Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Software Engineering

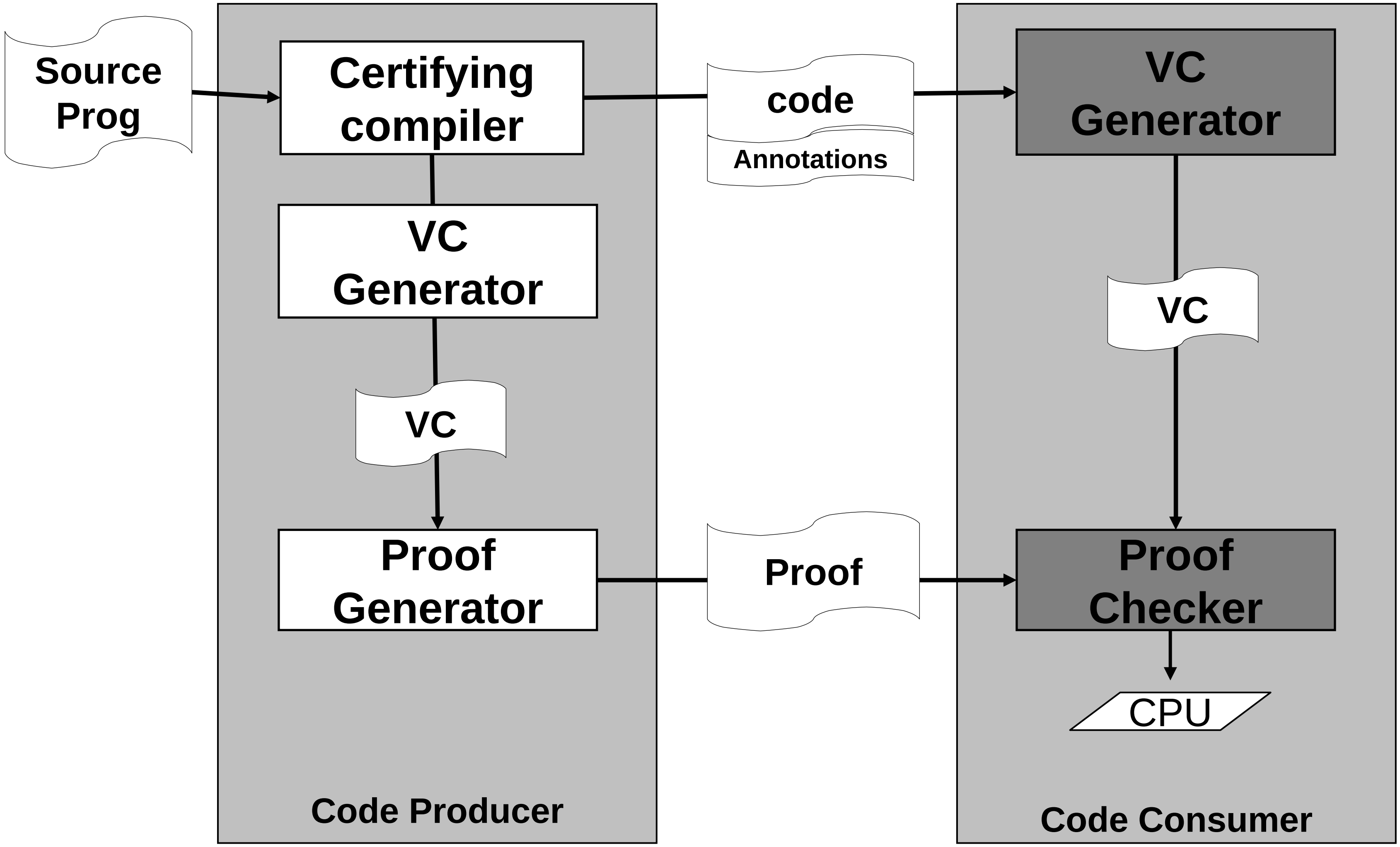
Proof-Transforming Compilation of Programs with Abrupt Termination

Peter Müller and Martin Nordio

Microsoft Research (USA)

