Type Inference

CSCI 334
Stephen Freund

Limitations of Static Type Checking

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

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

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

Java 1.5 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);
```


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);
```


C++ Templates

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

add for ints

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

add(1,2);

add<int>:
 pushl    %ebp
 movl     %esp, %ebp
 movl     12(%ebp), %eax
 addl     8(%ebp), %eax
 leave
 ret
```

7

add for reals

```
add(1.3,2.3);

add<double>:
 pushl %ebp
 movl %esp, %ebp
 subl $16, %esp
 movl 8(%ebp), %eax
 movl 12(%ebp), %edx
 movl %eax, -8(%ebp)
 movl %edx, -4(%ebp)
 movl 16(%ebp), %eax
 movl 20(%ebp), %edx
 movl %eax, -16(%ebp)
 movl %edx, -12(%ebp)
 fldl 8(%ebp)
 faddl 16(%ebp)
 leave
 ret
```

8

Code Specialization

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

int main() {
  int x = f(100,100,10,30);
  int y = f(100,90,1,3);
  int z = f(35,52,11,11);

  for (int i = 0; i < 100000; i++) {
    ... f(20,20,i,3) ...
  }
}
```

9

Code Specialization

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

int main() {
  int x = f<100,100>(10,30);
  int y = f<100,90>(1,3);
  int z = f<35,52>(11,11);

  for (int i = 0; i < 100000; i++) {
    ... f<20,20>(i,3) ...
  }
}
```

10

Code Specialization

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

int main() {
  int x = f0(10,30);
  int y = f2(1,3);
  int z = f3(11,11);

  for (int i = 0; i < 100000; i++) {
    ... f1(i,3) ...
  }
}
```

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

```
f1(int x, int y) {
  return 8000*x + y/400;
}
```

11