

2009. 10. 9

MODELS 2009



TOKYO INSTITUTE OF TECHNOLOGY  
DEPARTMENT OF COMPUTER SCIENCE

# Generating Assertion

## Code from OCL:

### A Transformational Approach Based on Similarities of Implementation Languages

Rodion Moiseev Shinpei Hayashi Motoshi Saeki

Department of Computer Science  
Tokyo Institute of Technology  
Japan

# Abstract

- Generation Assertion Code from OCL
  - **Transformational Approach:**  
Regarding OCL-to-code as model transformation
  - **Similarities of Implementation Languages:**  
Constructing a language hierarchy for effective developing of transformation rules
- Results
  - A translation framework has been developed
  - Showed ~50% of effort to construct translation rules saved



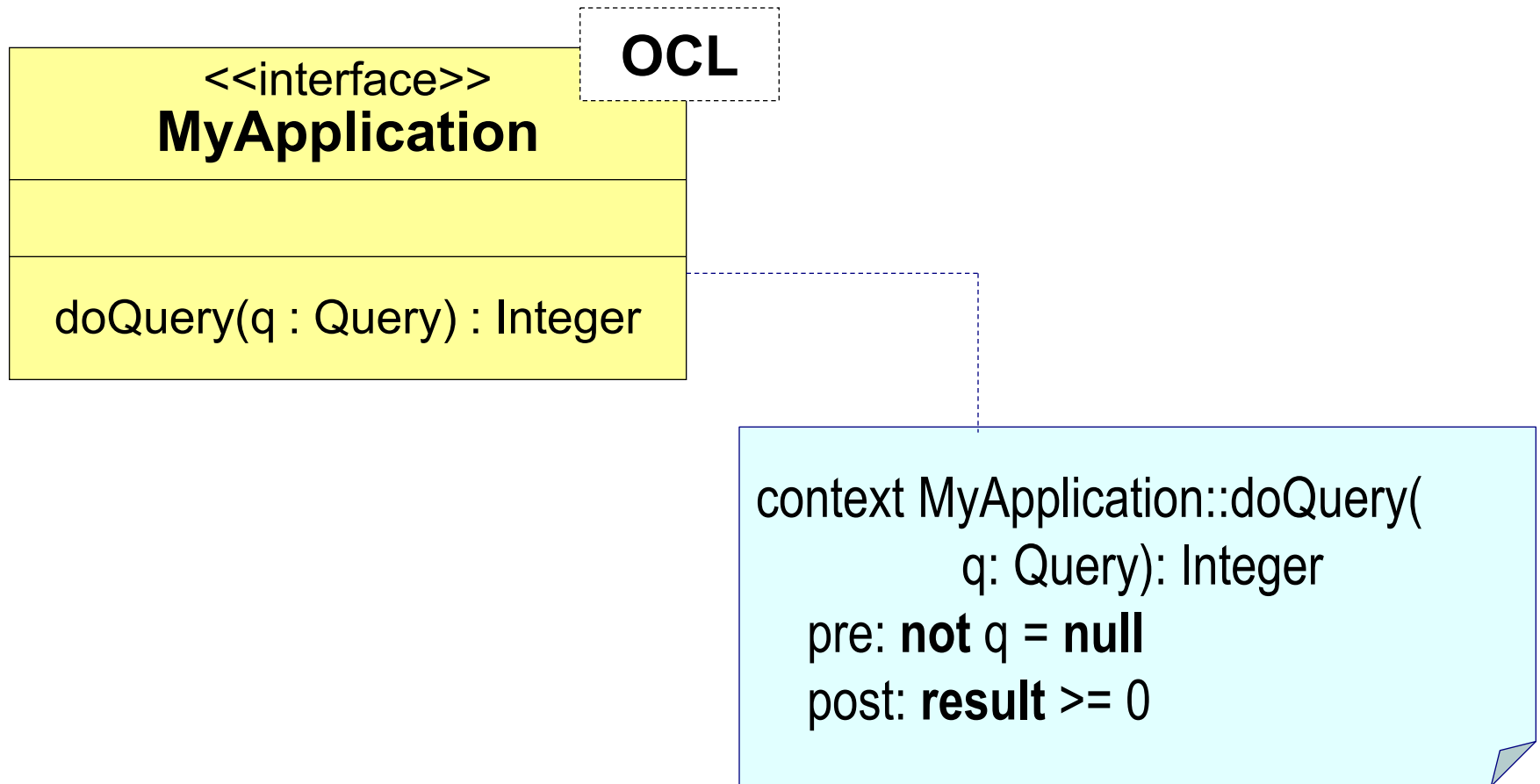
# Background

- Model-centric approaches are becoming important in academia and industry
- OCL is often the language of choice for stipulating constraints on models
- Successful application depends on mature tool support:
  - Support for multiple programming languages
  - Customizable assertion code/test case generation