ETH Zurich

Proof-Carrying Code



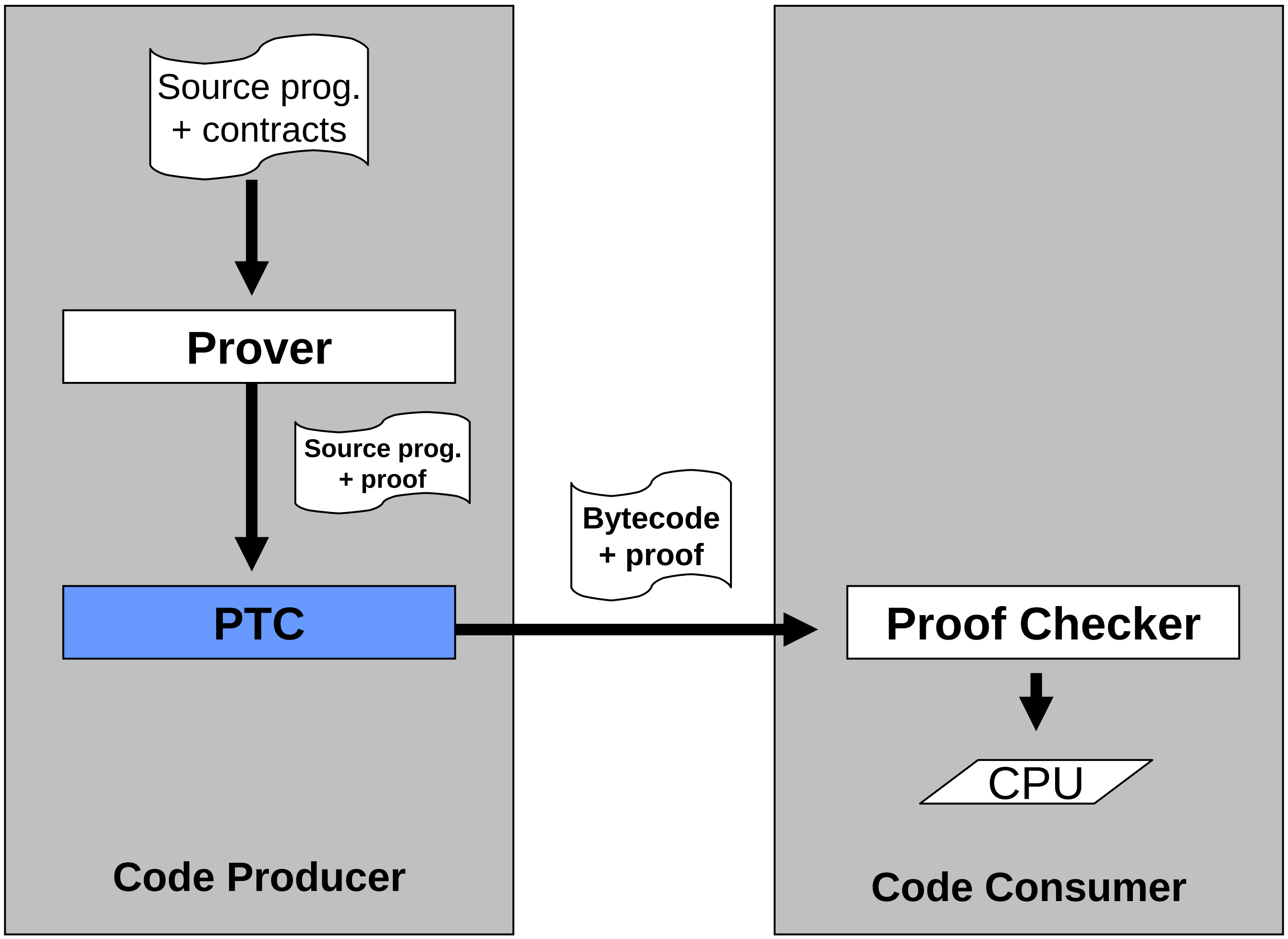
Develop the proof for the Bytecode

- **Logics for intermediate languages such as Java Bytecode and CIL were developed**

