Software Verification

ETH Zurich, September-December 2011

Proof-Carrying Code
&
Proof-Transforming Compilation

Chair of Software
Engineering

ETH
Eidgenössische Technische Hochschule Zürich
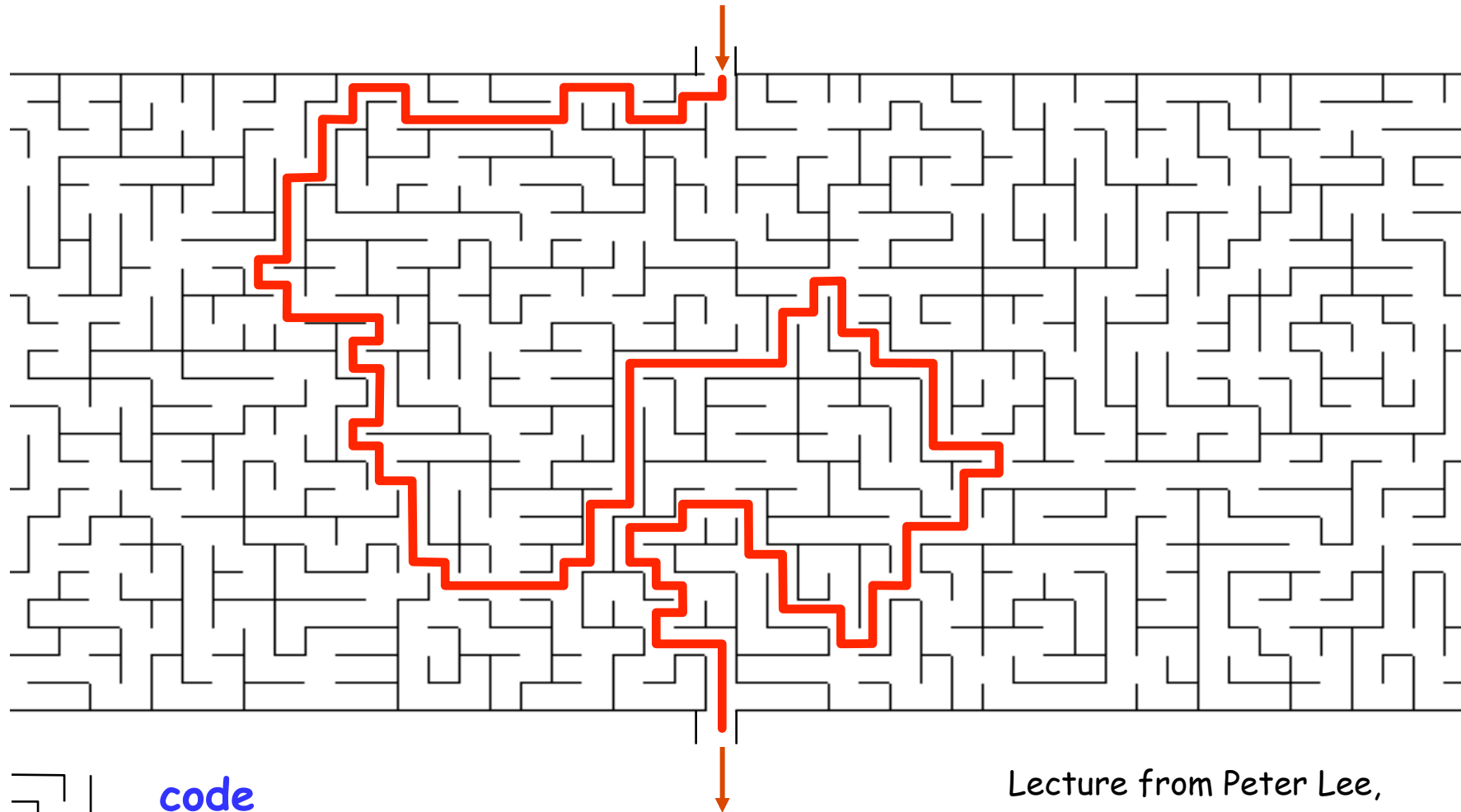Swiss Federal Institute of Technology Zurich