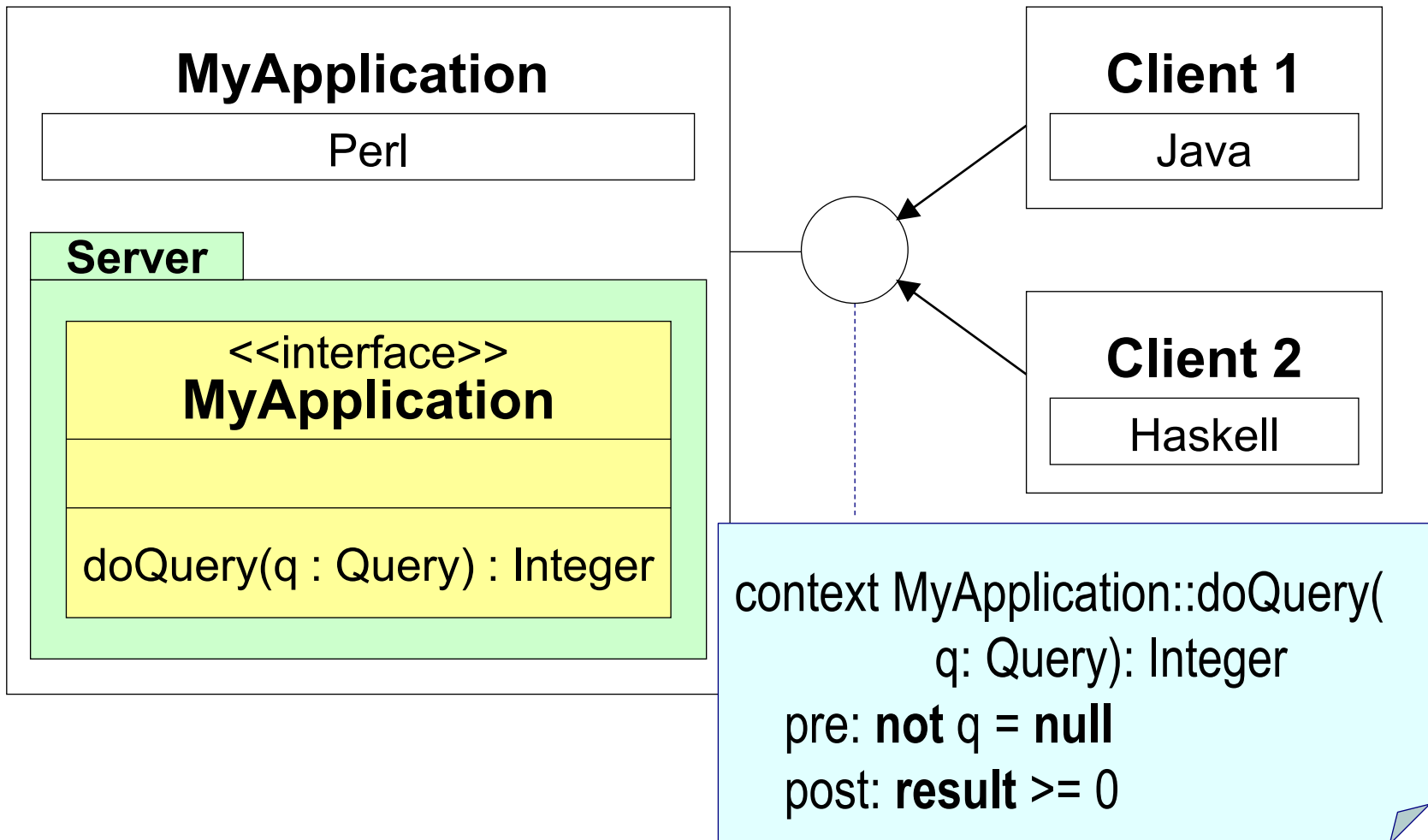
# Motivating Example

- Application design: UML with OCL

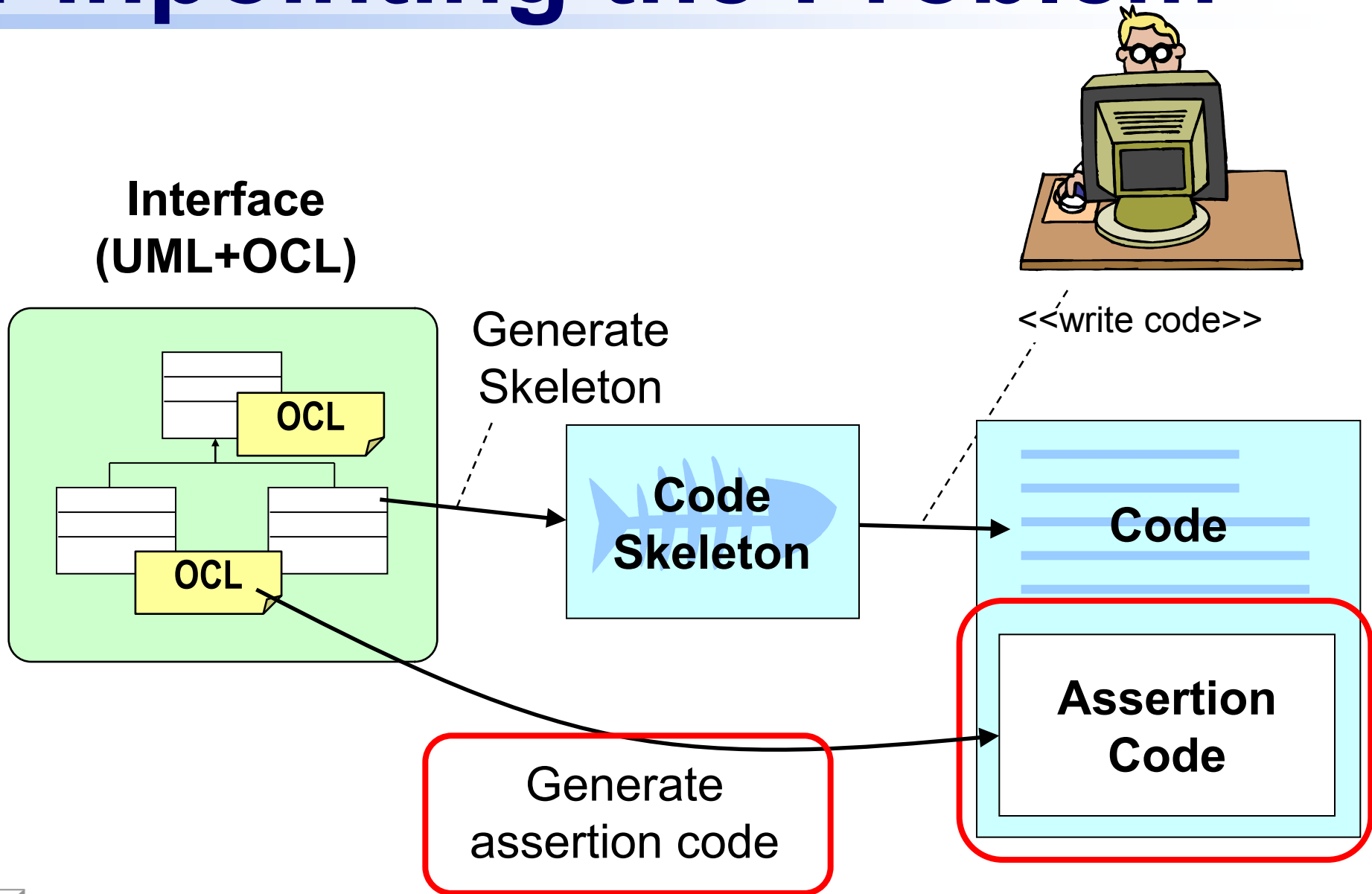


# Motivating Example

- A need to check OCL for multiple languages



# Pinpointing the Problem



# Problems

- Troublesome to make individual OCL translators
  - Many languages exist
    - Most of existing tools are tailor-made for a specific language
  - Creating a translator requires efforts
    - Specifying OCL translation rules for each individual OCL concept, operator, etc.
