Midterm Winter 2002

Karl Lieberherr

COM 1205

- has just been renamed from
  - Software Design and Development to
  - Advanced Pattern Matching and Brain Exercise
- motivation:
  - software development will change, but
  - your pattern matching skills will be useful for the rest of your lives

Three-way pattern matching

- Our brains are good at matching patterns
  - Given two instances of a pattern, find the correspondences between the instances and the pattern
    - pattern = class dictionary
    - instance 1: sentence
    - instance 2: object

```
pointcut X1(S s1) : this(s);
```

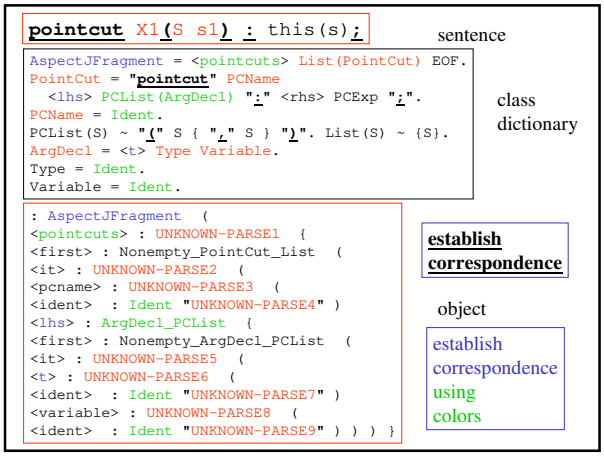
sentence

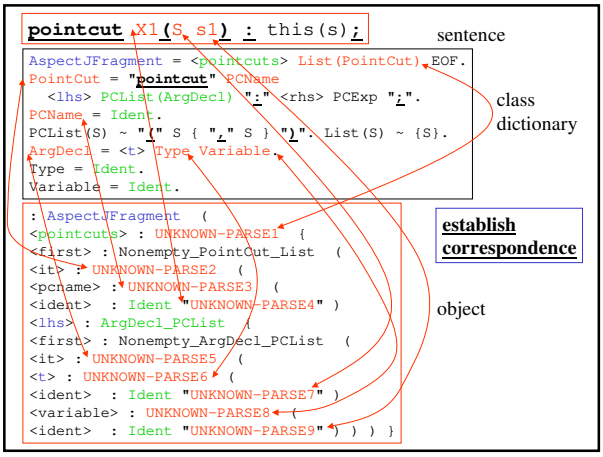
```
AspectJFragment = <pointcuts> List(PointCut) EOF.
PointCut = "pointcut" PCName
  <lhs> PCList(ArgDecl) ":" <rhs> PCExp ";".
PCName = Ident.
PCList(S) ~ "(" S { "," S } ")". List(S) ~ {S}.
ArgDecl = <t> Type Variable.
Type = Ident.
Variable = Ident.
```

class dictionary

```
: AspectJFragment  (
<pointcuts> : UNKNOWN-PARSE1  {
<first> : Nonempty_PointCut_List  (
<it> : UNKNOWN-PARSE2  (
<pcname> : UNKNOWN-PARSE3  (
<ident>  : Ident "UNKNOWN-PARSE4" )
<lhs> : ArgDecl_PCList  {
<first> : Nonempty_ArgDecl_PCList  (
<it> : UNKNOWN-PARSE5  (
<t> : UNKNOWN-PARSE6  (
<ident>  : Ident "UNKNOWN-PARSE7" )
<variable> : UNKNOWN-PARSE8  (
<ident>  : Ident "UNKNOWN-PARSE9" ) ) ) }
```

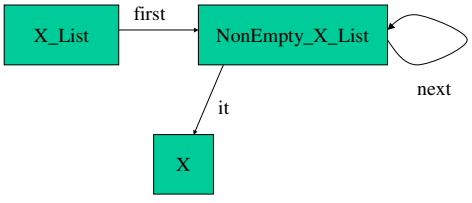
object

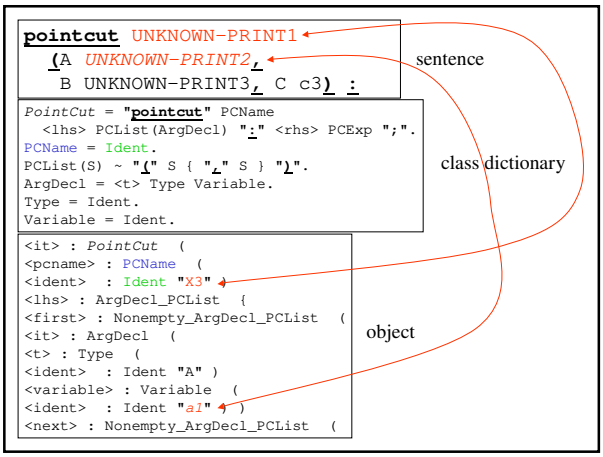




For phases 3 and 4: List Structure

- List(X)





