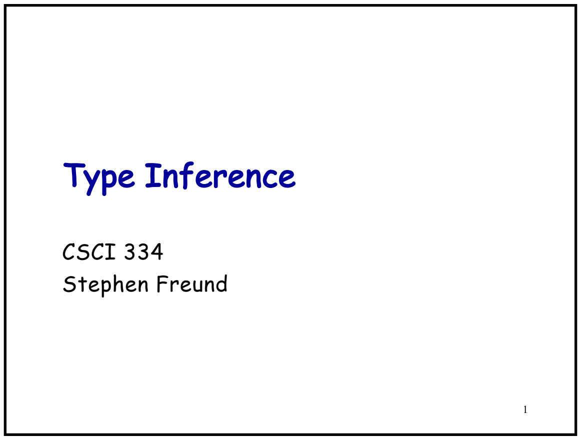
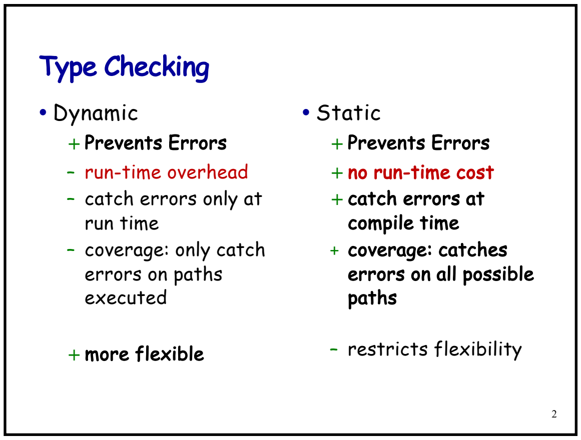


Type Inference

CSCI 334
Stephen Freund

1



Type Checking

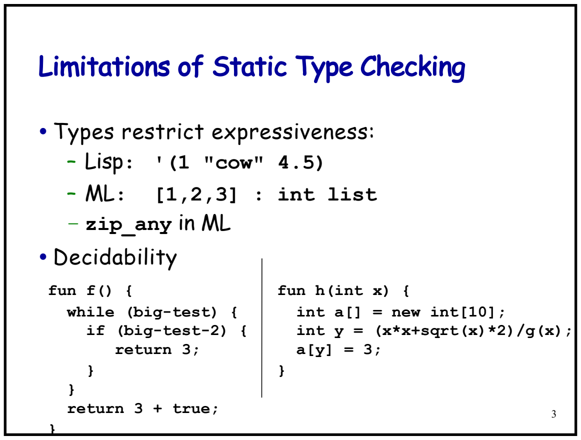
- Dynamic
  + Prevents Errors
  - run-time overhead
  - catch errors only at run time
  - coverage: only catch errors on paths executed

  + more flexible

- Static
  + Prevents Errors
  + no run-time cost
  + catch errors at compile time
  + coverage: catches errors on all possible paths

  - restricts flexibility

2



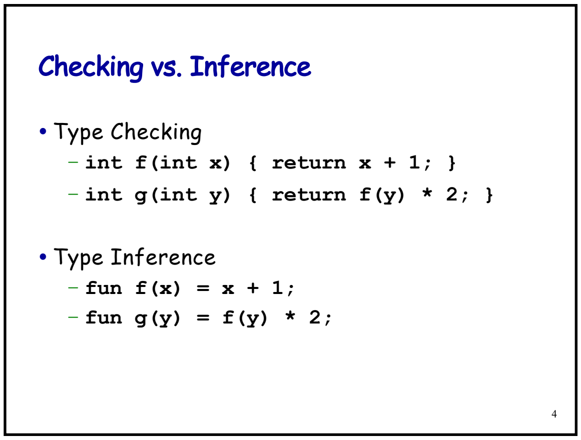
Limitations of Static Type Checking

- Types restrict expressiveness:
  - Lisp: `'(1 "cow" 4.5)`
  - ML: `[1,2,3] : int list`
  - `zip_any` in ML
- Decidability

```
fun f() {
  while (big-test) {
    if (big-test-2) {
       return 3;
    }
  }
  return 3 + true;
}
```

```
fun h(int x) {
  int a[] = new int[10];
  int y = (x*x+sqrt(x)*2)/g(x);
  a[y] = 3;
}
```

