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

Satellite Tracking Page
Running Programs in a Browser

www.nasa.gov

HTML
applet
files
monitor
printer
network

Virtual Machine
applet

Running Programs in a Browser

www.nasa.gov

HTML
applet

hard_drive.erase();

Sandbox Security Model

Virtual Machine Sandbox
security manager
security policy
runtime library
files
printer
monitor
network

applet
applet

Sandbox Security Model

Virtual Machine Sandbox
security manager
security policy
runtime library
files
printer
monitor
network

BAD!

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
  Java Virtual Machine
  bytecode interpreter
  and libraries
  ```
- 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
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 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 i>
5 return

Does stack top have two integers?
**Stack Type** | **Var 0 Type** | **Var 1 Type**
---|---|---
S₀ | V₀ | W₀
S₁ | V₁ | W₁
S₂ | V₂ | W₂
S₃ | V₃ | W₃
S₄ | V₄ | W₄
S₅ | V₅ | W₅

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

<table>
<thead>
<tr>
<th>Stack Type</th>
<th>Var 0 Type</th>
<th>Var 1 Type</th>
</tr>
</thead>
</table>
nil | A | int
A::nil | A | int
int::A::nil | A | int
int::int::A::nil | A | int
nil | A | int

S₀ = nil
S₁ = V₀::S₀
S₂ = W₀::S₁
S₃ = int::S₂
S₄ = int::a
S₅ = int::A::b
S₆ = 'b

V₀ = A
V₁ = V₀
V₂ = V₂
V₃ = V₃
V₄ = V₄
W₀ = int
W₁ = W₀
W₂ = W₂
W₃ = W₃
W₄ = W₄

**Processor Clock Speeds**

**Multi-Core Chips**
Concurrent Programming With Threads

Amazon.com

Network

Multithreaded Program Execution

Thread A
... 
    t1 = bal; 
    bal = t1 + 100; 
... 
Thread B
... 
    t2 = bal; 
    bal = t2 - 100; 
...

Race Condition

Thread A
... 
    t1 = bal; 
    bal = t1 + 100; 
... 
Thread B
... 
    t2 = bal; 
    bal = t2 - 100; 
...
Avoiding Race Conditions

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

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

- Common, Hard to Detect, Costly to Fix

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

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

}
Inline Blocks

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

Declarations

```plaintext
val Pi = 3.14;
fun for(lo,hi,f) = ...
fun build(...) = ...
```

Function Calls

```plaintext
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;
}
```

Activation Record

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

Activation Record

```
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); 

fun add(x, y) = 
  x + y; 
val a = ref 1; 
val b = ref 2; 
add(!a, !a); 
add(a, b); 

val m = 5; 
fun force(a) = m * a; 
fun cow(y) = 
  let m = y * y in 
  force(m) 
end; 
cow(10); 

cow(10);
Accessing Globals

```haskell
val m = 5;
funs force(a) = m * a;
funs cow(y) =
  let m = y * y in
  force(m)
end;
cow(10);
```

Dynamic Scope: `force(10)`
follow control links


Examples of Dynamic Scoping

```haskell
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

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

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

Fails if Applet code is not trusted