

Reference Attributed Grammars

Program Transformation and Generation Lecture 6

Eelco Visser

Software Engineering Research Group
Delft University of Technology
Netherlands

February 5, 2008

Part I

JastAdd: Attribute Grammars in Java

reference: www.jastadd.org

Binary Numbers

$N ::= L$

$N ::= L "." L$

$L ::= B$

$L ::= L B$

$B ::= "0"$

$B ::= "1"$

0 0

1 1

101 5

0.1 0.5

0.01 0.25

10.01 2.25

1101.01 13.25

[http://jastadd.org/jastadd-tutorial-examples/
knuths-binary-numbers](http://jastadd.org/jastadd-tutorial-examples/knuths-binary-numbers)

Binary Numbers

$B ::= "0"$
 $v(B) = 0$

$B ::= "1"$
 $v(B) = 2 ** s(B)$

$L ::= B$
 $v(L) = v(B)$
 $s(B) = s(L)$
 $l(L) = 1$

$L1 ::= L2 B$
 $v(L1) = v(L2) + v(B)$
 $s(B) = s(L1)$
 $s(L2) = s(L1) + 1$
 $l(L1) = l(L2) + 1$

$N ::= L$
 $v(N) = v(L)$
 $s(L) = 0$

$N ::= L1 "." L2$
 $v(N) = v(L1) + v(L2)$
 $s(L1) = 0$
 $s(L2) = - l(L2)$

Legend
 v : value
 s : scale of rightmost bit
 l : length

Representing Abstract Syntax Trees

```
abstract BinaryNumber;
IntegralNumber : BinaryNumber
  ::= IntegralPart:BitList;
RationalNumber : BinaryNumber
  ::= IntegralPart:BitList FractionalPart:BitList;
```

```
abstract BitList;
SingularBitList : BitList
  ::= Bit;
PluralBitList : BitList
  ::= BitList Bit;
```

```
abstract Bit;
Zero : Bit ::= ;
One  : Bit ::= ;
```

Declaring Attributes

```
// Attributes for Bits
syn double Bit.value();
inh int Bit.scale();

// Attributes for BitLists
syn double BitList.value();
syn int BitList.length();
inh int BitList.scale();

// Attributes for BinaryNumbers
syn double BinaryNumber.value();
```

Attribute Equations (1)

```
eq Zero.value() = 0;
```

```
eq One.value() = java.lang.Math.pow(2.0, scale());
```

```
eq SingularBitList.value() = getBit().value();
```

```
eq SingularBitList.getBit().scale() = scale();
```

```
eq SingularBitList.length() = 1;
```

```
eq PluralBitList.value()  
  = getBit().value() + getBitList().value();
```

```
eq PluralBitList.getBit().scale() = scale();
```

```
eq PluralBitList.getBitList().scale() = scale() + 1;
```

```
eq PluralBitList.length() = getBitList().length() + 1;
```

Attributes Equations (2)

```
eq IntegralNumber.value() = getIntegralPart().value();  
eq IntegralNumber.getIntegralPart().scale() = 0;
```

```
eq RationalNumber.value()  
  = getIntegralPart().value()  
    + getFractionalPart().value();  
eq RationalNumber.getIntegralPart().scale() = 0;  
eq RationalNumber.getFractionalPart().scale()  
  = -getFractionalPart().length();
```

Attribute Modules

```
aspect BinaryNumberValue {  
  ... attribute declarations ...  
  ... attribute equations ...  
}
```

attribute declarations and equations defined in *aspects*
good for extensibility

Implementation

- Each constructor in AST represented by (abstract) Java class
- Tree nodes have pointer to parent node
- Implication: tree copy by pointer sharing not possible
- Dynamic scheduling: attributes are methods, called on demand

```
public class ASTNode<T extends ASTNode> ... {
    public void flushCache() {...}
    public ASTNode<T> clone() {...}
    public ASTNode<T> copy() {...}
    public ASTNode<T> fullCopy() {...}
    public ASTNode() { super(); }
    ... settings ...
    public T getChild(int i) { ... }
    public static ASTNode getChild(ASTNode that, int i) { ...}
    public int getIndexOfChild(ASTNode node) { ... }
    public void addChild(T node) { ... }
    public void setChild(T node, int i) { ... }
    public void insertChild(T node, int i) { ... }
    public void removeChild(int i) { ... }
    public ASTNode getParent() { ... }
    public void setParent(ASTNode node) { ... }
    public java.util.Iterator<T> iterator() { ... }
    public static void reset() { ... }
}
```

PluralBitList.java: Class Hierarchy

JastAdd AST

```
abstract BitList;  
PluralBitList : BitList ::= BitList Bit;
```

