CS 536 Announcements for Wednesday, April 5, 2023

Last Time
- name analysis
- static vs dynamic scoping
- scoping issues to consider
- name analysis in brevis
  - scoping rules
  - symbol table
  - handling records

Today
- type checking
- type-system concepts
- type-system vocabulary
- brevis
  - type rules
  - how to apply type rules

Next Time
- runtime environments

Name analysis: handling classes

Similar to handling aggregate data structures
- also need to be able to search the class hierarchy

Idea:
Symbol table for each class with two nesting hierarchies
1) for lexical scoping within methods
2) for inheritance hierarchy

To resolve a name
- first
- then
What is a type?

Short for **data type**

- classification identifying kinds of data
- a set of possible values that a variable can possess
- operations that can be done on member values
- a representation (perhaps in memory)

Type intuition – is the following allowed?

```java
int a = 0;
int *pointer = &a;
float fraction = 1.2;
a = pointer + fraction;
```

Components of a type system

- base types (built-in/primitive)
- rules for constructing types
- means of determining if types are compatible or equivalent
- rules for inferring the type of an expression
Type rules of a language specify

What types the operands of an operator must be

double a;
int b;
a = b;
b = a;

What type the result of an operator is

Type coercion
• implicit cast from one data type to another

• type promotion

Places where certain types are expected

if (x = 4) {
  ...
}

Type checking: when do we check?

static typing

dynamic typing

combination of the two
Type checking: when do we check? (cont.)

Static vs dynamic trade-offs

- static
- dynamic

Duck typing - type is defined by methods and properties

```python
class bird:
    def quack() : print("quack")

class robobird
    def quack() : print("0100101101")
```

Type checking: what do we check?

Strong vs weak typing

- degree to which type checks are performed
- degree to which type errors are allowed to happened at runtime

General principles

- statically typed →
- more implicit casting allowed →
- fewer checks performed at runtime →

Example

```python
union either { real(2) + 2.0
    int i;
    float f;
} u;

u.i = 12;

float val = u.f;
```
Type safety

- All successful operations must be allowed by the type system
- Java is explicitly designed to be type safe

- C is not
  
  ```c
  printf("%s", 1);
  struct big {
      int a[100000];
  }
  struct big *b = malloc(1);
  ```

- C++ is a little better
  
  ```c
  class T1 { char a; }
  class T2 { int b; }
  int main() {
      T1 *myT1 = new T1();
      T2 *myT2 = new T2();
      myT1 = (T1 *)myT2;
  }
  ```

Type system of brevis

brevis's type system

- primitive types
- type constructors
- coercion
Type errors in brevis

Operators applied to operands of wrong type
- arithmetic operators
- logical operators
- equality operators
  - must have operands of the same type
  - can't be applied to
- other relational operators
- assignment operator
  - must have operands of the same type
  - can't be applied to

Expressions that, because of context, must be a particular type but are not
- expressions that must be boolean (in brevis)
- reading
- writing

Related to function calls
- invoking (i.e., calling) something that is not a function
- invoking a function with
  - wrong number of arguments
  - wrong types of arguments
- returning a value from a void function
- not returning a value from a non-void function
- returning wrong type of value in a non-void function
Type checking

Recursively walks the AST to
- determine the type of each expression and sub-expression using the type rules of the language
- find type errors

Add a `typeCheck` method to AST nodes

Type checking: binary operator

Type "checking": literal

Type checking: IdNode

Type checking: others
- call to function `f`
  - get type of each actual parameter of `f`
  - match against type of corresponding formal parameter of `f`
  - pass `f`’s return type up the tree
- statement `s`
  - type check constituents of `s`
Type checking (cont.)

Type checking: errors

Goals:
- report as many distinct errors as possible
- don’t report same error multiple times – avoid error cascading

Introduce internal error type
- when type incompatibility is discovered
  - report the error
  - pass error up the tree
- when a type check gets error as an operand
  - don’t (re)report an error
  - pass error up the tree

Example:

```plaintext
integer a;
boolean b;
a = true + 1 + 2 + b;
b = 2;
```