3

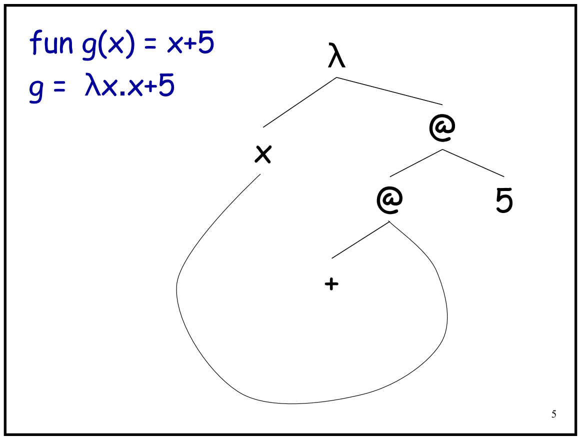


Checking vs. Inference

- Type Checking
  - `int f(int x) { return x + 1; }`
  - `int g(int y) { return f(y) * 2; }`

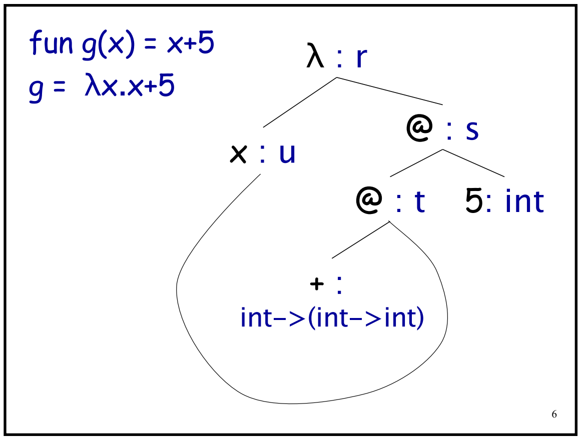
- Type Inference
  - `fun f(x) = x + 1;`
  - `fun g(y) = f(y) * 2;`

4



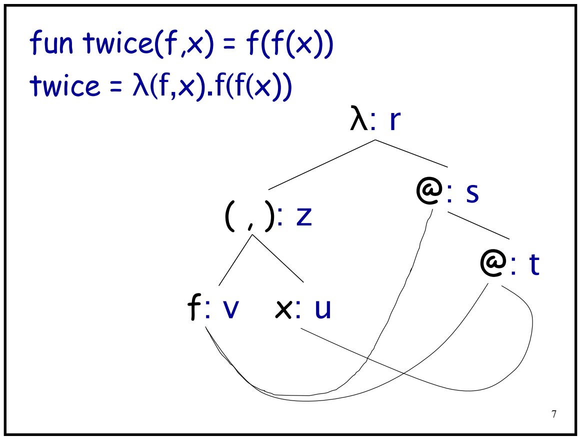
fun g(x) = x+5
g = λx.x+5

5



fun g(x) = x+5
g = λx.x+5

6



fun twice(f,x) = f(f(x))

twice = λ(f,x).f(f(x))

λ: r

(,): z

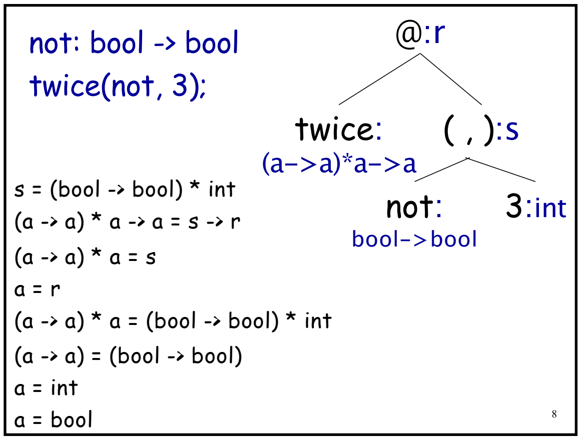
@: s

@: t

f: v

x: u

7



not: bool -> bool

twice(not, 3);

