UMASSCOMPSCI
DEPARTMENT OF COMPUTER SCIENCE

Automating Instruction Selector Generation in Jikes RVM

Adam Fidel — **Texas Tech University**
Chujiao Ma — **Franklin W. Olin College**

Faculty Advisors: Dr. J. Eliot Moss, Dr. Charles Weems

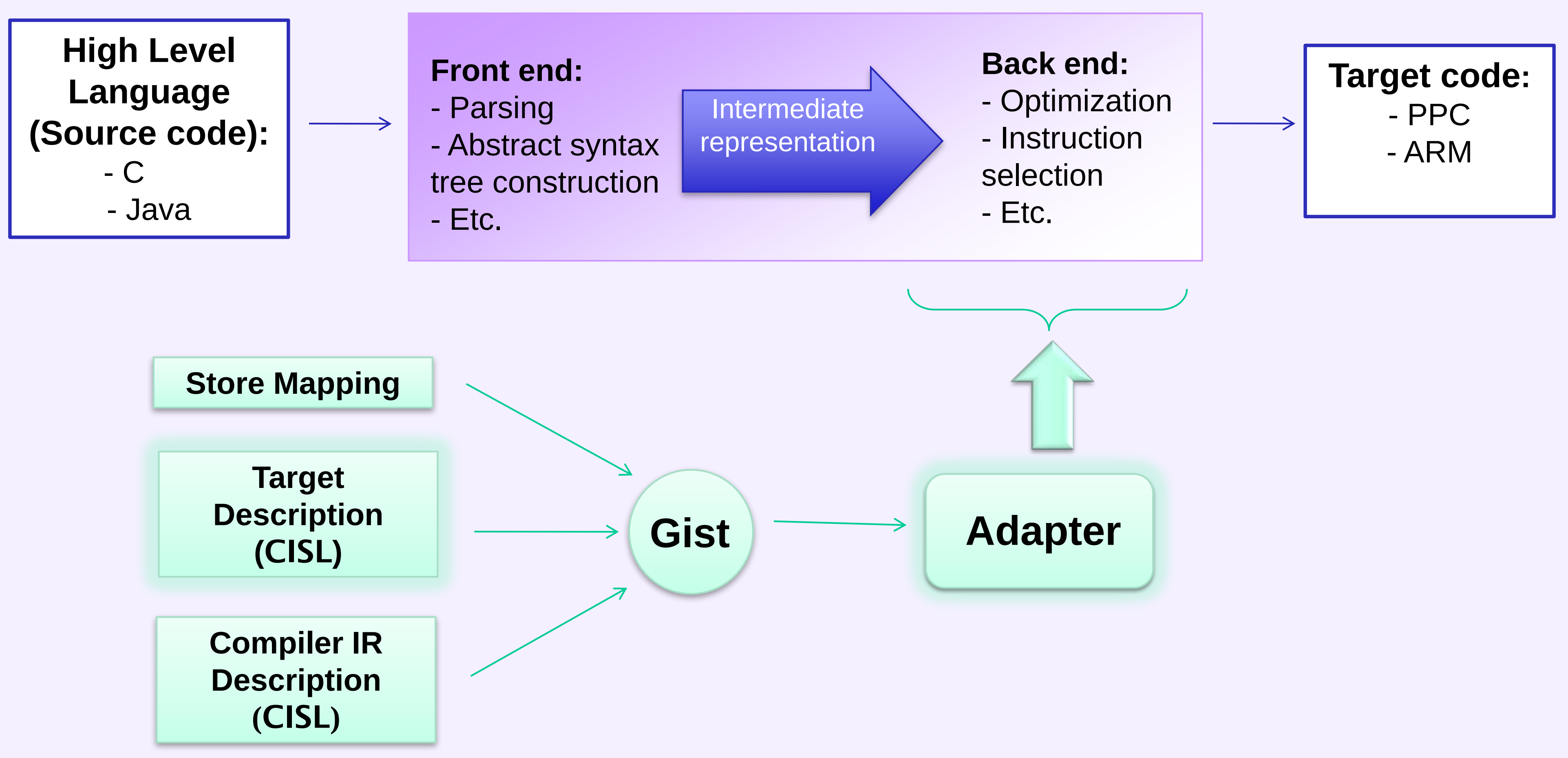
Graduate Mentors: Tim Richards, Ed Walters

CoGenT

CoGenT is a simulator and compiler environment driven by higher level specifications. It includes machine description languages to describe parts of machine architectures. Our research focused on the Gist component of the CoGenT project.

