程序分析与编程语言设计

从哥德尔不完备定理说起

熊英飞
北京大学
2015

报告人介绍 – 熊英飞

- 2000~2004，电子科技大学本科
- 2004~2006，北京大学研究生
  - 导师：梅宏、杨芙清
- 2006~2009，日本东京大学博士
  - 导师：胡振江、武市正人
- 2009~2011，加拿大滑铁卢大学博士后
  - 导师：Krzysztof Czarnecki
- 2012~，北京大学“百人计划”研究员
- 研究方向：软件分析、编程语言设计

软件缺陷可能导致灾难性事故

2003年美加停电事故：由于软件故障，美国和加拿大发生大面积停电事故，造成至少11人丧生

事故原因：电网管理软件内部实现存在重大缺陷，无法正确处理并行事件。

2006年巴西空难：由于防撞系统问题，巴西两架飞机相撞，造成154名人员丧生

事故原因：软件系统没有实现对防碰撞硬件系统故障的检测

2005年，东京证券交易所出现了人类历史上最长停机事故，造成的资金和信誉损失难以估算

事故原因：由于输入错误的升级指令，导致软件版本不匹配

能否彻底避免软件中出现缺陷？

- 程序分析
  - 给定程序，判断程序中是否没有某种类型的缺陷
- 编程语言设计
  - 设计编程语言，使得写出的程序就不具备某种类型的缺陷
- 两个问题是相关的
  - 以通过程序分析作为通过编译的标准

库尔特·哥德尔(Kurt Gödel)

- 20世纪最伟大的数学家、逻辑学家之一
- 爱因斯坦语录
  - “我每天会去办公室，因为路上可以和哥德尔聊天”
- 主要成就
  - 哥德尔不完备定理

希尔伯特计划
Hilbert's Program

- 德国数学家大卫·希尔伯特在20世纪20年代提出
- 背景：第三次数学危机
  - 罗素悖论：$R = \{X \mid X \notin X\}, R \in R?$
- 目标：提出一个形式系统，可以覆盖现在所有的数学定理，并且具有如下特点：
  - 完备性：所有真命题都可以被证明
  - 一致性：不可能推出矛盾，即一个命题要么是真，要么是假，不会两者都是
  - 保守型：任何抽象域导出来的具体结论可以直接在具体域中证明
  - 可判断性：存在一个算法来确定任意命题的真假

哥德尔不完备定理
Gödel's Incompleteness Theorem

- 1931年由哥德尔证明
- 蕴含皮亚诺算术公里的一致系统是不完备的
- 皮亚诺算术公理=自然数
  - 0是自然数
  - 每个自然数都有一个后继
  - 0不是任何自然数的后继
  - 如果b,c的后继都是a，则b=c
  - 自然数仅包含0和其任意多次后继
- 对任意能表示自然数的系统，一定有定理不能被证明

哥德尔不完备定理与程序正确性判定

- 主流程序语言的语法+语义=能表示自然数的形式系统
- 设有表达式T不能被证明
  - if (T) return;
  - buggy_method();
- 若T为永真式，则程序没有缺陷，否则存在缺陷

莱斯定理(Rice's Theorem)

- 我们可以把任意程序看成一个从输入到输出上的部分函数（Partial Function），该函数描述了程序的行为
- 关于程序行为的任何非平凡属性，都不存在可以检查该属性的通用算法
  - 平凡属性：要么对全体程序都为真，要么对全体程序都为假
  - 非平凡属性：不是平凡的所有属性
  - 关于程序行为：即能定义在函数上的属性
- 也就是说，只要某类缺陷有可能在程序中出现，我们就没法精确的判断程序中是否有该类缺陷