@:r

twice:

(a−>a)*a−>a

(,):s

not:

bool−>bool

3:int

s = (bool -> bool) * int

(a -> a) * a -> a = s -> r

(a -> a) * a = s

a = r

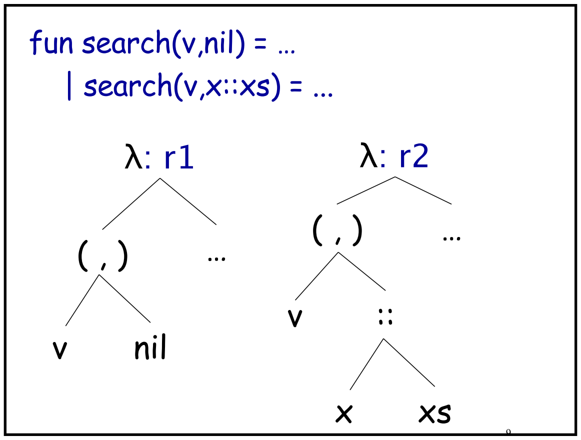
(a -> a) * a = (bool -> bool) * int

(a -> a) = (bool -> bool)

a = int

a = bool

8



fun search(v,nil) = …

| search(v,x::xs) = …

λ: r1

(,)

…

v

nil

λ: r2

(,)

…

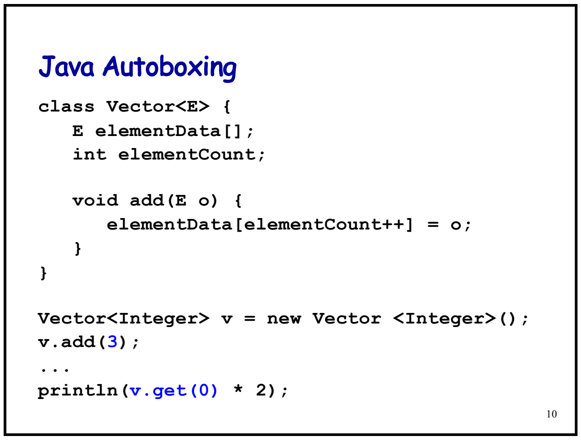
v

::

x

xs

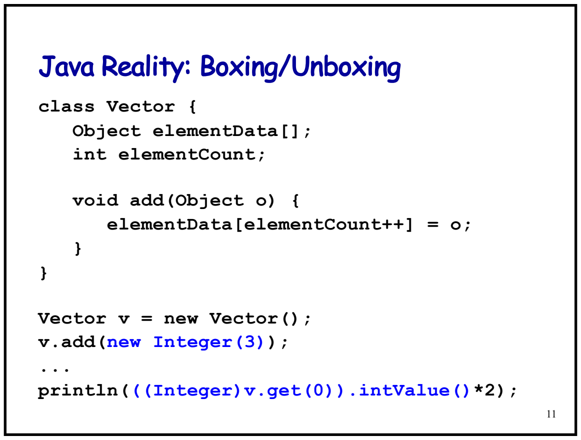
9



Java Autoboxing

```
class Vector<E> {
   E elementData[];
   int elementCount;

   void add(E o) {
      elementData[elementCount++] = o;
   }
}

Vector<Integer> v = new Vector <Integer>();
v.add(3);
...
println(v.get(0) * 2);
```