Gist, illustrated in the figure on the right, is a tool in the CoGenT suite that automates the process of generating a compiler's instruction selector.

- **Universal:** automatically generates instruction selection patterns for any combination of target instruction set and compiler IR.
- **Ease of use:** writing target-specific machine descriptions and compiler-specific adapters greatly reduces the amount of effort required to build an instruction selector



High Level Language (Source code):
- C
- Java
Front end:
- Parsing
- Abstract syntax tree construction
- Etc.
Intermediate representation
Back end:
- Optimization
- Instruction selection
- Etc.
Target code:
- PPC
- ARM
Store Mapping
Target Description (CISL)
Compiler IR Description (CISL)
Gist
Adapter

Fig 1. Illustration of a compiler's process (purple) and Gist's automatic instruction selector generation (green).

Process

Instruction Selection

Our focus is on providing compiler IR and target architecture descriptions in CoGenT's own language (CISL) coupled with compiler-specific adapters to plug generated instruction selectors into existing compiler frameworks.

Specifically, we concentrated on the PowerPC architecture for the target. For the source IR, we use a Java compiler provided by Jikes RVM.

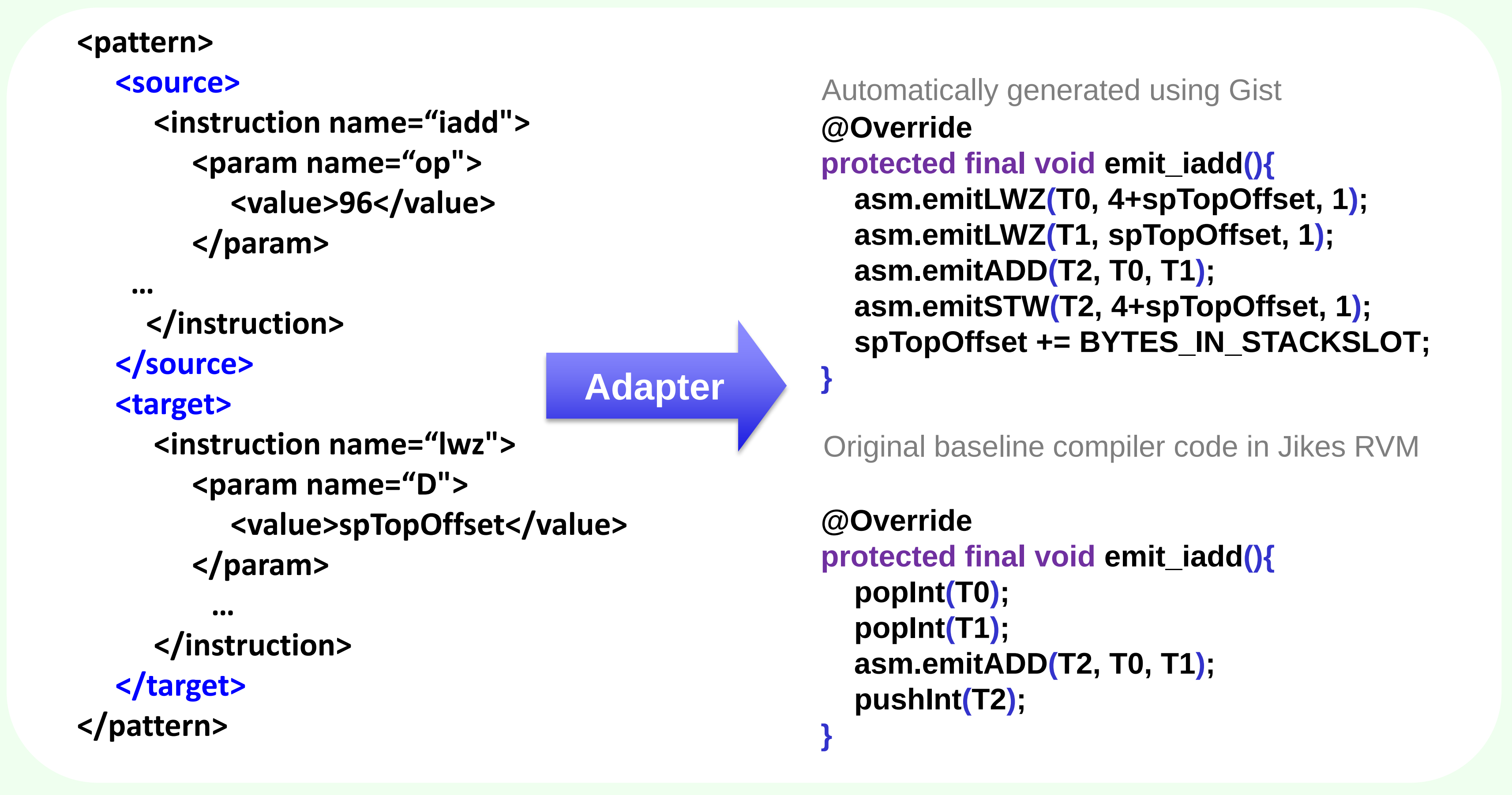
```
instruction class add extends XOForm_RT_RA_RB {
  fun encode() {
    OPCD = 31;
    XO   = 266;
  }
  fun effect() {
    R[RT] = R[RA] + R[RB];
  }
}
```