Overview

- Proof-Carrying Code

- Proof-Transforming Compilation
  - Semantics for Java and Eiffel
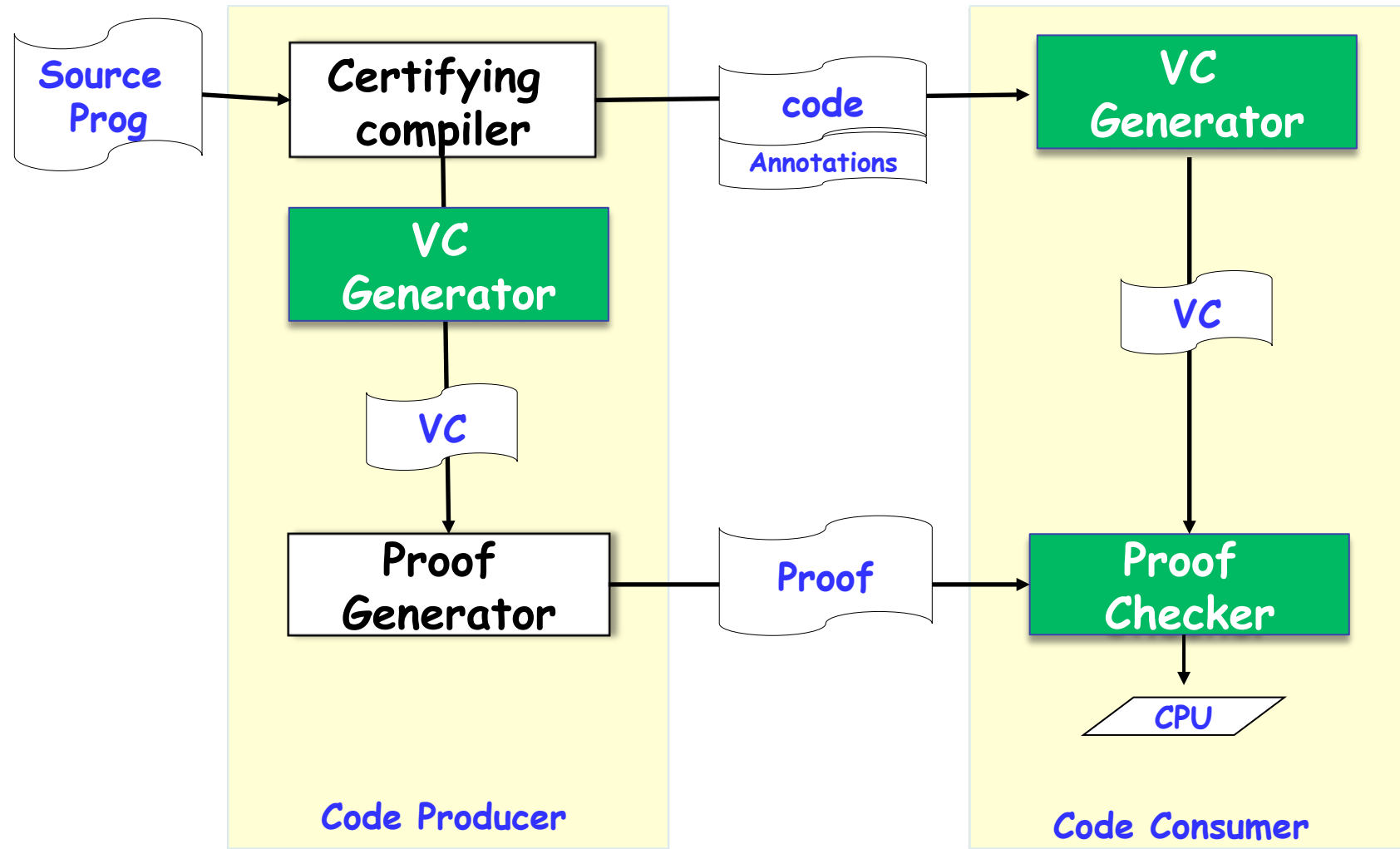 - A Hoare-style logic for Bytecode
  - Proof Translation

Chair of Software
Engineering

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Mobile Code

How to verify mobile code?

Chair of Software
Engineering

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Proof-Carrying Code

code

proof

Lecture from Peter Lee,
2003, University of Oregon

Proof-Carrying Code

Source Prog → Certifying compiler → code / Annotations → VC Generator

Certifying compiler → VC Generator → VC → Proof Generator → Proof → Proof Checker

VC Generator → VC → Proof Checker → CPU

Code Producer

Code Consumer

Chair of Software Engineering

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

What do we gain?

The process of checking the proof is fast and automatic

There is no loss of performance in the bytecode program

The overhead of developing the proof is done once and for all by the code producer

The code consumer does not need to trust the code producer

Chair of Software
Engineering

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich
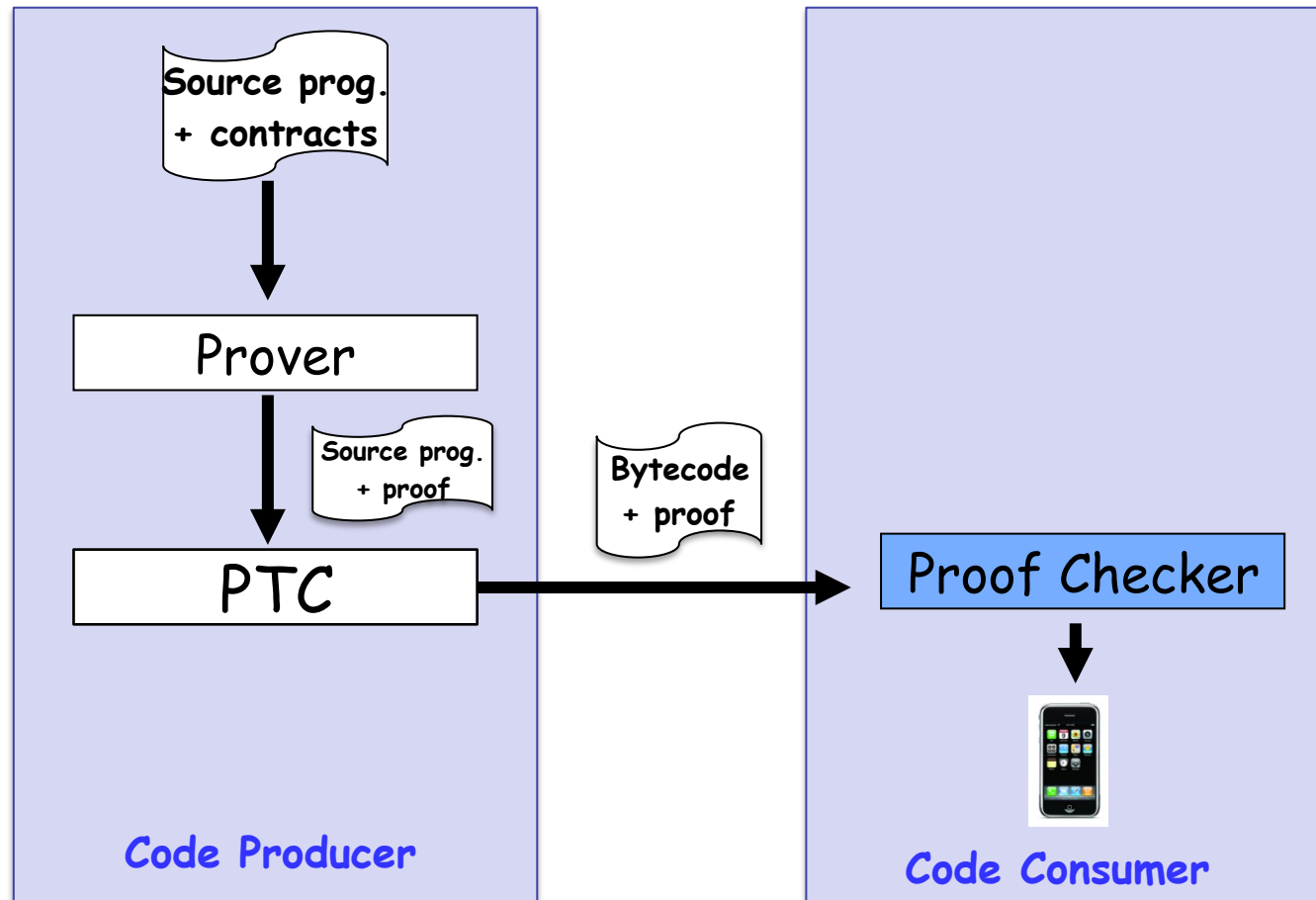
Limitations

