CONSTANT FOLDING AND TYPE CHECKING

OUR COMPILER PROJECT

Created by Evan Roncevich and Irakli Zhuzhunashvili

THE PROBLEMS

Large number of intermediate variables

Too many conditionals

Slow graph-coloring

Overall, too many lines of code to debug

CONSTANT FOLDING

The problem

```
def f():
        x = 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10 + 11 + 12 + 13\
        + 14 + 15 + 16 + 17 + 18 + 19 + 20+ 21 + 22 + 23 + 24 + 25\
        + 26 + 27 + 28 + 29 + 30 + 31 + 32 + 33 + 34 + 35 + 36 + 37
        return 1 + 2 + 3 + 4 + x
print f()
```

Lines without folding

```
3231 tests/big_no_fold.s
```

Lines with folding

```
387 tests/bigfold.s
```

EXAMPLE

```
def f():
        y = 12
        x = 1 + 2 + input() + 12 + y + input() + 5
        return x

print f()
```

Converts into:

```
def f():
        y = 12
        x = 20 + input() + y + input()
        return x

print f()
```

EXAMPLE

```
def f():
        y = 12
        x = 1 + 2 + 3 + - (input() + 12 + 2 + 1) + 20 + y + - (1 + in
        return x

print f()
```

Converts into:

```
def f():
        y = 12
        x = 10 + - input() + y + - input()
        return x

print f()
```

FOLDING IMPLEMENTATION

If child nodes include Const, add it to a list

If it includes another Add node, recurse on it

For anything else, recurse on it and add to another list

CONSTANT PROPAGATION

(NOT FULLY IMPLEMENTED)

Idea:

```
def f():
        x = 14
        y = 7 - x / 2
        return y * (28 / x + 2)
```

Converts into

```
def f():
        x = 14
        y = 0
        return 0
```

PROPAGATION IMPLEMENTATION

Create a dictionary mapping variables to values

At every node, update the dictionary

If variable's value is known, store it, otherwise set it to 'unknown'

