

Inferencia de facetas de declasificación

Propuesta de solución

Matías Meneses C.

1. Descripción del Problema
2. Inferencia Global vs Inferencia Local
3. Propuesta: Proceso iterativo de inferencia asistida
4. Resolución de Constraints

Descripción del Problema

Se quiere implementar una herramienta de inferencia de facetas públicas para un sistema de tipos de dos facetas, en el contexto de information flow control.

Se quiere implementar una herramienta de inferencia de facetas públicas para un sistema de tipos de dos facetas, en el contexto de information flow control.

- El objetivo es facilitar el trabajo al programador, evitando que anote manualmente las anotaciones y cree las interfaces respectivas.

Se quiere implementar una herramienta de inferencia de facetas públicas para un sistema de tipos de dos facetas, en el contexto de information flow control.

- El objetivo es facilitar el trabajo al programador, evitando que anote manualmente las anotaciones y cree las interfaces respectivas.
- La faceta privada es dada.

Inferencia Global vs Inferencia Local

La inferencia global es un problema no decidible, y la presencia de subtyping no mejora la situación.

```
class Person {  
    Person foo(Person a, Person b) {  
        return a.bar(b);  
    }  
  
    Person bar(Person a) {  
        return a.foo(null, null);  
    }  
}
```


Inferencia Global vs Inferencia Local

La inferencia global es un problema no decidible, y la presencia de subtyping no mejora la situación.

```
class Person {  
    @??? Person foo(@PersonBar Person a, @PersonFoo Person b) {  
        return a.bar(b);  
    }  
  
    @??? Person bar(@PersonFoo Person a) {  
        return a.foo(null, null);  
    }  
}
```

Podemos realizar inferencia local, exigiendo que la firma de los métodos se encuentre completamente anotada.

Podemos realizar inferencia local, exigiendo que la firma de los métodos se encuentre completamente anotada. Mucho trabajo para el programador!

Si una expresión exterior al método actual tiene una faceta pública no definida, entonces se infiere **top** para parámetros de métodos y variables, y **bottom** para tipos de retorno de métodos.

Si una expresión exterior al método actual tiene una faceta pública no definida, entonces se infiere **top** para parámetros de métodos y variables, y **bottom** para tipos de retorno de métodos.

- Si el parámetro de un método es \top , entonces cualquier llamada $e1.m(e2)$ será válida, puesto que $t2 \leq \top$.

Si una expresión exterior al método actual tiene una faceta pública no definida, entonces se infiere **top** para parámetros de métodos y variables, y **bottom** para tipos de retorno de métodos.

- Si el parámetro de un método es \top , entonces cualquier llamada $e1.m(e2)$ será válida, puesto que $t2 \leq \top$.
- Si una variable es \top , entonces cualquier asignación $e1 = e2$ será válida, puesto que $t2 \leq \top$.

Si una expresión exterior al método actual tiene una faceta pública no definida, entonces se infiere **top** para parámetros de métodos y variables, y **bottom** para tipos de retorno de métodos.

- Si el parámetro de un método es \top , entonces cualquier llamada $e1.m(e2)$ será válida, puesto que $t2 <: \top$.
- Si una variable es \top , entonces cualquier asignación $e1 = e2$ será válida, puesto que $t2 <: \top$.
- Si el tipo de retorno de un método es \perp , entonces cualquier asignación $e1 = e2.m(e3)$ será válida, puesto que $\perp <: t1$. Además, si lo usamos como argumento, también cualquier expresión $e1.m1(e2.m2(e3))$ será válida.

```
class Person {  
  Person foo(Person a, Person b) {  
    return a.bar(b);  
  }  
  
  Person bar(Person a) {  
    return a.foo(null, null);  
  }  
}
```

