CSE431 – Translation of Computer Languages

- Semantic Analysis

Doug Shook

Control Structure Analysis

- Three major phases for Java:
  - Type Checking
    - How might this apply to control structures?
  - Reachability Analysis
    - Can we reach this statement?
  - Throwing and Exceptions
    - If we throw something, where do we end up?

Type Checking Review

- How would the following statement be type checked?

```
if (i > 7 + x) ...; else ...;
```

- Constraints are often language specific
  - Could the predicate be a type other than boolean?

Reachability Analysis

- Performed by a ReachabilityVisitor
  - Responsible for setting two flags:
    - *isReachable*
      - Can we get here?
    - *terminatesNormally*
      - Does this statement terminate normally?

- Why do we care?

Reachability Rules

- If *isReachable* is true for a list of statements, then it must also be true for the first statement in the list
  - Corresponding rule for *terminatesNormally*
- For statements in a list, there is a relationship between *isReachable* and *terminatesNormally*
- Statement lists for method bodies or static initializers are always considered reachable

Reachability Rules

- Some statements will *never* terminate normally
  - Also, some special cases
- Some statements will *always* terminate normally
- How should we handle null statements?
- Example:

```
void example() {
   int v; v++; return;   ; v = 10; v = 20;}
```

Reachability – Control Structures

- if/else
  - Assume all parts are reachable
    - Unless…?
  - Terminates normally if…..?
- Loops
  - Assume it terminates normally
    - Unless…?
  - Body is reachable
    - Unless…?

Reachability – Control Structures

- break/continue
  - Never terminates normally
  - What happens to statements that follow?
- return
  - Check the type – does it match return type of method?
  - Always terminates abnormally

Reachability – Control Structures

- switch/case
  - Case labels must be type checked
    - Checked against what?
  - Reachability of case labels?
  - Should the switch block as a whole terminate normally?
    - What determines this?
- Method body
  - Can it terminate abnormally?
  - Can it terminate normally?

Exercises

- Determine the reachability of the final assignment statement in each of the following code snippets:

```
for (i=0; i<10; i++)
    a += i;
a = 0;
```

```
for (;;)
    a++;
a = 0;
```

Exercises

- Determine the reachability of the final assignment statement in each of the following code snippets:

```
for (i = 0; i == new Random(i).nextInt();)
    a += i;
a = 0;
```

```
for (i=0; i<10; i++) {
    a += i;
    if (i > 0) return i; }
a = 0;
```

Exercises

- Determine the reachability of the final assignment statement in each of the following code snippets:

```
for (i = 0; i == new Random(i).nextInt();)
    a += i;
a = 0;
```

```
for (i=0; i<10; i++) {
    a += i;
    if (i > 0) return i; }
a = 0;
```

Exercises

- Determine the values of *isReachable* and *terminatesNormally* for the following statements:

```
a = 0;
for (;;) {
    i = new Random(a).nextInt();
    a += i;
    if (i == 10)
       break;
    else
       continue;
    i++;
} // Include the For List flags here
if (i == 10)
    return i;
return a+i;
```

Exceptions

- Exceptions allow for abnormal termination of a block
  - How does this differ from return?
- Exceptions have a *type*
  - Can be checked or unchecked
    - Difference?

Semantics - Exceptions

- Try
  - Check body normally, no special treatment
- Catch/finally
  - Type checking?
  - Multiple catches?
- Exception scope
  - Catch lists

Reachability - Exceptions

- Throw
  - Always terminates abnormally
- Try / Finally
  - Reachability?
  - Termination?
- Catch
  - Reachability?
    - Checked vs. Unchecked?
  - Termination?

Exception Example

```
class ExitComputation extends Exception{};
try { ...
   if (cond)
     throw new ExitComputation();
   if (v < 0.0)
     throw new ArithmeticException();
   else a  = Math.sqrt(v);
   ... }
catch (e ExitComputation) {return 0;}
```

- How will this be processed?

Semantics – Method calls

- Pieces of a method call
  - Identifier
  - Qualifiers (optional)
  - Parameter list
- Disambiguation
  - Gather all possible matches
  - Choose the nearest possible match
    - What can we use to make this decision?

Call Disambiguation

- Identify the class
  - Qualifier: Type vs. Instance?
- Collect matching method names
  - Current class
  - Super classes
- Filter based on parameter types
  - Length of list
  - Bindable
    - Do parameters have to have the exact same type?

Call Selection

- Examine the list that was generated
  - What if it is empty?
  - What if there's only one method?
  - What if there are multiple methods?
    - Which to match first?

Calls - Example

```
class A { void M(A parm) {...}
          void M() {...} }
class B extends A { void M(B parm) {...}
          void test(B arg) {M(arg);}}
```

- How should `M(arg)` be resolved?