Scope and Memory Management

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

Programs on the Web

Running Programs in a Browser

Running Programs in a Browser
Running Programs in a Browser

Sandbox Security Model

Enforcing Sandbox Boundaries

Using Type Safety for Security

Enforcing Sandbox Boundaries

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

Using Type Safety for Security

- Compiler rejects programs with type errors:
  ```
  Java Compiler
  A.java
  A.class
  ```
  ```
  Java Virtual Machine
  bytecode interpreter and libraries
  ```
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

A obj.  Var 0  Var 1  Stack
A obj.  Var 0
A obj.  Var 0
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

Var 0 | Var 1
---|---
A obj. |
100 |
100 |
Stack

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

Var 0 | Var 1
---|---
A obj. |
100 |
100 |
Stack

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

Var 0 | Var 1
---|---
A obj. |
100 |
100 |
Stack

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

Var 0 | Var 1
---|---
A obj. |
100 |
100 |
Stack

Variable and Stack Types

Does stack top have two integers?
Concurrency and Race Conditions

int bal = 0;

Thread 1

<table>
<thead>
<tr>
<th>Thread 1</th>
<th>Thread 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>bal = t1 + 10</td>
<td>bal = t2 - 10</td>
</tr>
</tbody>
</table>

Thread 2

<table>
<thead>
<tr>
<th>bal == -10</th>
</tr>
</thead>
</table>

Race Conditions

- Common
- Hard to find via testing
- Scheduler dependent
- No resource is safe...

2003 Blackout ($6 Billion)

Mars Rovers

Therac-25
Concurrency and Race Conditions

Lock m = new Lock();
int bal = 0;

Thread 1
synchronized(m) {
    t1 = bal
    bal = t1 + 10
}

Thread 2
synchronized(m) {
    t2 = bal
    bal = t2 - 10
}

Thread 1
acquire(m)
release(m)

Thread 2
acquire(m)
release(m)

Type Inference to Identify Races

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

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

Simplified Machine Model

Inline Blocks

{ int x = 2;
  int y = 10
  { int z = 2;
    int x = 3;
    x = z + x + y;
  }
  print x;
}

Function Calls

int sumSquares(int n) {
    int i, sum = 0;
    for (i = 0; i < n; i++)
        sum = sum + i * i;
    return sum;
}

...}

{ int x = sumSquares(15);
  print x;
}
fun fact(n) =  
  if n <= 1 then 1 
  else fact(n-1)*n; 
val y = fact(2); 

fun fact(n) =  
  if n <= 1 then 1 
  else fact(n-1)*n; 
val y = fact(2); 

fun fact(n) =  
  if n <= 1 then 1 
  else fact(n-1)*n; 
val y = fact(2); 

fun fact(n) =  
  if n <= 1 then 1 
  else fact(n-1)*n; 
val y = fact(2); 

fun swap(x, y) =  
  let val t = !x in  
  x := !y; y := t  
  end; 
val a = ref 1; 
val b = ref 2; 
swap(a, b); 
swap(!a, !b); <= BAD 

fun add(x, y) =  
  x + y; 
val a = ref 1; 
val b = ref 2; 
add(!a, !a); 
add(a, b); <= BAD
Why Does it Matter?
• Side Effects
• Aliasing
• Efficiency

int add(x, y) {
    x = x + 1;
    return x + y;
}  
z = 5;
print add(z, z);

AccessingGlobals

val m = 5;
fun force(a) = m * a;

fun cow(y) = let m = y * y in
force(m)
end;
cow(10);

Examples of Dynamic Scoping

fun formatBuffer(buffer) =
... setColor(highlightColor) ...

let highlightColor = Blue in
formatBuffer(b);

fun playGame() =
... if strategy(...) = goLeft then ...

let fun strategy (...) = ...
in playGame();

Stack Inspection
• Permission depends on:
  - permission of calling method
  - permission of all methods above it on stack

System.main()
LocalClass.f()
LocalClass.g()
FileInputStream.open("log")
**Stack Inspection**

- Permission depends on:
  - permission of calling method
  - permission of all methods above it on stack

```java
void open(String s) {
    SecurityManager.checkRead();
    ...
}
```

Fails if Applet code is not trusted