- **Our solution:**  
Usage of language similarities



# Language Similarities

- E.g., for-loop iterating over a collection



```
For Each apple In apples Do:  
  If apple.color == "red" Then  
    ...  
Next
```

Imperative  
Pseudo-language

```
for (Iterator it = apples.iterator(); it.hasNext(); ) {  
  Apple apple = (Apple) it.next();  
  if (apple.color == "red") {  
    ...  
  }  
}
```

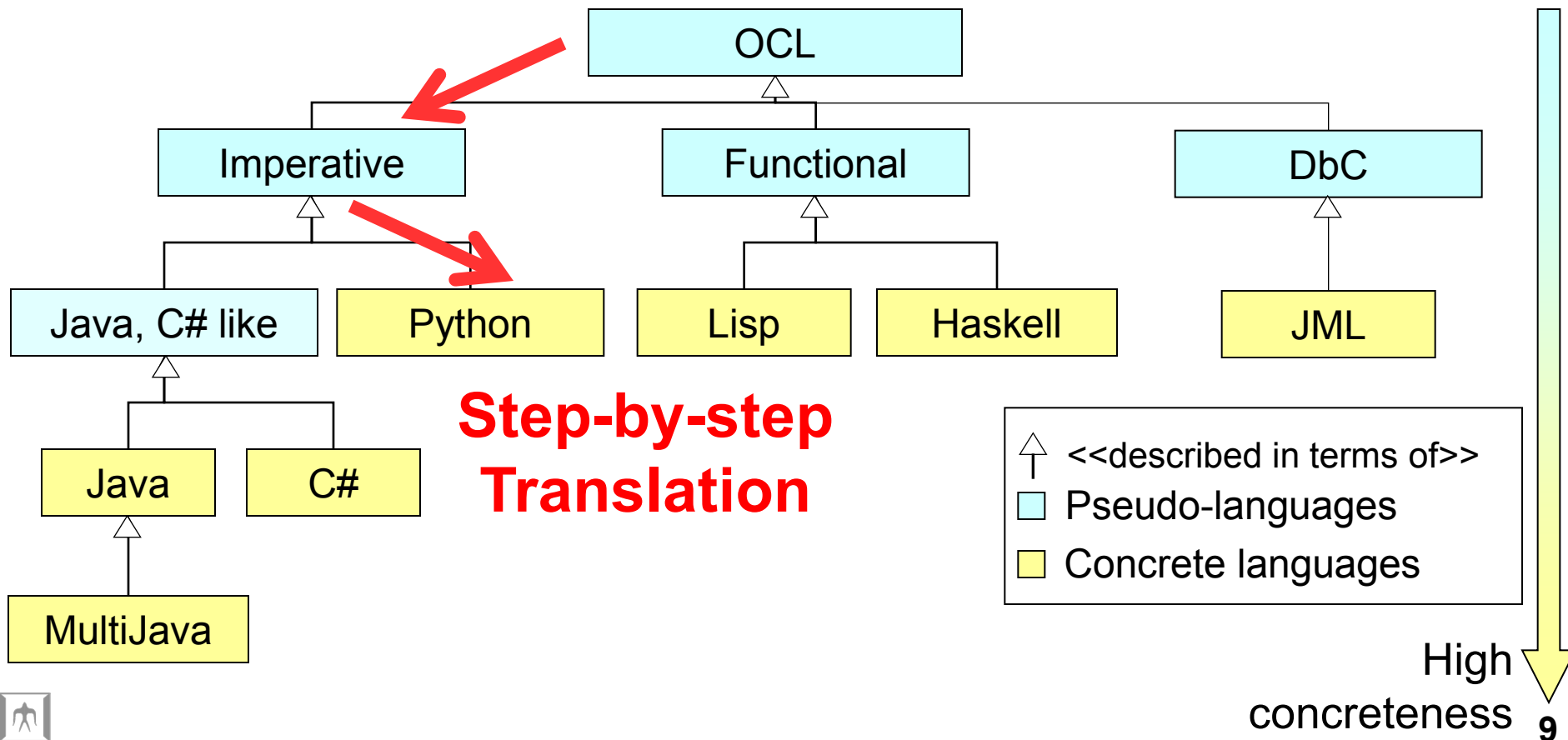
Java

```
for apple in apples:  
  if apple.color == "red":  
    ...
```

