Generative Programming, Interface-Oriented Programming, and Source Transformation Systems

L. Robert Varney
D. Stott Parker

University of California, Los Angeles

UCLA
COMPUTER SCIENCE DEPARTMENT

Background

- Implementation Bias in Programming Languages and Generative Systems:
  - Interface clients and class specializers must explicitly name and compose implementation-oriented units
- Interface-Oriented Programming (IOP):
  - A solution to the implementation bias problem:
    - Strict separation of interface from implementation (even at instantiation dependencies)
    - Partial representations (implementing part of an interface)
    - Representation inference (generative mechanism for automatic composition of partial representations)

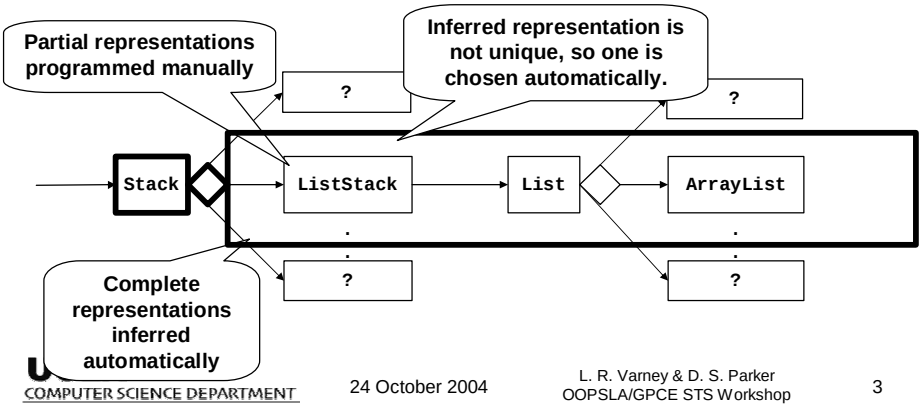
Interface-Oriented Programming

```
Stack s = new ListStack( new ArrayList() );
```

Implementation-Biased OOP

```
Stack s = new Stack();
```

IOP



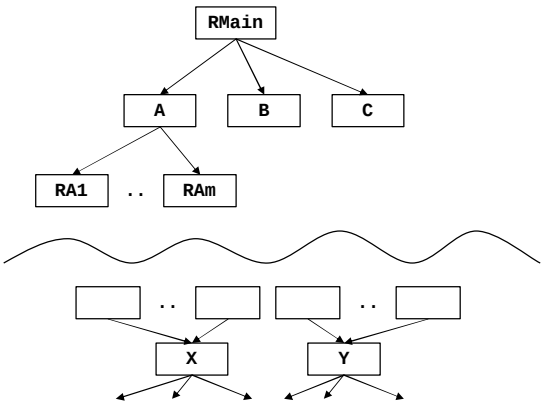
Evolutionary Representation Selection

```
Main m = new Main(args);
```

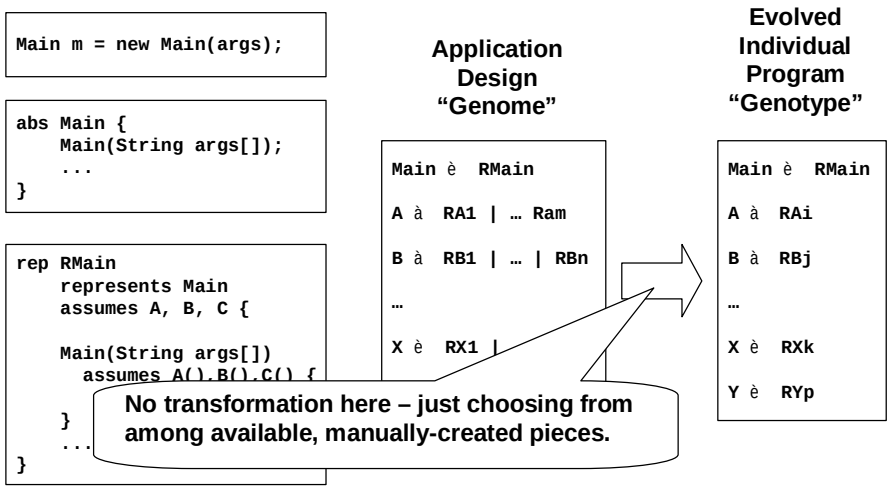
```
abs Main {
    Main(String args[]);
    ...
}
```

```
rep RMain
    represents Main
    assumes A, B, C {

    Main(String args[])
      assumes A(),B(),C() {
        ...
    }
    ...
}
```



Evolutionary Representation Selection
(Grammatical Evolution)

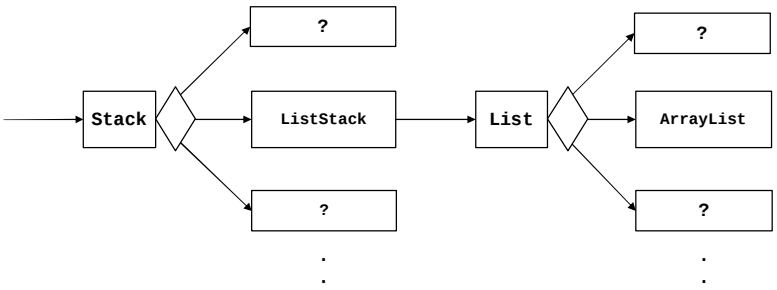


IOP and STS

- IOP:
  - An incremental programming model that avoids implementation bias, BUT:
    - Use of abstraction increases the need for optimizations
    - Separation of interface from implementation makes it harder to apply source transforms.
    - Representation inference and selection need a rich source of implementation variants.
- STS:
  - A way to optimize (specialize) abstract code, BUT:
    - Difficult to use specialzed code without violating encapsulation [Guyer99]
    - Optimization opportunity depends on choice and order of specific transformations [Baage03]
    - Manual transform application fixes order and loses opportunities for optimization [Guyer99]
    - ***STS specializes abstractions but introduces another form of implementation bias …?***
- IOP + STS:
  - Complementary strengths and weaknesses?

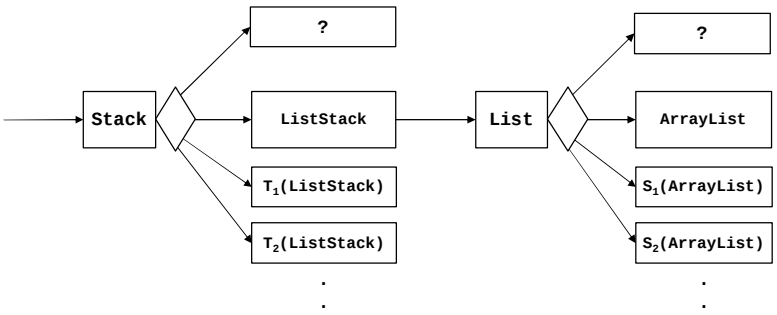
Research Question 1

- How can we use source transformations to generation implementation variants in an IOP system?



Research Question 1

- How can we use source transformations to generation implementation variants in an IOP system?



Research Question 2

- Can we apply IOP-like abstraction and inference techniques to reduce implementation bias of source transformations?
  - Can we infer specific complete transformations from abstract partial ones?
  - What is the “interface” of a transformation (as opposed to its “implementation”)?