Scrap your Boilerplate with Object Algebras!

Author1[1] Author2[2]

[1]The University of Hong Kong
author1@cs.hku.hk
[2] The University of Hong Kong
author2@cs.hku.hk

Abstract. This is the abstract ...

1 Introduction

This is the Introduction.

2 Overview

Raise the problem.

```
public interface Exp {
 Value query();
}
public interface Value {
 Integer getInt();
 List<String> getStringList();
}
public class VInt implements Value {
 int x;
 public VInt(int x){this.x = x;}
 @Override
 public Integer getInt() {
  return x;
 }
 @Override
 public List<String> getStringList() {
  return null;
 }
}
public class IntLit implements Exp{
 int x;
 public IntLit(int x){this.x = x;}
 @Override
 public Value query() {
  return new VInt(x);
 }
}
public class IntAdd implements Exp{
 Exp left, right;
```

```
public interface FreeVarsExpAlg extends ExpAlg<String[]>{
 @Override
 default String[] Var(String s){
  return new String[]{s};
 }
 @Override
 default String[] Lit(int i){
  return new String[]{};
 }
 @Override
 default String[] Add(String[] e1, String[] e2){
  int e1len = e1.length;
  int e2len = e2.length;
  String[] res = new String[e1len+e2len];
  System.arraycopy(e1, 0, res, 0, e1len);
  System.arraycopy(e2, 0, res, e1len, e2len);
  return res;
 }
}
```

Fig. 1. Query Example: Get Free variables

```
 public IntAdd(Exp left, Exp right){
  this.left = left;
  this.right = right;
 }
 @Override
 public Value query() {
  return new VInt(left.query().getInt() + right.query().getInt());
 }
}
```

It is easy to add new data variant, but adding new operation is difficult. Object Algebras is a good solution to Expression Problem.

```
public interface ExpAlg<Exp> {
 Exp Var(String s);
 Exp Lit(int i);
 Exp Add(Exp e1, Exp e2);
}
```

Now think about two very useful operations, query and transform.

3 Queries

How to write generic queries by hand

Introduce Monoid

```
public interface Monoid<R> {
   R join(R x, R y);
   R empty();
   default R fold(List<R> lr){
```

```
public class LitIncreaseTransform implements ExpAlg<G_Exp>{
 @Override
 public G_Exp Add(G_Exp e1, G_Exp e2) {
  return new G_Exp() {
   @Override
   public <Exp> Exp accept(ExpAlg<Exp> alg) {
    return alg.Add(e1.accept(alg), e2.accept(alg));
   }
  };
 }
 @Override
 public G_Exp Lit(int i) {
  return new G_Exp() {
   @Override
   public <Exp> Exp accept(ExpAlg<Exp> alg) {
    return alg.Lit(i*2);
   }
  };
 }
 @Override
 public G_Exp Var(String s) {
  return new G_Exp() {
   @Override
   public <Exp> Exp accept(ExpAlg<Exp> alg) {
    return alg.Var(s);
   }
  };
 }
}
```

Fig. 2. Transform Example: Substitute Variables

```
    R res = empty();
    for (R r: lr){
     res = join(res, r);
    }
    return res;
  }
}
```

Free Variables Monoid
Generic Query by hand with Monoid
Free Vars Query

4 Transformations

Generic classes with accept methods

```
public interface G_Exp {
 <Exp> Exp accept(ExpAlg<Exp> alg);
}
```

```
public class FreeVarsMonoid implements Monoid<String[]>{
 @Override
 public String[] empty() {
  return new String[]{};
 }
 @Override
 public String[] join(String[] e1, String[] e2) {
  int e1len = e1.length;
  int e2len = e2.length;
  String[] res = new String[e1len+e2len];
  System.arraycopy(e1, 0, res, 0, e1len);
  System.arraycopy(e2, 0, res, e1len, e2len);
  return res;
 }
}
```

Fig. 3. Free Variables Monoid

Generic Transformation by hand with acceptor interface
Substitute Variables Transformation

5 Object Algebras Framework

Users still need to write generic queries and transformation by hand. It will be great if these boilerplate codes can be generated automatically. Solution: Object Algebra Framework Java annotation tool

```
@Algebra
public interface ExpAlg<Exp> {
 Exp Var(String s);
 Exp Lit(int i);
 Exp Add(Exp e1, Exp e2);
}
```

With the annotation "‘Algebra"’, the framework will generate the boilerplate codes for us automatically. Generic Query using monoid Generic Acceptors Generic transformation with acceptor interfaces

Furthermore, the monoid interface is also included in the Object Algebras Framework

For query, only need to write interesting cases, for example, the return free variable names, and the specific monoid needed, which in Free Variables case will be A String List Monoid.

6 Other Features

One More section

```
public class QueryExpAlg<Exp> implements ExpAlg<Exp> {
  private Monoid<Exp> m;
  public Monoid<Exp> m() { return m; }
  public QueryExpAlg(Monoid<Exp> m) {
    this.m = m;
  }
  public Exp Add(Exp p0,Exp p1) {
    Exp res = m.empty();
    res = m.join(res, p0);
    res = m.join(res, p1);
    return res;
  }
  public Exp Lit(int p0) {
    Exp res = m.empty();
    return res;
  }
  public Exp Var(String p0) {
    Exp res = m.empty();
    return res;
  }
}
```

Fig. 4. Generic Query by hand with Monoid

```
public class FreeVarsQueryExpAlg extends QueryExpAlg<String[]>{
  public FreeVarsQueryExpAlg(Monoid<String[]> m) {super(m);}
  public String[] Var(String s){return new String[]{s};}
}
```

Fig. 5. Free Vars Query

7 Case Study

Case study section

8 Related Work

Finally related work.

9 Conclusion

And conclusion.

Acknowledgements We should thank someone!

```
public interface ExpAlgTransform extends ExpAlg<G_Exp> {
 @Override
 default G_Exp Add(G_Exp e1, G_Exp e2) {
  return new G_Exp() {
   @Override
   public <Exp> Exp accept(ExpAlg<Exp> alg) {
    return alg.Add(e1.accept(alg), e2.accept(alg));
   }
  };
 }
 @Override
 default G_Exp Lit(int i) {
  return new G_Exp() {
   @Override
   public <Exp> Exp accept(ExpAlg<Exp> alg) {
    return alg.Lit(i);
   }
  };
 }
 @Override
 default G_Exp Var(String s) {
  return new G_Exp() {
   @Override
   public <Exp> Exp accept(ExpAlg<Exp> alg) {
    return alg.Var(s);
   }
  };
 }
}
```

Fig. 6. Generic Transformation by hand with acceptor interface

```
public class SubstVarsTransform implements ExpAlgTransform{
 private String s;
 private G_Exp gExp;
 public SubstVarsTransform(String s, G_Exp gExp){
  this.s = s;
  this.gExp = gExp;
 }
 @Override
 public G_Exp Var(String ss) {
  return new G_Exp() {
   @Override
   public <Exp> Exp accept(ExpAlg<Exp> alg) {
    if (ss.equals(s)) return gExp.accept(alg);
    else return alg.Var(ss);
   }
  };
 }

}
```

Fig. 7. Substitute Variables Transformation