Types

Lecture 15

Kathleen Fisher

```
#include <stdio.h>

union {
float f;
int   i;
} unsafe;

void main(){
unsafe.f = 0.0 + 1;
printf("%d\n",unsafe.i);  -- it prints 1065353216
}
```

Information from Type Inference

- Consider this function…

```
fun reverse (nil) = nil
  | reverse (x::xs) = reverse(xs);
```

- … and its most general type:

```
reverse : 'a list → 'b list
```

- What does this type mean?

  Reversing a list does not change its type, so there must be an error in the definition of `reverse`!

Type Inference: Key Points

- Type inference computes the types of expressions
  - Does not require type declarations for variables
  - Finds the *most general type* by solving constraints
  - Leads to polymorphism
- Sometimes better error detection than type checking
  - Type may indicate a programming error even if no type error.
- Some costs
  - More difficult to identify program line that causes error
  - Natural implementation requires uniform representation sizes.
  - Complications regarding assignment took years to work out.
- Idea can be applied to other program properties
  - Discover properties of program using same kind of analysis

Example: Swap Two Values

- ML

```
- fun swap(x,y) =
    let val z = !x in x := !y; y := z end;
val swap = fn : 'a ref * 'a ref -> unit
```

- C++

```
template <typename T>
void swap(T& x, T& y){
      T tmp = x;   x=y;   y=tmp;
}
```

Declarations look similar, but compiled very differently

Implementation

- ML
  - `Swap` is compiled into one function
  - Type inference determines how function can be used
- C++
  - `Swap` is compiled into linkable format
  - Linker duplicates code for each type of use
- Why the difference?
  - The local x is a pointer to value on heap, so its size is constant.
  - C++ arguments passed by reference (pointer), but local x is on the stack, so its size depends on the type.

Parametric Polymorphism: ML vs C++

- ML polymorphic function
  - Declarations require no type information.
  - Type inference uses type variables to type expressions.
  - Type inference substitutes for variables as needed to instantiate polymorphic code.
- C++ function template
  - Programmer must declare the argument and result types of functions.
  - Programmers must use explicit type parameters to express polymorphism.
  - Function application: type checker does instantiation.

ML also has module system with explicit type parameters

Polymorphism vs Overloading

- Parametric polymorphism
  - Single algorithm may be given many types
  - Type variable may be replaced by *any* type
  - if `f:t→t` then `f:int→int`, `f:bool→bool`, ...
- Overloading
  - A single symbol may refer to more than one algorithm
  - Each algorithm may have different type
  - Choice of algorithm determined by type context
  - Types of symbol may be arbitrarily different
  - + has types `int*int→int`, `real*real→real`, *no others*

Summary

- Types are important in modern languages
  - Program organization and documentation
  - Prevent program errors
  - Provide important information to compiler
- Type inference
  - Determine best type for an expression, based on known information about symbols in the expression
- Polymorphism
  - Single algorithm (function) can have many types
- Overloading
  - One symbol with multiple meanings, resolved at compile time