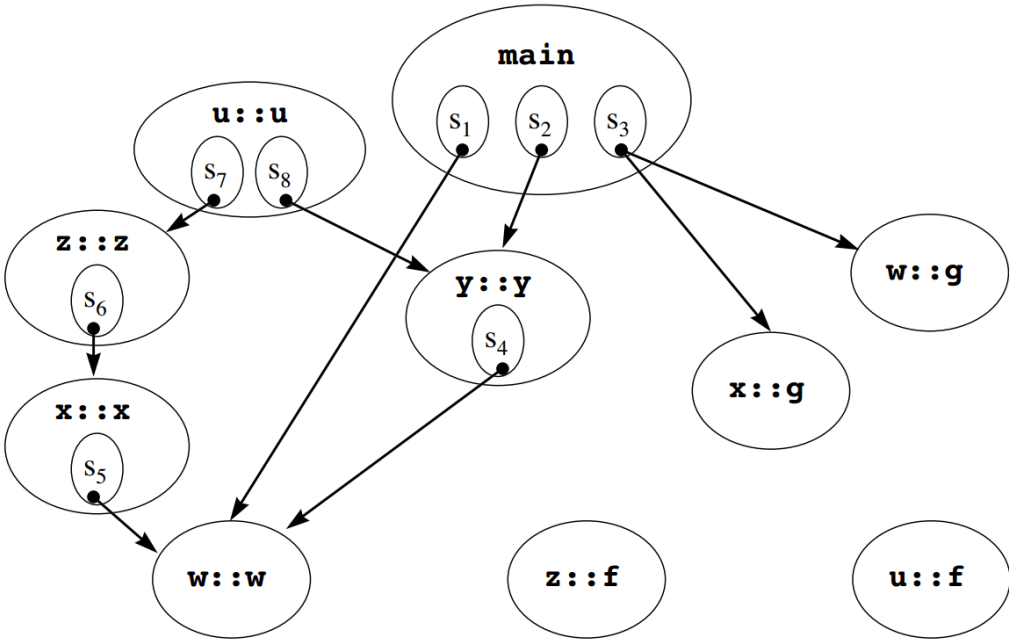
```
void main(){
        w* wp = new w;
        y* yp = new y;
        wp->g();
}
```

Program Virtual Call Graph

```
void main(){
        w* wp = new w;
        y* yp = new y;
        wp->g();
}
```



Program Virtual Call Graph



PVG is a tuple < F, S, I, R> where:

- F is the set of function nodes
- S is the set of call site subnodes
- I is the set of call instance edges
- R is the set of roots of the call graph

Rapid Type Analysis

```
class A{
        public:
                virtual int foo() { return 1; };
};

class B: public A{
        public:
                virtual int foo() { return 2;};
                virtual int foo(int i) { return i+1;};
};

void main(){
        B* p = new B;
        int result1 = p->foo(1);
        int result2 = p->foo();
        A* q = p;
        int result3 = q->foo();
}
```

It also use PVG generated by CHA

Rapid Type Analysis

```
class A{
        public:
                virtual int foo() { return 1; };
};

class B: public A{
        public:
                virtual int foo() { return 2;};
                virtual int foo(int i) { return i+1;};
};

void main(){
        B* p = new B;
        int result1 = p->foo(1);
        int result2 = p->foo();
        A* q = p;
        int result3 = q->foo();
}
```

Rapid Type Analysis

```
class A{
        public:
                virtual int foo() { return 1; };
};

class B: public A{
        public:
                virtual int foo() { return 2;};
                virtual int foo(int i) { return i+1;};
};

void main(){
        B* p = new B;
        int result1 = p->foo(1);
        int result2 = p->foo();
        A* q = p;
        int result3 = q->foo();
}
```

Notice: class A object is never instantiated

Rapid Type Analysis

```
class A{
        public:
                virtual int foo() { return 1; };
};

class B: public A{
        public:
                virtual int foo() { return 2;};
                virtual int foo(int i) { return i+1;};
};

void main(){
        B* p = new B;
        int result1 = p->foo(1);
        int result2 = p->foo();
        A* q = p;
        int result3 = q->foo();
}
```

Notice: class A object is never instantiated

Rapid Type Analysis

```
1   rapidTypeAnalysis(F, S, I, R)
2       Q_V ← ∅
3       C_L ← F_L ← S_L ← I_L ← ∅
4       for each f ∈ R
5           analyze(f, false)

6   analyze(f ∈ F, isbase ∈ Boolean)
7       if IsConstructor(f) and not isbase
8           instantiate(ClassOf(f))
9       if f ∈ F_L
10          return
11      F_L ← F_L ∪ {f}
12      for each s ∈ S, t ∈ F, P ∈ 2^C : < s, f, t, P > ∈ I
13          Let i = < s, f, t, P >
14          if s ∈ S_D or (s ∈ S_V and C_L ∩ P ≠ ∅)
15              addCall(i)
16          else
17              addVirtualMappings(P, i)
```

C_L : classes
F_L : functions
S_L : call sites
I_L : call instance

Rapid Type Analysis

```
18  addCall(i ∈ I)
19      Let < s, f, t, P >= i
20      I_L ← I_L ∪ {i}
21      S_L ← S_L ∪ {s}
22      analyze(t, IsBaseConstructorCall(i))

23  instantiate(c ∈ C)
24      if c ∈ C_L
25          return
26      C_L ← C_L ∪ {c}
27      for each i ∈ I : < c, i >∈ Q_V
28          if i ∉ I_L
29              addCall(i)
30              Q_V ← Q_V − {< c, i >}

31  addVirtualMappings(P ∈ 2^C, i ∈ I)
32      for each p ∈ P
33          Q_V ← Q_V ∪ {< p, i >}
```

Evaluation

Code Size Deduction

Benchmarks	Eliminated By		Not Eliminated	
	CHA	RTA	Unexecuted	Live
sched	1644	0	6684	91560
ixx	33540	1216	73692	70188
lcon	13920	392	25172	12458
hotwire	14924	544	3652	26296
simulate	11016	0	2712	15172
idl	26868	16436	93016	107428
taldict	12876	44	0	7596

Evaluation

Analysis Time

Benchmark	Size(lines)	CHA(sec)	RTA(sec)	RTA Overhead
sched	5712	1.90	1.94	< 0.1%
ixx	11157	5.12	5.22	1.4%
lcon	17278	6.27	6.50	3.0%
hotwire	5335	2.05	2.06	1.3%
simulate	6672	2.67	2.75	5.6%
idl	30288	5.71	6.42	1.4%
taldict	11854	1.66	1.78	4.0%
deltablue	1250	0.42	0.44	2.4%
richards	606	0.30	0.32	3.6%

Conclusion

- RTA resolved an average of 71% of virtual function calls
- Ran at an average of 3300 non-blank source lines per second
- CHA and RTA are essentially identical for reducing code size

Thank you!
And Questions?