Jeremy Siek
University of Colorado

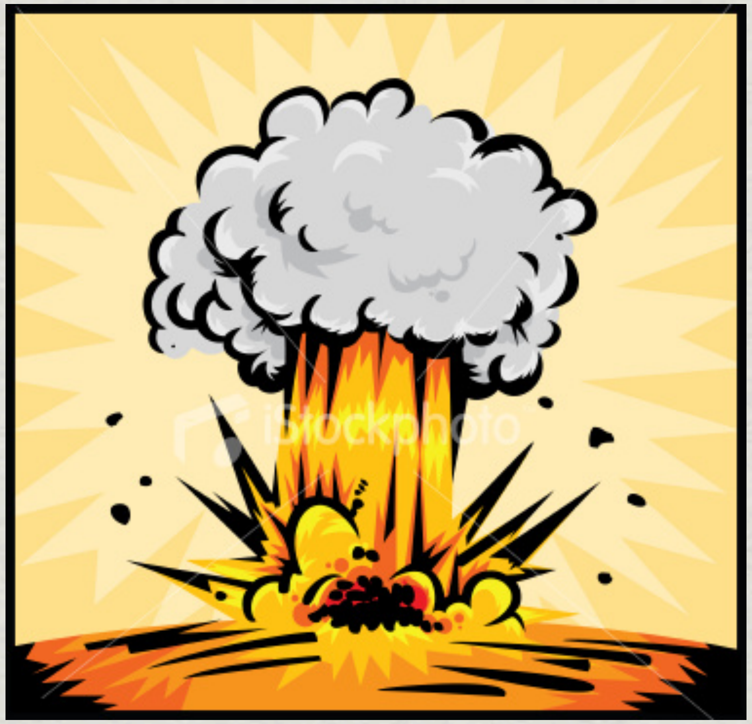
Blame Tracking

JOINT WORK WITH PHILIP WADLER

an explosive combination!

- dynamic languages are great!
- software libraries are great!
- the combination is explosive!

IT'S HAPPENED TO YOU:
YOU'VE CALLED A LIBRARY FUNCTION AND THEN... FROM SOMEWHERE DEEP INSIDE... KAABLAM!



Traceback... TypeError

SUBJECT:
GDATA LOGIN ISSUES (TYPEERROR: SEQUENCE ITEM 0: EXPECTED STRING, INT FOUND)
FROM:
LDK (LIAM...@GMAIL.COM)
DATE:
FEB 13, 2009 8:56:22 AM
LIST:
COM.GOOGLEGROUPS.GDATA-PYTHON-CLIENT-LIBRARY-CONTRIBUTORS

I'm having problems simply connecting to gdata from app engine. I realize that there are several ways to connect but I was hoping to use the ProgrammaticLogin method. I've created a package that includes the followin connectToGoogle function which always fails at the ProgrammaticLogin() call. I've attached the error below. I've confirmed that both the password and email are correct.

```
import gdata
from google.appengine.ext import webapp

def connectToGoogle(password):
        gd_client = gdata.docs.service.DocsService()
        gd_client.email = 'li...gmail.com'
        gd_client.password = password
        gd_client.source = 'mysite'
        gd_client.ProgrammaticLogin()
        return gd_client

Traceback (most recent call last):
  File "/Applications/GoogleAppEngineLauncher.app/Contents/Resources/GoogleAppEngine-default.bundle/Contents/Resources/google_appengine/google/appengine/ext/webapp/__init__.py", line 500, in __call__ handler.post(*groups)
  File "/Users/liamks/personalsite/main.py", line 97, in post gd = gsite.connect.connectToGoogle(password)
  File "/Users/liamks/personalsite/gsite/connect.py", line 16, in connectToGoogle gd_client.ProgrammaticLogin()
  File "/Users/liamks/personalsite/gdata/service.py", line 749, in ProgrammaticLogin
    headers={'Content-Type':'application/x-www-form-urlencoded'})
  File "/Users/liamks/personalsite/atom/http.py", line 134, in request
    connection.putheader(header_name, all_headers[header_name])
  File "/Applications/GoogleAppEngineLauncher.app/Contents/Resources/GoogleAppEngine-default.bundle/Contents/Resources/google_appengine/google/appengine/dist/httplib.py", line 174, in putheader
    line = '\r\n\t'.join(lines)
TypeError: sequence item 0: expected string, int found
```

