Type Inference Extras

CSCI 334
Stephen Freund

Type Inference Applications
- Compilers
  - are values used consistently with some type?
- C++ template expansion
  - must we generate a new template version?
- JVM Safety Checking
- Race condition analysis

Running Programs in a Browser

My Computer

```
<html>
<body>
<applet>
</applet>
</body>
</html>
```

Running Programs in a Browser

```
<html>
<body>
<applet>
"send me page …"
</applet>
</body>
</html>
```
Running Programs in a Browser

Sandbox Security Model

Enforcing Sandbox Boundaries
- **Problem**: Prevent direct access to resources
- **Enforcement** through type safety
  - permit library calls, but no "unsafe" operations
  - example:
    ```java
    char *s = "moo";
    s = s - 1000;  // BAD
    print s;
    ```
  - another example:
    ```java
    byte b[] = { 0x12, 0xa3, 0x05, ... };
    ((function)b());  // REALLY BAD
    ```

Using Type Safety for Security
- Compiler rejects programs with type errors:
- Why not sufficient for the Web?
class A extends Object {
    int i;
    void f(int val) { i = val + 1; }
}

Method void f(int)
0 aload 0
1 iload 1
2 iconst 1
3 iadd
4 putfield #4 <Field int i>
5 return

Java vs. Java Bytecodes

class A extends Object {
    int i;
    void f(int val) { i = val + 1; }
}

Method void f(int)
0 aload 0
1 iload 1
2 iconst 1
3 iadd
4 putfield #4 <Field int i>
5 return
Java vs. Java Bytecodes

class A extends Object {
    int i;
    void f(int val) { i = val + 1; }
}

Method void f(int)
0 aload 0
1 iload 1
2 iconst 1
3 iadd
4 putfield #4 <Field int i>
5 return

Java vs. Java Bytecodes

class A extends Object {
    int i;
    void f(int val) { i = val + 1; }
}

Method void f(int)
0 aload 0
1 iload 1
2 iconst 1
3 iadd
4 putfield #4 <Field int i>
5 return

Variable and Stack Types

Method void f(int)
0 aload 0
1 iload 1
2 iconst 1
3 iadd
4 putfield #4 <Field int i>
5 return

Stack Types

Method void f(int)
0 aload 0
1 iload 1
2 iconst 1
3 iadd
4 putfield #4 <Field int i>
5 return

S0 = nil
V0 = V0
W0 = W0
S1 = V0::S0
V1 = V1
W1 = W1
S2 = W0::S1
V2 = V2
W2 = W2
S3 = int::S2
V3 = V3
W3 = W3
S4 = int::'a
V4 = V4
W4 = W4
S5 = 'b
V5 = V5
W5 = W5
### Stack

<table>
<thead>
<tr>
<th>Type</th>
<th>Var 0</th>
<th>Type</th>
<th>Var 1</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>nil</td>
<td>A</td>
<td>int</td>
<td>nil</td>
<td>A</td>
</tr>
<tr>
<td>int:A:nil</td>
<td>A</td>
<td>int</td>
<td>int:int:A:nil</td>
<td>A</td>
</tr>
<tr>
<td>int:A:nil</td>
<td>A</td>
<td>int</td>
<td></td>
<td></td>
</tr>
<tr>
<td>nil</td>
<td>A</td>
<td>int</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Method `void f(int)`

0 `aload 0`  
1 `iload 1`  
2 `iconst 1`  
3 `iadd`  
4 `putfield #4 <Field int i>`  
5 `return`

\[ S_0 = \text{nil} \]
\[ V_0 = A \]
\[ W_0 = \text{int} \]
\[ S_1 = V_0::S_0 \]
\[ V_1 = V_0 \]
\[ W_1 = W_0 \]
\[ S_2 = W_1::V_0::S_0 \]
\[ V_2 = V_1 \]
\[ W_2 = W_1 \]
\[ S_3 = \text{int}:W_1::V_0::S_0 \]
\[ V_3 = V_2 \]
\[ W_3 = W_2 \]
\[ S_4 = \text{int}:\text{int}:\text{a} \]
\[ V_4 = V_3 \]
\[ W_4 = W_3 \]
\[ S_5 = \text{b} \]
\[ V_5 = V_4 \]
\[ W_5 = W_4 \]

---

**Processor Clock Speeds**

![Processor Clock Speeds](image)

**Intel 4004  (1971)**  
2,300 transistors

**Intel 8086  (1978)**  
50,000 transistors

50,000,000 transistors

**Intel Core i7  (2010)**  
2,000,000,000 transistors

**Multi-Core Chips**

![Multi-Core Chips](image)
Concurrent Programming With Threads

Multithreaded Program Execution

Thread A
... 
\[ t_1 = \text{bal}; \]
\[ \text{bal} = t_1 + 100; \]
...

Thread B
... 
\[ t_2 = \text{bal}; \]
\[ \text{bal} = t_2 - 100; \]
...

bal is 500

bal is 400

Race Condition

Thread A
... 
\[ t_1 = \text{bal}; \]
\[ \text{bal} = t_1 + 100; \]
...

Thread B
... 
\[ t_2 = \text{bal}; \]
\[ \text{bal} = t_2 - 100; \]
...

bal is 500

bal is 400
Avoiding Race Conditions

Thread A
acquire(m);
t1 = bal;
bal = t1 + 100;
release(m);

Thread B
acquire(m);
t2 = bal;
bal = t2 - 100;
release(m);

m is “mutual exclusion lock”
must acquire “m” before using “bal”

Thread A
acquire(m);
t1 = bal;
bal = t1 + 100;
release(m);

Thread B
acquire(m);
t2 = bal;
bal = t2 - 100;
release(m);

Thread A
acquire(m);
t1 = bal;
bal = t1 + 100;
release(m);

Thread B
acquire(m);
t2 = bal;
bal = t2 - 100;
release(m);

bal is 500

Thread A
synchronized(m){
t1 = bal;
bal = t1 + 100;
}

Thread B
synchronized(m){
t2 = bal;
bal = t2 - 100;
}

synchronized(m) {
   synchronized(l) {
      x := 10;
   }
   synchronized(m) { 
      synchronized(l) {
         x := !y + 1;
      }
      synchronized(m) {
         synchronized(l) {
            x := !y + 1;
         }
      }
      y := 2;
   }
}

Thread 1
Thread 2

Common, Hard to Detect, Costly to Fix

Type Inference to Identify Races

Thread 1
synchronized(l) {
   x := 10;
}
synchronized(m) {
   print !y;
}
synchronized(m) {
   synchronized(l) {
      x := !y + 1;
   }
   synchronized(m) {
      synchronized(l) {
         x := !y + 1;
      }
   }
   y := 2;
}

Thread 2
synchronized(m) {
   print !y;
}

{l}
{l,m}
{l,m}
{l}
{l,m}