Python

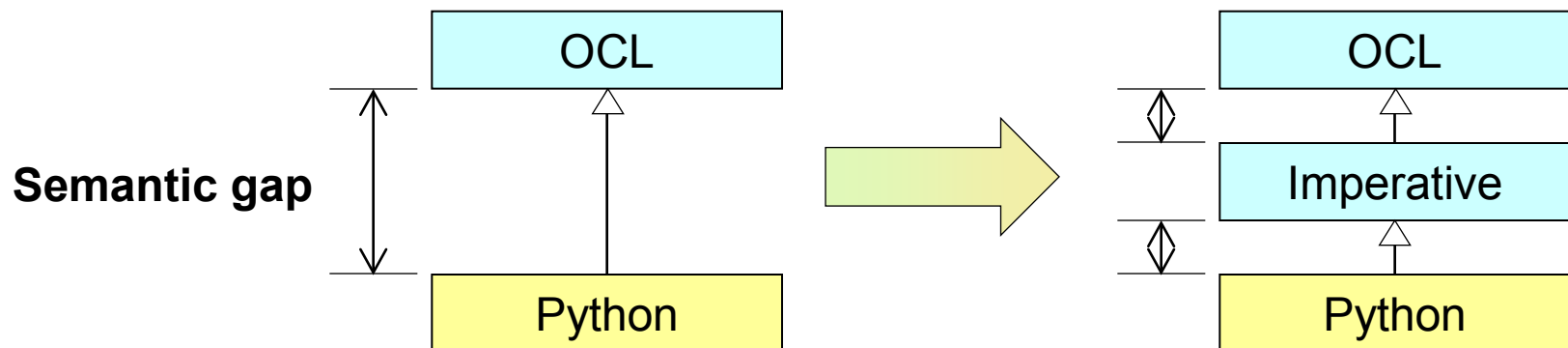
# The Approach

- Hierarchy of programming languages based on their structural similarities

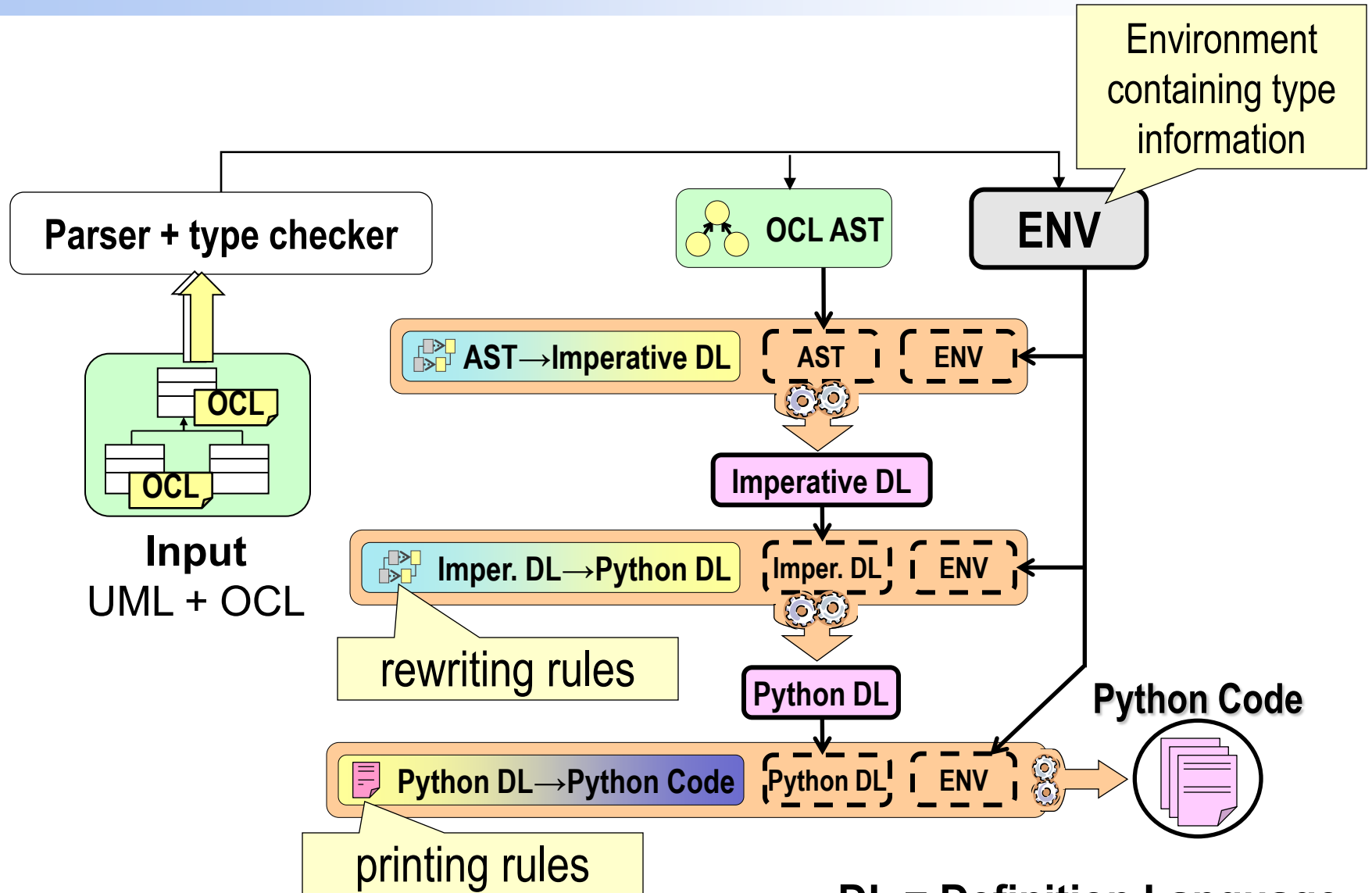


# Advantages

- Provide a single framework for creating OCL translators for any language
  - Reuse translator implementations
- Bridges semantic gap
  - Effort of understanding OCL is alleviated
  - Manual effort of creating OCL translators is reduced



# Generation Process

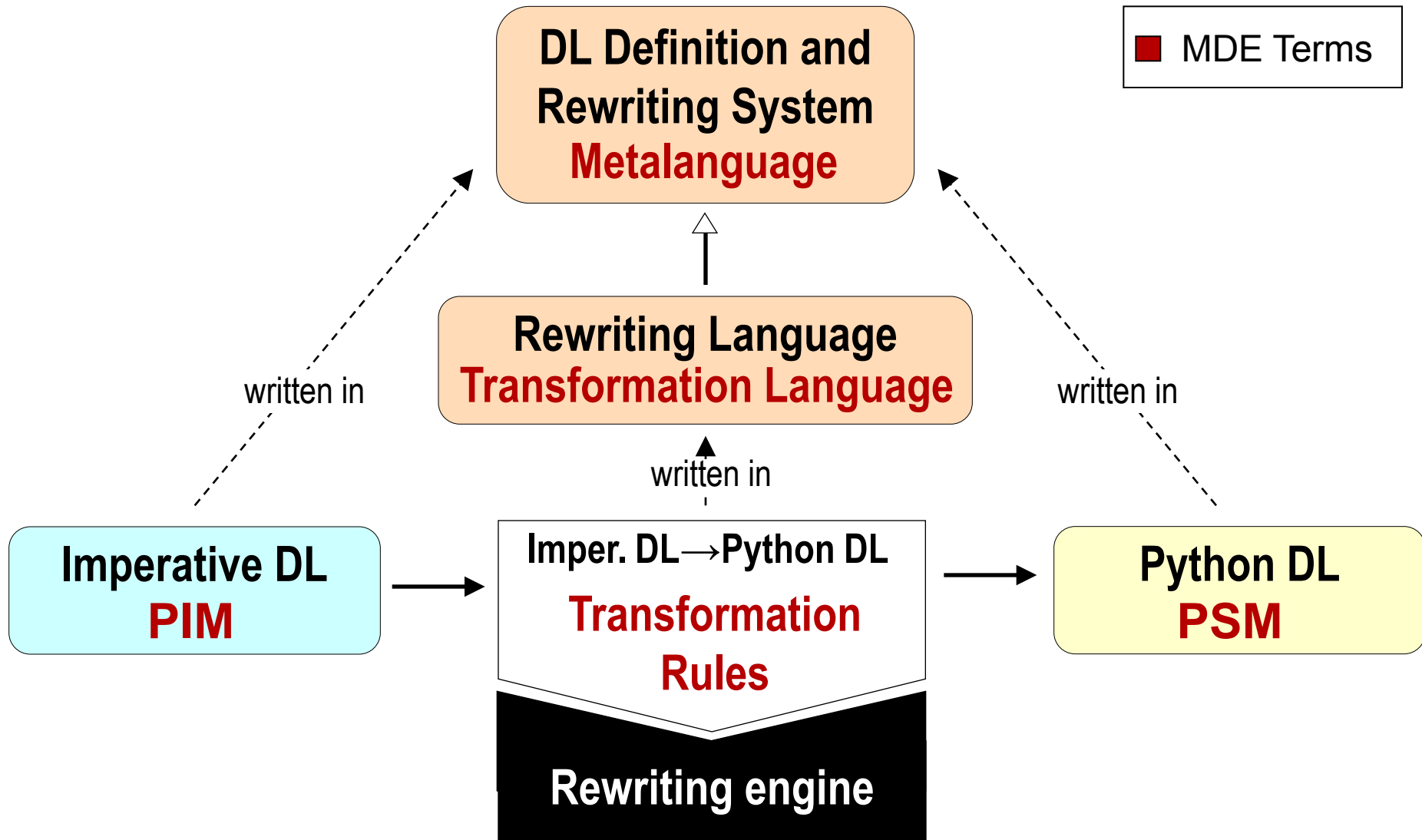


**DL = Definition Language**

(Structural representation of pseudo/implementation languages) 11



# Comparison with MDE



# Transformation Example

*All employees in the company earn more than 10,000*  
**employees->forall( e : Employee | e.salary > 10000)**

OCL AST:

iteratorExp

```
(assocEndCallExp simpleName("self") . simpleName("employees"))  
-> simpleName("forall") ( varDecl(simpleName("e"), pathName("Employee")) |  
  operCallExp(  
    attrCallExp simpleName("e") . simpleName("salary"),  
    simpleName(">"),  
    intLitExp("10000")  
  )  
)  
)
```



# Rewriting Rules

`iteratorExp OE -> simpleName("forAll") `( varDecl(SN, TYPE) | OE" `)`



```
impMethodExtract(  
  {  
    impForLoop( OE, impVarDecl(TYPE, SN),  
      {  
        impIf( impNot(OE"),  
          {  
            impReturn( boolLitFalse )  
          }  
        }  
      )  
    }  
    ; impReturn( boolLitTrue )  
  }, pathName("Boolean")) .
```