难道，世界真的就没有希望了吗？

近似法拯救世界

- 近似法：允许在得不到精确值的时候，给出不精确的答案

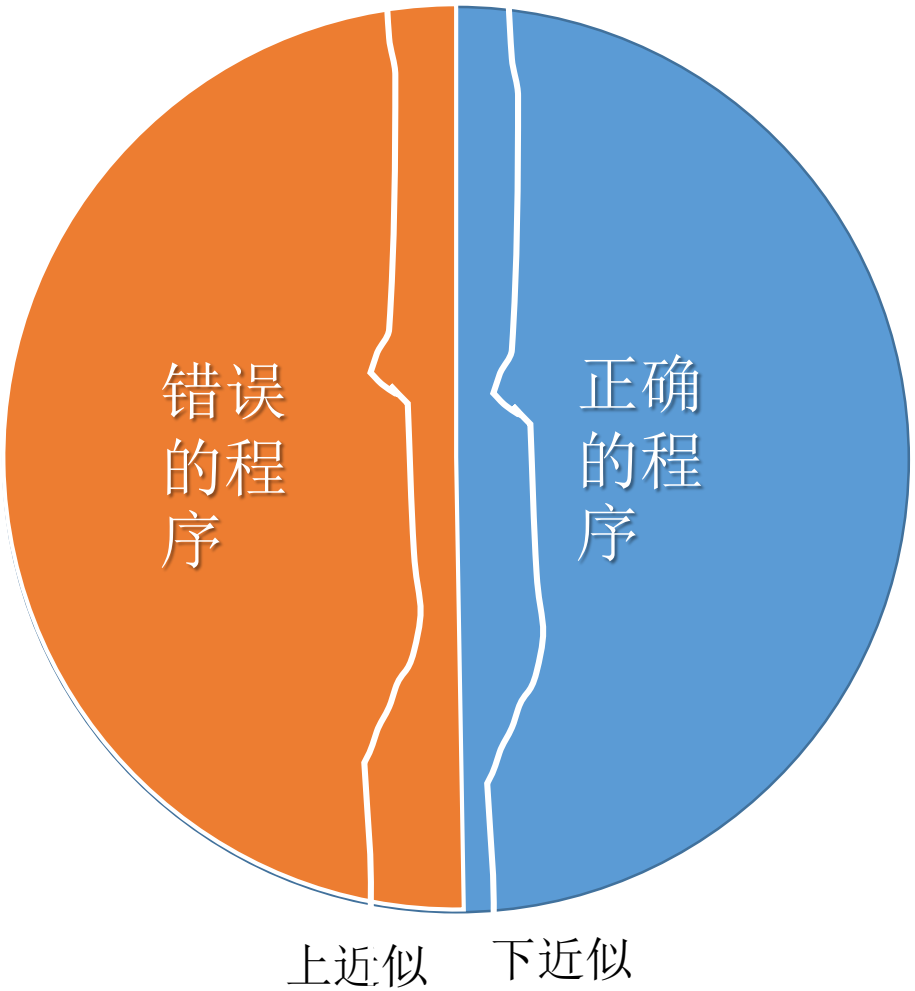
- 对于程序正确性问题，不精确的答案就是
  - 不知道

近似求解判定问题

- 原始问题：输出“正确”或者“错误”
- 近似求解问题：输出“正确”、“错误”或者“不知道”

- 两个变体
  - 只输出“正确”或者“不知道”
    - 下近似(lower approximation)
  - 只输出“错误”或者“不知道”
    - 上近似(upper approximation)

近似法判断程序正确性



各种常见技术

- 程序分析 – 16:50 唐浩、高庆的报告
  - 允许回答程序是“正确”、“错误”、或者“不知道”。
- 编程语言设计 – 稍后介绍
  - 只允许写能确定是正确的程序
  - 利用下近似分析
- 测试 – 14:30 陈振宇老师的报告
  - 只查找确定是错误的程序
  - 利用上近似分析
- 模型检查 – 13:30 田聪老师的报告
  - 高效查找错误
  - 利用上近似分析

求近似解基本原则——抽象

- 给定表达式语言

  term := term + term
  | term – term
  | term * term
  | term / term
  | integer

- 判断结果的符号是正是负

- 限制条件：分析在一台只有一个两位寄存器，没有内存的机器上进行

限制条件

- 无限制条件：直接求出表达式的值，然后查看符号位

- 有限制条件：
  - 无法分析出精确的答案
  - 将结果映射到一个可分析的抽象域

抽象域

- 正 ={所有的正数}
- 零={0}
- 负= {所有的负数}

- 乘法运算规则：
  - 正*正=正
  - 正*零=零
  - 正*负=负
  - 负*正=负
  - 负*零=零
  - 负*负=正
  - 零*正=零
  - 零*零=零
  - 零*负=零

问题

- 正+负=?

- 解决方案：增加抽象符号表示“不知道”
  - 正 ={所有的正数}
  - 零={0}
  - 负= {所有的负数}
  - 踝={所有的整数和NaN}

运算举例

+	正	负	零	粿
正	正			
负	粿	负		
零	正	负	零	
粿	粿	粿	粿	粿

/	正	负	零	粿
正	正	负	零	粿
负	负	正	零	粿
零	粿	粿	粿	粿
粿	粿	粿	粿	粿

如何分析一个程序

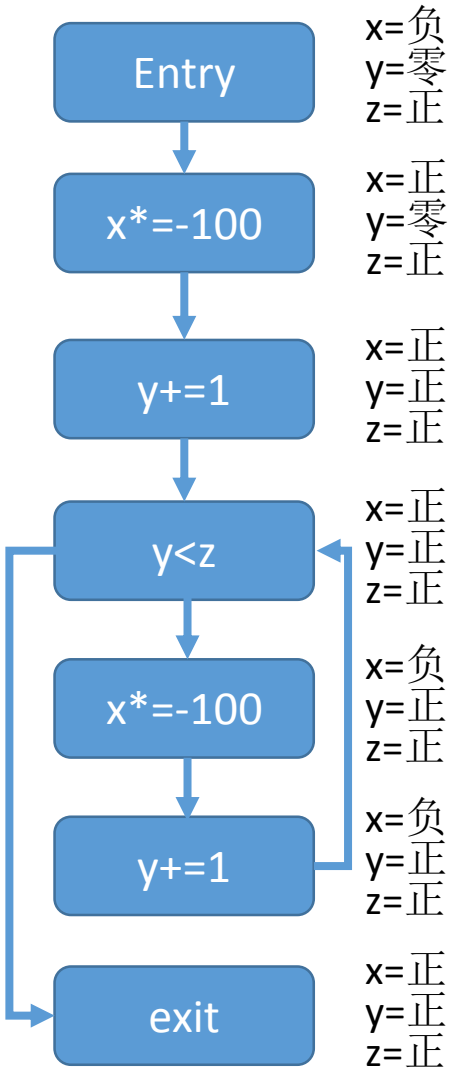

```
x*=-100;
y+=1;
while(y < z) {
    x *= -100;
    y += 1;
}
```