Do the folding in second recursion using the dictionary

COMPILE-TIME TYPE CHECKING

Enormous conditionals

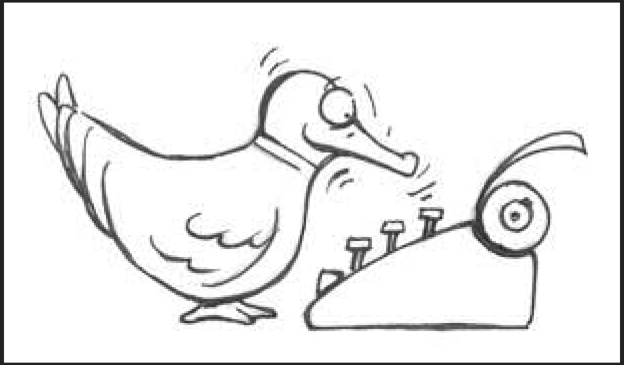
Explication is often useless

Hard to debug

GOAL

Determine if we know variables' typing at compile-time

Don't explicate if the type is known

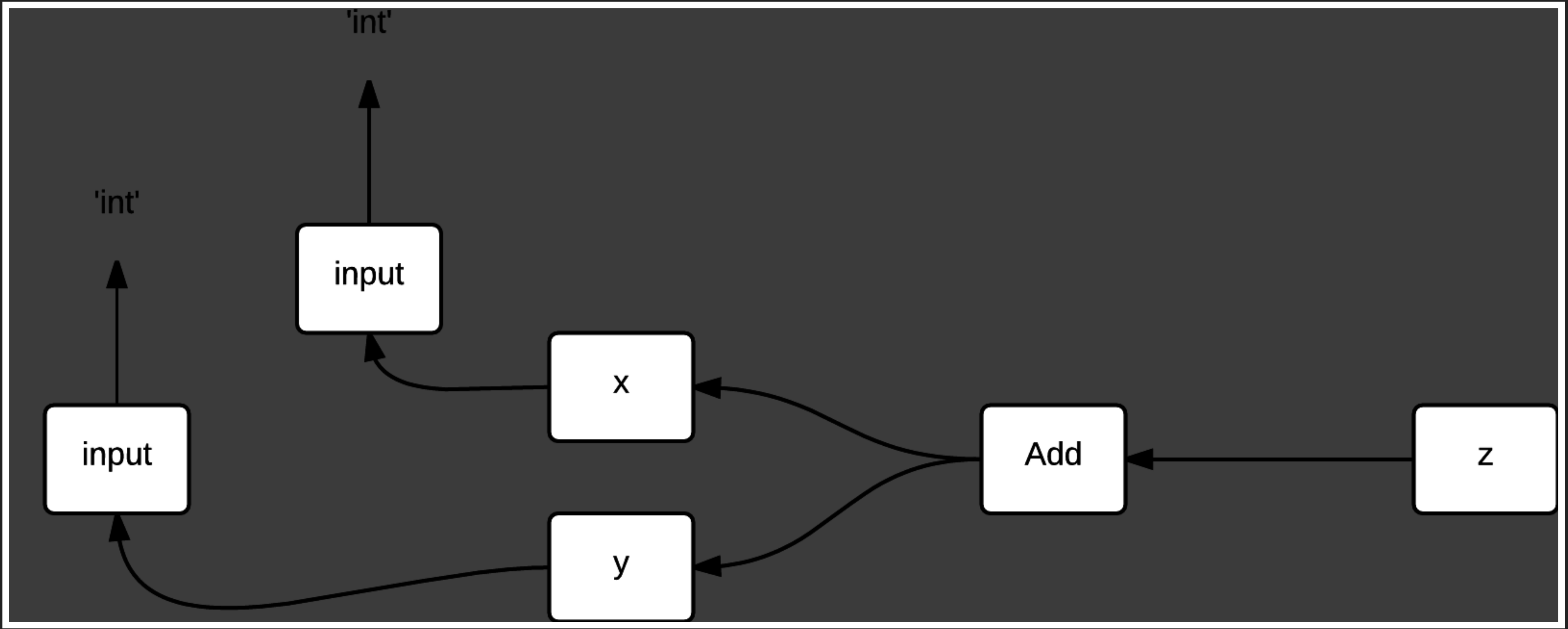


TYPE-CHECKING IMPLEMENTATION

At every statement, store all variable types in a dictionary

Handle if statements and function arguments

At explicate pass, use the dictionary to avoid explication



TYPE-CHECKING CODE

```
#adding l and r:
        if (lType == 'int' or lType =='bool')
        and (rType == 'int' or rType=='bool'):
                return InjectFrom('int', Add(
                (ProjectTo('int',l),ProjectTo('int',r))))
        if lType == "big" and rType == "big":
                return InjectFrom('big',CallFunc(Name("add"),
                [ProjectTo('big',l),ProjectTo('big',r)]))
```

Instead of:

```
#old method
    name1 = self.getNewTmp()
    name2 = self.getNewTmp()

    correctType = IsType('small',[name1,name2])
    correctBig = IsType('big',[name1,name2])
    ifExp = IfExp(correctType,InjectFrom('int', Add((ProjectTo
        ('int',name1),ProjectTo('int',name2)))),IfExp(correctBig
        ,InjectFrom('big',CallFunc(Name("add"),[ProjectTo('big'
        ,name1),ProjectTo('big',name2)])),ThrowErr('add_error')))

    return Let(name1, l,Let(name2,r,ifExp))
```

EXAMPLE

```
x=input()
y=input()
z=x+-y
print z
```

The contents of the dictionary at each point

```
Assign([AssName('True a0', 'OP_ASSIGN')], Const(1.25))
--> {'True a0': 'bool', 'fvs': 'unknown'}
Assign([AssName('False a0', 'OP_ASSIGN')], Const(0.25))
--> {'False a0': 'bool', 'fvs': 'unknown', 'True a0': 'bool'}
Assign([AssName('x a0', 'OP_ASSIGN')], CallFunc(Name('input'),
        [], None, None))
--> {'False a0': 'bool', 'True a0': 'bool', 'fvs': 'unknown',
        'x a0': 'int'}
Assign([AssName('y a0', 'OP_ASSIGN')], CallFunc(Name('input'),
        [], None, None))
--> {'False a0': 'bool', 'fvs': 'unknown', 'y a0': 'int', 'Tru
        'bool', 'x a0': 'int'}
Assign([AssName('z a0', 'OP_ASSIGN')], Add((Name('x a0'),
    UnarySub(Name('y a0')))))
--> {'False a0': 'bool', 'fvs': 'unknown', 'y a0': 'int', 'z a
    'int', 'True a0': 'bool', 'x a0': 'int'}
Printnl([Name('z a0')], None)
--> {'False a0': 'bool', 'y a0': 'int', 'z a0': 'int', 'True a
    'bool', 'fvs': 'unknown', 'x a0': 'int'}
--> {'False a0': 'bool', 'y a0': 'int', 'z a0': 'int', 'True a
    'bool', 'fvs': 'unknown', 'x a0': 'int'}
```

DRASTIC IMPROVEMENT

Compile time of infamous simplex test without type checking:

```
59038 lines, 1 min 10 secs compile
```

Compile time with our type checking:

```
10065 lines, 5 secs compile
```

Eliminates massive conditionals of addition and comparisons

Benefits to debugging and runtime performance

FUTURE USAGE AND DEVELOPMENT

Constant Propagation of If, While, and Functions

Analysis for heapified variables and lists

Type-error checking and warnings at compile-time

Removing unnecessary InjectFrom and ProjectTo lines