Language-Independent Detection of Object-Oriented Design Patterns

Johan Fabry
Vrije Universiteit Brussel, Programming Technology Lab
johan.fabry@vub.ac.be

Overview

- Research Goal
- DMP and Soul
- Soul to SoulJava
- Base-Level Reification
- Design Pattern Detection
- Conclusion & Future Work

Research Goal

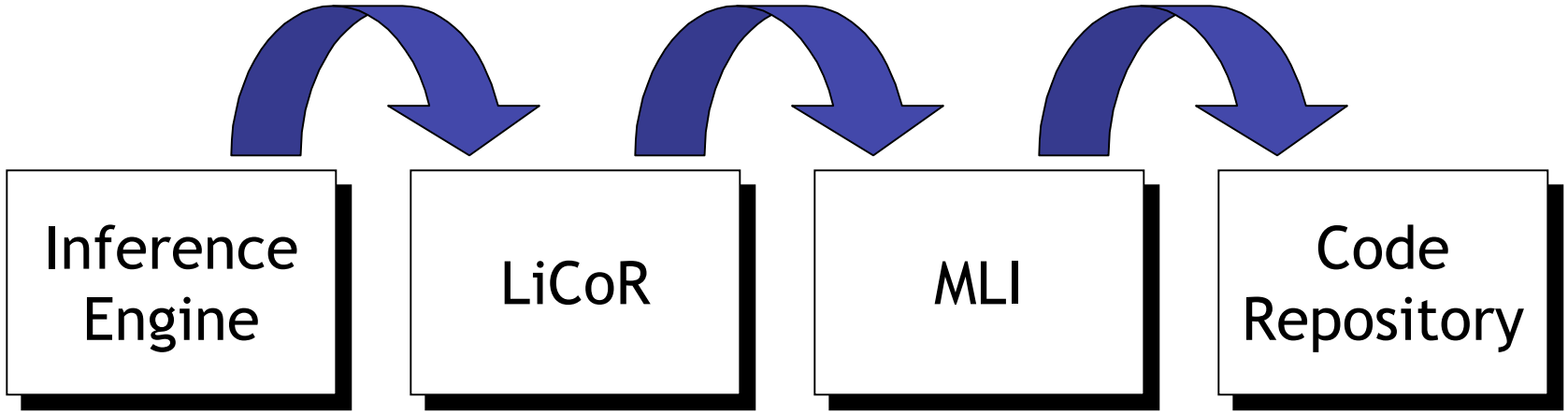
- Language-agnostic reasoning about OO programs
- Validate research in different OO - languages
  - Soul: Reasoning about Smalltalk (coding conventions, DP, bad smells, …)
- Clear separation language indep / dep
- Validation: ST + Java detect BPP, DP

DMP and Soul

- DMP:
  - Declarative language at meta-level
  - OO language at base level
  - Meta-level programs reason about structure of base-level
- Soul:
  - Meta-level prolog-like programming language
  - Smalltalk code can be embedded in prolog code

Soul Layering

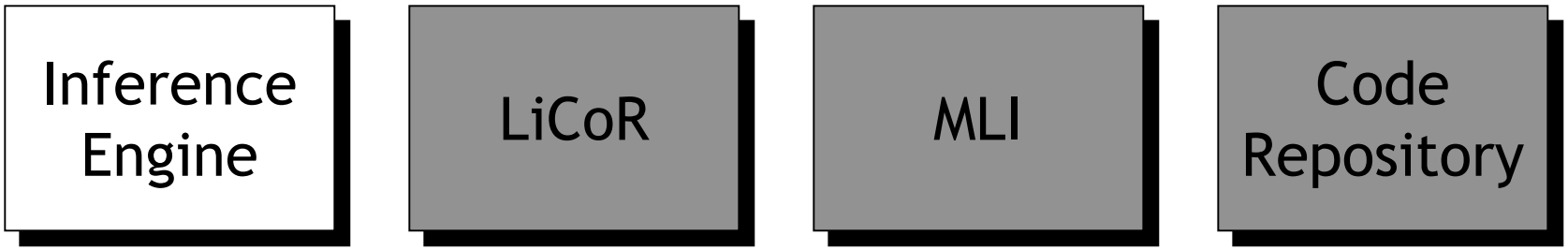
'Uses' Layering:



Inference engine

- Logic query with a variable:

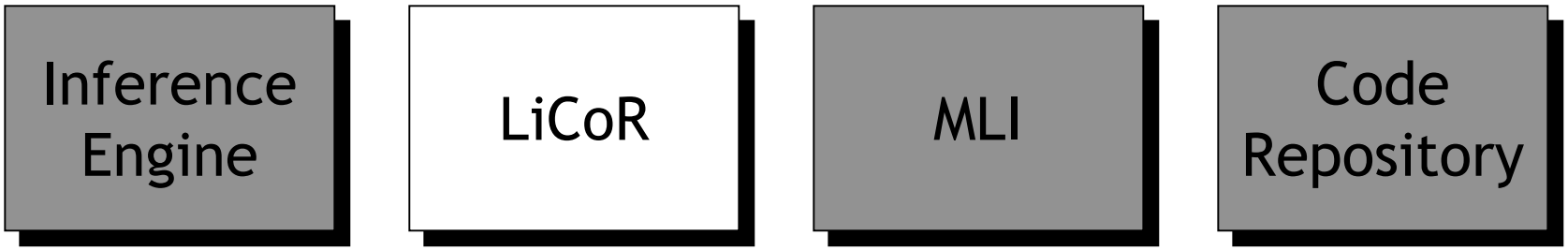
```
If class(?x)
```



LiCoR

- Appropriate rule is triggered:

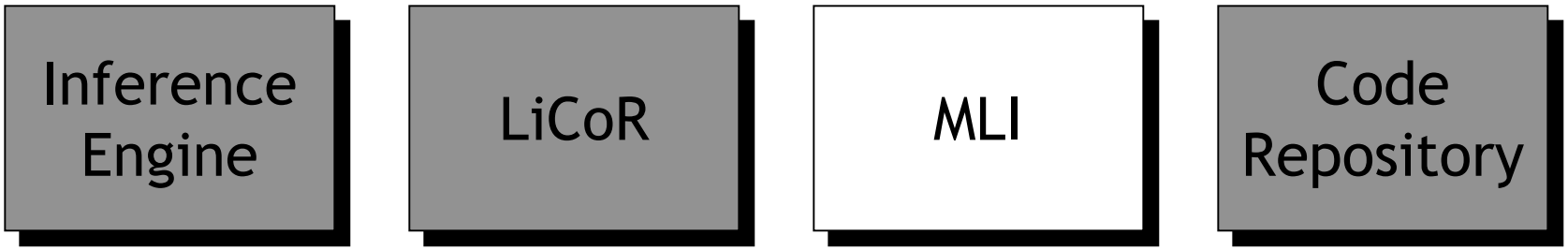
```
class(?c) if variable(?c),
    generate(?c,[ExplicitMLI
        current allClasses)])
```



Meta Level Interface

- Link to the base-level:

```
allClasses
        ^Smalltalk allClasses
```

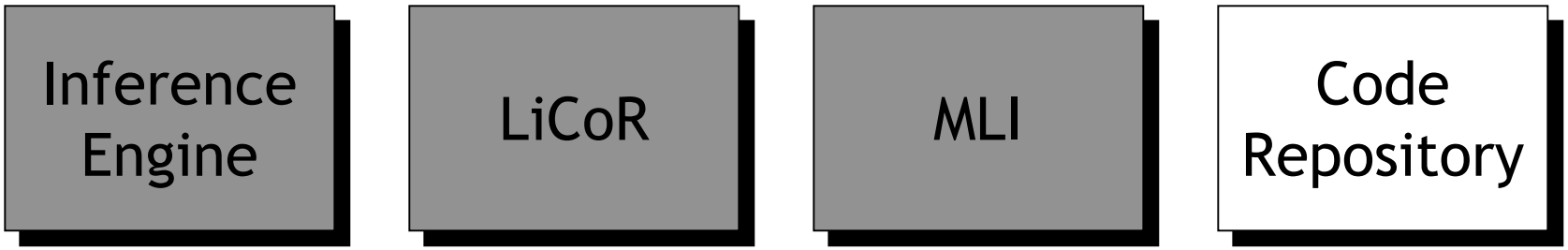


Code Repository

- Investigate the code:

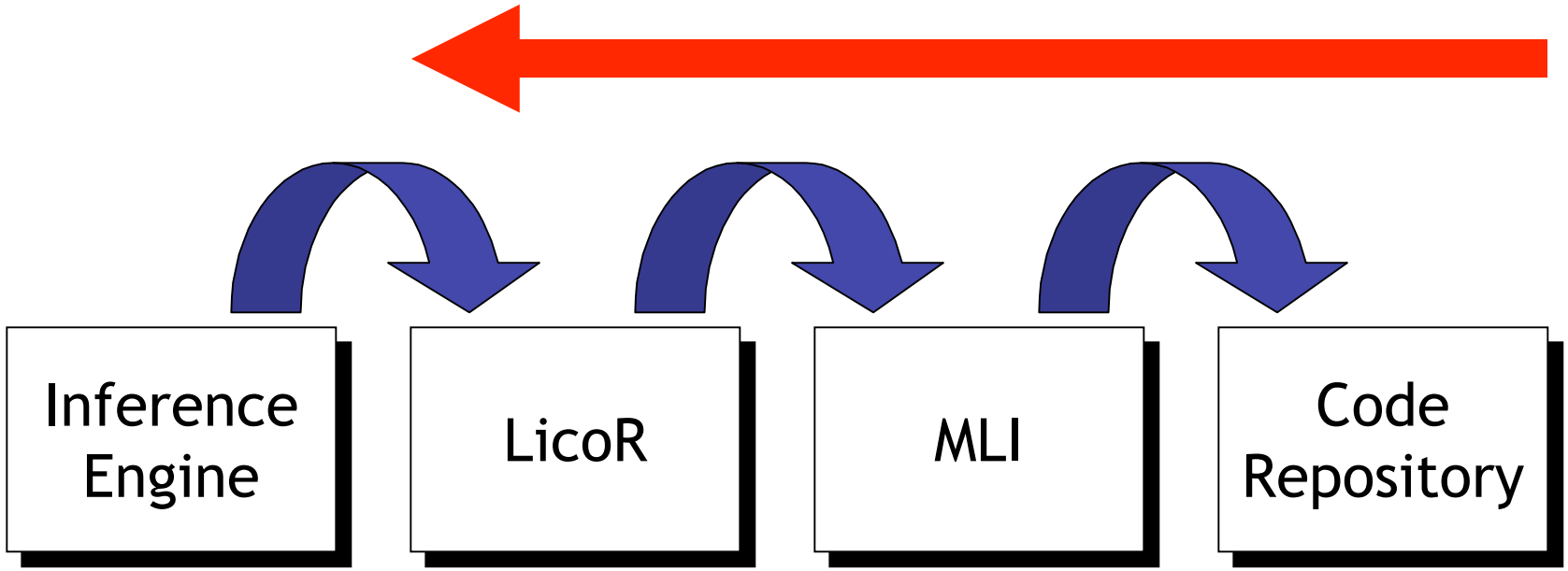
  `allClasses`

  …



Soul to SoulJava

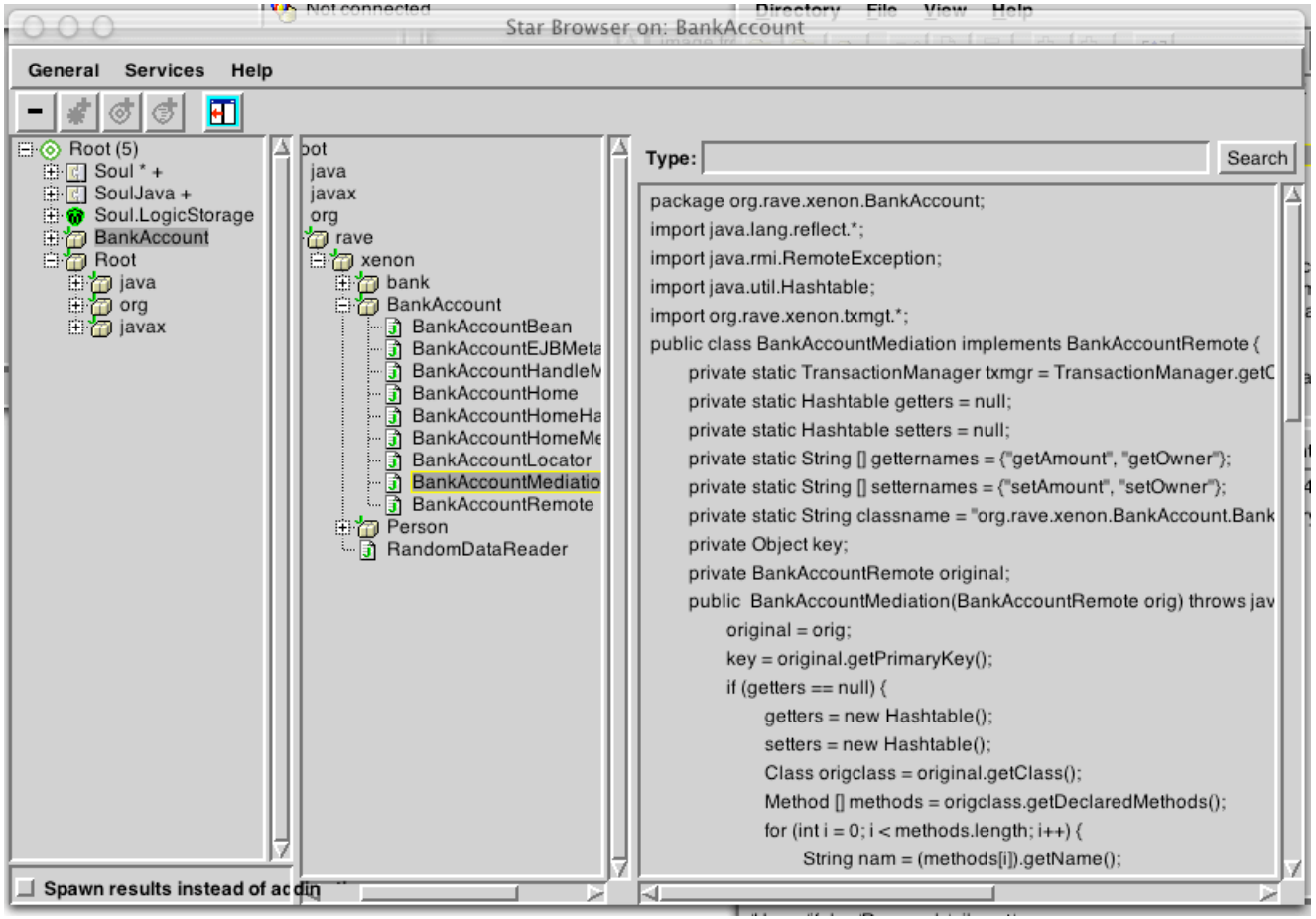
We discuss changes in inverse order.



Java Code Repository

- Use Frost to parse Java Code
  - Assume correct java code !
  - Java 1.0 (+epsilon) parser
- Java Code = parse trees
- .java ‘File in’ in File Browser
- Java Code Browser

Java Code Browser



MLI For Java

- Uses Java Code Repository
- Implements
  - all methods of ST MLI
  - methods for interface support

Base-Level Reification

- Namespace [name|super|sub|classes]
  - ->Through MLI
- Class [name|super|sub|ns|methods|iv]
  - ->Through MLI
- InstVar [name|inClass|type]
  - ->Through MLI
- Method [name|arglist]
  - ->Through MLI

Method Body Reification

- Method bodies = logic parse tree form
- Smalltalk: `count set: 0` =
  `send(variable([#count])),[#set:],<[0]>)`
- Java: `count.set(0)` =
  `send(variable([#count])),[#set],<[0]>)`