输入：x为负，y为零，z为正

输出：x为踩，y为正，z为正

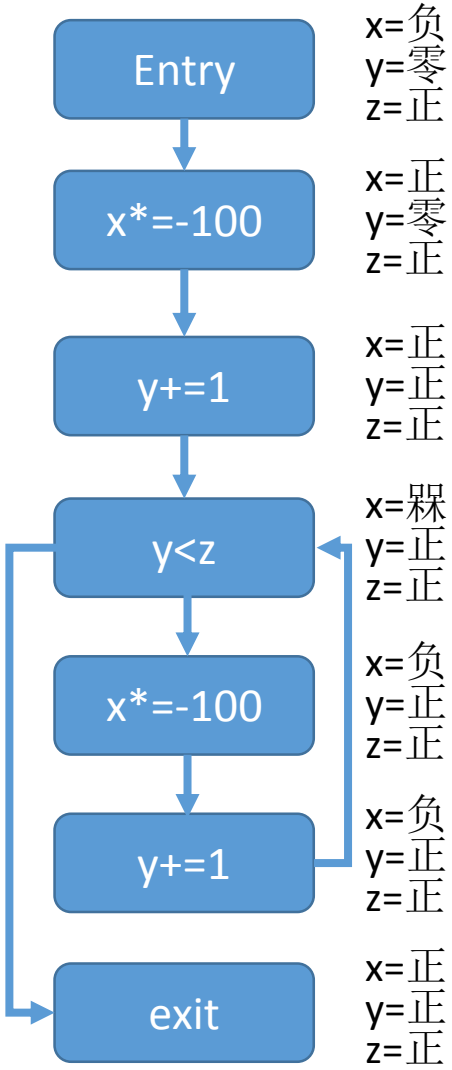
数据流分析

```
x*=-100;
y+=1;
while(y < z) {
    x *= -100;
    y += 1;
}
```



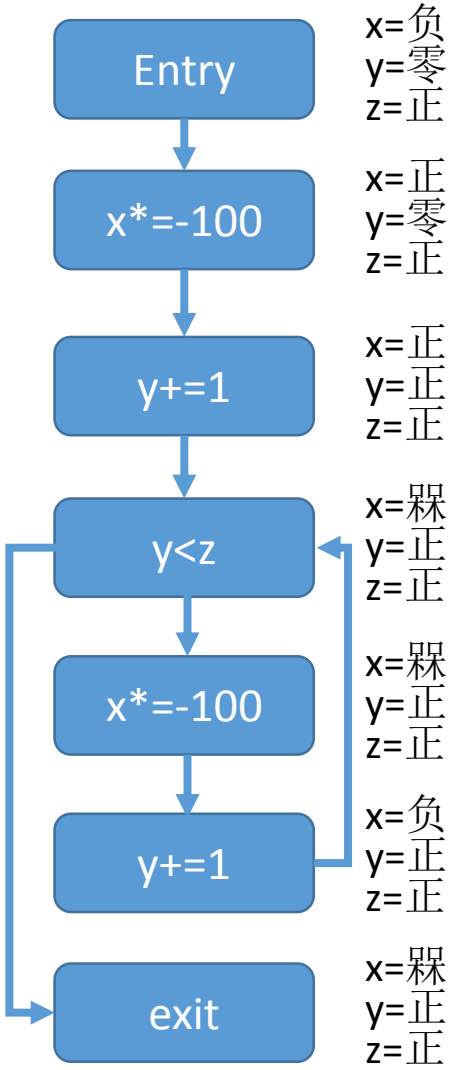
数据流分析

```
x*=-100;
y+=1;
while(y < z) {
    x *= -100;
    y += 1;
}
```



