Gradual Python with Colored Local Type Inference

Joe Angell

April 29, 2009
Motivation

- Python programmers use dynamic typing
- Let's type check what we can, and what the programmer wants
- Gradual Typing lets you use type annotations to statically check code portions
- In Python nearly everything is first class, so to fit that paradigm we want first class parametric polymorphism
- Type inference saves lives
Overview

- Gradual Typing
- Colored Local Type Inference
- Integration of the two
def div(a, b):
    if b == 0:
        return "err"
    else:
        return a // b

def tydiv(a : int, b : int) -> dynamic:
    if b == 0:
        return "err"
    else:
        return a // b
from gp import *
@generic("T")
@generic("S")
    return fn(a, b)

def tydiv(a : int, b : int) -> dynamic:
    if b == 0:
        return "err"
    else:
        return a / b

def div(a : int, b : int) -> float:
    ## returns int in Python 2.5
    return a / b

## myapply<int,dynamic>(fdiv, 5, 2)
x = inst_(myapply, int, dynamic)(tydiv, 5, 0)
Aimed at implicit instantiation, lambda signature inference, and local variable inference
Based on $F_{\leq}$ a second-order lambda calculus with subtyping
Bidirectional partial propagation of types
Local Subtype Constraint Solving
Extend the language without undecidability
Fall back on type annotation
from gp import *
@generic("T")
@generic("S")
    return fn(a, b)

def tydiv(a : int, b : int) -> dynamic:
    if b == 0:
        return "err"
    else:
        return a / b

def div(a : int, b : int) -> float:  # returns int in Python 2.5
    return a / b

## myapply<int,dynamic>(fdiv, 5, 2)
x = myapply(tydiv, 5, 0))
# x : dynamic
print(myapply(div, x, 2))  # runtime check from dyn to int
Infer type of Function Body with ? as Guess

Infer type of Argument with Parameter type as Guess

Subtype Constraint: Argument <: Parameter

Subtype Constraint: Return Type <: Inherited Guess

Solve Constraints, Substitute TV's in Return Type with Smallest Supertype of Guess
Future Work

- Extend inference to classes and object types
- Extend to lambda
- Will Local variable inference be useful?
- Soundness and Completeness proofs
Consistency

\((C_{\text{Refl}}) \tau \sim \tau\)  \hspace{1cm} \frac{\sigma_1 \sim \tau_1}{\sigma_1 \rightarrow \sigma_2 \sim \tau_1 \rightarrow \tau_2} \hspace{1cm} \frac{\sigma_2 \sim \tau_2}{\tau} \sim \sigma_1 \rightarrow \sigma_2 \sim \tau_1 \rightarrow \tau_2}

\((C_{\text{UnR}}) \tau \sim ?\)  \hspace{1cm} \frac{? \sim \tau}{?} \sim \tau\)
class Record:
    def getkey(self):
        return self.key

r = Record()
r.key = file.readline()
## setattr(object, name, value)
sattr(r, file.readline(), parse(file.readline()))

if isinstance(r, Record):
    ...

elif isinstance(r, Foo):
    ...

r.key = eval("5 if x == 0 else True")