Proofs are big

Good for safety but not yet termination

Certifying compilers can generate proof automatically only for a restricted set of properties

In Lee and Necula's implementation, they consider machine code... portability?

Chair of Software
Engineering

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Verification Process based on Proof-Transforming Compilation (PTC)

Source prog. + contracts

↓

Prover

↓

Source prog. + proof

PTC

→

Bytecode + proof

Proof Checker

↓

Code Producer

Code Consumer

☐ untrusted tool   ☐ trusted tool

Chair of Software Engineering

ETH
Eidgenössische Technische Hochschule Zürich
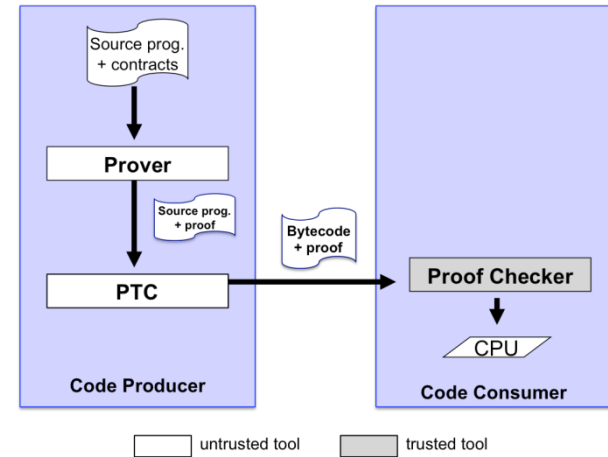Swiss Federal Institute of Technology Zurich