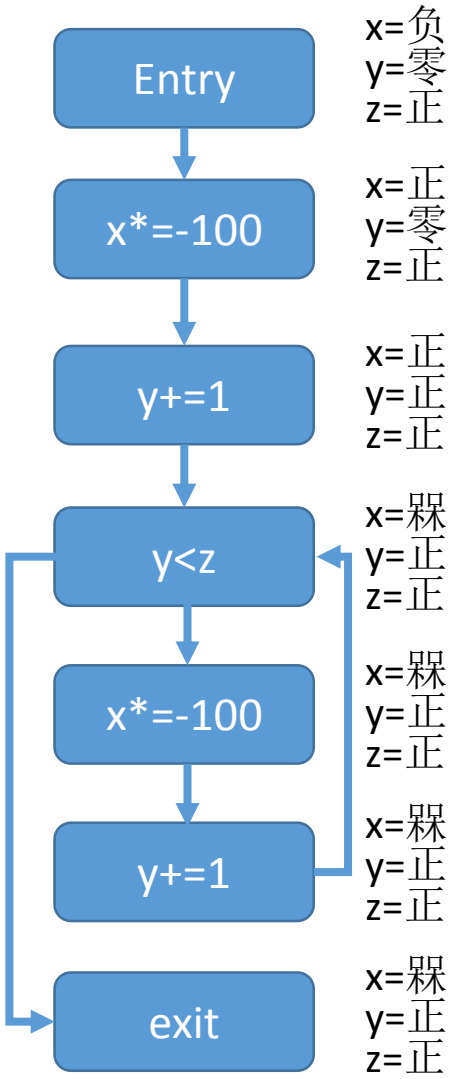
数据流分析

```
x*=-100;
y+=1;
while(y < z) {
    x *= -100;
    y += 1;
}
```



数据流分析

```
x*=-100;
y+=1;
while(y < z) {
    x *= -100;
    y += 1;
}
```



编程语言设计

- 类型系统
  - 表达程序上的约束
  - 避免特定类型的错误
  - 类型推导的过程就是程序分析的过程
- C语言的类型系统
  - 避免不同类型变量一起运算的错误
  - 类型推导过程也就是回答“某个表达式的计算结果是什么类型”的程序分析

示例：一个限定符号的编程语言


```
void (x:负, y:零, z:正):粿 {
    x*=-100;
    y+=1;
    while(y < z) {
        x *= -100;
        y += 1;
    }
    ensures y is 正 /\ z is 正;
    return x;
}
```

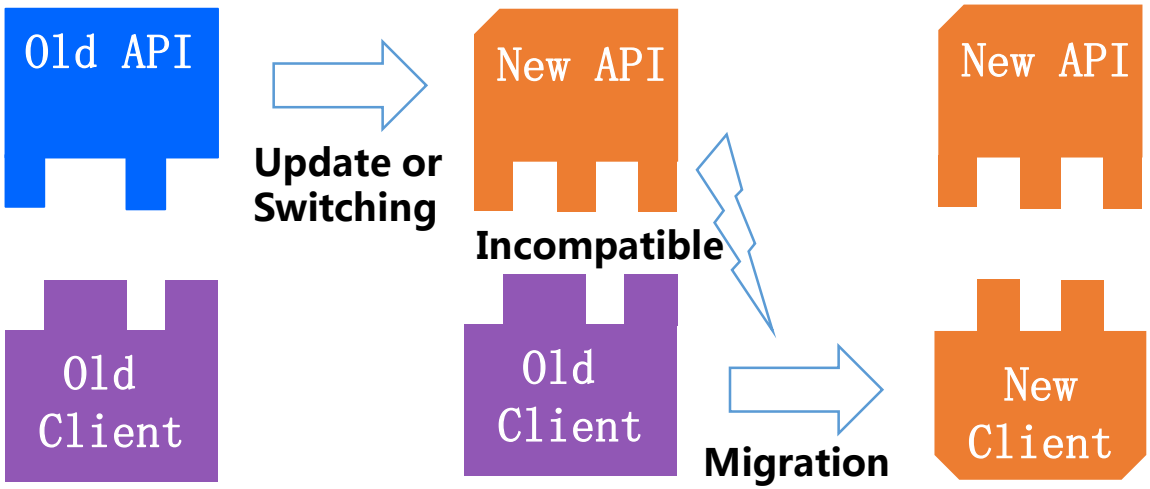
我们小组的近两年研究

- 程序分析基础：
  - 加速过程间分析 （POPL’15, 16:50 唐浩）
- 调试方面的程序分析应用：
  - 基于数据流分析的内存泄露自动修复（ICSE’15 17:10 高庆）
  - 范围修复技术（TSE’14）
  - 克隆的维护（TSE’14）
- 测试方面的程序分析应用
  - 浮点数误差查找（ICSE’15）
  - 测试驱动排序（TOSEM’14）
  - 蜕变关系挖掘（张洁、陈俊洁 ASE’14)
- 编程语言设计：
  - 高可信的程序迁移语言 （PEPM’15, 稍后介绍）

