

Lecture 32: Truthiness

"It would appear that we have reached the limits of what it is possible to achieve with computer technology, although one should be careful with such statements, as they tend to sound pretty silly in five years."

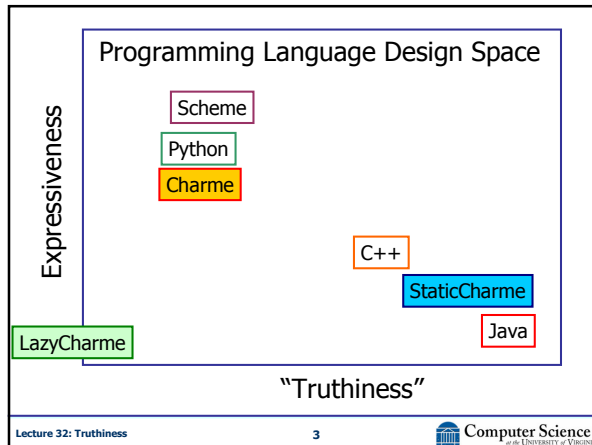
John Von Neumann, 1949

Problem Set 8

- Understand and modify a dynamic web application
- Will be posted on the web site before midnight tonight
- Due next Friday (April 13)

Problem Set 9

- Team requests and ideas due Monday (email me before midnight)



Types of Types

Charme

StaticCharme

Latent \Rightarrow Manifest
change grammar, represent types

Dynamically Checked \Rightarrow Statically Checked
typecheck expressions before eval

Manifest Types

Need to change the grammar rules to include types in definitions and parameter lists

Definition ::= (define Name : Type Expression)

Parameters ::= ϵ | Parameter Parameters

Parameter ::= Name : Type

Type ::= ??

Types in Charme

CType ::= CPrimitiveType

CType ::= CProcedureType

CType ::= CProductType

CPrimitiveType ::= Number | Boolean

CProcedureType ::= (CProductType \rightarrow Type)

CProductType ::= (CTypeList)

CTypeList ::= CType CTypeList


CTypeList ::=

```

CType ::= CPrimitiveType | CProcedureType | CProductType
CPrimitiveType ::= Number | Boolean
CProcedureType ::= (CProductType -> Type)
CProductType ::= (CTypeList)
CTypeList ::= CType CTypeList
CTypeList ::=

```

3
Number
+
((Number Number) -> Number)
(+ 3 3)
Number
Changed parameters grammar rule:
Parameter ::= Name : Type
(lambda (x:Number y:Number) (> x y))
((Number Number) -> Boolean)

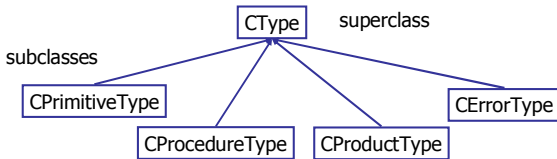
Lecture 32: Truthiness 7  Computer Science

Representing Types

```

CType ::= CPrimitiveType | CProcedureType | CProductType
CPrimitiveType ::= Number | Boolean
CProcedureType ::= (CProductType -> Type)
CProductType ::= (CTypeList)
CTypeList ::= CType CTypeList
CTypeList ::=


```



```

graph TD
    CType -- superclass --> CPrimitiveType
    CType -- superclass --> CProcedureType
    CType -- superclass --> CProductType
    CType -- superclass --> CErrorType

```


Lecture 32: Truthiness 8  Computer Science

```

class CType:
  @staticmethod ← No self object
  def fromString(s):
    # create type from string
    tparse = parse(s)
    return CType.fromParsed(tparse[0])

  @staticmethod
  def fromParsed(typ):
    ... # create type from parsed type
    # These methods are overridden by subclasses
  def isPrimitiveType(self): return False
  def isProcedureType(self): return False
  def isProductType(self): return False
  def isError(self): return False

```

Lecture 32: Truthiness 9  Computer Science


CPrimitiveType

```

class CPrimitiveType(CType):
  def __init__(self, s):
    self._name = s
  def __str__(self):
    return self._name
  def isPrimitiveType(self):
    return True
  def matches(self, other):

```

class X(Y):
means X is a subclass of Y


Lecture 32: Truthiness 10  Computer Science

CProcedureType

```

class CProcedureType(CType):
  def __init__(self, args, rettype):
    self._args = args
    self._rettype = rettype
  def __str__(self):
    return "(" + str(self._args) + " -> " \
      + str(self._rettype) + ")"
  def isProcedureType(self): return True
  def getReturnType(self): return self._rettype
  def getParameters(self): return self._args
  def matches(self, other):

```


Lecture 32: Truthiness 11  Computer Science

CProductType

```

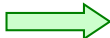
class CProductType(CType):
  def __init__(self, types): self._types = types
  def __str__(self): ...
  def isProductType(self): return True
  def matches(self, other):