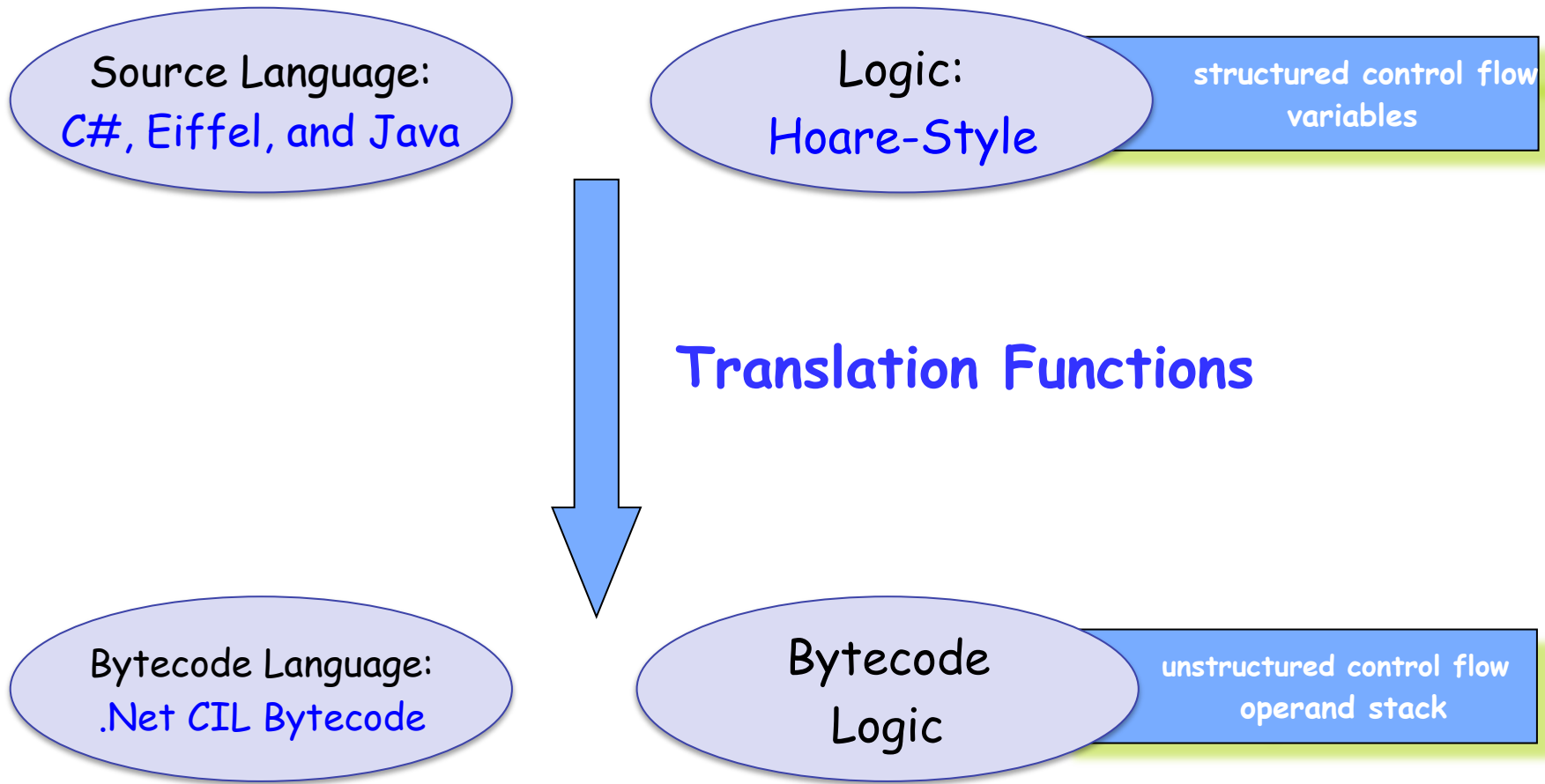# Advantages

Verification of functional properties

PTCs are not part of the trusted computing base

Small trusted computing base: Proof Checker

Verification on the source language

Chair of Software
Engineering

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Basics of our PTC

Source Language:
C#, *Eiffel*, *and Java*

Logic:
Hoare-Style

structured control flow
variables

Translation Functions

Bytecode Language:
.Net CIL Bytecode

Bytecode
Logic

unstructured control flow
operand stack

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Overview

- Proof-Carrying Code

- Proof-Transforming Compilation
  - ➤ Semantics for Java and Eiffel
  - ➤ A Hoare-style logic for Bytecode
  - ➤ Proof Translation

Chair of Software
Engineering

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

The Subset of Java

Assignment and compound

```
foo () {
    int b=1;
    b++;
} b = 2
```

Try-finally and throw

```
foo () {
    int b=1;
    try {
        throw new Exception();
    }
    finally {
        b++;
    }
} b = 2    Exception
```

While and break

```
foo () {
    int b=1;
    while (true) {
        b++;
        break;
    }
} b = 2
```

Other features:
  Try-catch
  If then else
  Read and write fields
  Routine invocation
  Single inheritance

Why is this Subset of Java interesting?

```
foo () {
  int b=1;
  b++;
}

foo () {
  int b=1;
  while (true) {
    b++;
    break;
  }
}

foo () {
  int b=1;
  try {
    throw new Exception();
  }
  finally {
    b++;
  }
}
```

Chair of Software
Engineering

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Why is this Subset of Java interesting?
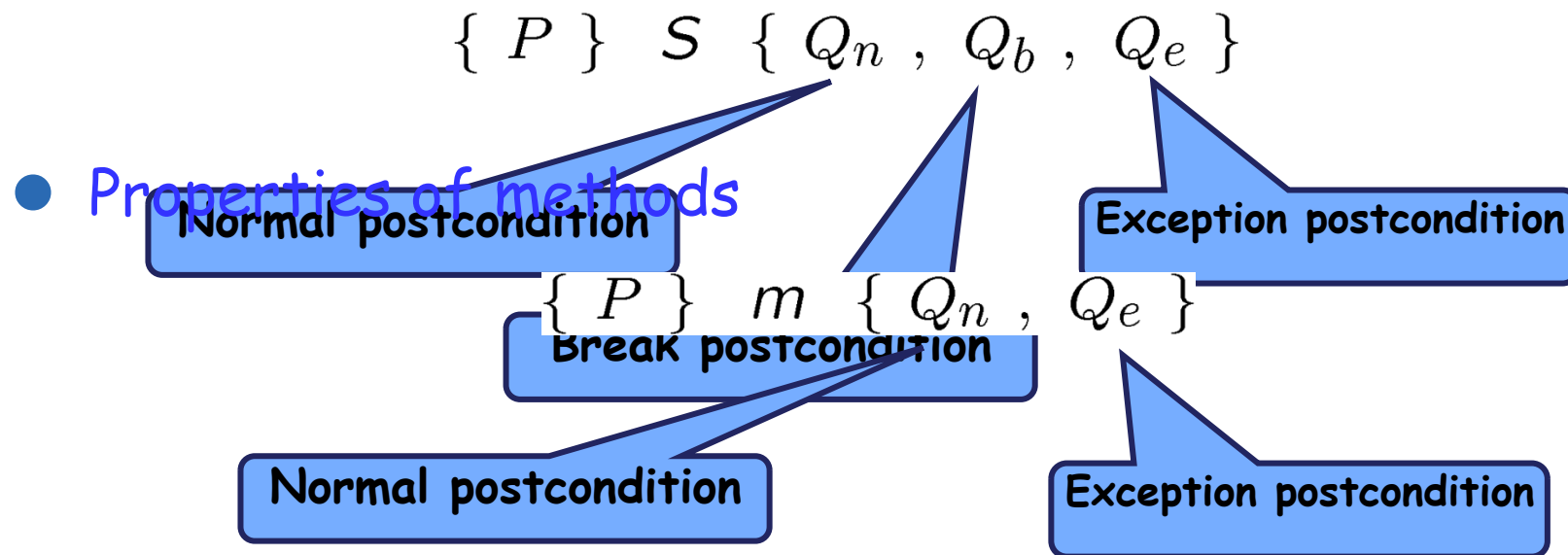
```
foo () {
    int b=1;          b = 1
    while (true) {
        try {
            b++;          b = 2
            throw new Exception ();  b = 2
        }
        finally {
            b++;          b = 3
            break;        b = 3
        }
    }
    b++;          b = 4
}
```

