Reading:
• Mitchell, Ch. 6

Objectives
1. be able to describe and give examples of how types are used to name and organize concepts
2. be able to describe and give examples of how types are used to make sure that bit sequences are used consistently
3. be able to describe how static typing makes languages like predominately statically typed languages like ML/Java/Scala more efficient than predominately dynamically typed languages like Lisp/Scheme/Python/Perl/SmallTalk
4. be able to describe how dynamic and static typing interact in Java
5. be able to describe why Java is categorized as type safe but C is not, citing specific language features that illustrate the difference
6. be able to translate short ML code fragments to lambda-calculus AST's capturing the type relevant interactions
7. be able to perform Hindley-Milner type inference on lambda-calculus AST's

Outline
1. purposes
   (a) name and organize concepts
   (b) using bit sequences consistently
   (c) providing information to compiler
2. type safety
   (a) example outcomes of type errors
      i. segmentation fault or buffer-overflow exploit
      ii. runtime exception
      iii. compile-time error message
   (b) run-time type-checking
   (c) static type checking
i. compile-time

ii. type system

(d) formal type system

i. formal, rule-based categorization of different syntactic program elements in a program language

ii. type rules can be combined to construct proofs that particular programs are correctly typed

iii. prove type safety of language (i.e., type correct programs will not exhibit type errors)

A. type preservation: as a well-typed program’s state changes during execution it stays well-typed

B. progress: a program never gets stuck in a state from which there is no defined transition

3. Hindley-Milner type inference

(a) type inference

(b) Hindley-Milner algorithm

i. assign a type / type variable to the expression / each subexpression

ii. generate constraints among the types / type variables

iii. solve the constraints through unification

4. Polymorphism

(a) parametric

i. implicit (using Hindley-Milner)

ii. explicit using Java generics

iii. explicit using C# templates

iv. explicit using C++ templates

(b) ad hoc polymorphism (aka overloading)

(c) subtype polymorphism (object-oriented)

**Vocabulary**

type, type error, type safety, type checking, dynamic (runtime) type checking, static (compile-time) type-checking, type inference, Hindley-Milner type inference, polymorphism, parametric polymorphism, implicit parametric polymorphism, explicit parametric polymorphism, ad-hoc polymorphism, overloading), subtype polymorphism