CSC 520, Spring 2020

Principles of Programming Languages

Michelle Strout

THE UNIVERSITY OF ARIZONA

Version: 1

March 30, 2020

Plan

- **Announcements**
  - HW7 is due Friday
  - Going to start using a waiting room in office hours
- **Last time**
  - Algebraic datatypes
  - Types, Patterns, Exceptions
  - ML Traps and Pitfalls
- **Today**
  - Type Systems
  - A type system for two types

Type Systems

- **What they do**
  - Guide coding
  - Document code in a way that is checked by the compiler or interpreter
  - Rule out certain errors
- **How they work**
  - Compile-time prediction of set of possible values at runtime
  - World's most widely deployed static analysis
- **Trajectory in 520**
  - Formalize familiar, monomorphic type systems (like C)
  - Learn polymorphic type systems
  - Eventually, infer polymorphic types

Types classify “terms”

- **Example “terms” and their type**

```
n + 1 : int

"hello" ^ "world" : string

(fn n => n * (n - 1)) : int -> int

if p then 1 else 0 : int
```

Type Systems: Questions answered and not

- **Questions type systems can answer**
  - What kind of value does it evaluate to (if it terminates)?
  - What is the type contract of the function?
  - Is the function called with the right number of arguments?
  - Who has the rights to look at it/change it?
  - Is the number in terms of miles or millimeters?

- **Questions type systems generally cannot answer**
  - Will my program contain a division by zero?
  - Will my program contain an array bounds error?
  - Will my program take the car of '()?
  - Will my program terminate?

Decidability and Type Checking

- Suppose L is a “Turing-Complete” Language.
- TP is the set of programs in L that terminate.
- Wish: a type system to statically classify terminating programs
- Question: Is that possible? Why or why not?

Static vs. Dynamic Type Checking

THE UNIVERSITY OF ARIZONA
Computer Science

- **Most Languages use a combination of static and dynamic checks**

- **Static: “for all inputs”**
  - Input independent
  - Efficient at runtime
  - Approximate: rules out some programs that won’t trigger errors, (e.g. (if false then 2 else “hi”) ^ “there”
  - ➔ Question: would the above expression work in uScheme?

- **Dynamic: “for some inputs”**
  - Dependent on input
  - Run-time overhead
  - precise

Why are we learning about type systems?

- **Effectively using type systems and testing in coordination will improve the overall quality of your code.**
- **The ideas behind type systems apply any time you need to validate user input.**
- **Your introduction to static analysis, which is used for code improvement and security**

Type System and Checker for a Simple Language

- **Define an AST for expressions with**
  - Simple integer arithmetic operations: + - *
  - Numeric comparisons
  - Conditional
  - Numeric literal

```
datatype exp = ARITH of arithop * exp * exp
             | CMP of relop * exp * exp
             | LIT of int
             | IF of exp * exp * exp
and arithop = PLUS | MINUS | TIMES | ...
and relop = EQ | NE | LT | LE | GT | GE

datatype ty = INTTY | BOOLTY
```

Examples to rule out

- **Can't add an integer and a boolean**

```
3 + (3 < 99)

(ARITH(PLUS, LIT 3, CMP (LT, LIT 3, LIT 99)))
```

- **Can't compare an integer to a boolean**

```
3 < (4=24)

Q: What is the AST for this?
(CMP (LT, LIT 3, (CMP (EQ, LIT 4, LIT 24))))
```

Inference rules to define a type system

THE UNIVERSITY OF ARIZONA
Computer Science

- **Form of judgement, Context |– term : type**
  - Given Context, expression e has type tau
  - Written `|- e : tau`
  - (Right now, the empty context)
- **Inference rules determine how to write a type checker function** `typeof : exp -> ty`
- **Q: What inference rules do you recommend for this language?**
  - ARITH
  - LIT
  - CMP
  - IF

```
Informal example
|- 3 : int        |- 5 : int
-----------------------------
|- 3 + 5 : int
```

Rule for arithmetic operators

- **Informal example**

```
|- 3 : int        |- 5 : int
--------------------------
|- 3 + 5 : int
```

- **Rules out**

```
|- ‘z’ : char    |- 5 : int
--------------------------
|- ‘z’ + 5 : int
```