Does this program compile in C#?

Semantics for Java

- Operational and axiomatic semantics
- The logic is based on the programming logic developed by P. Müller and A. Poetzsch-Heffter
- Properties of method bodies are expressed by Hoare triples of the form

$$\{ P \}\ S\ \{ Q_n\ ,\ Q_b\ ,\ Q_e \}$$

- Properties of methods

$$\{ P \}\ m\ \{ Q_n\ ,\ Q_e \}$$

Normal postcondition

Exception postcondition

Break postcondition

Normal postcondition

Exception postcondition

The subset of Eiffel

Basic instructions such as assignments, if then else, and loops

Exception handling: rescue clauses

Once routines

Multiple inheritance

Eiffel: Exception Handling

```
connect_to_server
    --Connect to Madrid, York, or Zurich.
  local
    i : INTEGER
  do
    if i = 0 then connect_to_madrid  end
    if i = 1 then connect_to_york    end
    if i = 2 then connect_to_zurich  end
  rescue
    if i < 3 then
      i := i + 1
      Retry := True
    else
      failed := True
    end
  end
```

Eiffel: Once Functions

```
f (i: INTEGER): INTEGER
    once
        Result := i + 1
    end
```

j := f (2)

{ j = 3 }

k := f (4)

{ j = 3 and k = 3 }

Chair of Software
Engineering

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Semantics for Eiffel

Operational and axiomatic semantics

Based on the logic by P. Müller and A. Poetzsch-Heffter

Properties of routines and routine bodies are expressed by Hoare triples of the form

$$\{ P \} \ S \ \{ Q_n , Q_e \}$$

Normal postcondition

Exception postcondition

Proof of soundness and completeness

Chair of Software Engineering

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Logic: Assignment Rule

$$\{ \begin{array}{l} (safe(e) \ \wedge \ P[e/x]) \ \vee \\ (\neg safe(e) \ \wedge \ Q_e) \end{array} \} \ x \ := \ e \ \{ \ P \ , \ Q_e \ \}$$

Logic: Compound

$$\frac{\{\ P\ \}\ \ s_1\ \{\ ???\ ,\ ???\ \}\quad\{\ ???\ \}\ \ s_2\ \{\ ???\ ,\ ???\ \}}{\{\ P\ \}\ \ s_1; s_2\ \{\ ???\ ,\ ???\ \}}$$

Chair of Software
Engineering

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Example1: Hoare Logic

Example 2: Exceptions

Example

$$\{\ true\ \}\quad balance := b \ \backslash\backslash\ i\ \{\ i \neq 0\ \wedge\ balance = b\ \backslash\backslash\ i\ \wedge\ ,\quad i = 0\ \}$$

Assignment Rule

$$\{i \neq 0\ \wedge\ balance = b\ \backslash\backslash\ i\ \}\quad credit := b + 10\quad \left\{ \begin{array}{l} i \neq 0\ \wedge\ balance = b\ \backslash\backslash\ i\ \wedge \\ credit = b + 10 \end{array}\ ,\quad false \right\}$$

Compound Rule

$$\{\ true\ \}\ balance := b\ \backslash\backslash\ i\ ;\ credit := b + 10\quad \left\{ \begin{array}{l} i \neq 0\ \wedge\ balance = b\ \backslash\backslash\ i\ \wedge \\ credit = b + 10 \end{array}\ ,\quad i = 0 \right\}$$

Chair of Software
Engineering

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Rescue

$$P \Rightarrow P'$$

$$\{INV \wedge P'\} \quad b \quad \{INV \wedge Q \;,\; Q'\}$$

$$\{Q'\}\; r \;\{INV \wedge (\textbf{Retry} \Rightarrow P') \wedge (\neg\, \textbf{Retry} \Rightarrow R) \;,\; INV \wedge R\}$$

$$\overline{}$$

$$\{INV \wedge P\}\; \textbf{do } b \textbf{ rescue } r \textbf{ end}\; \{INV \wedge Q \;,\; INV \wedge R\}$$

Normal postcondition

Error postcondition

Chair of Software
Engineering

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Example: rescue

safe_division   (x, y: INTEGER): INTEGER
   local
      z: INTEGER
   do
      Result := x // (y+z)
   ensure
      y = 0 **implies Result** = x
      y /= 0 **implies Result** = x // y
   rescue
      z := 1
      Retry := **true**
   end

$$\{\ true\ \}\quad MATH{:}safe\_division\quad \{\ Q\ ,\ false\ \}$$

where

$$Q \equiv (y = 0 \Rightarrow Result = x) \wedge (y/ = 0 \Rightarrow Result = x//y)$$

