Gradual Security Typing
(for Java)
APLS 2015

Luminous Fennell, Peter Thiemann

Wednesday 2015-12-16
Gradual Security Typing

apply Gradual Typing to Security Typing
Gradual Security Typing

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

Dynamic

- Flexible
- Simple
- Robust
- Fast

Static

- Conservative
- or Complex
- Fragile
- Slow
Gradual Typing

- Define a static type system
- Add a special type \textbf{Dynamic}
- and use \texttt{casts} at the static/dynamic border
Casts

```java
class C {
    int i;

    Dyn maybeIncI(Dyn x) {
        if (i_am_sure_x_is_an_int) {
            this.i = this.i + (int ⇅ Dyn)x
        }
    }
}
```

... 

```java
Dyn s = (Dyn ⇅ String) "Hello";
c.maybeIncI(s)
```
Casts

```java
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, ...
Gradual Security Typing

apply *Gradual Typing* to *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*/
}
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

- Define a static type system
- Add a special type **Dynamic**
- and use **casts** at the static/dynamic border
Example: Gradual Security Types

```java
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

Easy for static security levels:

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

But how to include “Type Dynamic”?
Interpretation of Constraints

Easy for static security levels:

- $\mathcal{C} = \{\alpha \subseteq \text{LOW}, \text{HIGH} \subseteq \beta, \alpha \subseteq \beta\} \ldots$
- Find a solution for $\alpha, \beta, \ldots$
  consistent with the security lattice i.e., such that $\text{HIGH} \not\subset \text{LOW}$.

But how to include “Type Dynamic”?

Naive Solution

\[ T \subseteq \star \]

- 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” ★

dynamic information  static information
Including “Type Dynamic”

```
“dynamic”  Casts  static information
```

```
dynamic information
```

Fennell, Thiemann
Gradual Security Typing
Wednesday 2015-12-16
int max(int x, int y)
where { @x ⊑ @return, @y ⊑ @return } {
    if (x ≤ y) {
        x = y;
    }
    return x;
}
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
}

}
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
}

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

Fennell, Thiemann  Gradual Security Typing  Wednesday 2015-12-16
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

```c
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

```java
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
    }
}
```

Static updates cannot be checked in dynamic contexts
Dynamic updates need the context represented at run-time
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
  }
}
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, \text{HIGH}\} ?

\[ \text{dynamic information} \quad \text{static information} \]
Including “Type Dynamic”

Greatest lower bound of \{\star, \text{HIGH}\}?
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
In the LJGS core calculus, all values carry run-time labels. They are the mirror image to security types:

\[ \text{D}(\top) \]
\[ \text{D}(\bot) \]

Casts

“public label”

“static label”

**dynamic information**

**static information**
Run-Time Labels

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

\[ D(\top) \]

\[ ... \]

\[ D(\bot) \]

Casts

“static label”

“public label”

\[ S \]

dynamic information  static information

static labels carry no information and could be erased
Dynamic IFC

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

- 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)