OCL AST → Imperative DL



# Imperative DL

```
impMethodExtract(  
  {emptyCtx |  
    impForLoop(  
      assocEndCallExp simpleName("self"). simpleName("employees"),  
      impVarDecl(pathName("Employee"),simpleName("e")),  
      { "ctx:iter" = simpleName("it_e") |  
        impIf( impNot(operCallExp(  
          attrCallExp simpleName("e"). simpleName("salary"),  
          simpleName(">"),  
          intLitExp("10000"))),  
          {emptyCtx |  
            impReturn(boolLitFalse)  
          }  
        )  
      }  
    )  
  };  
  {emptyCtx |  
    impReturn(boolLitTrue)  
  },pathName("Boolean"))
```

# Generation Result

```
class Employee:  
    def __init__(self, name, salary):  
        self.name = name  
        self.salary = salary
```

**From UML**

```
class Company:  
    def __init__(self, employees):  
        self.employees = employees
```

---

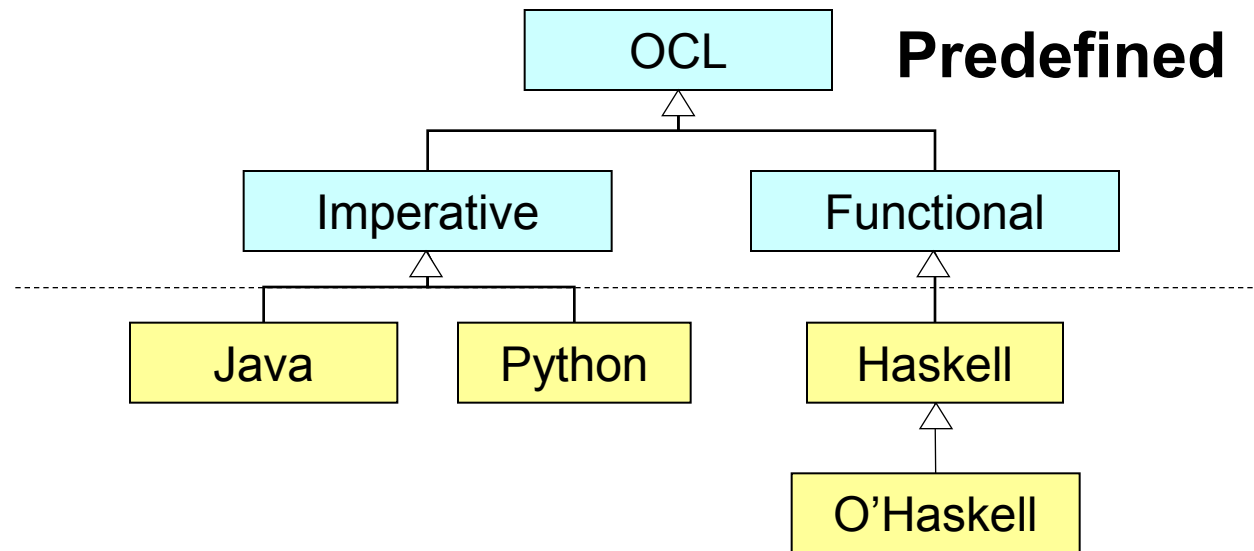
```
def callExtMethod_A(self):  
    for e in self.employees:  
        if (not (e.salary > 10000)):  
            return False  
    return True  
def inv(self):  
    return self.callExtMethod_A()
```