Java Implementation

```
public abstract class BitList  
    extends ASTNode  
    implements Cloneable  
{  
    ...  
}  
public class PluralBitList  
    extends BitList  
    implements Cloneable  
{  
    ... next slides ...  
}
```

PluralBitList.java: Tree Operations

JastAdd AST

```
abstract BitList;  
PluralBitList : BitList ::= BitList Bit;
```

Java Implementation in PluralBitList.java

```
public PluralBitList() { ... }  
public PluralBitList(BitList p0, Bit p1) {  
    setChild(p0, 0); setChild(p1, 1);  
}  
... clone() copy() fullCopy() flushCache() ...  
protected int numChildren() { return 2; }  
public void setBitList(BitList node) { setChild(node, 0); }  
public BitList getBitList() { return (BitList)getChild(0); }  
public void setBit(Bit node) { setChild(node, 1); }  
public Bit getBit() { return (Bit)getChild(1); }
```

JastAdd Attribute Equation

```
eq PluralBitList.value()  
  = getBit().value() + getBitList().value();
```

Java Implementation in PluralBitList.java

```
protected boolean value_visited = false;  
public double value() {  
    if(value_visited)  
        throw new RuntimeException(  
            "Circular definition of attr: value in class: ");  
    value_visited = true;  
    double value_value = value_compute();  
    value_visited = false;  
    return value_value;  
}  
private double value_compute() {  
    return getBit().value() + getBitList().value();  
}
```

JastAdd Attribute Declaration

```
inh int BitList.scale();
```

Java Implementation in BitList.java

```
protected boolean scale_visited = false;
public int scale() {
    if(scale_visited)
        throw new RuntimeException(
            "Circular definition of attr: scale in class: ");
    scale_visited = true;
    int scale_value = getParent().Define_int_scale(this, null);
    scale_visited = false;
    return scale_value;
}
```

JastAdd Equations

```
eq PluralBitList.getBit().scale() = scale();  
eq PluralBitList.getBitList().scale() = scale() + 1;
```

Java Implementation in PluralBitList.java

```
public int Define_int_scale(ASTNode caller, ASTNode child) {  
    if(caller == getBitListNoTransform()) {  
        return scale() + 1;  
    }  
    if(caller == getBitNoTransform()) {  
        return scale();  
    }  
    return getParent().Define_int_scale(this, caller);  
}
```

JastAdd Equations

```
eq RationalNumber.getIntegralPart().scale() = 0;  
eq RationalNumber.getFractionalPart().scale()  
  = -getFractionalPart().length();
```

Java Implementation in RationalNumber.java

```
public int Define_int_scale(ASTNode caller, ASTNode child) {  
    if(caller == getFractionalPartNoTransform()) {  
        return -getFractionalPart().length();  
    }  
    if(caller == getIntegralPartNoTransform()) {  
        return 0;  
    }  
    return getParent().Define_int_scale(this, caller);  
}
```

Part II

WebDSL Data Models in JastAdd

Abstract Syntax Tree Definition (1)

```
module foo

  entity User {
    username :: String
    password :: Secret
    topics   -> Set<Topic>
    email    :: Email
  }

  entity Topic {
    name     :: String
    content  :: WikiText
  }

  extend entity Topic {
    acl -> ACL
    authors -> Set<User> (inverse=User.topics)
  }

  entity ACL {
    viewers -> Set<User>
    editors -> Set<User>
  }
```

Abstract Syntax Tree Definition (1)

```
Program ::= Module*;  
Module ::= <ID : String> AbstractSection*;  
  
abstract AbstractSection;  
Section : AbstractSection ::= <ID : String> Declaration*;  
Imports : AbstractSection ::= <ID : String>;  
  
abstract Declaration;  
SortDeclaration : Declaration ::= <ID : String> ;  
BuiltinDecl      : SortDeclaration;  
GenericSortDecl  : SortDeclaration;  
  
Entity : SortDeclaration ::= Property*;  
ExtendEntity : Declaration ::= <ID : String> Property*;
```

Abstract Syntax Tree Definition (2)

```
abstract Property ::= <ID : String> Sort Annotation*;
SimpleProperty  : Property;
CompProperty    : Property;
RefProperty     : Property;

abstract Sort ;
SimpleSort      : Sort ::= <ID : String> ;
GenericSort    : Sort ::= <ID : String> Sort* ;

abstract Annotation;
SimpleAnnotation : Annotation ::= <ID:String>;
InverseAnnotation : Annotation
    ::= SimpleSort <PropertyId : String>;
```