基于搜索的软件工程

高可信的程序迁移语言

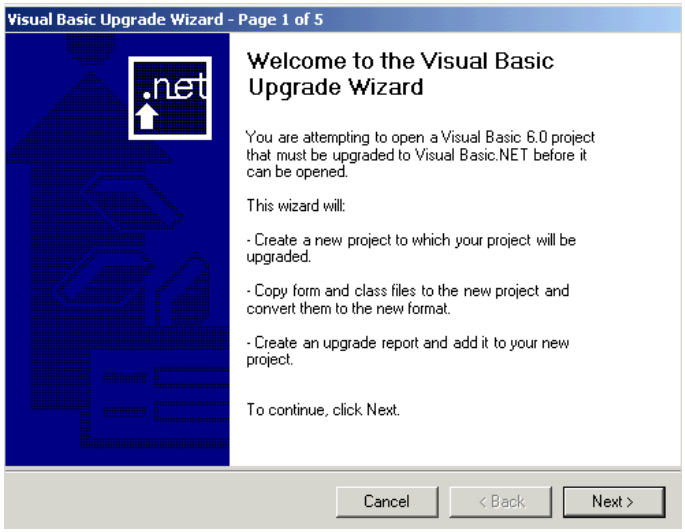
Motivation



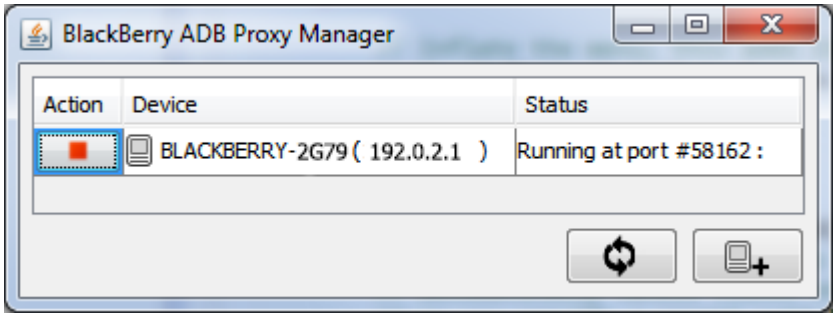
API Migration

- 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

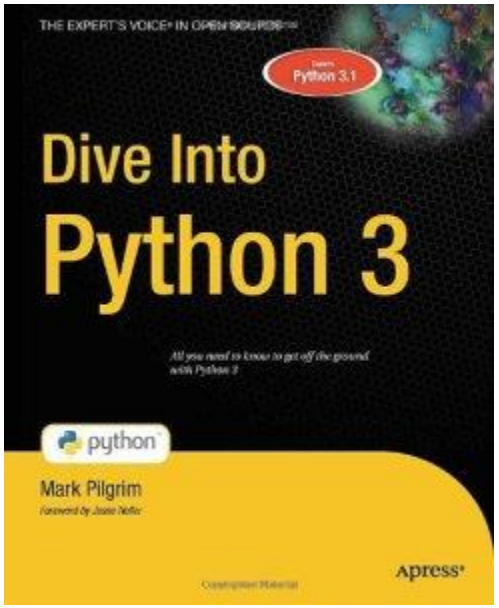
Static vs Dynamic Program Adaptation

- 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: safe 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 (PEPM'14)

- Type-safe program transformation

Safety in dynamics for Java (Submitted)

- Semantics-preserving program transformation

Generality

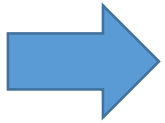
- Transformations beyond a single language

Transformation Language

- Changes from JDK 1.0 to 2.0

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

Old client code



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

New client code

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

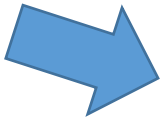
Transformation Code

Transformation Language

- Changes in Jboss from 3.2.5 to 3.2.6

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

Old client code



New client code

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

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

Transformation Code

Problems in statics

- Are the following transformations safe?

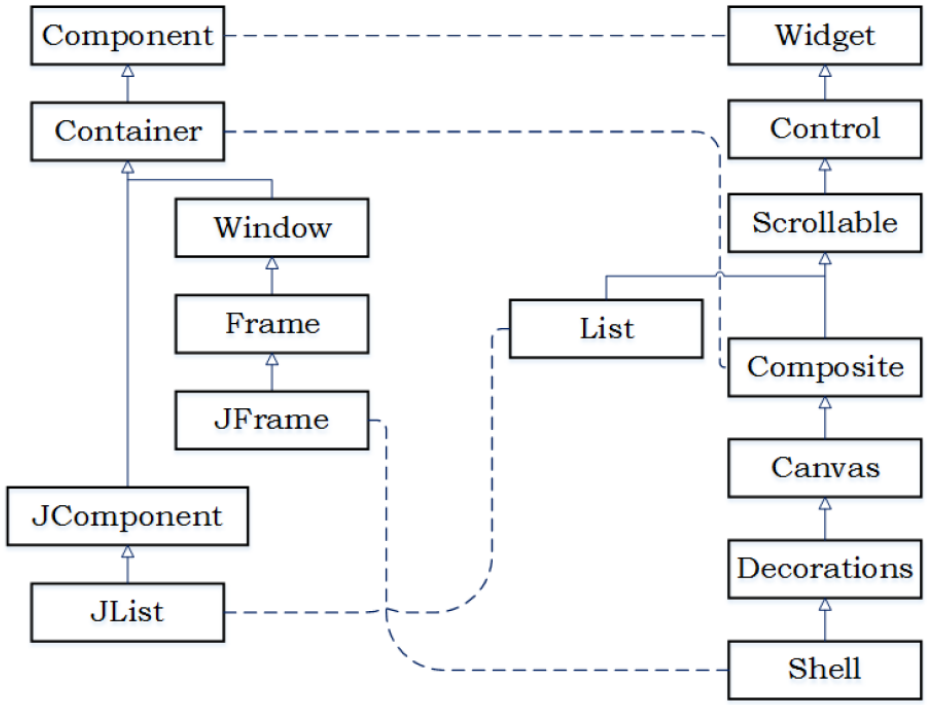
```
(t: Hashtable  ->> HashMap)
{
  - t.elements();
  + t.yingfei_xiong();
}
```

```
(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?

