Pseudo

An algorithm design language
Overview

- Concise Pythonic syntax
- Rich list support
- Robust type inference
- On-demand objects
Compiler Architecture

Preprocessor → Scanner → Parser → L1 Inference → L2 Inference → Codegen

Type Inference Modules
def main():
    print "Hello world!"
def main():
    a = 42
    b = 41.15
    c = a + b

    d = "Introduction to" ^ " Algorithms"

    e = true
    f = e and false
    g = a < 2
def main():
    print fib(5)

def fib(n):
    if n == 1 or n == 0:
        return 1
    else:
        return fib(n - 1) + fib(n - 2)
Lists

- Variable-sized collections of data
- Supports primitive types and objects
- Backed by LLVM arrays
- Rich built-in functions
  - get(), set(), length(), insert(), remove(), push(), pop(), enqueue(), dequeue()
Lists

def main():
    a = [5, 7, 8]
    a.insert(1, 6)
    a.remove(3)
    print(a)

[5.00, 6.00, 7.00]

def main():
    b = ["C", "L", "R"]
    b.push("S")
    for letter in b:
        print(letter)
Type Inference (L1)

- Primitives: num, bool, string
- Infer the types of variables based on their usage (Hindley-Milner)
- Doubles as a semantic checker
def main():
    a = 42
    b = foo(a)
    c = not b

def foo(x):
    if x == 42:
        return true
    else:
        return false

Fdecl({ name=main; params=[]; body=
    [
        AAssign(a, ANum_lit(Num, 42.), Num);
        AAssign(b, ACall(foo, [AVal(a, Num)], Bool), Bool);
        AAssign(c, AUnop(Not, AVal(b, Bool), Bool), Bool)
    ]})

Fdecl({ name=foo; params=[(Num, x)]; body=
    [
        If(ABinop(AVal(x, Num), Eq, ANum_lit(Num, 42.), Bool),
            [Return(ABool_lit(Bool, true))],
            [Return(ABool_lit(Bool, false))]
        ]})
Objects

- Containers of arbitrary data
- Backed by LLVM structs
- Flexible and on-demand
Objects

```python
def main():
    person.name = "Foo"
    person.age = 42
    person.alive = true

    node.visited = false
    node.flow = 7
```

Users never explicitly define or create objects!
Object Inference (L2)

- Novel object inference algorithm

- We determine at compile-time:
  - Whether each variable is an object, and if so:
    - Its object type (an integer id)
    - The set of fields in the object type
    - The types of those fields
Object Inference Algorithm

1. Collect fields for each object variable
2. Collect equalities for object variables
3. Unify object variables to obtain object types
4. Annotate SAST with object type ids
5. Pass to codegen the object types
Object Inference Algorithm

Coercive Object Equality Scheme

- Two objects are of the same type if they are used in a manner that would require them to be the same type
- Examples: assignment, in a list with other objects, arguments to a function
Object Inference Algorithm

Coercive Object Equality Scheme

```python
def main():
    node1.visited = false
    node2.color = "maroon"
    list = [node1, node2]
```
Object Initialization

- The first field assignment determines the scope of an object.
- Other fields are automatically initialized to default values.
- `init` keyword can also be used to initialize objects in a scope.

```python
def main()
    init a, b, c
    if true:
        a.foo = true
    else:
        a.bar = 42
```
Object Printing

- Our built-in `print` function also supports objects!

```python
def main():
    person.name = "Foo"
    person.age = 42
    person.alive = true

    print person

person.name: Foo
person.age: 42.00
person.alive: 1
```
Key Accomplishments

- **Lists**
  - Types supported: num, bool, string, Object
  - Rich built-in function support

- **Objects**
  - Novel object inference algorithm
  - Object field types supported: num, bool, string, Object (1 level), List

- **High Dimension Type Inference**
  - (All primitives) × (Lists) × (Objects)
Pseudo

An algorithm design language