See the Pet in the Beast: How to Limit Effects of Aliasing

Franz Puntigam

Vienna University of Technology
Vienna, Austria

franz@complang.tuwien.ac.at

Aliasing = Pet = Beast

Goal:

- Limit effects of aliasing without restricting aliasing itself

Approach:

- Types specify what can be done through object references
- Object references carry tokens as limited resources
- Static type checking ensures proper use of tokens

Specifying Abstract Object States

```
class Window {
   Window[init] () {...}
   [init -> shown,ready] void initialize (...) {...}
   [ready -> ready] void update (...) {...}
   [shown -> icon] void iconify () {...}
   [icon -> shown] void uniconify () {...}
   [shown,ready ->] void close () {...}
   int getCreationTime () {...}
}
```

Tokens in Reference Types

```
class Test {

   void foo (Window[icon -> shown] w)
      { w.uniconify(); }

   Window[ready] win;

   [unique -> unique] Window[ready] set (Window[ready->] w)
      { Window[ready] old = win; win = w; return old; }

   Test[unique] () { win = null; }
}
```

Type Checking

- Separate compilation (class by class)

Type checker ensures:

- Cannot have several tokens of same name for same object
- Invocations only through references carrying needed tokens
- No token duplication when introducing aliases
- Exclusive access to instance variables

Tokens Depending on Tokens

```
class IconButtons {

   IconButtons[up] (Window[shown->] w) { window = w; }

   Window[icon for down][shown for up] window;

   [down -> up] void pressUp() { window.uniconify(); }

   [up -> down] void pressDown() { window.iconify(); }
}
```

Tokens Depending on Values

```
class SwapButton {

   SwapButton[unique] (Window[shown->] w)
      { window = w; state = 1; }

   int state;
   Window[icon if state < 0][shown if state > 0] window;

   [unique -> unique] void press() {
      if (state < 0) { window.uniconify(); state = 1; }
      else if (state > 0) { window.iconify(); state = -1; }
   }
}
```

Tokens as Values

```
class SwapButton2 {

   SwapButton2[unique] (Window[shown->] w)
      { window = w; }

   Window[?] window;

   [unique -> unique] void press() {
      if ([icon]window) { window.uniconify(); }
      else if ([shown]window) { window.iconify(); }
   }
}
```

Genericity

```
class IconList<W extends Window> {
   IconList[i] () {}
   [i -> i] void add (String s, W[icon] w) {...}
   [i -> s] void uniconifyAll () {...}
   [s -> s] W[shown] delete (String s) {...}
   [s -> s] W get (String s) {...}
}

class IconList2<W[?] extends Window> {...}
```

Conclusions

- Tokens associated with references limit effects of aliasing
- Static type checking ensures proper use of tokens
- Most difficult part: Tokens carried by shared variables
- Future work: Token inference for shared variables