Gradual Security Typing (for Java)

APLS 2015

Albert-Ludwigs-Universität Freiburg

UNI FREIBURG

Luminous Fennell, Peter Thiemann

Wednesday 2015-12-16

Introduction

UNI
FREIBURG

Gradual Security Typing

=

apply *Gradual Typing* to *Security Typing*

Introduction

UNI FREIBURG

Gradual Security Typing

=

apply **Gradual Typing** to *Security Typing*

Gradual Typing

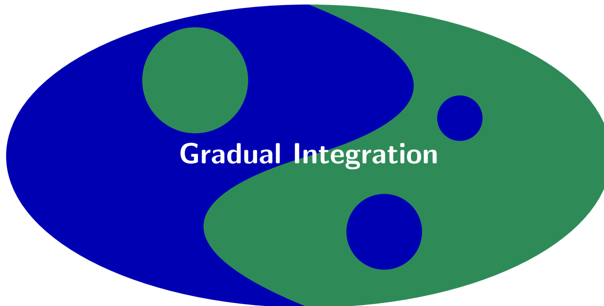
- ⊕ Flexible
- ⊕ Simple
- ⊖ Fragile
- ⊖ Slow

- ⊖ Conservative
- ⊖ or Complex
- ⊕ Robust
- ⊕ Fast

Gradual Typing

UNI FREIBURG

Dynamic Fragment **Static Fragment**



- Define a static type system
- Add a special type **Dynamic**
- and use **casts** at the static/dynamic border

Casts

```
class C {
  int i;

  Dyn maybeIncI(Dyn x) {
      if (i_am_sure_x_is_an_int) {
        this.i = this.i + (int ⇐ Dyn)x
      }
  }
}
...
Dyn s = (Dyn ⇐ String) "Hello";
c.maybeIncI(s)
```

Casts

UNI FREIBURG

```
class C {
  int i;

  Dyn maybeIncI(Dyn x) {
     if (i_am_sure_x_is_an_int) {
       this.i = this.i + (int ⇐ Dyn)x
     }
  }
}
...
Dyn s = (Dyn ⇐ String) "Hello";
c.maybeIncI(s)
```

- Extensions: Datatypes, Higher-order casts, Refinement types, Objects
- Optimization, Inference
- Implementations: C#, Racket, Clojure,...

Introduction

UNI FREIBURG

Gradual Security Typing

=

apply *Gradual Typing* to **Security Typing**

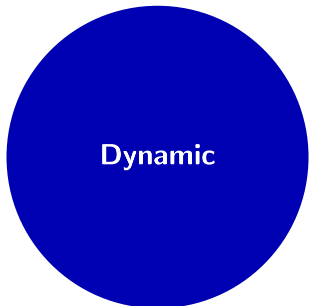
Security Typing

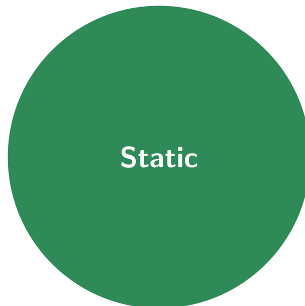
```
class C {
  String<LOW> low;

  int<HIGH> maxWithMessage(int<LOW> x, int<HIGH> y)
     effect { LOW }  {
    if (x ≤ y) { x = y; }
    this.low = "max was called";
    return x;
  }
}
...
int<HIGH> x = c.maxWithMessage(42, secret_pin)
int<HIGH> y = c.maxWithMessage(42, 42) /* subtyping */
int<LOW> z = c.maxWithMessage(42, secret_pin)  /*type error*/
if (secret_pin == 42) {
  c.maxWithMessage(0, 0);  /*type error*/
}
```

UNI FREIBURG

```
class C {
  String<LOW> low;

  int maxWithMessage(int x, int y)
    where { @x ⊑ @return
          , @y ⊑ @return}
    effect { LOW }  {
    if (x ≤ y) { x = y; }
    this.low = "max was called";
    return x;
  }
}
```

Gradual Security Typing



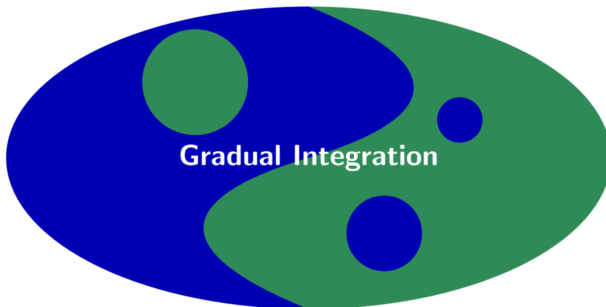


- ⊕ Flexible
- ⊕ Simple
- ⊖ Fragile
- ⊖ (Slow)

- ⊖ Conservative
- ⊖ or Complex
- ⊕ Robust
- ⊕ (Fast)

Gradual Security Typing

Dynamic Fragment **Static Fragment**



- Define a static type system
- Add a special type **Dynamic**
- and use **casts** at the static/dynamic border