Two-way pattern matching

- Given a pattern and and a use of it, find correspondences between pattern and usage.
  - pattern = method
  - usage = inlined code

```
Iterator it = used.iterator();
while (it.hasNext()){
    Variable curvar = (Variable) it.next();
    System.out.println(); curvar.print();
}
```

```
void                    java.util.Collection used
static UNKNOWN-ABS3 show(UNKNOWN-ABS4) {
    UNKNOWN-ABS5
    while (it.hasNext()){
        UNKNOWN-ABS6
        System.out.println(); curvar.print();
    }
```

establish correspondence

make method as general as possible! Who supports iterator()?

Two-way pattern matching

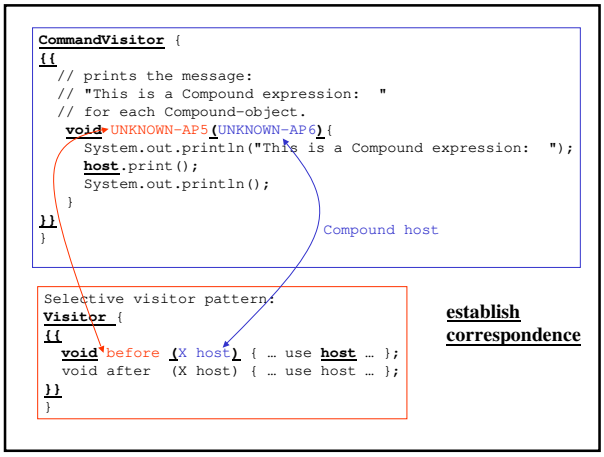
- Given a pattern and and a use of it, find correspondences between pattern and usage.
  - pattern = a programming pattern
  - usage = an application in a specific context

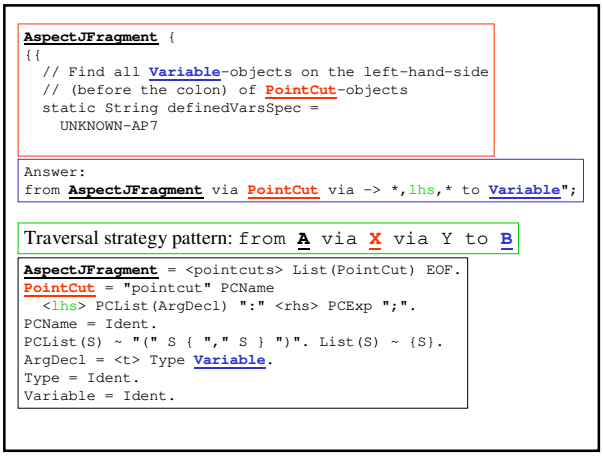
```
AspectJFragment {
{{
  // method process finds all Compound-objects inside
  // an AspectJFragment-object and prints the message:
  // "This is a Compound expression:  "
  // for each Compound-object.
  void process() {
    UNKNOWN-AP1;        CommandVisitor cV = new CommandVisitor();
    Main.cg.UNKNOWN-AP2(UNKNOWN-AP3, UNKNOWN-AP4, cV);
  }
}}
}
```

this

"from AspectJFragment to Compound"

DJ programming pattern:
class-graph.traverse(what, where-to-go, what-to-do)
what = an object
where-to-go = a traversal strategy
what-to-do = a visitor


CommandVisitor {
{{
// prints the message:
// "This is a Compound expression: "
// for each Compound-object.
void UNKNOWN-AP5(UNKNOWN-AP6){
System.out.println("This is a Compound expression: ");
host.print();
System.out.println();
}
}}
}
Compound host
Selective visitor pattern:
Visitor {
{{
void before (X host) { … use host … };
void after (X host) { … use host … };
}}
}
establish correspondence


AspectJFragment {
{{
// Find all Variable-objects on the left-hand-side
// (before the colon) of PointCut-objects
static String definedVarsSpec =
UNKNOWN-AP7
Answer:
from AspectJFragment via PointCut via -> *,lhs,* to Variable";
Traversal strategy pattern: from A via X via Y to B
AspectJFragment = <pointcuts> List(PointCut) EOF.
PointCut = "pointcut" PCName
<lhs> PCList(ArgDecl) ":" <rhs> PCExp ";".
PCName = Ident.
PCList(S) ~ "(" S { "," S } ")". List(S) ~ {S}.
ArgDecl = <t> Type Variable.
Type = Ident.
Variable = Ident.