Fig 2. Description of a target architecture (PowerPC)

```
instruction class iadd extends ByteCode {
  fun encode() {
    op = 96;
  }
  fun effect() {
    var word_t a = S.slot[direct(spTopOffset)];
    var word_t b = S.slot[direct(spTopOffset + 4)];

    S.slot[direct(spTopOffset + 4)] = a + b;
  }
}
```

Fig 3. Description of a compiler IR (Java bytecode)



```
<pattern>
    <source>
        <instruction name="iadd">
            <param name="op">
                <value>96</value>
            </param>
    ...
        </instruction>
    </source>
    <target>
        <instruction name="lwz">
            <param name="D">
                <value>spTopOffset</value>
            </param>
            ...
        </instruction>
    </target>
</pattern>
```

Automatically generated using Gist

```
@Override
protected final void emit_iadd(){
    asm.emitLWZ(T0, 4+spTopOffset, 1);
    asm.emitLWZ(T1, spTopOffset, 1);
    asm.emitADD(T2, T0, T1);
    asm.emitSTW(T2, 4+spTopOffset, 1);
    spTopOffset += BYTES_IN_STACKSLOT;
}
```

Original baseline compiler code in Jikes RVM

```
@Override
protected final void emit_iadd(){
    popInt(T0);
    popInt(T1);
    asm.emitADD(T2, T0, T1);
    pushInt(T2);
}
```

Fig 4. Illustration of compiler-specific adaption process

Compiler-specific Adaption

Given CISL descriptions for the PowerPC (Fig. 2) and Java's bytecode specification (Fig. 3), Gist will generate generic instruction selection patterns in XML (Fig. 4).

Our adapter parses these patterns and generates code conformable with Jikes RVM's baseline complier.

Results

The goal of Gist is to replace the instruction selector component of the compiler. Since Jikes RVM is a JIT compiler, we performed runtime benchmarks to demonstrate that the instruction selector generated by Gist does not hinder the efficiency of the compiler.

The benchmark suite we used is a standard Java runtime benchmarking package called DaCapo. We did ten iterations of all benchmark tests for the original implementation and the Gist generated instruction selector.

