Inferencia de facetas de declasificación

Propuesta de solución

Matías Meneses C.

Índice de Contenidos

Descripción del Problema

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.

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.

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

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.

- 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

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(a, this);
  }
}
```

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(a, this);
  }
}
```

Inferencia Global vs Inferencia Local

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

Inferencia Global vs Inferencia Local

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

Inferencia Global Restringida

Si una expresión exterior al método actual tiene una faceta pública no definida, entonces no se generará ninguna constraint (por defecto es bottom).

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

  Person bar(Person a) {
    return a.foo(a, this);
  }
}
```

Inferencia Global Restringida

Si una expresión exterior al método actual tiene una faceta pública no definida, entonces no se generará ninguna constraint (por defecto es bottom).

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

  @Person Person bar(@PersonFoo Person a) {
    return a.foo(a, this);
  }
}
```

Inferencia Global Restringida

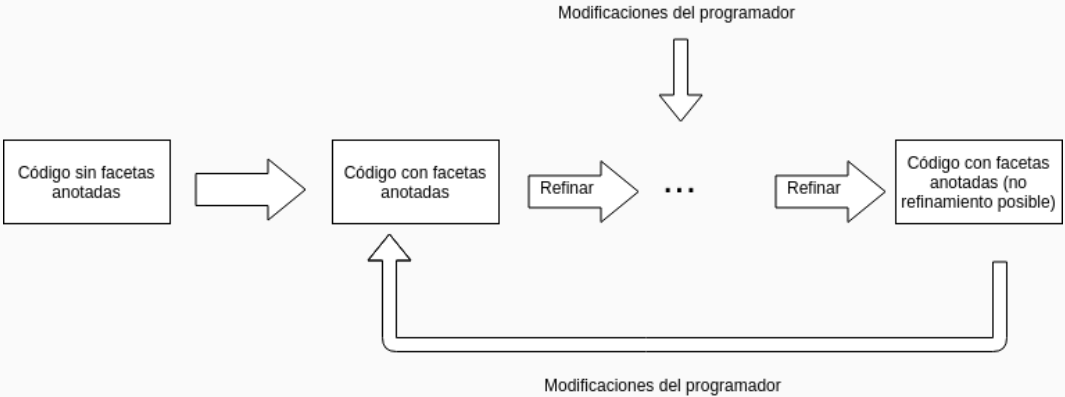
Si una expresión exterior al método actual tiene una faceta pública no definida, entonces no se generará ninguna constraint (por defecto es bottom).

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

  @Person Person bar(@PersonFoo Person a) {
    return a.foo(a, this);
  }
}
```

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,
      @Person 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,
      @Person 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);
  }

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

No hay errores :)

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 ::= O | TVar
  O ::= 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 ::= O | TVar | void
O ::= Obj(TVar). [list(m: U -> U)]

x variable
m method label
U public facet
DV Dart primitive value
```

Propuesta: Formalización

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

Propuesta: Formalización

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

$$\text{(seq)}$$

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

Propuesta: Formalización

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

(method)

$$\frac{\Gamma, \mathrm{M}, \mathrm{pc}, \mathrm{pt1} \vdash \mathrm{m} : \mathrm{t1} \to \mathrm{t2} \,|\, \mathrm{C1} \qquad \Gamma, \mathrm{M}, \mathrm{pc}, \mathrm{pt2} \vdash \mathrm{e} : \mathrm{t3} \,|\, \mathrm{C2}}{\Gamma, \mathrm{M}, \mathrm{pc}, \mathrm{pt} \vdash \mathrm{m(x)e} : \mathrm{t3} \,|\, \{\mathrm{t3} <: \mathrm{t2}\} \cup \mathrm{C2} \cup \mathrm{C1}}$$

Propuesta: Formalización

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

(assn)

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

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

$$\text{(if)}$$

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

Propuesta: Formalización

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

$$\text{(call)}$$

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

Resolución de Constraints

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.

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

Backup slides

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.