leaky abstractions

- The Law of Leaky Abstractions: “All non-trivial abstractions, to some degree, are leaky” – Joel Spolsky
- my take: many abstractions are air tight when everything is going as planned... its just when things go wrong that they start to leak
- this is especially true of libraries in dynamic languages

plugging the leaks

- We try to plug these leaks with input checking code...

```
function send(msg) {
  validateMsg(msg)
  msg.id = sendToServer(JSON.encode(msg))
  database[msg.id] = msg
}
function validateMsg(msg) {
  function isObject(v)
    v != null && typeof v == "object"

  function isAddress(a)
    isObject(a) && isObject(a.at) && typeof a.at[0] == "string"
    && typeof a.at[1] == "string" && typeof a.name == "string"

  if (!(isObject(msg) && isObject(msg.to) &&
        msg.to instanceof Array && msg.to.every(isAddress) &&
        isAddress(msg.from) && typeof msg.subject == "string" &&
        typeof msg.body == "string" && typeof msg.id == "number" &&
        uint(msg.id) === msg.id))
    throw new TypeError
}
```

declarative checking

- types (e.g. int, string) naturally express many of these checks

```
type Message  = { to:      [Addr],
                  from:    Addr,
                  subject: string,
                  body:    string,
                  id:      uint }

function send(msg : Message) {
  msg.id = sendToServer(JSON.encode(msg))
  database[msg.id] = msg
}
```

- compiler generates *run-time* checks

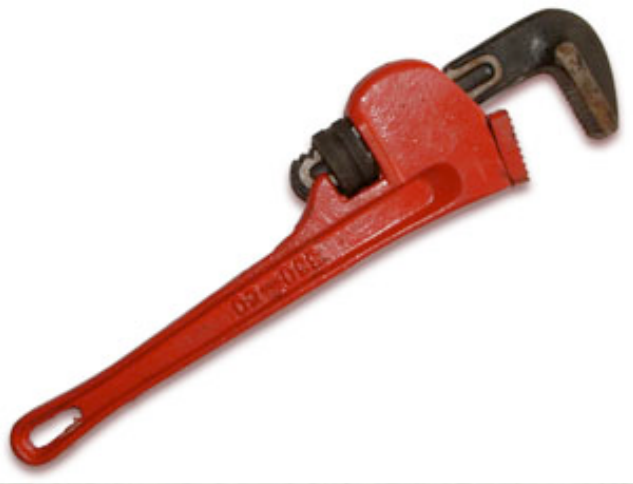
throwing a monkey wrench in the works

- callbacks & object methods cannot always be immediately checked

```
def g(cb : Int -> Int):
    ...
    cb(-1) + 5
    ....

def f(x):
    if 0 <= x:
        return 2
    else:
        return True

g(f)
```



← **WILL F RETURN AN INT? DON'T KNOW.**

blame tracking

```
1   def g(cb : Int -> Int):
2       ...
3       cb(-1) + 5
4       ...
5
6   def f(x):
7       if 0 <= x:
8           return 2
9       else:
10          return True
11
12  g(⟨Int -> Int⟩^12 f)
```