```

Lecture 32: Truthiness 12  Computer Science

Types of Types

Charme StaticCharme

Latent  Manifest
change grammar, represent types

Dynamically Checked  Statically Checked
typecheck expressions before eval

Lecture 32: Truthiness

13

 Computer Science
University of Virginia

Adding Type Checking

```
def evalLoop():
    initializeGlobalEnvironment()
    while True:
        ...
        for expr in exprs:
            typ = typecheck(expr, globalEnvironment)
            if typ and typ.isError():
                print "Type error:" + typ.getMessage()
            else:
                res = meval(expr, globalEnvironment)
                if res != None:
                    print str(res)
```

Lecture 32: Truthiness

14

 Computer Science
University of Virginia

Static Type Checking

```
def typecheck(expr, env):
    if isPrimitive(expr):
        return typePrimitive(expr)
    elif isConditional(expr):
        return typeConditional(expr, env)
    elif isLambda(expr):
        return typeLambda(expr, env)
    elif isDefinition(expr):
        typeDefinition(expr, env)
    elif isName(expr):
        return typeName(expr, env)
    elif isApplication(expr):
        return typeApplication(expr, env)
    else: evalError ("Unknown expression: " + str(expr))
```

Lecture 32: Truthiness

15

 Computer Science
University of Virginia

```
class Environment:
    # Store a [type, value] pair for each variable.
    ...
    def addVariable(self, name, typ, value):
        self._frame[name] = (typ, value)
    def lookupPlace(self, name):
        if self._frame.has_key(name): return self._frame[name]
        elif (self._parent): return self._parent.lookupPlace(name)
        else: return None
    def lookupVariableType(self, name):
        place = self.lookupPlace(name)
        if place: return place[0]
        else: return CErrorType("Name not found")
    def lookupVariable(self, name):
        return self.lookupPlace(name)[1]
    ...
```

Lecture 32: Truthiness

16

 Computer Science
University of Virginia

Typing Names

```
def typeName(expr, env):
    return env.lookupVariableType(expr)
```

```
def evalDefinition(expr, env):
    name = expr[1]
    value = meval(expr[4], env)
    typ = CType.fromParsed(expr[3])
    env.addVariable(name, typ, value)
```

Lecture 32: Truthiness

17

 Computer Science
University of Virginia

Static Type Checking

```
def typecheck(expr, env):
    if isPrimitive(expr):
        return typePrimitive(expr)
    elif isConditional(expr):
        return typeConditional(expr, env)
    elif isLambda(expr):
        return typeLambda(expr, env)
    elif isDefinition(expr):
        typeDefinition(expr, env)
    elif isName(expr):
        return typeName(expr, env)
    elif isApplication(expr):
        return typeApplication(expr, env)
    else: evalError ("Unknown expression: " + str(expr))
```

Lecture 32: Truthiness

18

 Computer Science
University of Virginia

```

def typeDefinition(expr, env):
    assert isDefinition(expr)
    if len(expr) != 5:
        evalError ("Bad definition: %s" % str(expr))
    name = expr[1]
    if isinstance(name, str):
        if expr[2] != ':':
            evalError ("Definition missing type: %s" % str(expr))
            typ = CType.fromParsed(expr[3])
            etyp = typecheck(expr[4], env)
            if not typ.matches(etyp):
                evalError("Mistyped definition: ..." % (name, typ, etyp))
    elif isinstance(name, list):
        evalError ("Procedure definition syntax not implemented")
    else: evalError ("Bad definition: %s" % str(expr))

```

Lecture 32: Truthiness

19

Static Type Checking

```

def typecheck(expr, env):
    if isPrimitive(expr):
        return typePrimitive(expr)
    elif isConditional(expr):
        return typeConditional(expr, env)
    elif isLambda(expr):
        return typeLambda(expr, env)
    elif isDefinition(expr):
        return typeDefinition(expr, env)
    elif isName(expr):
        return typeName(expr, env)
    elif isApplication(expr):
        return typeApplication(expr, env)
    else: evalError ("Unknown expression: " + str(expr))

```

Lecture 32: Truthiness

20

```

class Procedure:
    def __init__(self, params, typ, body, env):
        self._params = params
        self._body = body
        self._typ = typ
        self._env = env
    def getParams(self):
        return self._params
    def getParamTypes(self):
        return self._typ
    def getBody(self): return self._body
    def getEnvironment(self): return self._env
    def __str__(self):
        return "<Procedure %s / %s>" \
            % (str(self._params), str(self._body))

```

Add type to
Procedure

Lecture 32: Truthiness

21

```

def evalLambda(expr, env):
    assert isLambda(expr)
    if len(expr) != 3:
        evalError ("Bad lambda expression: %s" % (str(expr)))
    params = expr[1]
    paramtypes = []
    paramnames = []
    assert len(params) % 3 == 0
    for i in range(0, len(params) / 3):
        name = params[i*3]
        assert params[(i*3)+1] == ':'
        paramnames.append(name)
        typ = CType.fromParsed(params[(i*3)+2])
        paramtypes.append(typ)
    return Procedure(paramnames, paramtypes, expr[2], env)