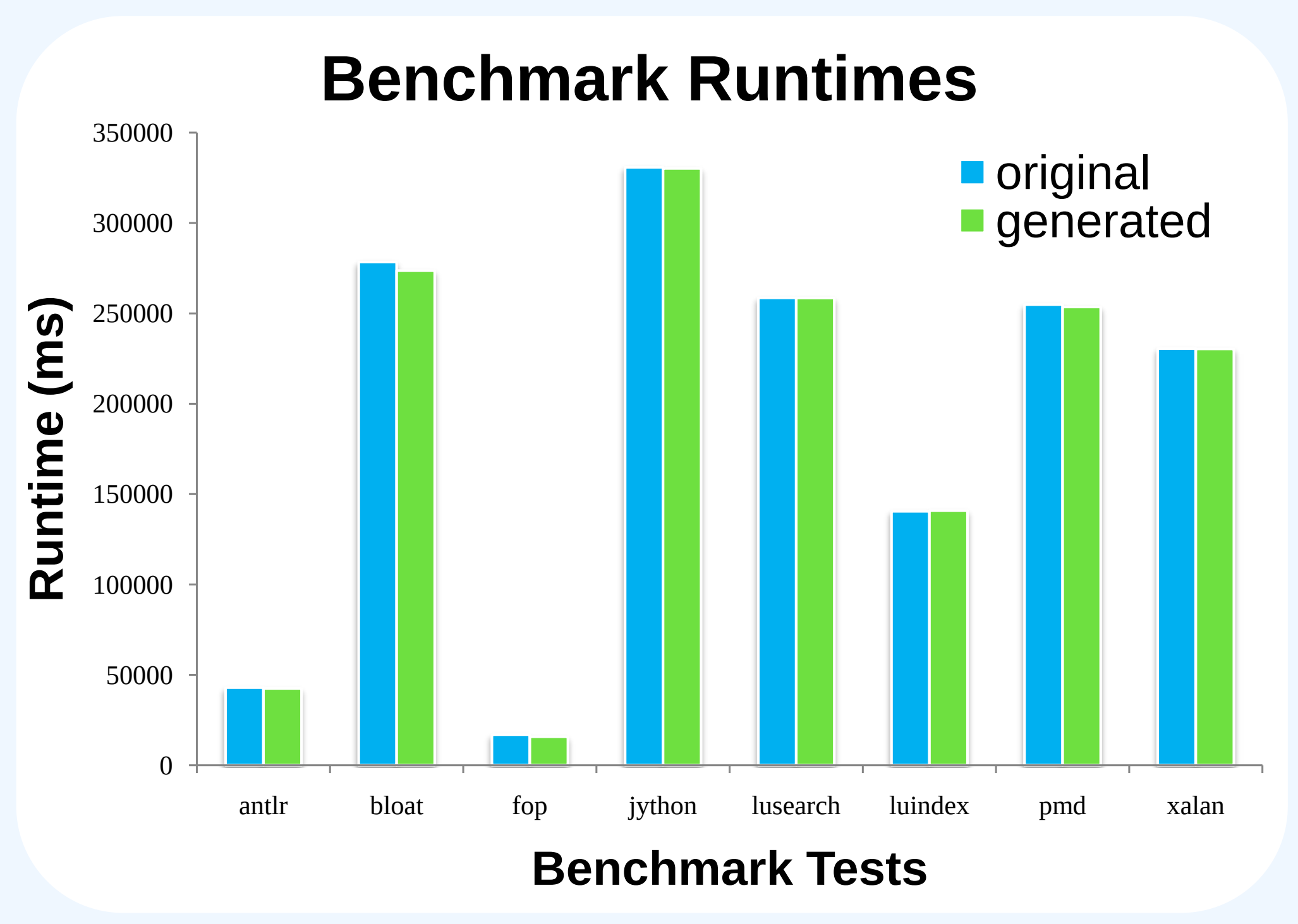


Fig 5. Graph of average runtime for various benchmarks.

Future Direction

For future work, there are plans to implement instruction selection pattern matching for more common combinations of targets and compilers, such as x86.

For CoGenT as a whole, we would like to work more on descriptions of micro-architectures as well as continuous refinement of Gist to be more efficient.

This work is partially supported by the Research Experience for Undergraduates Program of the National Science Foundation under NSF award number CCF-0755376. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the National Science Foundation.