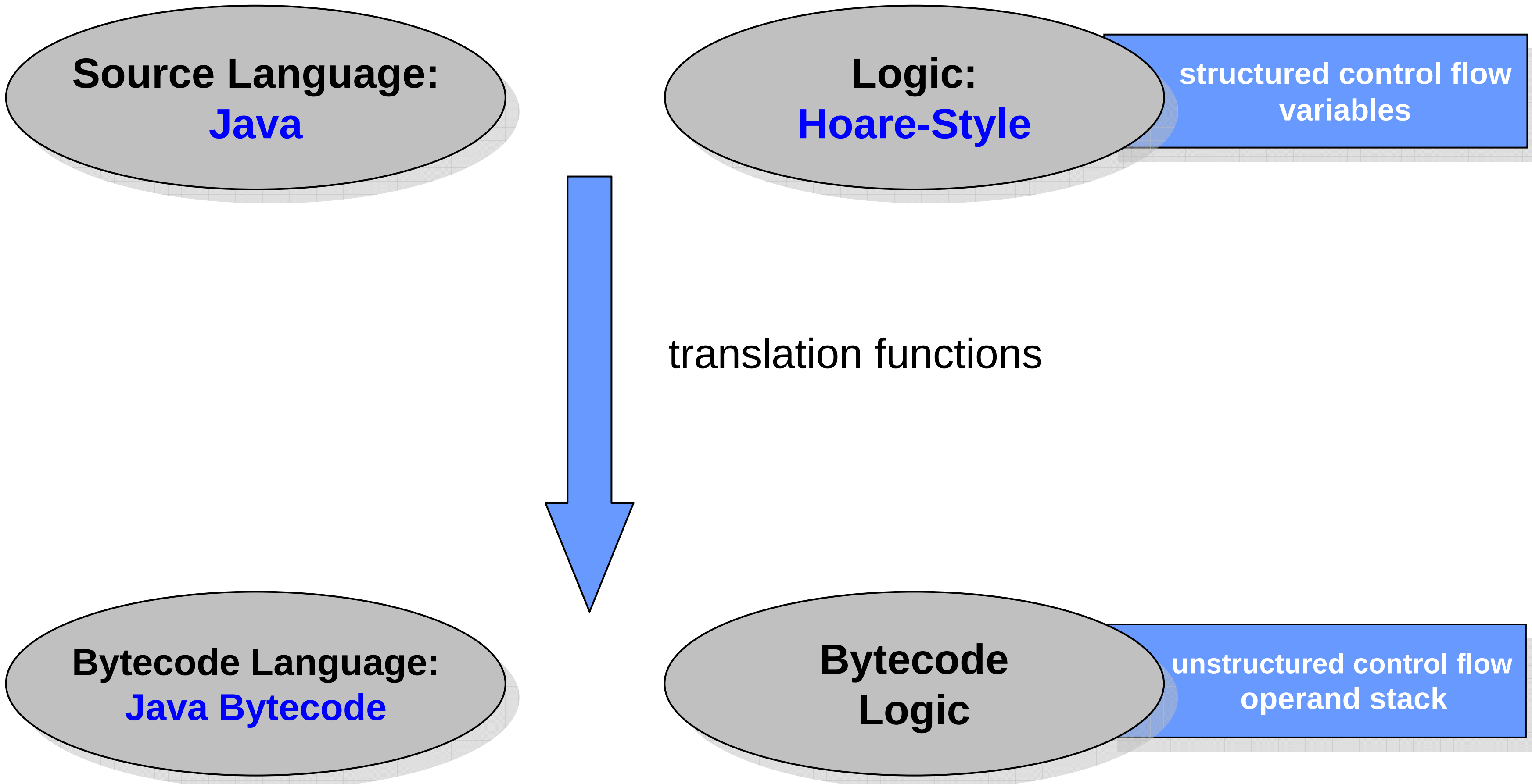
  (Müller and Bannwart)

- **Pro: It can produce the certificate needed**
- **Con: It is difficult and expensive**

Proof-Transforming Compilers (PTC)



PTC Elements



The bytecode Language

$$
\begin{array}{lcl}
\mathsf{bytecodeInstr} & ::= & \mathsf{pushc}\ \mathsf{v} \\
& | & \mathsf{pushv}\ \mathit{x} \\
& | & \mathsf{pop}\ \mathit{x} \\
& | & op_{op} \\
& | & \mathsf{goto}\ \mathit{l} \\
& | & \mathsf{brtrue}\ \mathit{l} \\
& | & \mathsf{nop} \\
& | & \mathsf{athrow}
\end{array}
$$

The bytecode Logic

- We use the bytecode logic developed by F. Bannwart and P. Müller
- Instruction specification

$$\{E_l\}\ l : I_l$$

The Source Language

- **Similar to a Java subset**

$$
\begin{array}{lcl}
exp & ::= & literal \mid var \mid exp\ op\ exp \\
stm & ::= & x = exp \mid stm; stm \mid \boxed{\texttt{while}\ (exp)\ stm} \\
 & \mid & \boxed{\texttt{break ;}} \mid \texttt{if}\ (exp)\ stm\ \texttt{else}\ stm \\
 & \mid & \texttt{try}\ stm\ \texttt{catch}\ (type\ var)\ stm \\
 & \mid & \boxed{\texttt{try}\ stm\ \texttt{finally}\ stm} \mid \texttt{throw}\ exp\ \texttt{;}
\end{array}
$$

Logic for Java subset

- **The logic is based on the programming logic developed by A. Poetzsch-Heffter and N. Rauch.**

- **Properties of method bodies are expressed by Hoare triples of the form**

$$\{\ P\ \}\quad \mathit{comp}\quad \{\ \underbrace{Q_n}_{\text{normal}}\ ,\ \underbrace{Q_b}_{\text{break}}\ ,\ \underbrace{Q_e}_{\text{exception}}\ \}$$

Example: try-finally statements

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

b= 4 Normal

Compilation: try-finally statements

```
try {
    s1
}
finally {
    s2
}
```

$\nabla_S \ (s_1)$

$\nabla_S \ (s_2)$

l_c : goto l_h

l_d : pop $eTmp$

$\nabla_S \ (s_2)$

l_f : pushv $eTmp$

l_g : athrow

l_h : ...

