Katja Laboratory
Abstract Syntax Trees in Java with Katja
Lecture Compiler and Language Processing Tools 2011

Jan Schäfer (slides by Jean-Marie Gaillourdet)

Software Technology Group
TU Kaiserslautern

What's Katja?

- code generator for order-sorted datatypes in Java (and Haskell and Isabelle/HOL)
- goal: simplify generation and handling of ASTs in Java
- enable the implementation of attribute grammar systems (not quite yet)

Design decisions

- functional terms in Java
  - immutable
  - globally shared
  - strutcural equality implies reference equality
- declarative specification as order-sorted datatypes
- statically typed (as far as possible)

Katja - The Command Line Tool

Executing Katja

```
java -jar katja.jar -b java -o -j spec.katja
```

Input: Specification of order-sorted datatype (`*.katja`)

Output:

- SpecificationName.jar
  - Generated sources and class files
- katja-common.jar (only with `-j`)
  - Base library of Katja

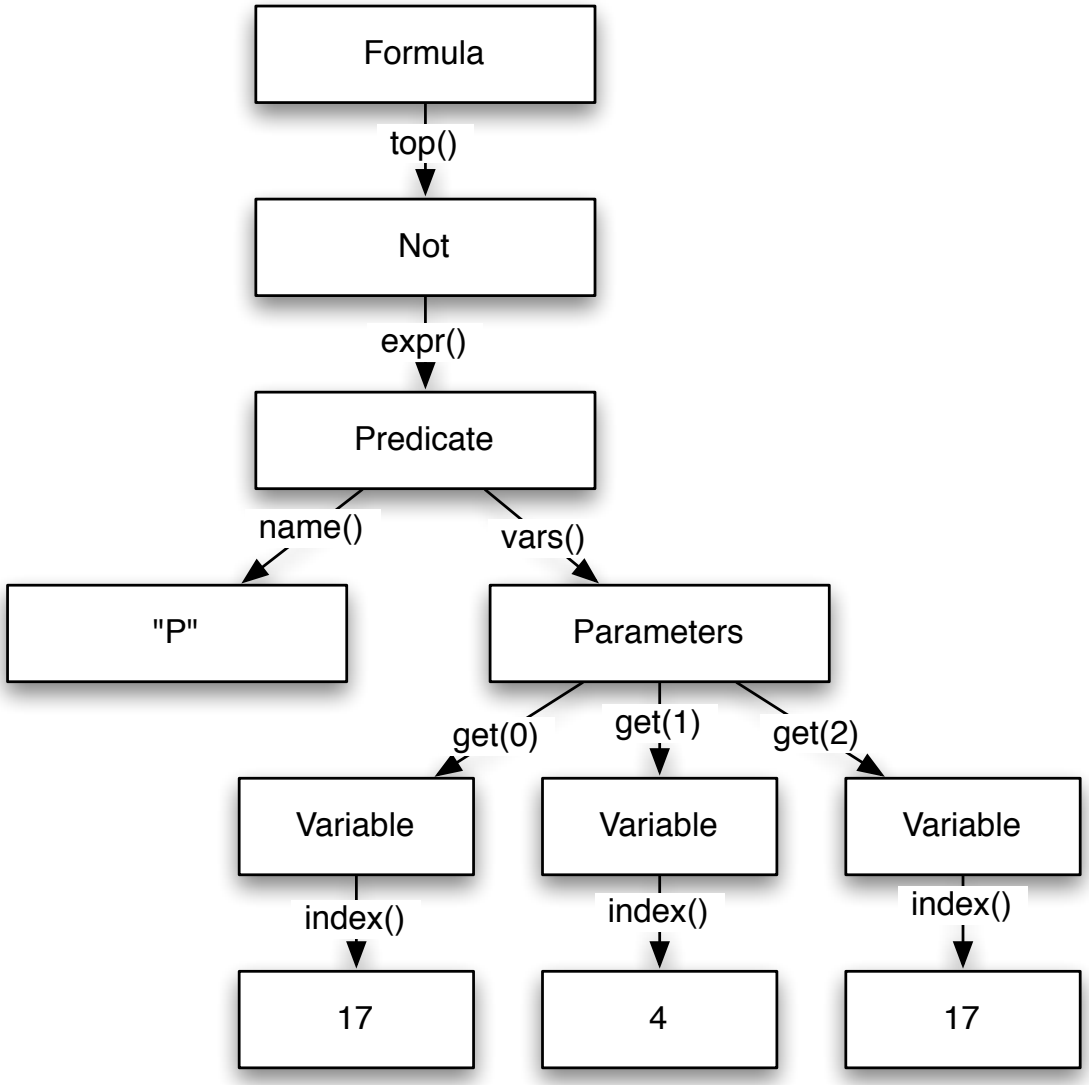
Katja Specifications

```
specification Formulas

external Integer
external String

Formula    ( Expression top )

Expression = False | Implies | Not | Predicate
False      ( )
Implies    ( Expression left, Expression right )
Not        ( Expression expr )
Predicate  ( String name, Parameters vars )

Parameters * Variable
Variable   ( Integer index )
```