- Allows for easy verification:

```
isMessageSend(send(?receiver,
  ?message,?arguments))
```

Method Body Reification

- But Java has more PT elements!
  - if-then-else, while, for, …
- Introduce new logic functor for each
- Logic Java Parse trees have different form!
  - No transformation to 'ST-compatible' form
  - Logic PT as similar as possible to Java code
  - Needed for later work: Java PT manipulations
- Users of parse tree must take this into account

Parse Tree Traversal Layer

- PTT: Specific layer in LiCoR
- Recursively traverses parse trees
- Called with:
  - ?found predicate
  - ?process predicate
  - ?env for results
- Provide lang-spec implementation

Parse Tree Traversal Layer

- PTT: Specific layer in LiCoR
- Recursively `isMessageSend` trees
- Called with:
  - ?found predicate
  - ?process predicate
  - ?env for results
- Provide lang-spec implementation

Parse Tree Traversal Layer

- PTT:
- Recu
- Calle
  - ?found predicate
  - ?process predicate
  - ?env for results
- Provide lang-spec implementation

```
messageSendMessage(
        send(?rec,?msg,?args),
        ?args)
```

Parse Tree Traversal Layer

- PTT: Specific layer in LiCoR
- Recursively traverses parse trees
- Called with:
  - ?found predicate
  - ?process predicate
  - ?env for results
- Provide lang-spec implementation

Idioms

- Lang specific naming and coding conventions
- Accessor method name:
  - Smalltalk: var name
  - Java: ‘get’ + capitalized var name

Pattern Detection

- 4 patterns in 4 apps

Smalltalk	Java
HotDraw	Drawlets
RefactoringBrowser	JRefactory

Best Practice Patterns

Double Dispatch

- One language-independent rule
- Two idiom rules
  - selfReference
  - varName
- HotDraw: 0, Drawlets:3
- RefactoringBrowser: 17, JRefactory 174

Best Practice Patterns

Getting Method

- One language-independent rule
- Two idiom rules
  - methodSelector
  - gettingMethodName
- HotDraw: 35/75, Drawlets:33/270
- RBrowser: 125/531, JRefactory 134/721

Design Patterns

Template Method

- Three language-independent rules
- Three idiom rules
  - abstractSelector, abstractMethod
  - selfReference
- HotDraw Fig: 3, Drawlets Fig: 42 (19 I)
- RBrowser: 43, JRefactory: 50

Design Patterns

Template Method

- Three language-independent rules
- Three idiom rules
  - abstractSelector, abstractMethod
  - selfReference
- H
- RBrowser: 43, JRefactory: 50

Takes care of interfaces in Java

Design Patterns

Template Method

- Three language-independent rules
- Three idiom rules
  - abstractSelector, abstractMethod
  - selfReference

- HotDraw Fig: 3, Drawlets Fig: 42 (19 I)
- RBrowser: 43, JRefactory: 50

Design Patterns

Visitor

- Two language-independent rules
  - Use double-dispatch rule
- Zero idiom rules

- HotDraw: 0, Drawlets: 0
- RBrowser: 14, JRefactory: 174

Conclusion

- Feasible to reason about OO software in a language-independent way
- Soul to SoulJava
  - Source Code Repository (Parser & Storage)
  - Meta-level Interface (Implement API)
  - Parse tree & traversal (Convertor & Logic Rules)
  - Idiom layer (As Needed)
- Growth Idiom Layer < growth Detection layer