Example: Gradual Security Types

```
class D {
  String<HIGH> high; String<LOW> low;
  String<⋆> dyn;

  void doSomeUpdates (String s)
    where {@s ⊑ ⋆ }
    effect { LOW , ⋆ } {
    this.dyn = s;
    this.high = (HIGH ⇚ ⋆) this.dyn;
    if (i_am_sure_s_is_low) {
      this.low = (LOW ⇚ ⋆) s;
    }
  }
}
```

Interpretation of Constraints

UNI FREIBURG

Easy for static security levels:

- $\mathcal{C} = \{\alpha \sqsubseteq \texttt{LOW}, \texttt{HIGH} \sqsubseteq \beta, \alpha \sqsubseteq \beta\} \dots$
- Find a solution for $\alpha, \beta, \dots$
  consistent with the security lattice i.e., such that
  $\texttt{HIGH} \not\leq \texttt{LOW}$.

But how to include "Type Dynamic"?

Interpretation of Constraints

UNI FREIBURG

Easy for static security levels:

- $\mathcal{C} = \{\alpha \sqsubseteq \texttt{LOW}, \texttt{HIGH} \sqsubseteq \beta, \alpha \sqsubseteq \beta\} \dots$
- Find a solution for $\alpha, \beta, \dots$
  consistent with the security lattice i.e., such that
  $\texttt{HIGH} \not\leq \texttt{LOW}$.

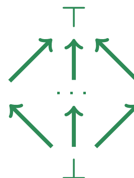
But how to include "Type Dynamic"?

Naive Solution

$$\top \sqsubseteq \star$$

- **if**`(`**this**`.sField)` **this**`.dynField = 42;` is allowed
- Result:
  - no clear separation of the static/dynamic fragments
  - run-time checks with static types "all over the place"

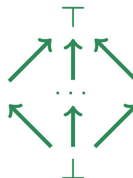
Including “Type Dynamic”



static information

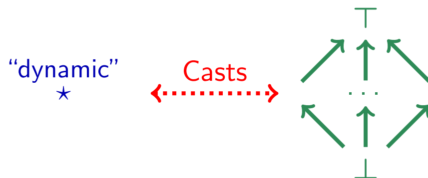
Including “Type Dynamic”

“dynamic”
$\star$



dynamic information **static information**

Including "Type Dynamic"



dynamic information **static information**

Type Dynamic for Values

```
int max(int x, int y)
  where { @x ⊑ @return, @y ⊑ @return } {
  if (x ≤ y) {
    x = y;
  }
  return x;
}
```

Type Dynamic for Values

UNI FREIBURG

```
int max(int x, int y)
  where { @x ⊑ @return, @y ⊑ @return } { ... }


int<⋆> d1; int<⋆> d2; int<HIGH> sH;

void callingMax() where { } effect { ⋆ , HIGH } {
  this.d1 = max(this.d1, this.d2);    // ok
  this.sH = max(this.sH, this.sH);    // ok




}
```

Type Dynamic for Values

```
int max(int x, int y)
  where { @x ⊑ @return, @y ⊑ @return } { ... }


int<⋆> d1; int<⋆> d2; int<HIGH> sH;

void callingMax() where { } effect { ⋆ , HIGH } {
  this.d1 = max(this.d1, this.d2);   // ok
  this.sH = max(this.sH, this.sH);   // ok
  this.d1 = max(this.d1, this.sH); // type error
   // no solution for @return
  this.sL = this.d1                // type error
   // ⋆ ⋢ LOW
}
```

Type Dynamic for Values

UNI FREIBURG

```
int max(int x, int y)
  where { @x ⊑ @return, @y ⊑ @return } { ... }


int<⋆> d1; int<⋆> d2; int<HIGH> sH;

void callingMax() where { } effect { ⋆ , HIGH } {
  this.d1 = max(this.d1, this.d2);   // ok
  this.sH = max(this.sH, this.sH);   // ok
  this.d1 = max(this.d1, (⋆ ⇐ HIGH) this.sH); // ok
   // value cast to dynamic
  this.sL = (LOW ⇐ ⋆) this.d1      // run-time error
   // failing value cast from dynamic
}
```

Type Dynamic for Values

```
int max(int x, int y)
  where { @x ⊑ @return, @y ⊑ @return } { ... }


int<⋆> d1; int<⋆> d2; int<HIGH> sH;

void callingMax() where { } effect { ⋆ , HIGH } {
  this.d1 = max(this.d1, this.d2);   // ok
  this.sH = max(this.sH, this.sH);   // ok
  this.d1 = max(this.d1, (⋆ ⇚ HIGH) this.sH); // ok
   // value cast to dynamic
  this.sL = (LOW ⇚ ⋆) this.d1       // run-time error
   // failing value cast from dynamic
}
```

Value casts

- Declare security levels to be represented at run-time
- Check statically unknown security levels