Exception Table

From	to	target	type
l_a	l_b	l_d	any

Example: try-finally statements

$$\nabla_S \begin{pmatrix} b++; \\ throw\ new\ Exception() \end{pmatrix}$$

$$\nabla_S \begin{pmatrix} b++; \\ break \end{pmatrix}$$

l_c : goto l_h

l_d : pop $eTmp$

$$\nabla_S \begin{pmatrix} b++; \\ break \end{pmatrix} \rightarrow \text{goto } l_i$$

l_f : pushv $eTmp$

l_g : athrow

l_h...

$l_i : \nabla_S \begin{pmatrix} b++; \end{pmatrix}$

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

Logic for try-finally statements

$$\frac{\begin{array}{c}\{\ P\ \}\ \ s_1\ \ \{\ Q_n\ ,\ Q_b\ ,\ Q_e\ \} \\ \{\ Q\ \}\ \ s_2\ \ \{\ R\ ,\ R'_b\ ,\ R'_e\ \}\end{array}}{\{\ P\ \}\ \ try\ s_1\ finally\ s_2\ \ \{\ R'_n\ ,\ R'_b\ ,\ R'_e\ \}}$$

finally / try	N	B	E_2
N	N	B	E_2
B	B	B	E_2
E_1	E_1	B	E_2

where

$$Q \equiv \left(\begin{array}{l} (Q_n \ \wedge\ \mathcal{X}Tmp = normal)\ \vee (Q_b\ \wedge\ \mathcal{X}Tmp = break)\ \vee \\ \left(\ Q_e[eTmp/excV]\ \wedge \mathcal{X}Tmp = exc\ \wedge\ eTmp = excV\ \right) \end{array} \right)$$

$$and$$

$$R \equiv \left(\begin{array}{l} (R'_n\ \wedge\ \mathcal{X}Tmp = normal)\ \vee (R'_b\ \wedge\ \mathcal{X}Tmp = break)\ \vee \\ (R'_e\ \wedge\ \mathcal{X}Tmp = exc) \end{array} \right)$$

Example 2: Exception Table

```
while (i<20) {
    try {
        try {
            try {
                ...
                break;
                ...
            }
            catch (Exception e) {
                i=9;
            }
        }
        finally {
            throw new Exception();
        }
    }
    catch (Exception e) {
        i=99;
    }
}
```

Example 2: Exception Table (cont.)

```
while (i<20) {
   try {
      try {
         try {
               ...

               ...
         }
         catch (Exception e) {
               i=9;
         }
      }
      finally {
         throw new Exception();
      }
   }
   catch (Exception e) {
      i=99;
   }
}
```

Exception

any

Exception

Example 2: Exception Table (cont.)

```
while (i<20) {
    try {
        try {
            try {
                ...
                break;
                ...
            }
            catch (Exception e) {
                i=9;
            }
        }
        finally {
            throw new Exception();
        }
    }
    catch (Exception e) {
        i=99;
    }
}
```

Exception

ETH Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Translation Function

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

$$\nabla_S \ : ProofTree \times List[Finally] \times ExceptionTable \rightarrow [BytecodeProof \times ExceptionTable]$$

$Finally$ is defined as a tuple of $[ProofTree\ ,\ ExceptionTable]$

PTC

- **Compositional statement**

$$[B_{S_1}, et_1] = \nabla_S \left(T_{S_1},\ f, et\right)$$
$$[B_{S_2}, et_2] = \nabla_S \left(T_{S_2} f, et_1\right)$$

$$[B_{S_1} + B_{S_2}\ ,\ et_2]$$

- **While**

$$Finally := \emptyset$$

- **try-finally**

$$Finally := [ProofTree, ExceptionTable] + Finally$$

- **Break**
  - **Translate the finally blocks dividing the exception table**
  - **Add a goto end-while**

Summary

- **Source Language:**
  - **Subset of Java**
  - **while, break,**
  - **try-catch, try-finally, throw**

- **Soundness proof**