

# Fugue: Annotations for Protocol Checking

Reading: ***The Fugue Protocol  
Checker: Is Your Software Baroque?***

17-654/17-765

Analysis of Software Artifacts

Jonathan Aldrich

# Find the Bug!

```
void CopyFile (string src, string dest)
{
    StreamReader fromFile = new StreamReader(src);
    StreamWriter toFile = new StreamWriter(dest);
    string line;
    while ((line = fromFile.ReadLine()) != null) {
        toFile.WriteLine(line);
    }
    fromFile.Close();
    ERROR: warning: StreamWriter resource 'toFile' becoming unreachable
    without calling StreamWriter.Close
}
```

# Find the Bug!

```
static public string DoSocketGet (string server)
{
    Socket s = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
    byte[] cmd = Encoding.ASCII.GetBytes("GET / HTTP/1.1\r\nHost: " +
        server + "\r\nConnection: Close\r\n\r\n");
    s.Send(cmd);
    ERROR: cannot call Socket.Send because 's' in state 'raw', but expected
    state 'connected'; did you forget to call Socket.Connect?
    // ...
}
```

# Specifications(1)

```
class StreamWriter
{
  [Creates]
  StreamWriter (string filename);

  [Disposes]
  void Close ();
}
```

# Specifications(2)

```
[WithProtocol("raw", "bound", "connected", "down")]
```

```
class Socket
```

```
{
```

```
  [Creates("raw")]
```

```
  public Socket (...);
```

```
  [ChangesState("raw", "bound")]
```

```
  public void Bind (EndPoint localEP);
```

```
  [ChangesState("raw", "connected"), ChangesState("bound", "connected")]
```

```
  public void Connect (EndPoint remoteEP);
```

```
  [InState("connected")]
```

```
  public int Send (...);
```

```
  [InState("connected")]
```

```
  public int Receive (...);
```

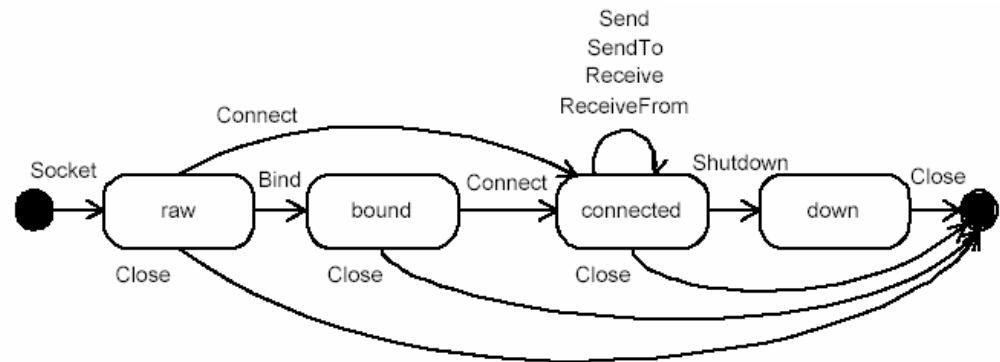
```
  [ChangesState("connected", "down")]
```

```
  public void Shutdown (SocketShutdown how);
```

```
  [Disposes(State.Any)]
```

```
  public void Close ();
```

```
}
```



# Specifications(3)

```
[WithProtocol("open", "closed")]
class WebPageFetcher
{
  [InState("connected", WhenEnclosingState="open"), NotAliased(WhenEnclosingState="open")]
  [Unavailable(WhenEnclosingState="closed")]
  private Socket socket;
```

```
[Creates("closed")]
public WebPageFetcher () { }
```

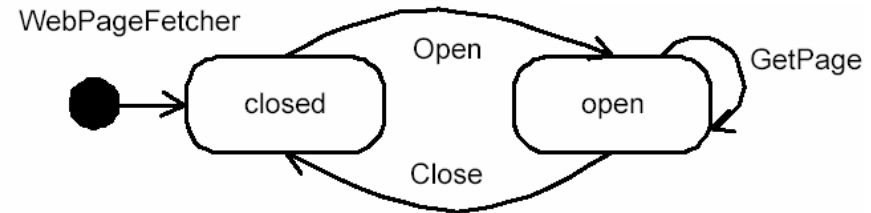
```
[ChangesState("closed","open")]
public void Open (string server)
{
  Socket newSock = new Socket( AddressFamily.InterNetwork, SocketType.Stream,
                               ProtocolType.Tcp);

  this.socket = newSock;
  IPAddress host = Dns.Resolve(server).AddressList[0];
  socket.Connect(new IPEndPoint(host, 80));
}
```

```
[InState("open")]
public string GetPage (string url)
{
  this.socket.Send( Encoding.ASCII.GetBytes("GET / HTTP/1.1\r\nHost: " +
                                             server + "\r\nConnection: Close\r\n\r\n"));

  //...
}
```

```
[ChangesState("open", "closed")]
public void Close ()
{
  this.socket.Send(Encoding.ASCII.GetBytes("QUIT\r\n"));
  this.socket.Close();
}
}
```



# Aliasing Challenges

a.Open(); b.Open();