ERROR IS CAUGHT HERE, BUT BLAMES LINE 12

COMPILER GENERATED WRAPPER

- Originates from work on contract checking by Findler and Felleisen [ICFP 2002]

wrapper bloat

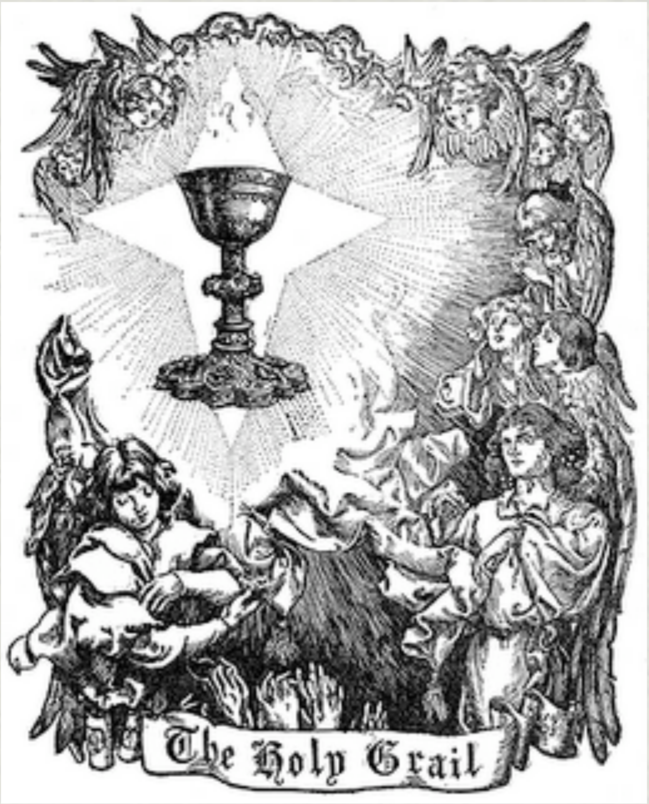
```
def even(n : Int, k : Dyn->Bool) -> Bool:
    if n == 0:
        return k(⟨Dyn->Bool⟩ True)
    else:
        return odd(n - 1, ⟨Bool->Bool⟩k)

def odd(n : Int, k : Bool->Bool) -> Bool:
    if n == 0:
        return k(False)
    else:
        return even(n - 1, ⟨Dyn->Bool⟩k)

even(99, k) ⟶ odd(98, ⟨Bool->Bool⟩k)
            ⟶ even(97, ⟨Dyn->Bool⟩⟨Bool->Bool⟩k)
              ...
```

- Example from Herman et al. [TFP 2007]

the quest for space efficiency



Coercion Calculus
[Henglein, SCP 1994]

↓

Space efficient checking
[Herman et al.,TFP 2007]

↓

Space efficient blame tracking
[Siek et al., ESOP 2009]

Mostly space efficient checking
[Siek and Taha, ECOOP 2007]

Threesomes [Siek and Wadler, in review]

twosomes

- twosomes, the standard way to represent wrappers:

$$\langle \mathtt{T} \Leftarrow \mathtt{S} \rangle e$$

- we've proven that a sequence of twosomes can always be collapsed to an equivalent pair of twosomes:

$$\langle \mathtt{T}_n \Leftarrow \mathtt{T}_{n-1} \rangle \ldots \langle \mathtt{T}_3 \Leftarrow \mathtt{T}_2 \rangle \langle \mathtt{T}_2 \Leftarrow \mathtt{T}_1 \rangle e$$

$$\langle \mathtt{T}_n \Leftarrow \mathtt{R} \rangle \langle \mathtt{R} \Leftarrow \mathtt{T}_1 \rangle e$$

greatest lower bound &

- The type R is the *greatest lower bound* of all the types in the sequence of twosomes

$$\langle T_n \Leftarrow R \rangle \langle R \Leftarrow T_1 \rangle e$$

$$R = T_n \ \& \ T_{n-1} \ \& \ \ldots \ \& \ T_2 \ \& \ T_1$$

```
        Int & Int = Int
(S -> T) & (S' -> T') = (S & S') -> (T & T')
          Dyn & T = T
          T & Dyn = T
```

threesomes

- We introduce "threesomes" simply as shorthand for a pair of twosomes:

$$\langle T_n \Leftarrow R \rangle \langle R \Leftarrow T_1 \rangle e$$

becomes

$$\langle T_n \Leftarrow R \Leftarrow T_1 \rangle e$$

threesomes with blame

- But what about the blame tracking information?