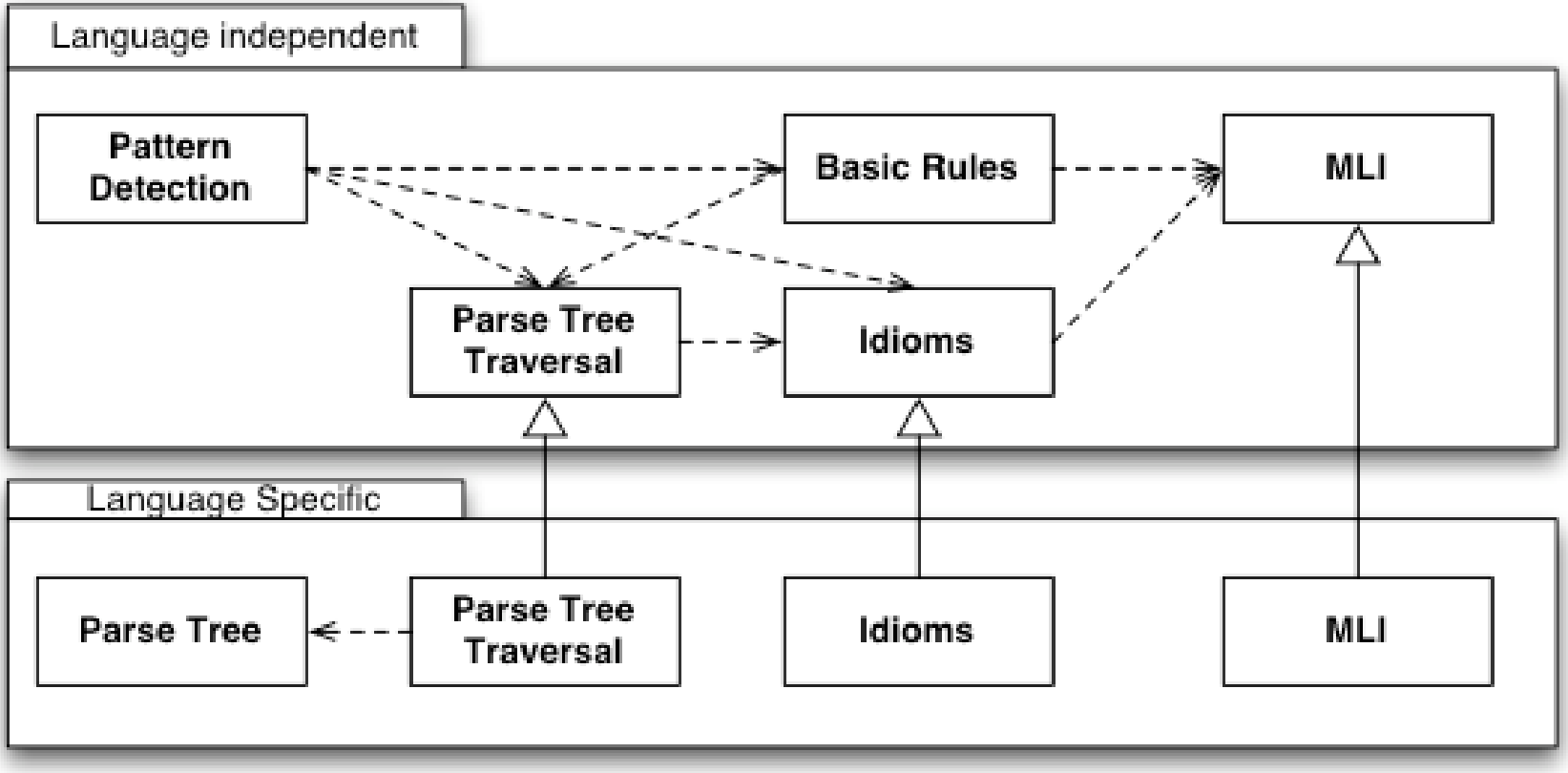
Future Work

- More pattern detection rules
  - Easy / Tricky / Too language-specific
- New languages
  - Small impact on current language-indep rules
- New language versions
- Reasoning about Java Bytecodes
- Type inferencing
- Language-Independent code generation
  - From UML, pattern descriptions

FAQ

- Static Typing vs Dynamic Typing
  - No Type Inferencing (Yet)
  - Types in var declarations: 1 idiom rule
- Interfaces
  - Not widely used in patterns: 1 idiom rule
- Accuracy of detection
  - No false positives, no false negatives
  - Similar limitations as other approaches

LiCoR Structure



```
public void getFoo(){
   if (foo == null)
     foo = factory.boot();
   return foo;}


public void getFoo() {
   foo = factory.boot(foo == null);
   return foo;}
```