Transformations you can trust
-- Towards high-confidence program transformations

Yingfei Xiong, Peking University

Collaborators:
Zhenjiang Hu, NII
Jun Li, Peking University
Chenglong Wang, Peking University
About the Speaker

• Yingfei Xiong
• 2012~Now, Assistant Professor @ Peking University
• 2009~2011, University of Waterloo, Postdoc
  • Advisor: Krzysztof Czarnecki
• 2006~2009, The University of Tokyo, Ph.D.
  • Advisor: Zhenjing Hu, Masato Takeichi
• 2004~2006, Peking University, Pre Ph.D. courses
  • Advisor: Hong Mei, Fuqing Yang

• Research Interests:
  • Software Engineering and Programming Languages
Motivation

• API update and API switching may introduce incompatibilities in the client code
Automatic Transformation Tool is Essential

Visual Basic Upgrade Wizard

Android to Blackberry Conversion Tool
Static vs Dynamic Program Adaptation

• **Static Program Transformation**
  • Change client code directly
  • Usually more difficult to implement
  • Incurs no runtime overhead

• **Dynamic Program Adaptation**
  • Create a proxy from the old API to the new API
  • Usually easier to implement
  • Incurs runtime overhead

• **Our target:** static program transformation
Automatic Transformation Tool is not Easy

• Python has an upgrade tool from Python 2.x to Python 3.x
• “Dive into Python 3” use a whole chapter to discuss how to deal with the bugs in the transformation tool.
• Syntax errors, typing errors, runtime errors, silent misbehavior
Our goal: high-confidence (=safety) program transformation language

• A language for program transformation between different APIs
• The programmers describe the mappings between APIs in a declarative way
• The programs using the source API are automatically transformed to programs using the target API
  • No compilation error -- safety in statics
  • No runtime misbehavior -- safety in dynamics
Roadmap

Safety in statics for Java
• Type-safe program transformation

Safety in dynamics for Java
• Behavior-preserving program transformation

Generality
• Transformations beyond a single language
Transformation Language

• Changes from JDK 1.0 to 2.0

```java
void f(Hashtable t) {
    Enumeration e = t.elements():
    while (Object o = e.next()){
        ...
    }
}
```

Old client code

```java
void f(HashMap t) {
    Iterator e = t.values().iterator():
    while (Object o = e.next()){
        ...
    }
}
```

New client code

```
(t: Hashtable -&gt; HashMap)
{
    - t.elements();
    + t.values().iterator();
}
```

Transformation Code
Transformation Language

- Changes in Jboss from 3.2.5 to 3.2.6

```java
SnmpPeer peer = new SnmpPeer(this.address);
peer.setPort(this.port);
peer.setServerPort(this.localPort);
```

Old client code

New client code

```java
SnmpPeer peer = new SnmpPeer(this.address, this.port, this.localPort);
```

(x: String -> String, y: String -> String, z: String -> String)

```java
{
    - SnmpPeer p = new SnmpPeer(x);
    - p.setPort(y);
    - p.setServerPort(z);
    + SnmpPeer p = new SnmpPeer(x, y, z);
}
```

Transformation Code
Problems in statics

• Are the following transformations safe?

```
(t: Hashtable  ->> HashMap)
{
  - t.elements();
  + t.yingfei_xiong();
}
```

```
(t: Hashtable  ->> HashMap)
{
  - t.elements();
  + t.iterator();
}
```

```
(t: Hashtable  t ->> HashMap)
{
  - t.elements();
  + t.iterator().elements();
}
```
Problems in Statics

• Is the following transformation from Swing to SWT safe?

(t: Container --> Composite)
{
  - new Container();
  + new Composite(new Shell() 0);
}
(t: --> HashMap)
{
  - t.elements();
  + t.iterator().elements();
}
Safety in statics: definition

\[ H \leftarrow p : t \implies H \leftarrow T(p) : T(t) \]

- $T$: a transformation written in our language
- $\vdash$: type derivation using Java typing rules
- $H$: hypothesis
- $p$: a Java program
- $t$: a Java type
Safety in statics: basic idea

• Type derivation tree

\[
\begin{align*}
100 &: \text{int} \\
\alpha &: \text{int} \\
2.0 &: \text{float} \\
100 + \alpha &: \text{int} \\
(100 + \alpha) \times 2.0 &: \text{float}
\end{align*}
\]