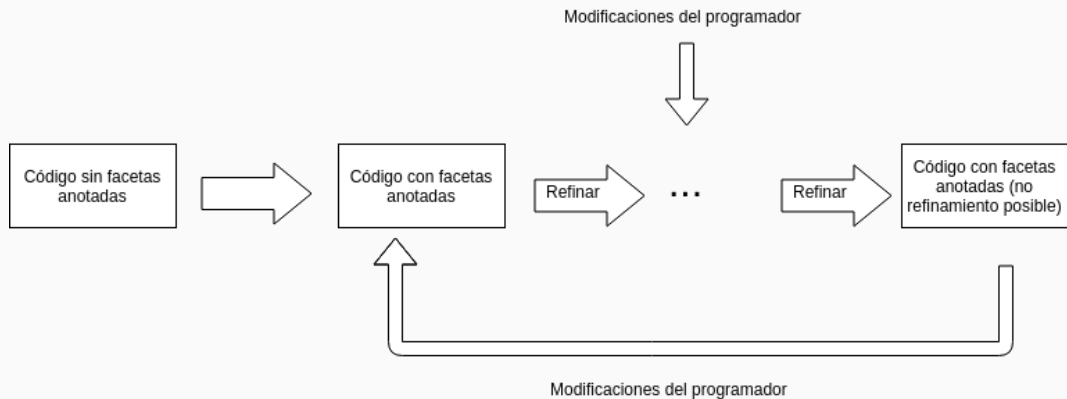


```
class Person {  
    @Person Person foo(@PersonBar Person a, @Top Person b) {  
        return a.bar(b);  
    }  
  
    @Person Person bar(@PersonFoo Person a) {  
        return a.foo(null, null);  
    }  
}
```

```
class Person {  
    @Person Person foo(@PersonBar Person a, @PersonFoo Person b) {  
        return a.bar(b);  
    }  
  
    @Person Person bar(@PersonFoo Person a) {  
        return a.foo(null, null);  
    }  
}
```

Propuesta: Proceso iterativo de
inferencia asistida

Propuesta: Proceso iterativo de inferencia asistida



Propuesta: Proceso iterativo de inferencia asistida

```
class Person {  
    void foo(Person a, Person b) {  
        a.bar(b);  
    }  
  
    void bar(Person p) {  
        p.baz(p);  
    }  
  
    void baz(Person p) {  
        print(p);  
    }  
}
```

Propuesta: Proceso iterativo de inferencia asistida

```
class Person {  
    void foo(Person a, Person b) {  
        a.bar(b);  
    }
```

```
    void bar(Person p) {  
        p.baz(p);  
    }
```

```
    void baz(Person p) {  
        print(p);  
    }  
}
```



Inferir facetas no definidas.

Propuesta: Proceso iterativo de inferencia asistida

```
class Person {  
    @void void foo(@PersonBar Person a,  
        @Top Person b) {  
        a.bar(b);  
    }  
  
    @void void bar(@PersonBaz Person p) {  
        p.baz(p);  
    }  
  
    @void void baz(@Person Person p) {  
        print(p);  
    }  
}
```

Propuesta: Proceso iterativo de inferencia asistida

```
class Person {  
    @void void foo(@PersonBar Person a,  
        @Top Person b) {  
        a.bar(b);  
    }  
  
    @void void bar(@PersonBaz Person p) {  
        p.baz(p);  
    }  
  
    @void void baz(@Person Person p) {  
        print(p);  
    }  
}
```



Flujo no permitido. Refinar facetas.

Propuesta: Proceso iterativo de inferencia asistida

```
class Person {  
    @void void foo(@PersonBar Person a,  
        @PersonBaz Person b) {  
        a.bar(b);  
    }  
  
    @void void bar(@Person Person p) {  
        p.baz(p);  
    }  
  
    @void void baz(@Person Person p) {  
        print(p);  
    }  
}
```

Propuesta: Proceso iterativo de inferencia asistida

```
class Person {  
    @void void foo(@PersonBar Person a,  
        @PersonBaz Person b) {  
        a.bar(b);  
    }  
}
```

```
@void void bar(@Person Person p) {  
    p.baz(p);  
}
```

```
@void void baz(@Person Person p) {  
    print(p);  
}  
}
```



Flujo no permitido. Refinar facetas.

