Redesigning the implementation of Mathematical formulas

The SBML Java™ library

Victor Kofia

Math Package Redesign

- ASTNode
  - Large class ~ 4000 lines of code
  - Represents abstract syntax tree nodes with diverse purposes

- Reasons for redesigning the math package:
  1. Adhere to Software Engineering principles
     - Abstraction
     - Modularity
     - Generality
     - Extensibility
  2. Increase similarity between JSBML and libSBML
  3. Efficiency gains …

What should I expect?

1. I am currently a user of JSBML. What new things should I expect?
   - More succinct and informative documentation
   - A more sensible type hierarchy for the math package
   - Less bugs

2. I am a developer currently working on JSBML, how will the addition of the new math package affect my work?
   - New compilers and parsers may break outdated code
   - Less work will be needed for future extensions
   - Easier to identify and fix bugs

Math Package

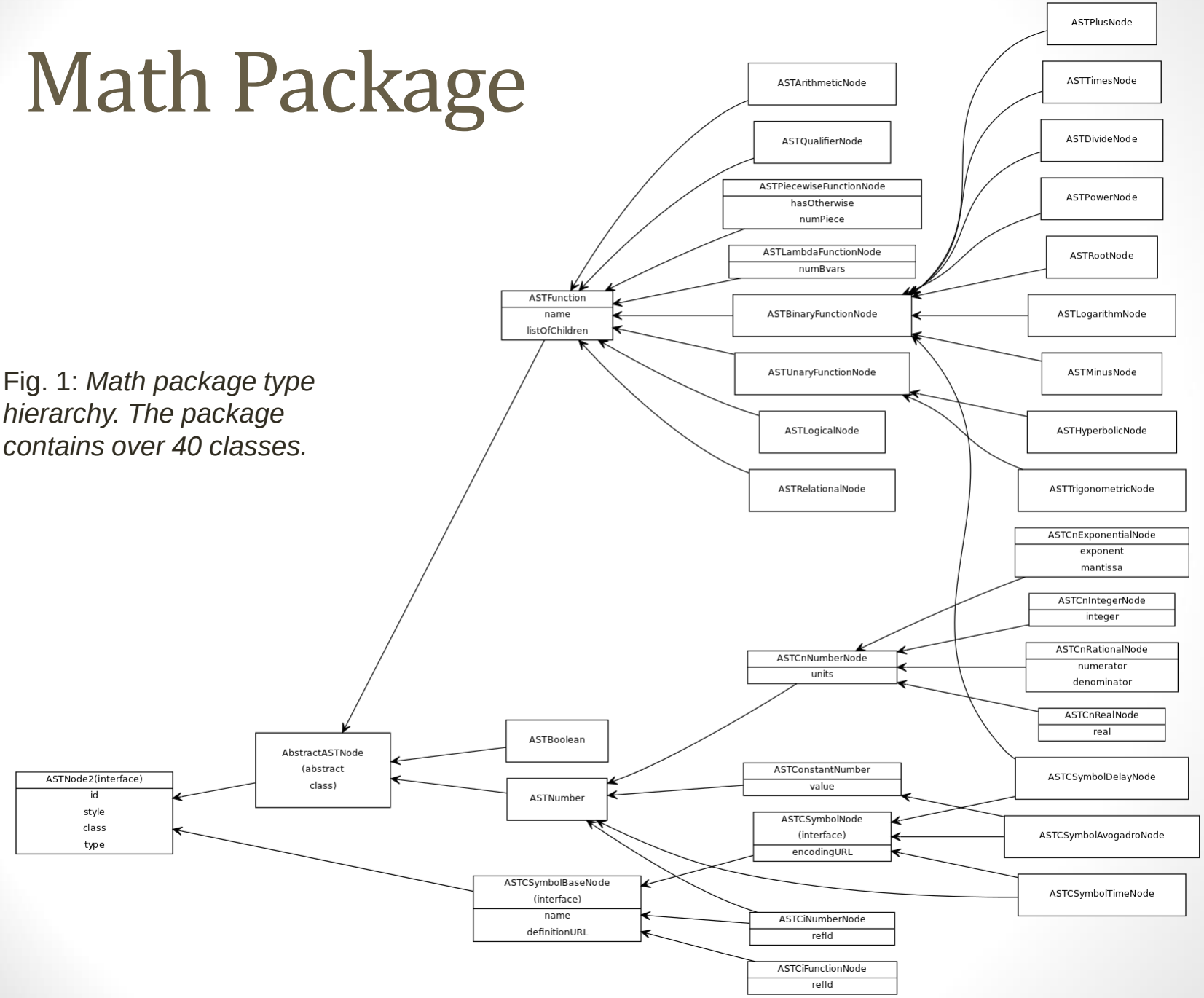


Fig. 1: *Math package type hierarchy. The package contains over 40 classes.*

ASTNode2

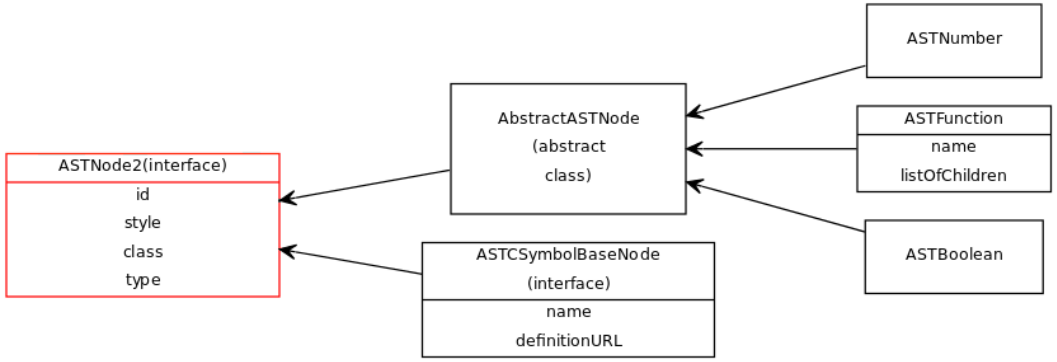


Fig. 2: *The ASTNode2 interface (red).*

- Interface
- Strictness
  - Default is set to true
  - Set to false only under extraordinary circumstances
  - ***NOT recommended***
- `IsAllowableType()`
  - `IllegalArgumentException` thrown for wrong types

ASTNumber

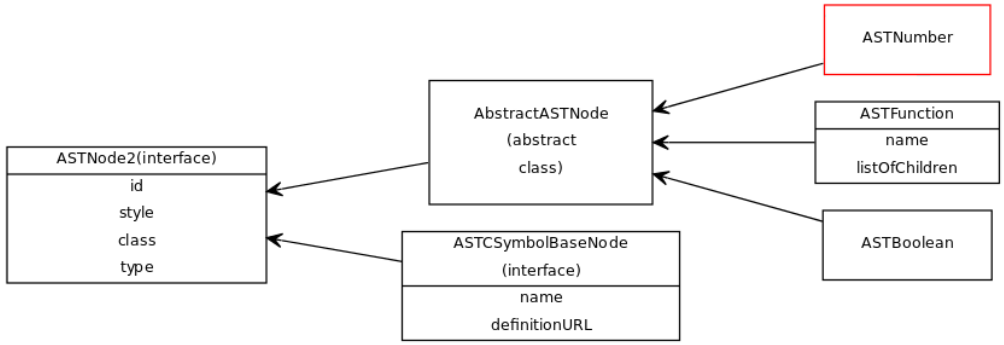


Fig. 3: *The ASTNumber class (red).*

- `getChildCount()`
  - `0`
- `getAllowsChildren()`
  - `false`
- `getChildAt()`
  - `IndexOutOfBoundsException`

ASTFunction

- Implementation
  - Each function regardless of identity stores its children in an `ArrayList`
  - No significant difference in implementation between unary, binary and *n*-ary
- Consequences
  - Users can violate SBML rules
- Alternatives
  1. Restrict child count in unary and binary
  2. **`Strictness`**