**From OCL**



# Approach Evaluation

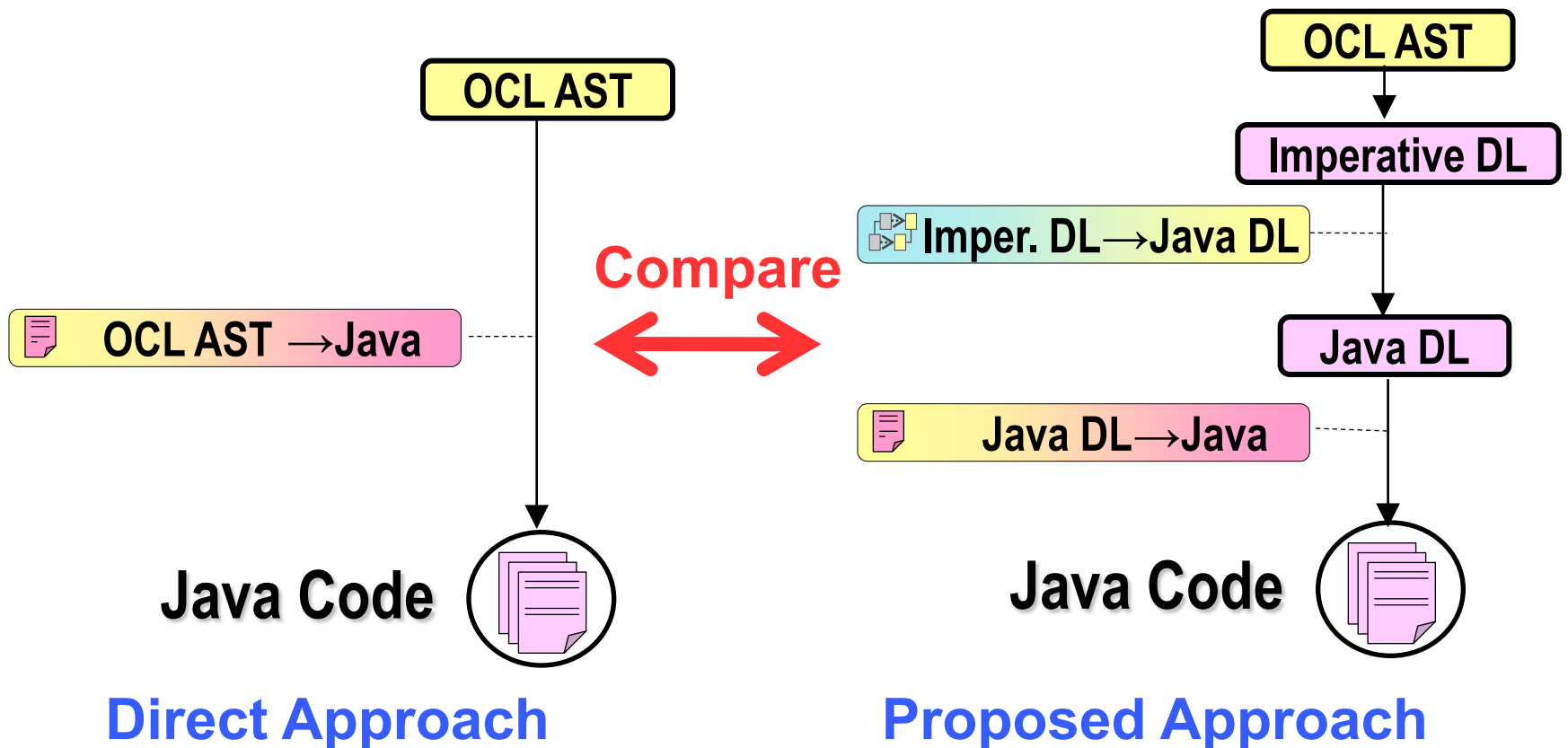
- Implemented a tool in the Maude System (\*)
- To show the ability to implement several OCL translators with significant effort savings
- Target languages
  - Imperative
    - Java
    - Python
  - Functional
    - Haskell
    - O'Haskell



\* <http://maude.cs.uiuc.edu/>

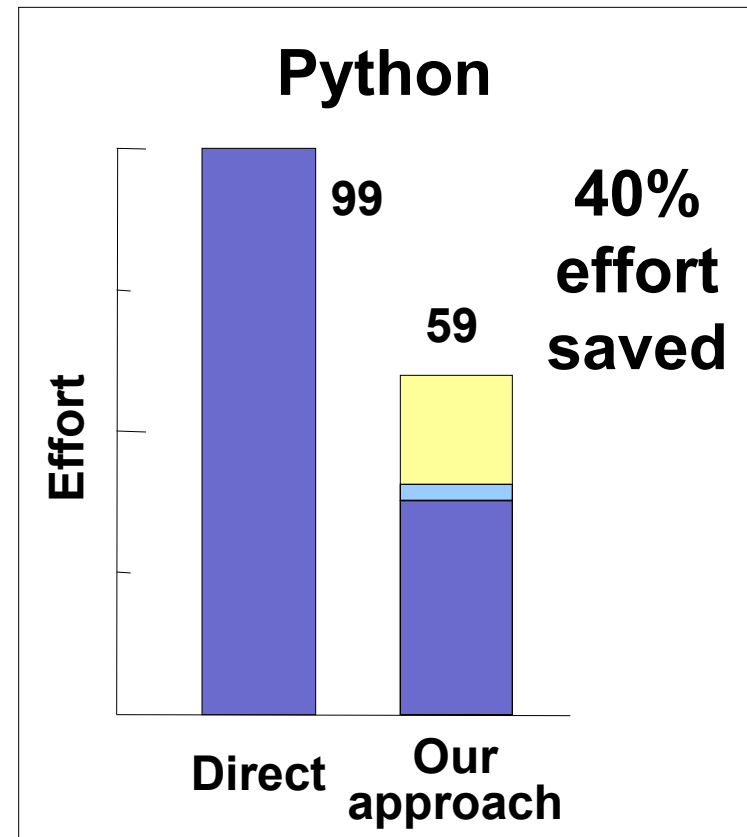
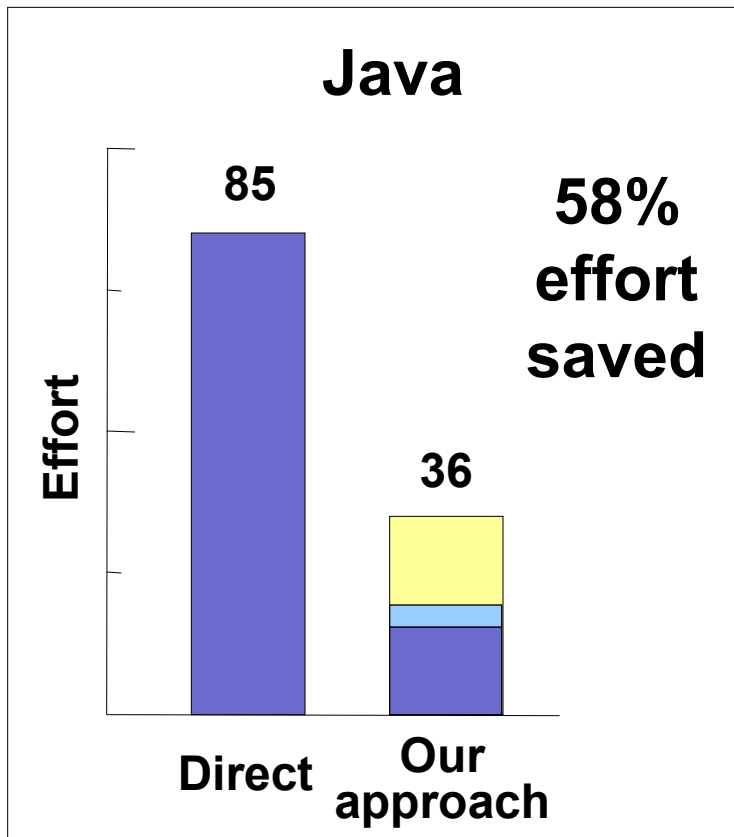
# Approach Evaluation