Katja Specifications With Syntactic Sugar

```
specification Formulas

external Integer
external String

Formula    ( Expression top )

Expression = False     ( )
           | Implies   ( Expression left, Expression right )
           | Not       ( Expression expr )
           | Predicate ( String name, Parameters vars )

Parameters * Variable
Variable   ( Integer index )
```

Katja Terms


Formula
top()
Not
expr()
Predicate
name()
vars()
"P"
Parameters
get(0)
get(1)
get(2)
Variable
Variable
Variable
index()
index()
index()
17
4
17

What is generated?

- an interface per sort
- a specification class with static constructor methods
- hidden implementations and utility classes/interfaces

Example

```
import static softech.formula.Formulas.*;
import softech.formula.*;

... {
  Formula f = Formula( Not( Predicate( "P",
    Parameters( Variable(17), Variable(4),
                Variable(42)))));
}
```

Last missing part

```
specification Formulas

backend java {
  package softech.formulas
  import java.lang.String
  import java.lang.Integer
}

external String
external Integer
...
```

Expressions

```
Literal    ( Integer value )
Variable   ( String name )
Plus       ( Expression left, Expression right )
Minus      ( Expression left, Expression right )
Mult       ( Expression left, Expression right )
Div        ( Expression left, Expression right )
Expression = Plus | Minus | Mult | Div | Literal | Variable
```