Example: rescue

```
safe_division (x,y: INTEGER): INTEGER
 local
    z: INTEGER
 do
```
$$\{\,(y \neq 0 \wedge z = 0) \vee (y = 0 \wedge (z = 1 \vee z = 0))\,\}$$
```
    Result := x // (y+z)
```
$$\left\{\left(\begin{array}{l}(y = 0 \Rightarrow Result = x)\ \wedge \\ (y \neq 0 \Rightarrow Result = x/y)\end{array}\right), (y = 0 \wedge z = 0)\right\}$$
```
 ensure
    y = 0 implies Result = x
    y /= 0 implies Result = x // y
 rescue
```
$$\{\ y = 0 \wedge z = 0\ \}$$
```
    z := 1
```
$$\{\ (y = 0 \wedge z = 1),\ false\ \}$$
```
    Retry := true
```
$$\left\{\left(\ Retry \wedge (y = 0 \wedge z = 1)\ \right),\ false\right\}$$
```
 end
```

Retry invariant

$$(y \neq 0 \wedge z = 0) \vee (y = 0 \wedge (z = 1 \vee z = 0))$$

Overview

- Proof-Carrying Code

- Proof-Transforming Compilation
  - ➢ Semantics for Java and Eiffel
  - ➢ A Hoare-style logic for Bytecode
  - ➢ Proof Translation

Chair of Software
Engineering

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

The bytecode Language

Bytecode language similar to .Net CIL bytecode
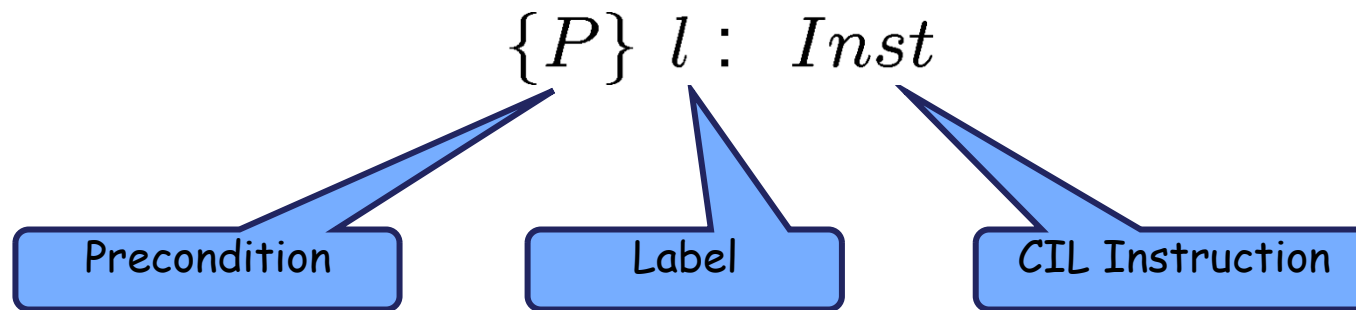
Boolean type

Instead of using an array of local variables like in .Net CIL, we use the name of the source variable

$$
\begin{aligned}
\text{bytecodeInstr} \quad ::= \quad & \text{pushc } v \\
| \quad & \text{pushv } x \\
| \quad & \text{pop } x \\
| \quad & op_{op} \\
| \quad & \text{goto } l \\
| \quad & \text{brtrue } l \\
| \quad & \text{nop} \\
| \quad & \text{athrow}
\end{aligned}
$$

The Bytecode Language and its Logic

- Bytecode Logic:
  - ➢ Logic developed by F. Bannwart and P. Müller
  - ➢ Instruction specification

$$\{P\}\ l:\ Inst$$

Precondition

Label

CIL Instruction

Chair of Software
Engineering

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

The bytecode Logic

Rules for instructions

$$\frac{E_l \Rightarrow wp_p^1(I_l)}{\mathsf{A} \vdash \{E_l\}\ l : I_l}$$

Chair of Software
Engineering

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

The bytecode Logic

I_l	$wp_p^1(I_l)$
pushc v	$unshift(E_{l+1}[v/s(0)])$
pushv x	$unshift(E_{l+1}[x/s(0)])$
pop x	$(shift(E_{l+1}))[s(0)/x]$
bin_{op}	$(shift(E_{l+1}))[s(1)ops(0)/s(1)]$
goto l'	$E_{l'}$
brtrue l'	$(\neg s(0) \Rightarrow shift(E_{l+1})) \wedge (s(0) \Rightarrow shift(E_{l'}))$
return	true
nop	E_{l+1}

$$shift(E) = E[s(i+1)/s(i) \text{ for all } i \in \mathbb{N}]$$
$$unshift = shift^{-1}$$

Chair of Software
Engineering

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Example Bytecode Proof

Source Program:

x := 5

y := 1

Compiled Program:

L00: push 5

L01: pop x

L02: push 1

L03: pop y

Chair of Software
Engineering

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Overview

- Proof-Carrying Code

- Proof-Transforming Compilation
  - ➢ Semantics for Java and Eiffel
  - ➢ A Hoare-style logic for Bytecode
  - ➢ Proof Translation