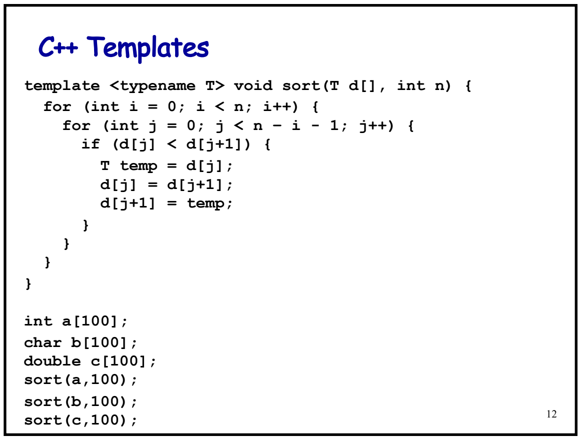
10



Java Reality: Boxing/Unboxing

```
class Vector {
   Object elementData[];
   int elementCount;

   void add(Object o) {
      elementData[elementCount++] = o;
   }
}

Vector v = new Vector();
v.add(new Integer(3));
...
println(((Integer)v.get(0)).intValue()*2);
```

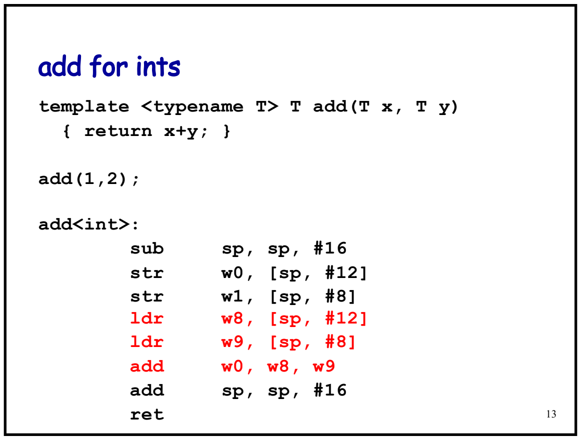
11



C++ Templates

```
template <typename T> void sort(T d[], int n) {
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n - i - 1; j++) {
      if (d[j] < d[j+1]) {
        T temp = d[j];
        d[j] = d[j+1];
        d[j+1] = temp;
      }
    }
  }
}

int a[100];
char b[100];
double c[100];
sort(a,100);
sort(b,100);
sort(c,100);
```

12

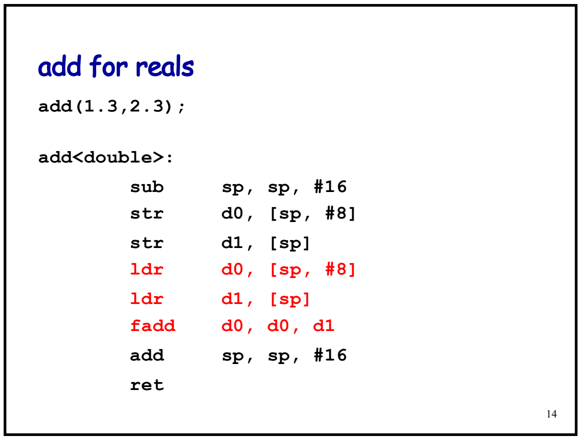


add for ints

```
template <typename T> T add(T x, T y)
  { return x+y; }

add(1,2);

add<int>:
        sub     sp, sp, #16
        str     w0, [sp, #12]
        str     w1, [sp, #8]
        ldr     w8, [sp, #12]
        ldr     w9, [sp, #8]
        add     w0, w8, w9
        add     sp, sp, #16
        ret
```

13

13

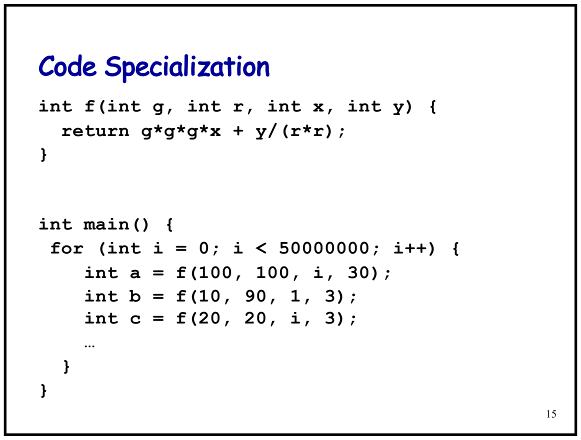


add for reals

```
add(1.3,2.3);

add<double>:
        sub     sp, sp, #16
        str     d0, [sp, #8]
        str     d1, [sp]
        ldr     d0, [sp, #8]
        ldr     d1, [sp]
        fadd    d0, d0, d1
        add     sp, sp, #16
        ret
```