$$\langle T_n \Leftarrow T_{n-1}\rangle^{bn-1} \ldots \langle T_3 \Leftarrow T_2\rangle^{b2}\langle T_2 \Leftarrow T_1\rangle^{b1} e$$

- We compress the blame information into the middle type

$$\texttt{Int}^{b1} \;\&\; \texttt{Int}^{b2} = \texttt{Int}^{b2}$$

$$(S \to T)^{b1} \;\&\; (S' \to T')^{b2} = (S \;\&\; S') \to^{b2} (T \;\&\; T')$$

$$\texttt{Dyn} \;\&\; T^{b} = T^{b}$$

$$T^{b} \;\&\; \texttt{Dyn} = T^{b}$$

preserving tail calls

- Compiler-generated wrappers can turn tail calls into non-tail calls, leading to bloat on the stack

```
def even(n : Int) -> Dyn:
    if n == 0:
        return ⟨Dyn⟩True
    else:
        return ⟨Dyn⟩odd(n - 1)
```

```
def odd(n : Int) -> Bool:
    if n == 0:
        return False
    else:
        return ⟨Bool⟩even(n - 1)
```

- Solution: inspect the stack and compress wrappers

dealing with failure

- When compressing wrappers in tail position, there may be a conflict in the types, in which case there is no GLB.
- We can't signal the error immediately, that would change the order of evaluation.
- We instead record the error as the type $\perp$

failure & blame

- The blame handling for type ⊥ is delicate
- Can't just annotate ⊥ with a single piece of blame info:

$$\langle \texttt{Int} \Leftarrow \texttt{Dyn} \rangle^{l} \langle \texttt{Dyn} \Leftarrow \texttt{Bool} \rangle^{m} \langle \texttt{Bool} \Leftarrow \texttt{Dyn} \rangle^{n} \langle \texttt{Dyn} \Leftarrow \texttt{Bool} \rangle^{o}\ \texttt{True} \longrightarrow \texttt{blame}\ l$$

$$\langle \texttt{Int} \Leftarrow \texttt{Dyn} \rangle^{l} \langle \texttt{Dyn} \Leftarrow \texttt{Bool} \rangle^{m} \langle \texttt{Bool} \Leftarrow \texttt{Dyn} \rangle^{n} \langle \texttt{Dyn} \Leftarrow \texttt{Int} \rangle^{o}\ \texttt{1} \longrightarrow \texttt{blame}\ n$$

- Can't choose between label l or n, both are needed.

failure and blame

- Need to remember two blame labels and a type

$\perp(\texttt{l},\texttt{T}^{\texttt{m}})$

$$\texttt{S} \;\&\; \perp(\texttt{m},\texttt{G}^{\texttt{p}}) = \perp(\texttt{m},\texttt{G}^{\texttt{p}})$$

$$\perp(\texttt{m},\texttt{G}^{\texttt{q}}) \;\&\; \texttt{T} = \perp(\texttt{m},\texttt{G}^{\texttt{p}})$$

$$\quad \texttt{where head(T)=}\texttt{G}^{\texttt{p}}$$

$$\perp(\texttt{m},\texttt{H}^{\texttt{l}}) \;\&\; \texttt{T} = \perp(\texttt{l},\texttt{G}^{\texttt{p}})$$

$$\quad \texttt{where head(T)=}\texttt{G}^{\texttt{p}} \texttt{ and H} \neq \texttt{G}$$

Conclusion

- Blame tracking provides improved modularity, better error messages
- Finding a way to do space-efficient blame tracking was non-trivial, but now it's a solved problem
- Threesomes provide a simple data structure and algorithm for representing sequences of wrappers