- Compare effort in our approach vs. direct approach:
  - # rewriting rules + # structures + # printing rules



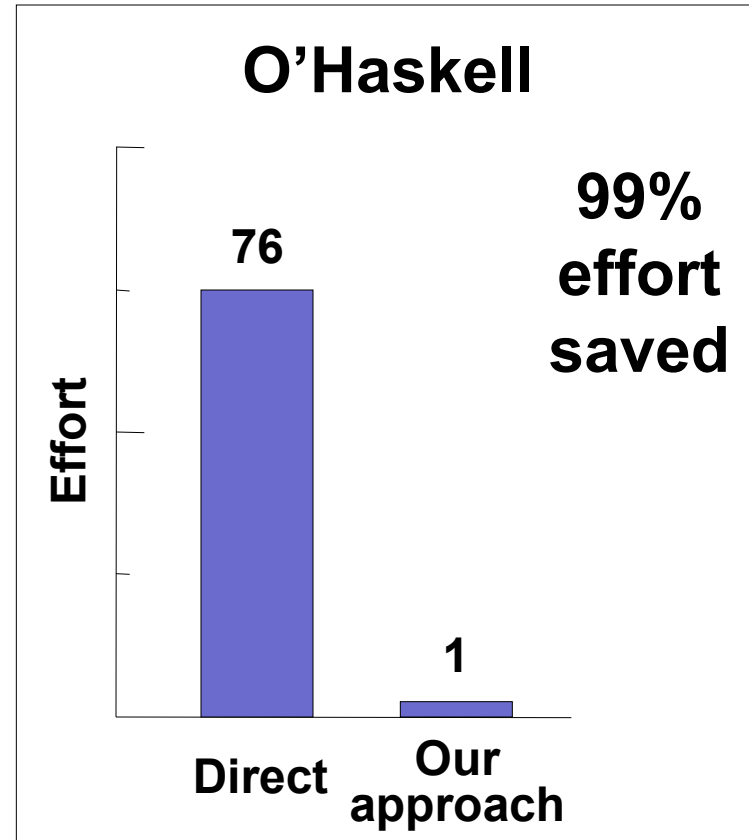
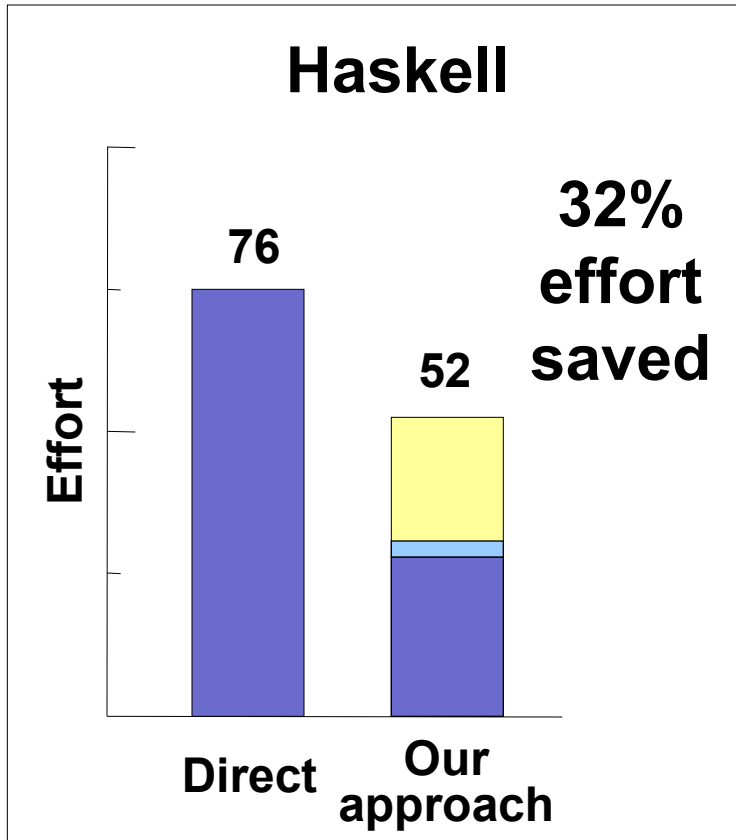
# Evaluation Results

- Imperative languages



# Evaluation Results

- Functional languages



# Evaluation Results

	Direct Approach	Proposed Approach
Java	85	36
Python	99	59
Haskell	76	52
O'Haskell	76	1
Total	316	148

~50% effort could be saved on average



# Conclusion

- Proposed a novel technique to generate assertion code from OCL constraints
- Used structural similarities of programming languages to enable reusing in OCL translators for multiple languages
  - Saving manual effort
- Run an experiment to show the claimed effort savings by using our approach



# Future Work

- Uncovered functionalities of OCL
  - History expressions (@pre)
  - OCL messages
  - allInstances, etc.
- Hierarchy Limitations
  - Difficult to extend hierarchy from the middle
  - Does not allow multiple parent nodes (e.g. useful for implementing an imperative/functional hybrid)
- Real case study