ASTBoolean

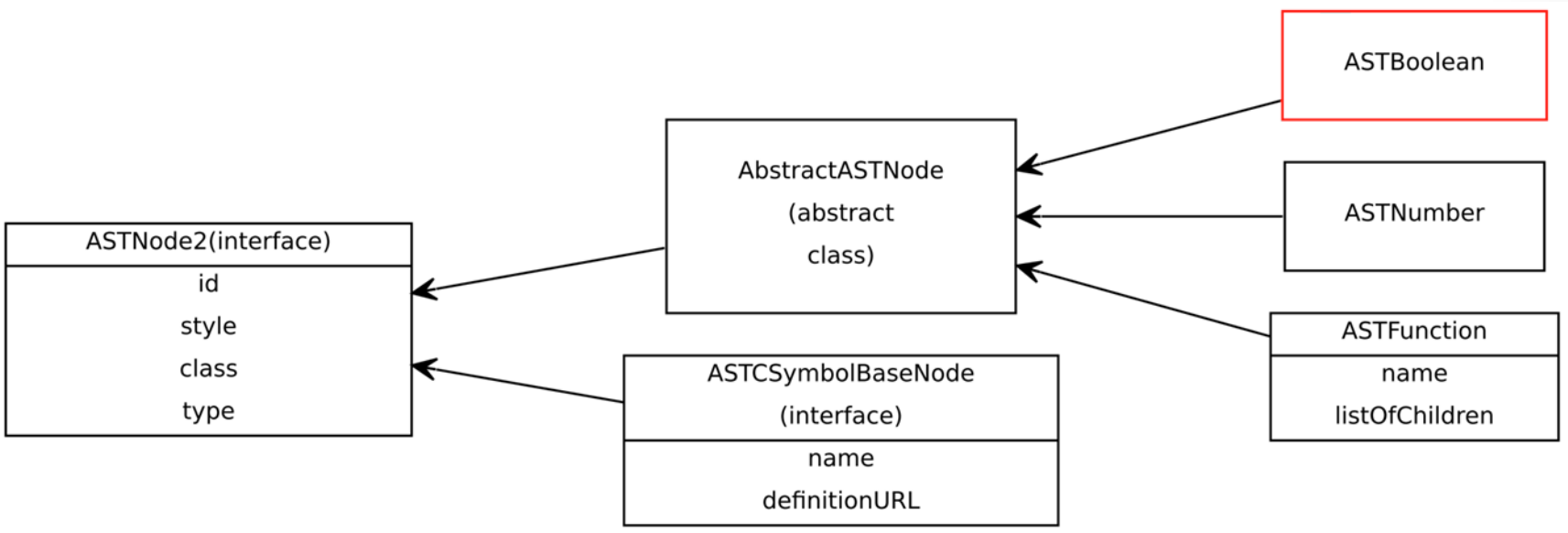


Fig. 3: *Snapshot of type hierarchy. ASTBoolean extends AbstractASTNode.*

ASTCSymbolBaseNode

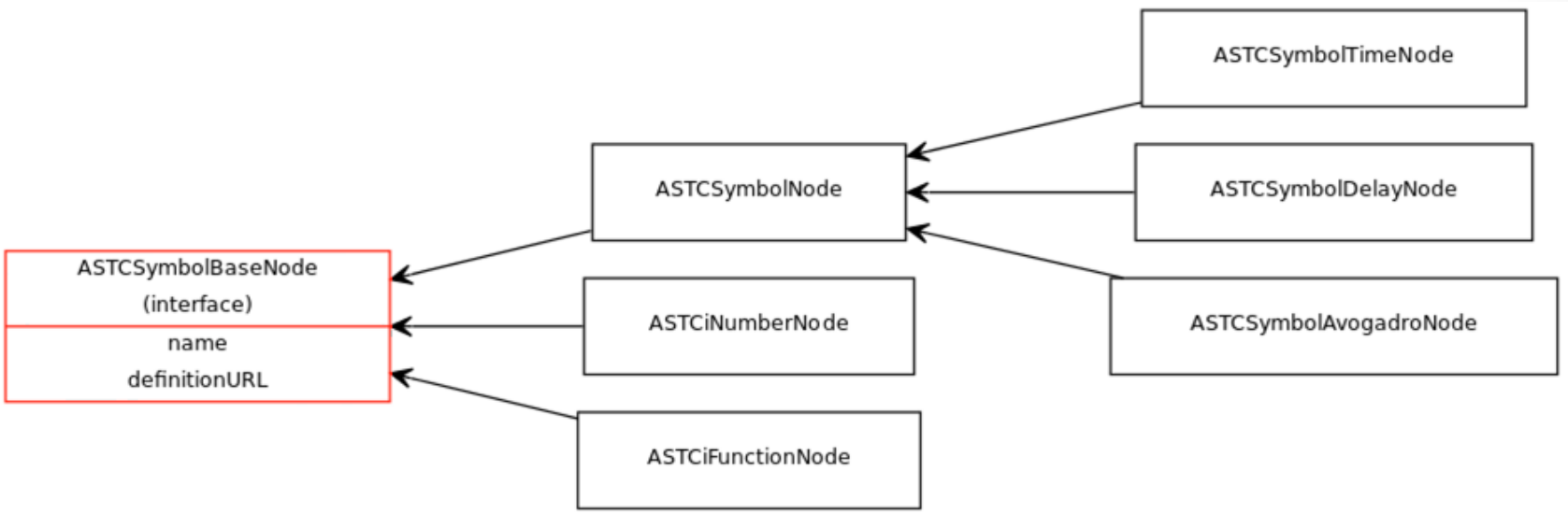
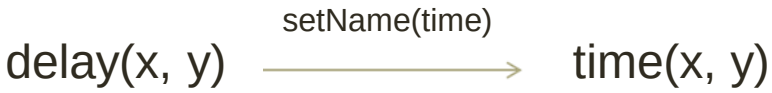


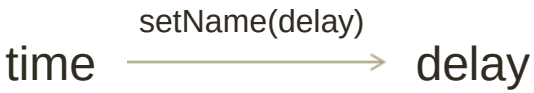
Fig. 4: *Snapshot of type hierarchy. All nodes extending ASTCSymbolBaseNode possess a name attribute that is mutable.*

ASTCSymbolBaseNode

- Name attribute is mutable
- Modifying name may potentially cause confusion (see below)
- Default names are sufficiently descriptive. Modify at your own risk.