Chair of Software
Engineering

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

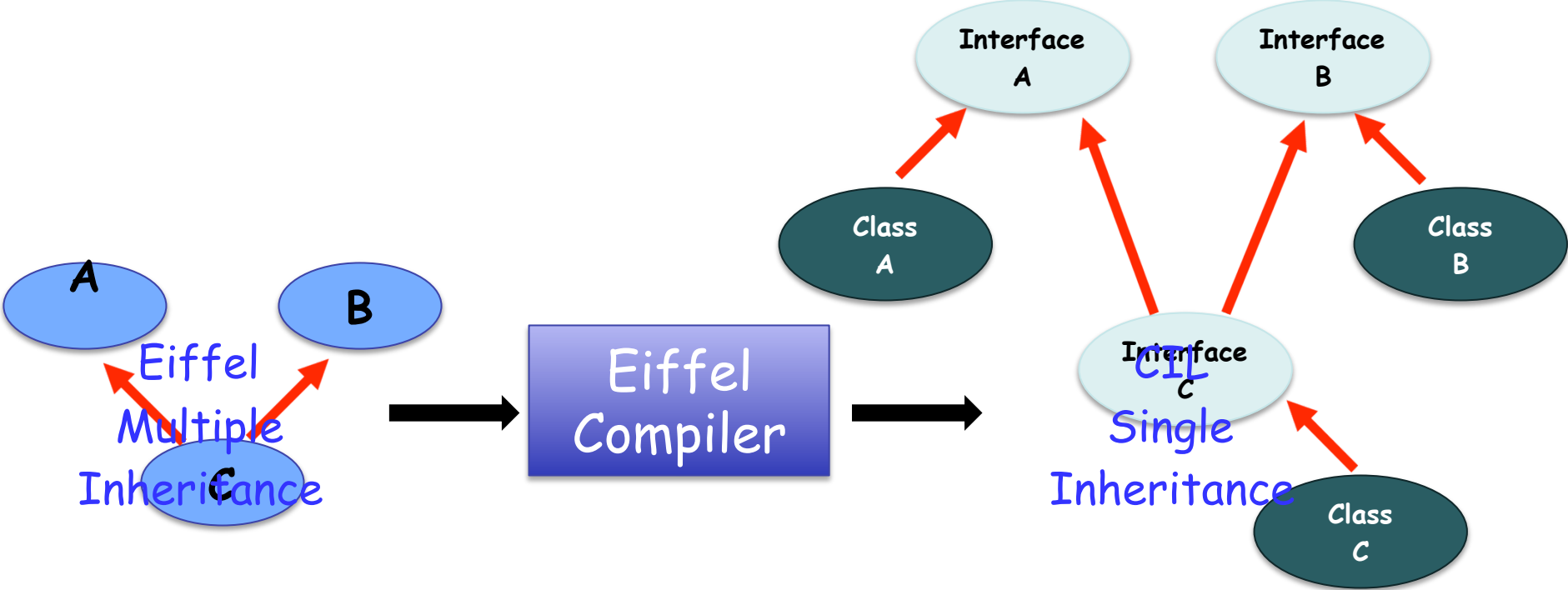# Proof-Transforming Compilation
for Eiffel

Contract Translator

> Deep embedding of contracts, pre- and postconditions

> Translation functions

- Input: Deep embedding of Boolean expressions
- Output: First Order Logic

$$\nabla_E \ : \ Precondition \times Expression \times Postcondition \times Label \rightarrow BytecodeProof$$
$$\nabla_S \ : \ ProofTree \times Label \times Label \times Label \rightarrow BytecodeProof$$

- Soundness Proof

Chair of Software
Engineering

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Compiling Eiffel to .Net CIL

Eiffel Multiple Inheritance

A

B

C

Eiffel Compiler

Interface A

Interface B

Class A

Class B

Interface C

CIL Single Inheritance

Class C

↑ Inheritance   Eiffel class   CIL interface   CIL class

Chair of Software Engineering

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Applications

$$\nabla_S \left(\begin{array}{c} \text{Compound Rule} \\ \{\, true \,\} \quad balance := b \;;\; credit := b + 10 \quad \left\{\begin{array}{l} balance = b \;\wedge \\ credit = b + 10 \end{array}\right\} \\[2em] \nabla_S \left(\left(\text{Assignment Rule} \atop \{\, true \,\} \; balance := b \; \{\, balance = b \,\} \right) \; l_{00}, l_{02} \; \left(\text{Assignment Rule} \atop \{\, balance = b \,\} \; credit := b + 10 \; \left\{\begin{array}{l} balance = b \;\wedge \\ credit = b + 10 \end{array}\right\} \right), l_{02}, l_{06} \right) \end{array} \;,\; l_{00}, \; l_{06} \right)$$
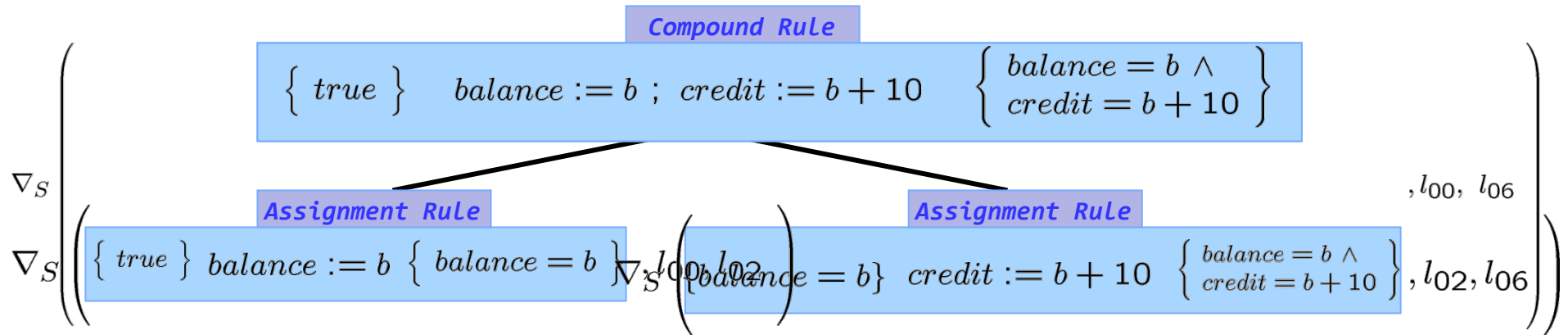
CIL proof:

$$\nabla_E \left(\{\, true \,\} \; l_{00}: \; \text{ldloc} \; b \; \{\, s(0) = b \,\} \right)$$

$$\{\, s(0) = b \,\} \; l_{01}: \; \text{stloc} \; balance$$

$$\nabla_E \left(\left\{\begin{array}{l} balance = b \end{array}\right\} \; l_{02}: \; \text{ldloc} \; b, \; \left(\begin{array}{l} balance = b \;\wedge \\ s(0) = b \end{array}\right) \right)$$

$$\nabla_E \left(\left(balance = b \;\wedge\; s(0) = b \right), \; l_{03}: \; \text{ldc} \; 10, \; \left(balance = b \;\wedge\; s(1) = b \;\wedge\; s(0) = 10 \right) \right)$$

$$\{\, balance = b \;\wedge\; s(1) = b \wedge\; s(0) = 10 \,\} \; l_{04}: \; \text{add}$$

$$\{\, balance = b \;\wedge\; s(0) = b + 10 \,\} \; l_{05}: \; \text{stloc} \; credit$$

Chair of Software Engineering

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Tool Support

XML file → XML parser → AST → Proof translator / Specification translator → CIL code + proof

Proof-Transforming Compiler
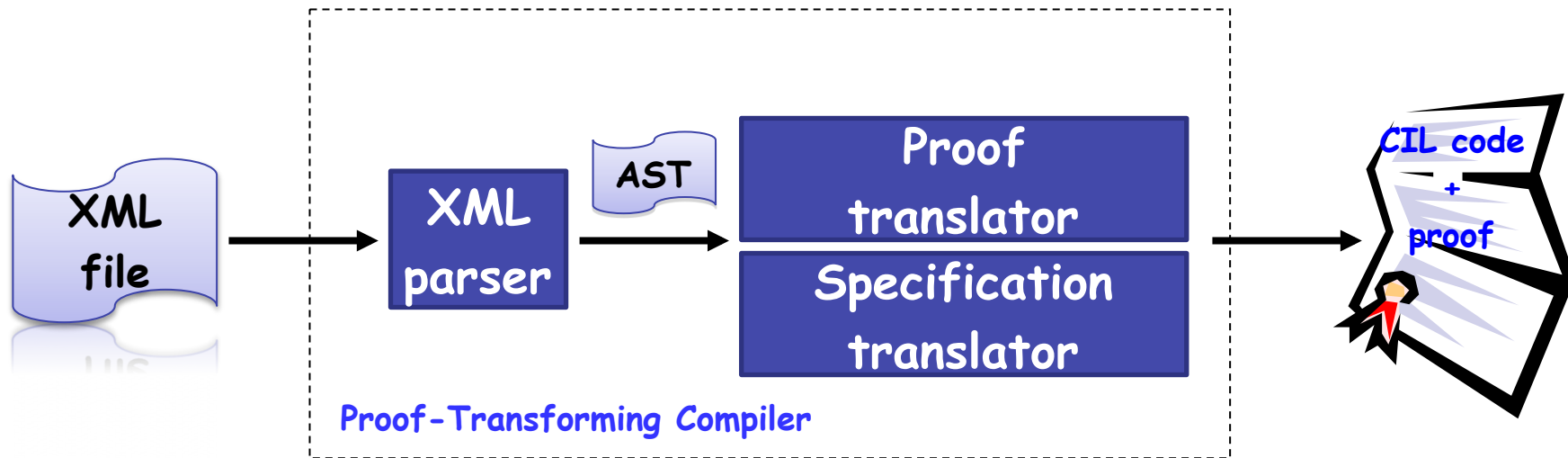
Experiments with PTC

Example	#Classes	#Routines	#lines Eiffel	#lines source proof
Boolean expressions	2	3	76	205
Attributes	3	5	83	167
Conditionals	1	2	55	154
Loops	1	1	31	73
Bank Account simple	1	3	57	108
Bank Account	1	5	57	130
Sum Integers	1	1	35	126
Subtyping	3	5	41	117
Demo	4	8	152	483
Total	**17**	**33**	**587**	**1563**

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Size of the proof

Example	#lines Eiffel	#lines source proof	#lines in Isabelle
Boolean expressions	76	205	711
Attributes	83	167	1141
Conditionals	55	154	510
Loops	31	73	305
Bank Account simple	57	108	441
Bank Account	57	130	596
Sum Integers	35	126	358
Subtyping	41	117	756
Demo	152	483	1769
Total	**587**	**1563**	**6587**

Chair of Software Engineering

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Experiments Proof Checker

Isabelle Example	#lines in Isabelle	Simplifier Proof Script (in sec)	Optimized Proof Script (in sec)
Boolean expressions	711	3.4	1.9
Attributes	1141	3.6	2.2
Conditionals	510	7.3	3.8
Loops	305	14.1	3.2
Bank Account simple	441	5.5	2.4
Bank Account	596	12.8	4.6
Sum Integers	358	45.2	6.3
Subtyping	756	4.3	2.3
Demo	1769	92.2	27.5
Total	**6587**	**192.4 (~3')**	**54.2**

Chair of Software
Engineering

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich