Active Typechecking and Translation in Ace

Cyrus Omar

Jonathan Aldrich

School of Computer Science
Carnegie Mellon University
Monolithic Languages

3,000,000 B.C.

WHAT IS IT?
WILL IT KILL US?
MAYBE WE SHOULD WORSHIP IT.

PRESENT DAY

WHY IS THERE ALWAYS CONSTRUCTION ON THIS ROAD?
MODERN ART IS SO PRETENTIOUS.
Reasons for New (DS)Languages

- Specialized syntax improves ease-of-use
- Specialized typechecking rules improve verifiability
- Specialized translation strategies improve performance
Safe and Natural Interoperability

Monolithic Languages

A + C
B + C
C

Extensible Languages

A
B
C

A → B
B → A
C → A
C → B
Reasons for New (DS)Languages

- Specialized syntax improves ease-of-use
- Specialized typechecking rules improve verifiability
- Specialized translation strategies improve performance
Ace

- An **extensible, statically-typed** programming language that lives inside *Python*.
  - Shares Python’s syntax
  - Uses Python as a metalanguage
  - Semantics are not the same as Python’s
from ace.C import C99, printf

print "Hello, compile-time world!"

@C99.fn
def main():
    hello = "Hello, run-time world!"
    printf(hello)
main = main.compile()

print "Goodbye, compile-time world!"

$ acec hello.py
Hello, compile-time world!
Goodbye, compile-time world!
$ cat hello.c
#include <stdio.h>
void main() {
    char* hello = "Hello, run-time world!";
    printf(hello);
}
from ace.OpenCL import OpenCL, get_global_id

@OpenCL.fn
def map(input, output, f):
    gid = get_global_id(0)
    output[gid] = f(input[gid])

@OpenCL.fn
def add5(x):
    return x + 5

from ace.OpenCL import int, double, global_ptr
gp_int = global_ptr(int)
gp_dbl = global_ptr(double)
add5_t = add5.ace_type

map_add5_int = map.compile(gp_int, gp_int, add5_t)
map_add5_dbl = map.compile(gp_dbl, gp_dbl, add5_t)

compile method on a generic function like map produces a concrete function (i.e. a function with a single type for each argument)
from ace.OpenCL import OpenCL, get_global_id

@OpenCL.fn
def map(input, output, f):
    gid = get_global_id(0)
    output[gid] = f(input[gid])

@OpenCL.fn
def add5(x):
    return x + 5

from ace.OpenCL import int, double, global_ptr
gp_int = global_ptr(int)
gp_dbl = global_ptr(double)
add5_t = add5.ace_type

map_add5_int = map.compile(gp_int, gp_int, add5_t)
map_add5_dbl = map.compile(gp_dbl, gp_dbl, add5_t)
class MonolithicTypechecker(ASTVisitor):

...  

def visit_Subscript(node, ctx):
    obj_t = visit(node.obj, ctx)
    slice_t = visit(node.slice, ctx)
    if (obj_t is a PtrType
        and slice_t is a IntegerType):
        return obj_t.target_type
    else:
        raise TypeError("<appropriate error msg>")
input[gid]

class ActiveTypechecker(ASTVisitor):
...

def visit_Subscript(node, ctx):
    obj_t = visit(node.obj, ctx)
    return obj_t.resolve_Subscript(node, ctx)

class PtrType(ace.Type):
    def __init__(self, target_type):
        self.target_type = target_type

    def resolve_Subscript(self, node, ctx):
        slice_t = visit(node.slice, ctx)
        if slice_t is a IntegerType:
            return self.target_type
        else:
            raise TypeError("<appropriate error msg>")
input[gid]

class MonolithicTranslator(ASTVisitor):

...

def visit_Subscript(node, ctx):
    obj_c = visit(node.obj, ctx)
    slice_c = visit(node.slice, ctx)
    return obj_c + "[" + slice_c + "]"
class ActiveTranslator(ASTVisitor):
...

def visit_Subscript(node, ctx):
    obj_t = typeof(node.obj, ctx)
    return obj_t.translate_Subscript(node, ctx)

class PtrType(ace.Type):
    def __init__(self, target_type):
        self.target_type = target_type

    def translate_Subscript(self, node, ctx):
        self_c = visit(node.obj, ctx)
        slice_c = visit(node.slice, ctx)
        return self_c + "[" + slice_c + "]"
Other Operations

- Attribute access: `o.field_name`
- Unary and binary operators: `e1 + e2`
- Call: `f(arg1, ..., argn)`
Examples

- Full OpenCL language implemented in this way
- Alternative types of pointers
  - First-class arrays (no pointer arithmetic)
  - OpenCL: tagged pointers
  - Managed pointers
- A variety of object systems
  - A class is a type indexed by the member signatures, base class(es), etc.
- A variety of higher-level parallel abstractions
  - Partitioned global address spaces (looks like an array, distributed underneath)
  - Higher-level message passing (e.g. Charm++)
- Security-oriented features
  - A special type for strings *statically* constrained to be within a regular language
Ace

- An extensible, statically-typed programming language that lives inside Python.
  - Shares Python’s syntax
  - Uses Python as a metalanguage
  - Semantics are not the same as Python’s
Safe and Natural Interoperability

Monolithic Languages

Extensible Languages
Active Typechecking & Translation

- Types are **first-class values** at compile-time.
- Dispatch to the type of a subexpression to control these phases of compilation, rather than build them into monolithic languages.