- **General form**

```
|- e1 : int     |- e2 : int
--------------------------
|- ARITH(_, e1, e2) : int
```

Rule for comparisons

- **Informal example**

```
|- 7 : int        |- 10 : int
-----------------------------
|- 7 < 10 : bool
```

- **General form**

```
|- e1 : int      |- e2 : int
-----------------------------
|- CMP(_, e1, e2) : bool
```

Rule for literals

- **Informal example**

```
|- 14 : int
```

- **General form**

```
---------------------------
|- LIT(n): int
```

Rule for conditionals

THE UNIVERSITY OF ARIZONA
Computer Science

- **General form**

```
|- e : bool
|- e1 : tau1
|- e2 : tau2       tau1 equiv tau2
---------------------------------
|- IF ( e, e1, e2) : tau1
```

- **Experience show it is better to test two types for equivalence than to write rule with same type appearing twice. Q: why?**

- **Typing rules let us read off what a type checker need to do**
  - Input to checker: e
  - Output from checker: tau

What is a type?

- **Working definition: a set of values**
- **Precise definition: classifier for terms!!**
- **A computation can have a type even if it doesn't terminate!**

Type checker in ML

```
val typeof : exp -> ty
exception IllTyped
fun typeof (ARITH (_, e1, e2)) =
      case (typeof e1, typeof e2)
        of (INTTY, INTTY) => INTTY
         | _              => raise IllTyped
  | typeof (CMP (_, e1, e2)) =
      case (typeof e1, typeof e2)
        of (INTTY, INTTY) => BOOLTY
         | _              => raise IllTyped
  | typeof (LIT _) = INTTY
  | typeof (IF (e,e1,e2)) =
      case (typeof e, typeof e1, typeof e2)
        of (BOOLTY, tau1, tau2) =>
           if eqType(tau1, tau2)
           then tau1 else raise IllTyped
         | _                    => raise IllTyped
```

Typing Rules: Context and Term Variables

- **Goal: Add variables and let binding to our language**
- **Questions (we will take one at a time)**
  - How do we need to extend our AST?
  - What could go wrong with variables?
  - What typing rules do we need for variables and let?

Extended language of expressions

THE UNIVERSITY OF ARIZONA.
Computer Science

- **Q: How do we need to extend our AST?**
  - Let x=e1 in e2

```
datatype exp = ARITH of arithop * exp * exp
             | CMP   of relop   * exp * exp
             | LIT   of int
             | IF    of exp     * exp * exp

             | VAR   of name
             | LET   of name    * exp * exp


and arithop = PLUS | MINUS | TIMES | ...
and relop = EQ | NE | LT | LE | GT | GE

datatype ty = INTTY | BOOLTY
```

What could go wrong?

THE UNIVERSITY OF ARIZONA
Computer Science

- **What could go wrong with variables?**

```
;; x can't be both an integer and a list

x + x @ x


;; y can't be both an integer and a string

let y = 10 in y ^ “hello” end
```

- **Need to track variable use to ensure consistency**

- **Key idea: Type environment Gamma that maps variable names to types**

Rule for var

- **What typing rule do we need for variables?**
- **General form**

```
x in dom Gamma        tau = Gamma(x)
--------------------------------------
Gamma |- VAR x : tau
```

Rule for let

THE UNIVERSITY OF ARIZONA
Computer Science

- **What typing rule do we need for let?**

- **General form**

```
Gamma |- e : tau
Gamma{x->tau} |- e' : tau'
-------------------------------
Gamma |- LET x = e in e' : tau'
```

Adding Gamma to the type checker


```
val typeof : ty env -> exp -> ty
fun typeof Gamma (ARITH ... ) = <as before>

  | typeof Gamma (VAR x) =
      case Gamma (x)
      of Some tau => tau
       | None     => raise IllTyped

  | typeof Gamma (LET x, e1, e2) =
           let tau1 = typeof Gamma e1
           in typeof (extend Gamma x tau1) e2
           end
```

Review

- **Today we discussed an abstract syntax for a simple language**
- **We came up with typing rules**
- **We talked about how to implement the type checker**
- **In HW7 and HW8**
  - You will design new syntax and typing rules for lists
  - You will read about and answer questions about extending an existing type checker
  - You will implement parts of a type checker from scratch
- **This is a big chunk of what language designers do**