```
() {
  - new Container();
  + new Composite(new Shell() 0);
}
() {
  - new JList();
  + new List();
}
```



Four conditions about safety

- 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

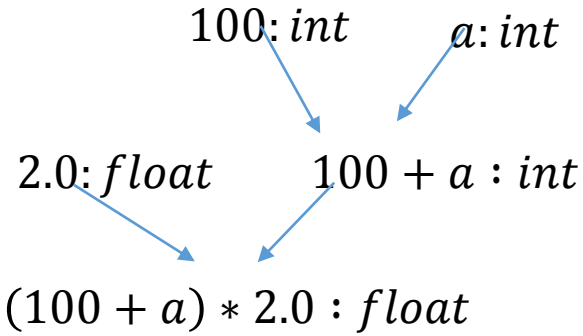
Safety in statics: definition

$$H \vdash p : t \Longrightarrow H \vdash T(p) : T(t)$$

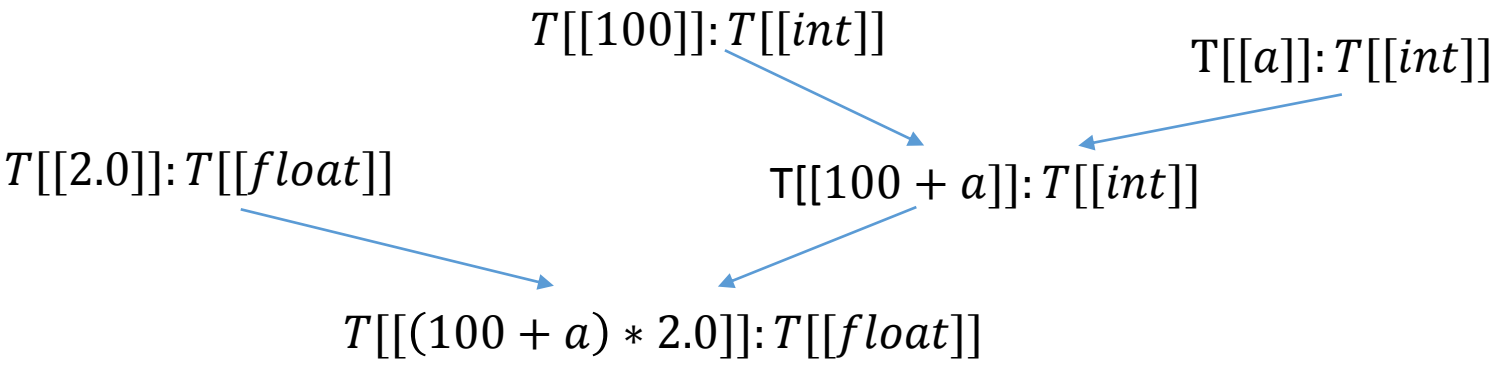
- T: a transformation written in our language
- $\vdash$: type derivation using Java typing rules
- H: hypothesis
- p: a Java program
- t: a Java type

Safety in statics: basic idea

- Type derivation tree



- Transformation should not break the structure of derivation tree



Roadmap

Safety in statics for Java (PEPM'14)

- Type-safe program transformation

Safety in dynamics for Java (Submitted)

- Semantics-preserving program transformation

Generality

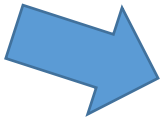
- Transformations beyond a single language

Problem in dynamics: Performing the transformation

- Changes in Jboss from 3.2.5 to 3.2.6

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

Old client code



New client code

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

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

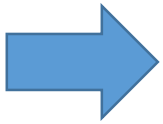
Transformation Code

Problem in dynamics: Performing the transformation

- Changes in Jboss from 3.2.5 to 3.2.6

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

Old client code

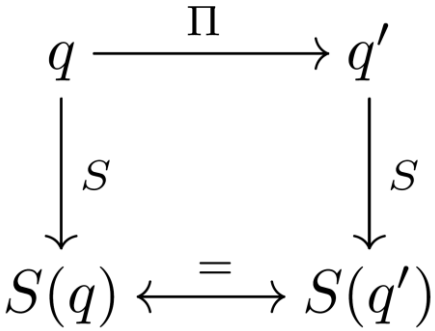