14

14

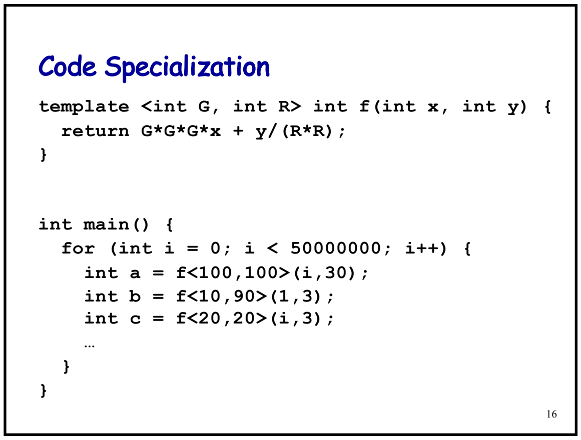


Code Specialization

```
int f(int g, int r, int x, int y) {
  return g*g*g*x + y/(r*r);
}


int main() {
 for (int i = 0; i < 50000000; i++) {
    int a = f(100, 100, i, 30);
    int b = f(10, 90, 1, 3);
    int c = f(20, 20, i, 3);
    …
  }
}
```

15

15

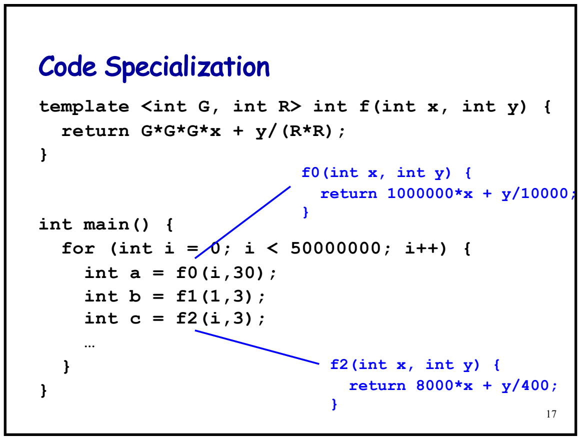


Code Specialization

```
template <int G, int R> int f(int x, int y) {
  return G*G*G*x + y/(R*R);
}


int main() {
  for (int i = 0; i < 50000000; i++) {
    int a = f<100,100>(i,30);
    int b = f<10,90>(1,3);
    int c = f<20,20>(i,3);
    …
  }
}
```

16

16



Code Specialization

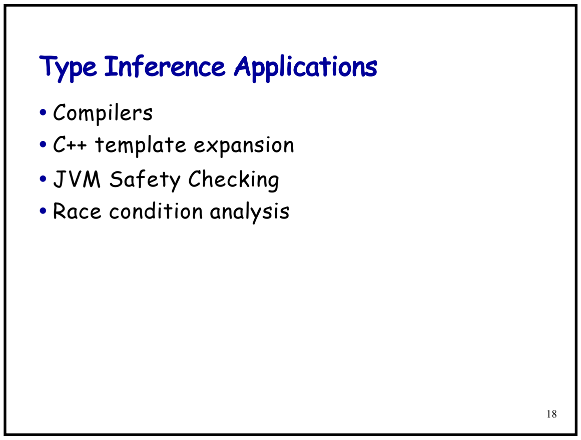
```
template <int G, int R> int f(int x, int y) {
  return G*G*G*x + y/(R*R);
}


int main() {
  for (int i = 0; i < 50000000; i++) {
    int a = f0(i,30);
    int b = f1(1,3);
    int c = f2(i,3);
    …
  }
}
```

```
f0(int x, int y) {
  return 1000000*x + y/10000;
}
```

```
f2(int x, int y) {
  return 8000*x + y/400;
}
```

17

17



Type Inference Applications

- Compilers
- C++ template expansion
- JVM Safety Checking
- Race condition analysis

18

18