Propuesta: Proceso iterativo de inferencia asistida

```
class Person {  
    @void void foo(@PersonBar Person a,  
        @Person Person b) {  
        a.bar(b);  
    }  
}
```

```
@void void bar(@Person Person p) {  
    p.baz(p);  
}
```

No hay errores :)

```
@void void baz(@Person Person p) {  
    print(p);  
}  
}
```

Propuesta: Formalización

Se extiende el lenguaje base de Type-based relaxed noninterference para adaptarlo a las necesidades y sintaxis de un subconjunto de Dart, incluyendo referencias, instrucciones condicionales y secuencias de instrucciones.

```
e ::= v | e.m(e) | x
v ::= [z: U => list(m(x)e)]
T,U ::= 0 | TVar
0 ::= Obj(TVar). [list(m: S -> S)]
S ::= T < U
```

x variable
m method label
S security type
U public facet, T private facet

Propuesta: Formalización

Se extiende el lenguaje base de Type-based relaxed noninterference para adaptarlo a las necesidades y sintaxis de un subconjunto de Dart, incluyendo referencias, instrucciones condicionales y secuencias de instrucciones.

```
e ::= v | e.m(e) | x | e;e | e = e | if e then e else e |  
      while e do e | m(x)e  
v ::= [z: U => list(m(x)e)] | DV  
U ::= 0 | TVar | void  
0 ::= Obj(TVar). [list(m: U -> U)]
```

x variable

m method label

U public facet

DV Dart primitive value

Se escriben las reglas de inferencia, que generan un set de constraints a resolver.

Se escriben las reglas de inferencia, que generan un set de constraints a resolver.

(seq)

$$\frac{\Gamma, M, pc, pt1 \vdash e1 : t1 \mid C1 \quad \Gamma, M, pc, pt2 \vdash e2 : t2 \mid C2}{\Gamma, M, pc, pt2 \vdash e1; e2 : t2 \mid C2 \cup C1}$$

Se escriben las reglas de inferencia, que generan un set de constraints a resolver.

(method)

$$\frac{\Gamma, M, pc, pt1 \vdash m : t1 \rightarrow t2 \mid C1 \quad \Gamma, M, pc, pt2 \vdash e : t3 \mid C2}{\Gamma, M, pc, pt \vdash m(x)e : t3 \mid \{t3 <: t2\} \cup C2 \cup C1}$$

Se escriben las reglas de inferencia, que generan un set de constraints a resolver.

(assn)

$$\frac{\Gamma, M, pc, pt1 \vdash e1 : t1 \mid C1 \quad \Gamma, M, pc, pt2 \vdash e2 : t2 \mid C2}{\Gamma, M, pc, pt \vdash e1 = e2 : \text{void} \mid \{t2 <: t1\} \cup \{pc <: t1\} \cup C2 \cup C1}$$

Se escriben las reglas de inferencia, que generan un set de constraints a resolver.

(if)

$$\frac{\begin{array}{l} \Gamma, M, pc, pt1 \vdash e1 : t1 \mid C1 \\ \Gamma, M, pc1, pt2 \vdash e2 : t2 \mid C2 \\ \Gamma, M, pc1, pt3 \vdash e3 : t3 \mid C3 \end{array}}{\Gamma, M, pc, pt \vdash \text{if } e1 \text{ then } e2 \text{ else } e3 : t \mid \{t2 <: t\} \cup \{t3 <: t\} \cup C3 \cup C2 \cup \{t1 <: pc1\} \cup C1}$$

Se escriben las reglas de inferencia, que generan un set de constraints a resolver.

(call)

$$\frac{\Gamma, M, pc, pt1 \vdash e1 : \text{Obj}(t1).[[m : t3 \rightarrow t4]] \mid C1 \quad \Gamma, M, pc, pt2 \vdash e2 : t2 \mid C2}{\Gamma, M, pc, pt \vdash e1.m(e2) : t4 \mid \{t2 <: t3\} \cup \{t1 <: [[m : t3 \rightarrow t4]]\} \cup C2 \cup C1}$$

Resolución de Constraints

Para resolver las constraints se usan los operadores **join** (\vee) y **meet** (\wedge) en la lattice que forman las facetas de declasificación.

- $\{t1 <: t2\}, \{t1 <: t3\} \implies \{t1 <: t2 \wedge t3\}$
- $\{t2 <: t1\}, \{t3 <: t1\} \implies \{t2 \vee t3 <: t1\}$

Sometimes, it is useful to add slides at the end of your presentation to refer to during audience questions.

The best way to do this is to include the `appendixnumberbeamer` package in your preamble and call `\appendix` before your backup slides.

metropolis will automatically turn off slide numbering and progress bars for slides in the appendix.