```
(x: String ->> String, y: String ->> String, z: String ->> String)
{
  - p = new SnmpPeer(x);
  - p.setPort(y);
  - p.setServerPort(z);
  + p = new SnmpPeer(x, y, z);
}
```

Transformation Code

Problem in dynamics: Defining the goal

- Absolute semantic equivalence is useless
  - Changing the API changes the semantics
- Abstract semantics
  - Interpretation function: S
  - The transformed program should be equal to the original program on the abstract semantics



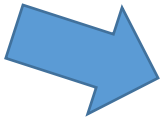
- But we do not know S, it is API/application-specific.

Solution

- Let the basic transformation define abstract semantics equivalence

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

Old client code



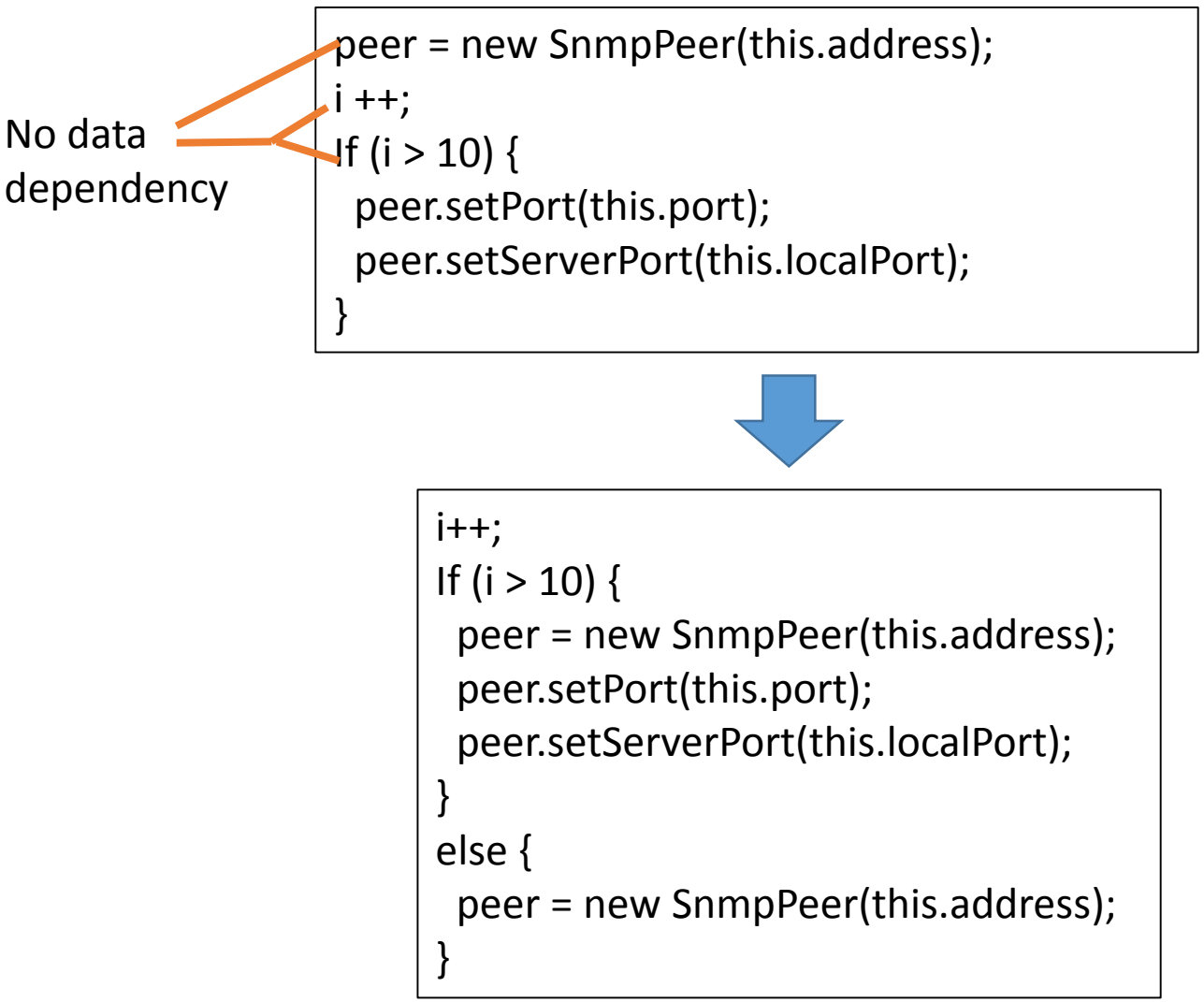
New client code

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

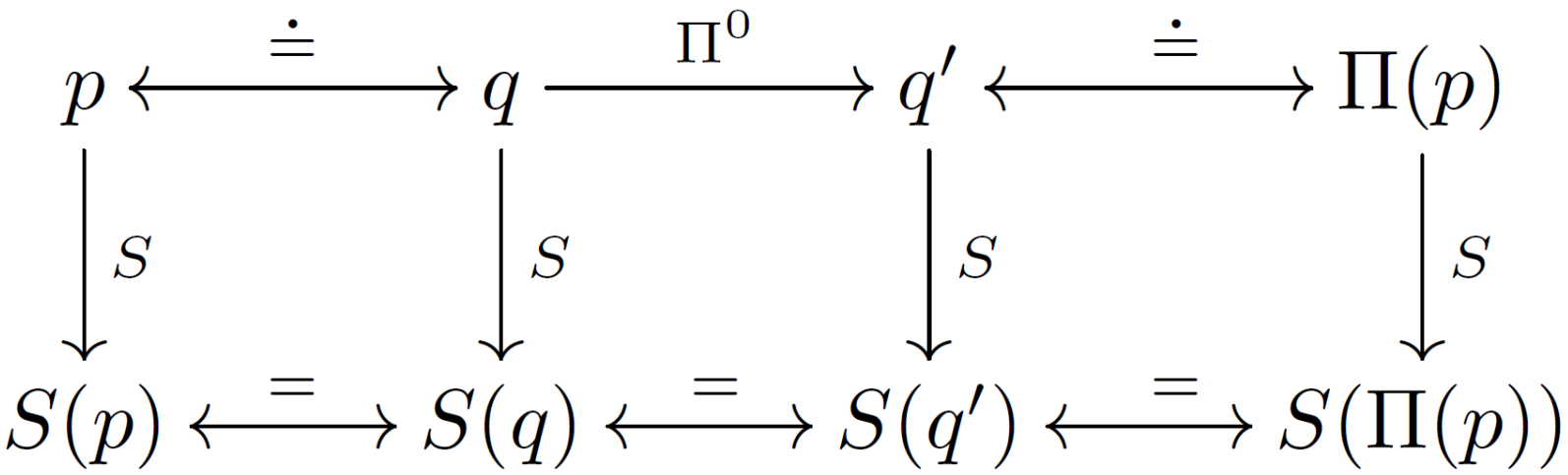
```
(x: String ->> String, y: String ->> String, z: String ->> String)
{
  - p = new SnmpPeer(x);
  - p.setPort(y);
  - p.setServerPort(z);
  + p = new SnmpPeer(x, y, z);
}
```

Transformation Code

Solution: Guided Normalization



The transformation diagram



Evaluation

Table 1. Subject Client Programs

Client	KLOC	Classes	Methods	Case
husacct	195.6	1187	5977	Jdom/Dom4j
serenoa	12.2	52	523	Jdom/Dom4j
openfuxml	112.5	727	4098	Jdom/Dom4j
clinicaweb	3.9	74	213	Calendar
blasd	9.7	199	729	Calendar
goofs	8.6	78	643	Calendar
Total	342.5	2317	12183	–

Evaluation

Table 2. Transformation Rules