• Transformation should not break the structure of derivation tree

\[
\begin{align*}
T[[100]] &: T[[\text{int}]] \\
T[[\alpha]] &: T[[\text{int}]] \\
T[[2.0]] &: T[[\text{float}]] \\
T[[100 + \alpha]] &: T[[\text{int}]] \\
T[[\alpha]] &: T[[\text{int}]] \\
T[[\alpha]] &: T[[\text{int}]] \\
T[[(100 + \alpha) \times 2.0]] &: T[[\text{float}]]
\end{align*}
\]
Initial result with single method replacement

• For each code snippet introduced by a transformation rule, the code snippet itself must be well-typed
• The type mapping must form a function
• The type mapping must preserve the subtyping relation
• The transformation rules must cover all type changes between the source API and target API
Problem in dynamics

• Changes in Jboss from 3.2.5 to 3.2.6

SnmpPeer peer = new SnmpPeer(this.address);
peer.setPort(this.port);
peer.setServerPort(this.localPort);

Old client code

SnmpPeer peer = new SnmpPeer(this.address, this.port, this.localPort);

New client code

(x: String -> String, y: String -> String, z: String -> String)
{
- SnmpPeer p = new SnmpPeer(x);
- p.setPort(y);
- p.setServerPort(z);
+ SnmpPeer p = new SnmpPeer(x, y, z);
}

Transformation Code
Problem in dynamics

- Changes in Jboss from 3.2.5 to 3.2.6

Old client code:
```java
SnmpPeer peer = new SnmpPeer(this.address);
If (someCondition) {
    peer.setPort(this.port);
    peer.setServerPort(this.localPort);
}
```

Transformation Code:
```java
(x: String ->> String, y: String ->> String, z: String ->> String)
{
    - SnmpPeer p = new SnmpPeer(x);
    - p.setPort(y);
    - p.setServerPort(z);
    + SnmpPeer p = new SnmpPeer(x, y, z);
}
```
Problem in dynamics

• Changes in Jboss from 3.2.5 to 3.2.6

Old client code

```java
SnmpPeer peer = new SnmpPeer(this.address);
If (someCondition) {
    peer.setPort(this.port);
    peer.setServerPort(this.localPort);
}
```

Transformation Code

```java
(x: String -> String, y: String -> String, z: String -> String)
{
    - SnmpPeer p = new SnmpPeer(x);
    - p.setPort(y);
    - p.setServerPort(z);
    + SnmpPeer p = new SnmpPeer(x, y, z);
}
```
Problem in dynamics

• Changes in Jboss from 3.2.5 to 3.2.6

Old client code:

```java
SnmpPeer peer = new SnmpPeer(this.address);
Port p = peer.getPort();
If (someCondition) {
    peer.setPort(this.port);
    peer.setServerPort(this.localPort);
}
System.out.print(p);  // Old client code
```

Transformation Code:

```java
(x: String ->> String, y: String ->> String, z: String ->> String)
{
    - SnmpPeer p = new SnmpPeer(x);
    - p.setPort(y);
    - p.setServerPort(z);
    + SnmpPeer p = new SnmpPeer(x, y, z);
}
```
Safety in dynamics: definition

\[
\forall i: \Phi(T(p), T(i)) = T(\Phi(p, i))
\]

- \( i \): an input to the program
- \( \Phi \): interpretation function, generating a trace of API invocations from a program and an input
- \( T \): a transformation written in our language
- \( p \): a Java program
Safety in dynamics: ideas

• Work in progress
• $\Phi$ can also be defined by logic rules

\[
\begin{align*}
\Phi[[100]] &= 100 \\
\Phi[[f(100 + a, x)]] &= f(120, 2.0) \\
\Phi[[a]] &= 10 \\
\Phi[[x]] &= 2.0 \\
\Phi[[100 + a]] &= 110
\end{align*}
\]

• Transformation should preserve the derivation
Generality

\[ H \vdash p: t \implies H \vdash T(p): T(t) \]
\[ \forall i: \Phi(T(p), T(i)) = T(\Phi(p, i)) \]

• ⊢ and Φ are given by logic rules
• Find parametric safe conditions working for a class of logic rules
Roadmap again

Safety in statics for Java
• Type-safe program transformation

Safety in dynamics for Java
• Behavior-preserving program transformation

Generality
• Transformations beyond a single language