Switch Classes

- simulate switch construct of Java over variants

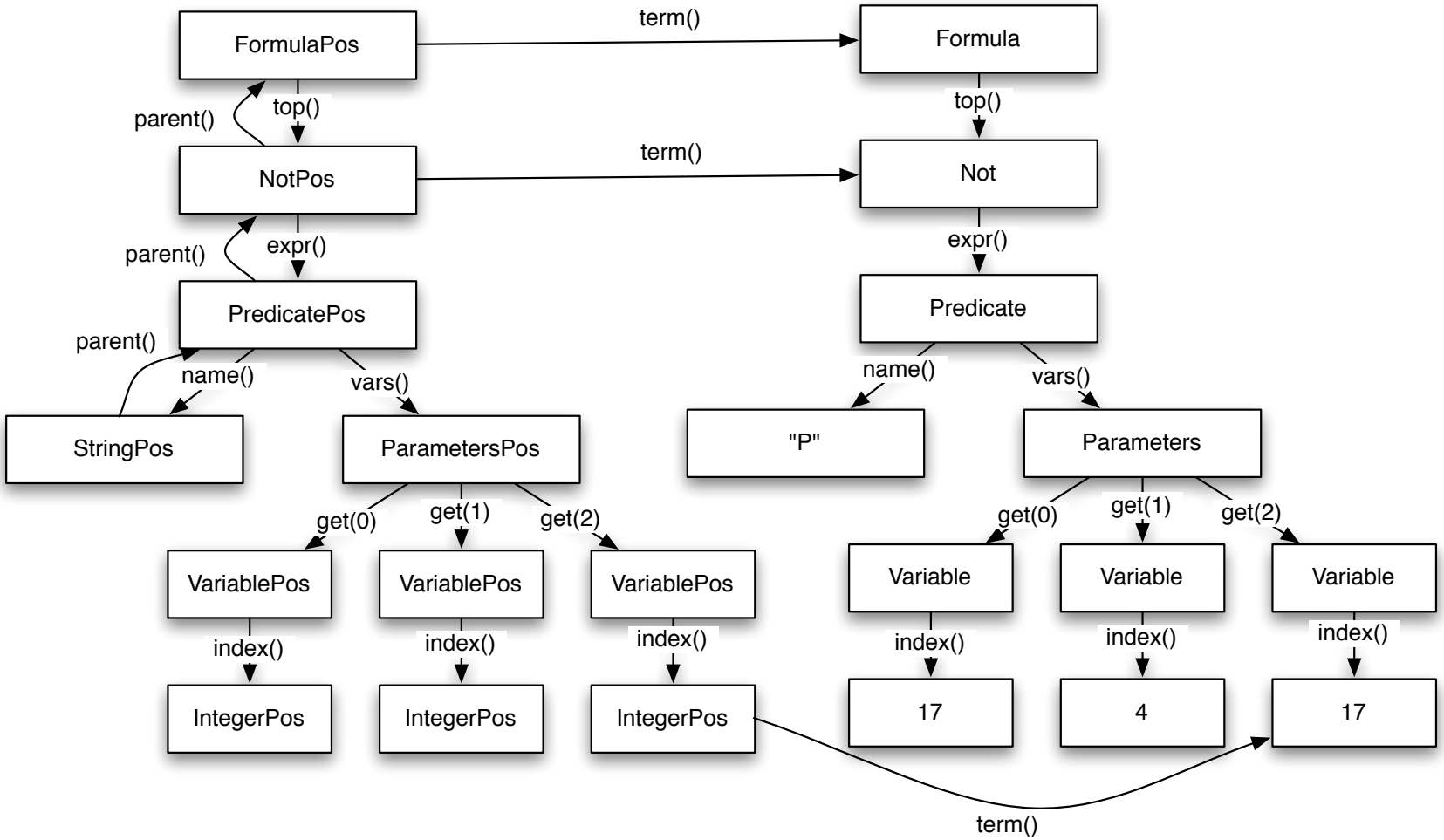
```
...
f.top().Switch( new Expression.Switch<Integer,NE> {
  public CaseFalse() throws NE { ... }
  public CaseImplies(Expression left, Expression right)
    throws NE  { ... }
  public CaseNot(Expression expr) throws NE  { ... }
  public CasePredicate(String name, Parameters vars)
    throws NE { ... }
});
```

Task 1

- create a Katja specification file for expressions
- compile it
- write a method `Integer eval(Expression e)` to evaluate expressions without variables
- use an exception to signal the presence of variables

Solution

Terms vs. Term-Positions



`root Formula Pos`

What is generated?

- a term position interface per sort
- hidden implementations
- one addition constructor method (FormulaPos)
- default visitor
- common super-type (`softech.formula.Formulas.SortPos`)

Visitors

```
interface FormulaPos ... {
  class DefaultVisitor ... {
    void visitFormulaPos(FormulaPos term) { ... }
    void visitNotPos(NotPos term) { ... }
    void visitImpliesPos(ImpliesPos term) { ... }
    ...
  }
}
```

Task 2

- implement a PrettyPrinter based on the DefaultVisitor
- decide whether to print out parenthesis based on the parent

Solution

Modifying Term Positions

- every term position can be replaced by a term of equivalent sort
- immutable!
- `KatjaSort<R> replace(Object o)`
- sometimes you have to use `.cast()`

Post- and Preorder

- every position has a `preOrder()` method, which goes to the next position in pre order
- every position has `postOrder` and `postOrderStart` methods

Task 3

- implement a method simplify which simplifies constant expressions to constants
- use a post order path through the positions