```

Lecture 32: Truthiness

22

```

def typeLambda(expr, env):
    assert isLambda(expr)
    if len(expr) != 3: evalError ("Bad lambda expression: %s" % str(expr))
    # this is a bit tricky - we need to "partially" apply it
    # to find the type of the body
    newenv = Environment(env)
    params = expr[1]
    paramnames = []
    paramtypes = []
    assert len(params) % 3 == 0
    for i in range(0, len(params) / 3):
        name = params[i*3]
        assert params[(i*3)+1] == ':'
        typ = CType.fromParsed(params[(i*3)+2])
        paramnames.append(name)
        paramtypes.append(typ)
    newenv.addVariable(name, typ, None)
    resulttype = typecheck(expr[2], newenv)
    return CProcedureType(CProductType(paramtypes), resulttype)

```

Lecture 32: Truthiness

23

Static Type Checking

```

def typecheck(expr, env):
    if isPrimitive(expr):
        return typePrimitive(expr)
    elif isConditional(expr):
        return typeConditional(expr, env)
    elif isLambda(expr):
        return typeLambda(expr, env)
    elif isDefinition(expr):
        return typeDefinition(expr, env)
    elif isName(expr):
        return typeName(expr, env)
    elif isApplication(expr):
        return typeApplication(expr, env)
    else: evalError ("Unknown expression: " + str(expr))

```

Lecture 32: Truthiness

24

Type Application

```
def typeApplication(expr, env):
  proctype = typecheck(expr[0], env)
  if not proctype.isProcedureType():
    evalError("Application of non-procedure: " + str(expr[0]))
  optypes = map(lambda op: typecheck(op, env), expr[1:])
  optype = CProductType(optypes)
  if not optype.matches(proctype.getParameters()):
    evalError("Parameter type mismatch: ..." \
              % (proctype.getParameters(), optype))
  return proctype.getReturnType()
```

Lecture 32: Truthiness

25

Static Type Checking

```
def typecheck(expr, env):
  if isPrimitive(expr):
    return typePrimitive(expr)
  elif isConditional(expr):
    return typeConditional(expr, env)
  elif isLambda(expr):
    return typeLambda(expr, env)
  elif isDefinition(expr):
    return typeDefinition(expr, env)
  elif isName(expr):
    return typeName(expr, env)
  elif isApplication(expr):
    return typeApplication(expr, env)
  else: evalError("Unknown expression: " + str(expr))
```

Lecture 32: Truthiness

26

Type Primitives

```
def typePrimitive(expr):
  if isNumber(expr):
    return CPrimitiveType('Number')
  elif isinstance(expr, bool):
    return CPrimitiveType('Boolean')
  elif callable(expr):
    return findPrimitiveProcedureType(expr)
  else:
    assert False
```

This is a kludgy procedure that looks through the global environment to find the matching procedure, and returns its type

Lecture 32: Truthiness

27

Static Type Checking

```
def typecheck(expr, env):
  if isPrimitive(expr):
    return typePrimitive(expr)
  elif isConditional(expr):
    return typeConditional(expr, env)
  elif isLambda(expr):
    return typeLambda(expr, env)
  elif isDefinition(expr):
    return typeDefinition(expr, env)
  elif isName(expr):
    return typeName(expr, env)
  elif isApplication(expr):
    return typeApplication(expr, env)
  else: evalError("Unknown expression: " + str(expr))
```

Left as possible
Exam 2 question!

Lecture 32: Truthiness

28

StaticCharme

```
StaticCharme> (+ 1 #t)
Error: Parameter type mismatch:
expected (Number Number), given (Number Boolean)
StaticCharme> (define square:((Number -> Number)
  (lambda (x:Number) (* x x)))
StaticCharme> (square #t)
Type error: Parameter type mismatch:
expected (Number), given (Boolean)
StaticCharme> (define badret:((Number -> Number)
  (lambda (x: Number) (> x 3)))
Error: Mismatched definition:
badret declared type ((Number) -> Number),
actual type ((Number) -> Boolean)
```

Lecture 32: Truthiness

29

Define Compose

Is it possible to define `compose` in StaticCharme?

```
(define compose:((((Number) -> Number)
  ((Number) -> Number))
  -> ((Number) -> Number))
(lambda (f:((Number) -> Number)
  g:((Number) -> Number))
  (lambda (x:Number)
    (f (g x))))
```

But, no way to define a `compose` that works for any functions.

Lecture 32: Truthiness

30

Charge

- PS9 project team requests and ideas due by midnight Monday
- PS8 will be posted soon