Type Dynamic for Contexts

```
int<⋆> d1; int<⋆> d2; int<HIGH> sH;

void updates() where { } effect { ⋆, HIGH } {
  if (this.d1 == 42) {
    this.d2 = this.d1; // ok


  }
  if (this.sH == 42) {
    this.sH += 1; // ok


  }
}
```

Type Dynamic for Contexts

UNI FREIBURG

```
int<⋆> d1; int<⋆> d2; int<HIGH> sH;

void updates() where { } effect { ⋆, HIGH } {
  if (this.d1 == 42) {
    this.d2 = this.d1; // ok
    this.sH = 42 // type error
    // static update in dynamic context
  }
  if (this.sH == 42) {
    this.sH += 1; // ok
    this.d1 = this.d2 // type error
    // dynamic update in static context
  }
}
```

Type Dynamic for Contexts

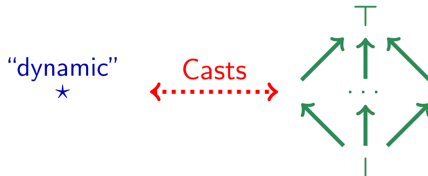
UNI FREIBURG

```
int<⋆> d1; int<⋆> d2; int<HIGH> sH;

void updates() where { } and { ⋆, HIGH } {
  if (this.d1 == 42) {
    this.d2 = this.d1; // ok
    (⋆ ⇛ HIGH) {this.sH = 42;} // ok

  }
  if (this.sH == 42) {
    this.sH += 1; // ok
    (HIGH ⇛ ⋆) {this.d1 = this.d2;} // ok

  }
}
```

- Static updates cannot be checked in dynamic contexts
- Dynamic updates need the context represented at run-time

Including "Type Dynamic"

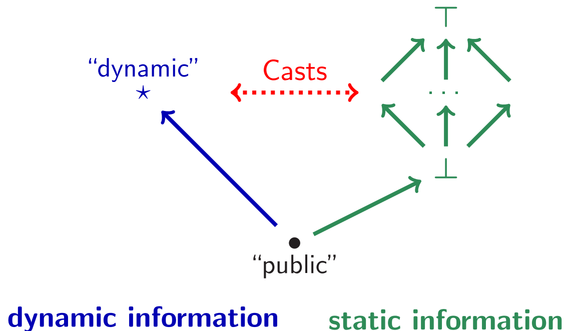
Greatest lower bound of $\{\star, \texttt{HIGH}\}$?



dynamic information **static information**

Including “Type Dynamic”

Greatest lower bound of $\{\star, \texttt{HIGH}\}$?



Public Contexts

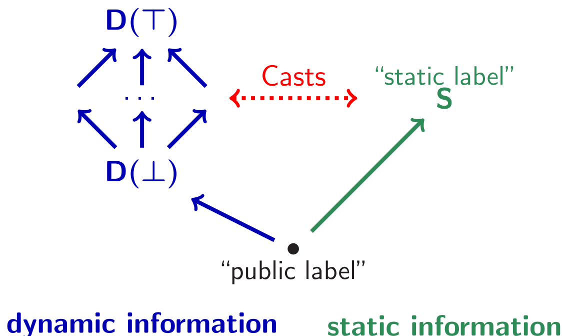
```
int<⋆> d1; int<⋆> d2; int<HIGH> sH;

void updates2() where { } effect { • } {
  this.d2 = this.d1; // ok
  this.s1 = 42       // ok

}
```

A context of type • can accept both, static and dynamic updates.

- ...as it is trivially secure
- calling `update2()` at the top-level is fine
- `if(s1 == 42){ update2(); }` is a type error

Run-Time Labels

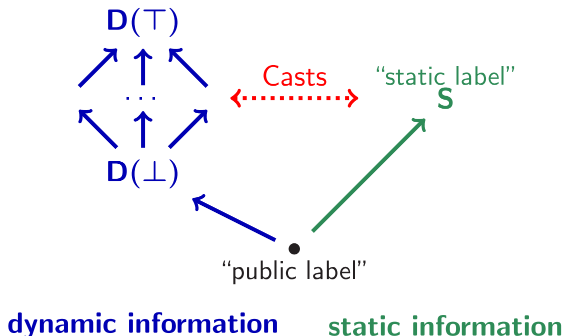
In the LJGS core calculus, all values carry run-time labels. They are the mirror image to security types:



Run-Time Labels

UNI FREIBURG

In the LJGS core calculus, all values carry run-time labels. They are the mirror image to security types:



static labels carry no information and could be erased

Dynamic IFC

UNI FREIBURG

- Via casts from dynamic to static
- for implicit flows:
    - NSU check
    - hybrid monitors
    - facets
    - ...

Conclusion

UNI FREIBURG

- Integrates static and dynamic IFC by gradual typing
- Static and dynamic code fragments interact through casts
- Ready: calculus based on Lightweight Java

Current work:

- Implementation
- Comparisons with other practical systems for IFC (e.g. JIF)