Transformation	Rules	Classes	Methods	M-to-m
Jdom/Dom4j	84	12	77	12(14.3%)
Calendar	42	14	45	21(50.0%)
Total	126	26	122	33(26.2%)

Evaluation

Table 3. Results of Transformation

Client	CF	CL	U	W	MM
husacct	42	852	0	0(0%)	0(0%)
serenoa	8	273	0	3(1%)	9(3.3%)
openfuxml	72	983	15	54 (6.6%)	2(0.2%)
clinicaweb	5	81	8	0 (0%)	34(42%)
blasd	5	26	0	7 (21.2%)	13(50%)
goofs	13	100	27	13 (15.3%)	27(27%)
Total	145	2315	50	77(3.2%)	85(3.7%)

CF = number of changed files, CL= number of changed lines, U = unconvertible lines, W = number of lines that generate warnings, percentages in W = W / (W + CL), MM = number of lines that are involved in many-to-many mappings, percentages in MM = MM / CL

Roadmap

Safety in statics for Java (PEPM'14)

- Type-safe program transformation

Safety in dynamics for Java (Submitted)

- Semantics-preserving program transformation

Generality

- Transformations beyond a single language

Generality

- Type checking and evaluation can be expressed by a set of logic rules
- Find a class of rules that cover mainstream languages, and whose safety can be ensured

Summary

- API migration is important
- Automating API migration is difficult
- Many accidental difficulties can be removed by designing a safe transformation language

课程建设
——其他想和大家交流的内容

- 《软件分析技术》
  - 程序分析技术
    - 数据流分析、过程间分析、符号执行、约束求解、抽象解释、模型检查
  - 智能化软件分析技术（微软亚洲研究院张冬梅老师小组）
    - 数据挖掘、信息检索、可视化等技术
- 《编程语言的设计原理》
  - 程序设计语言的类型理论
    - Lambda演算、类型系统的正确性、基本类型、引用类型、例外类型、子类型、递归类型、类型推导、System F等
  - 教材《Types and Programming Language》