- Sort of property is declared
- Entities declared only once
- Inverse annotation should refer to another defined entity which should have a property refer to the enclosing property of the annotation

lookupSort: Reference Attribute

Return a reference to a node in the tree

```
inh Declaration Sort.lookupSort(String name);

eq Module.getChild().lookupSort(String name)
  = localLookupSort(name);

syn lazy Declaration Module.localLookupSort(String name) {
  for (Declaration d : builtins()) {
    if(d.declaresSort(name)) { return d; }
  }
  for(Declaration d : getDeclarations()) {
    if(d.declaresSort(name)) { return d; }
  }
  return null;
}
```

builtins: Nonterminal (Higher-Order) Attributes

Return (a reference to) a new AST node as attribute value

```
aspect Builtins {  
  syn nta List<Declaration> Module.builtins() {  
    return new List<Declaration>()  
      .add(new BuiltinDecl("String"))  
      .add(new BuiltinDecl("Secret"))  
      .add(new BuiltinDecl("WikiText"))  
      .add(new BuiltinDecl("Email"))  
      .add(new GenericSortDecl("List"))  
      .add(new GenericSortDecl("Set"));  
  }  
}
```

getDeclarations: Collecting Tree Nodes

```
syn Collection<Declaration> Module.getDeclarations()  
    circular [new HashSet<Declaration>()]  
{  
    Set<Declaration> set = new HashSet<Declaration>();  
    for(AbstractSection a : getAbstractSections()) {  
        if (a instanceof Section) {  
            Section s = (Section)a;  
            for(Declaration d : s.getDeclarations()) {  
                set.add(d);  
            }  
        } else if (a instanceof Imports) {  
            Imports i = (Imports)a;  
            set.addAll(i.module().getDeclarations());  
        }  
    }  
    return set;  
}
```

```
syn boolean Declaration.declaresSort(String name)
  = false ;
```

```
eq SortDeclaration.declaresSort(String name)
  = name.equals(getID());
```

declaration: Reference to Declaration of Sort

```
syn Declaration Sort.declaration();
```

```
syn Declaration SimpleSort.declaration()  
  = lookupSort(getID());
```

```
syn Declaration GenericSort.declaration()  
  = lookupSort(getID());
```

Imports

```
aspect Imports {
  syn Module Imports.module() =
    lookupModule(getID());

  inh Module Imports.lookupModule(String name);

  eq Program.getChild().lookupModule(String name)
    = localLookupModule(name);

  syn lazy Module Program.localLookupModule(String name) {
    for(Module m : getModules()) {
      if(m.getID().equals(name))
        return m;
    }
    Module m = Program.parseModule(name + ".app");
    addModule(m);
    return m;
  }
}
```

```
aspect SortWellformedness {  
  
    syn boolean SimpleSort.wellformed()  
        = declaration() != null;  
        // a sort is wellformed if there is  
        // a declaration that defines it  
  
    public void SimpleSort.collectErrors(Collection<String> errors)  
        super.collectErrors(errors);  
        if(!wellformed())  
            errors.add("Sort " + getID() + " is not declared");  
    }  
  
}
```

EntityWellformedness

```
aspect EntityWellformedness {

syn boolean Entity.isUnique()
  // there can be no two entities with the same name
  = lookupSort(getID()) == this;

syn boolean Entity.doesNotDefineBuiltinSort()
  // entity name cannot be a builtin sort
  = !(lookupSort(getID()) instanceof BuiltinDecl);

syn boolean Entity.wellformed()
  = isUnique() && doesNotDefineBuiltinSort();

public void Entity.collectErrors(Collection<String> errors) {
  ... }
}
```

InverseAnnotationWellformedness

```
// annotation should refer to another defined entity which
// should have a property refer to the enclosing property
// of the annotation

syn Entity InverseAnnotation.entity() {
    Declaration d = getSimpleSort().declaration();
    if (d == null || !(d instanceof Entity)) return null;
    else return (Entity)d;
}

syn Property InverseAnnotation.property() {
    if (entity() == null) return null;
    else return entity().localLookupProperty(getPropertyId());
}

syn boolean InverseAnnotation.isInverse()
    = property() != null
    && property().getSort().canContain(enclosingEntity());

syn boolean InverseAnnotation.wellformed()
    = isInverse();
```

canContain

```
syn boolean Sort.canContain(Entity e) = false;

eq SimpleSort.canContain(Entity e)
  = declaration() == e;

eq GenericSort.canContain(Entity e)
  = (getID().equals("List") || getID().equals("Set"))
    && getNumSort() == 1
    && getSort(0).declaration() == e;
```

Next week: Rewritable Reference Attributed Grammars