- Legal only if  $a \neq b$

# Fugue Alias Analysis

- Annotations
  - NotAliased
    - Field or param is unique pointer to an object
    - Allows type system to track state changes
    - Warning (lost track of object) if assigned to Escaping parameter
  - MaybeAliased
    - May have aliases
    - May not call state-changing functions
    - If not escaping, error if assigned to field or passed to Escaping parameter
  - Escaping
    - A MaybeAliased parameter that may be (transitively) assigned to a field



# Fugue Alias Analysis

- Analysis information
  - Environment env: var → addr
  - Capabilities: addr → aliasInfo
  - aliasInfo: one of NotAliased, MaybeAliased, MaybeAliased/Escaping

# Example: Alias Analysis

```
void f([MaybeAliased][Escaping] x);  
void g([MaybeAliased] x);
```

```
void h([NotAliased] y) {
```

```
    z = y;
```

```
    v = new T();
```

```
    g(z);
```

```
    f(v);
```

```
}
```

<u>Environment</u>	<u>Capabilities</u>
$y \rightarrow a$	$a \rightarrow \text{NA}$

$y \rightarrow a, z \rightarrow a$	$a \rightarrow \text{NA}$
------------------------------------	---------------------------

$y \rightarrow a, z \rightarrow a, v \rightarrow b$	$a \rightarrow \text{NA}, b \rightarrow \text{NA}$
---	--

$y \rightarrow a, z \rightarrow a, v \rightarrow b$	$a \rightarrow \text{NA}, b \rightarrow \text{NA}$ <i>a still NotAliased</i>
---	---

$y \rightarrow a, z \rightarrow a, v \rightarrow b$	$a \rightarrow \text{NA}, b \rightarrow \text{MBA}$ <i>Warning: lost track of b</i>
---	--

# Flow Functions

- `init`
  - initialization based on param. annotations
- `x = y`
  - `env [x → env[y]]`
- `x = new T()`
  - `env[x → a]`
    - $a \notin \text{domain}(\text{cap})$
  - `cap[a → NotAliased]`
- `x = y.f`
  - *[slightly simplified rule]*
  - `env[x → a]`
    - $a \notin \text{domain}(\text{cap})$
  - `cap[a → annot(f)]`
- `x = f(y)`
  - if `cap[env[y]] == NotAliased` && `annot(f_arg) == Escaping`  
`warn("lost track of y")`  
`cap[env[y] → MaybeAliased]?`
  - `env[x → a]`
    - $a \notin \text{domain}(\text{cap})$
  - `cap[a → annot(f_return)]`
- Analysis is underspecified in paper
  - How to perform joins?
  - How to model MaybeAliased params?

# Type State Analysis

- Extended analysis information
- Environment
  - Symbolic address for references
  - Also stores constants (for constant prop.)
- Capabilities
  - Aliasing state
  - Symbolic object state
  - Contents of fields (symbolic addresses)

# Example: Type State Analysis

```
[WithProtocol("raw", "bound", "connected",  
"down")]
```

```
class Socket {  
  ...  
  [InState("connected")]  
  public int Send(...);  
  [Disposes(State.Any)]  
  public void Close();  
}
```

```
[WithProtocol("open", "closed")]
```

```
class WebPageFetcher {  
  [InState("connected",  
WhenEnclosingState="open"),  
NotAliased(WhenEnclosingState="open")]  
  private Socket socket;  
  ...  
  [ChangesState("open", "closed")]  
  public void Close() {  
    Socket sock = this.socket;  
    sock.Send(...);  
    sock.Close();  
  }  
}
```

2/22/2005

## Analysis Information

- Entry to Close
  - env: this  $\rightarrow a_0$
  - cap:  $a_0 \rightarrow (\text{WebPageFetcher}, \text{NA}, \text{"open"}, \emptyset)$
- Socket sock = this.Socket;
  - env: this  $\rightarrow a_0$ , sock  $\rightarrow a_1$
  - cap:  $a_0 \rightarrow (\text{WebPageFetcher}, \text{NA}, \text{"open"}, \{\text{socket} \rightarrow a_1\})$ ,  
 $a_1 \rightarrow (\text{Socket}, \text{NA}, \text{"connected"}, \emptyset)$
- sock.Send(...);
  - verify: sock in "connected" state (yes)
- sock.Close();
  - verify: sock in State.Any
  - verify: env[sock] is NotAliased
  - env: this  $\rightarrow a_0$ , sock  $\rightarrow a_1$
  - cap:  $a_0 \rightarrow (\text{WebPageFetcher}, \text{NA}, \text{"open"}, \{\text{socket} \rightarrow a_1\})$
  - sock and this.socket become dangling
- Exit of Close
  - verify: env[sock]  $\notin$  cap

# Experience

- Web server application
  - 16,000 lines of code
  - Well tested, deployed
  - Checked DB library usage
- Errors
  - Disposing command object (17 times)
  - Closing DB connections (9 times)
    - Could cause end of resources
- Observations
  - Added states to objects to track initialization
  - Annotated 24 methods and 6 fields
    - 3 more methods used library only intra-procedurally
- *How would Metal have done?*

# Fugue vs. Metal, PREfix

- Fugue
  - Manual annotations
  - Can find inter-procedural errors
  - Tracks aliases for soundness
- Metal
  - Fully automatic (once protocol specified)
  - Finds only intra-procedural errors
  - Unsound
- PREfix
  - Fully automatic
  - Finds only language errors
  - Unsound