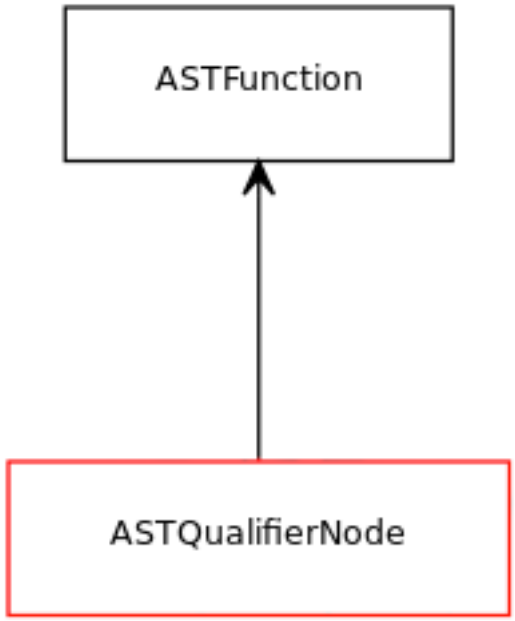
Example …





ASTQualifierNodes

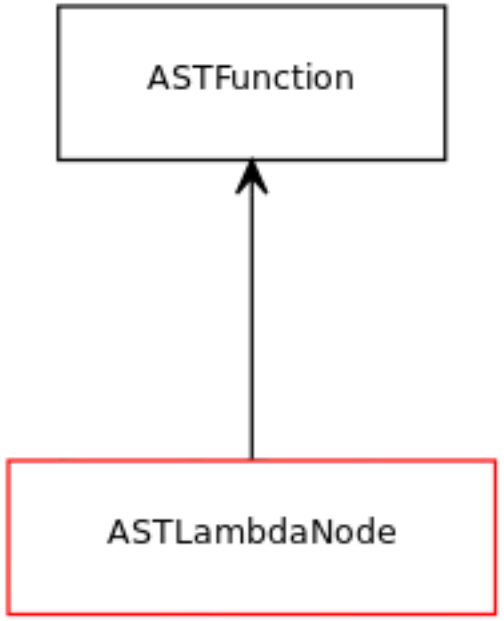
- Represent MathML qualifier element expressions
- bvar
  - `<bvar> </bvar>`
- degree
  - `<degree> </degree>`
- piece / otherwise
  - `<piece> </piece>`
  - `<otherwise> </otherwise>`
- logbase
  - `<logbase> </logbase>`



ASTLambdaFunction

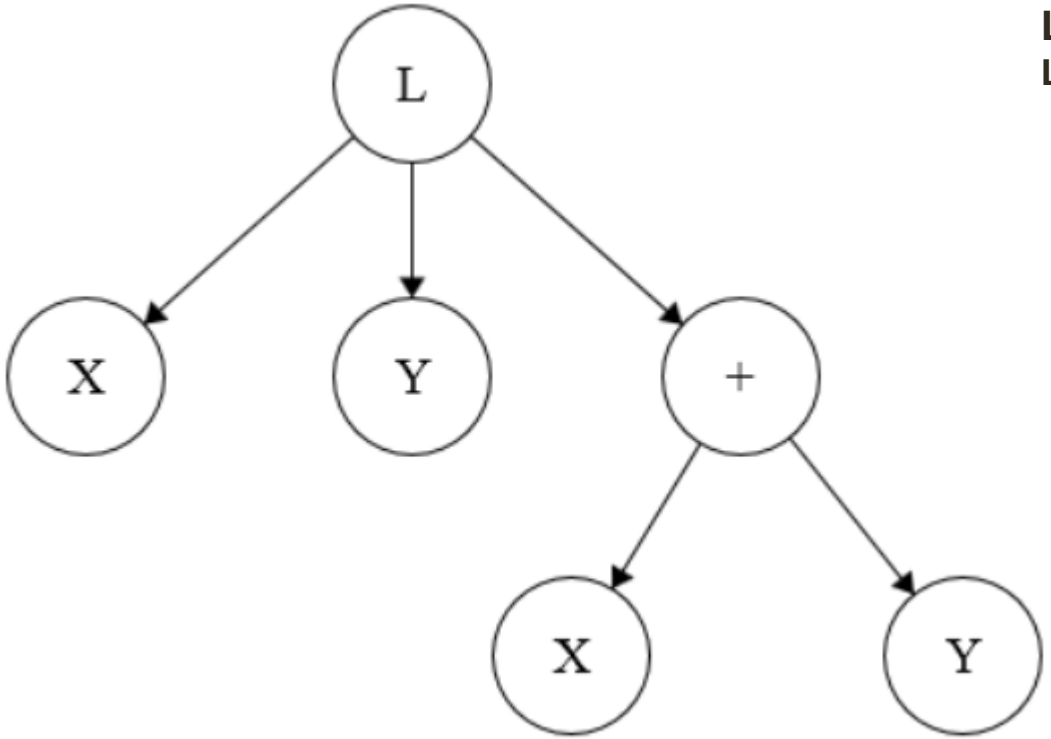
```
<lambda>
    <bvar>
         <ci> x </ci>
    </bvar>
    <bvar>
         <ci> y </ci>
    </bvar>
    <apply>
          <plus/>
          <ci> x </ci>
           <ci> y </ci>
    </apply>
 </lambda>
```

$$\lambda(x, y) = x + y$$



ASTLambdaFunction

Before …



Legend
L - `ASTLambdaFunctionNode`

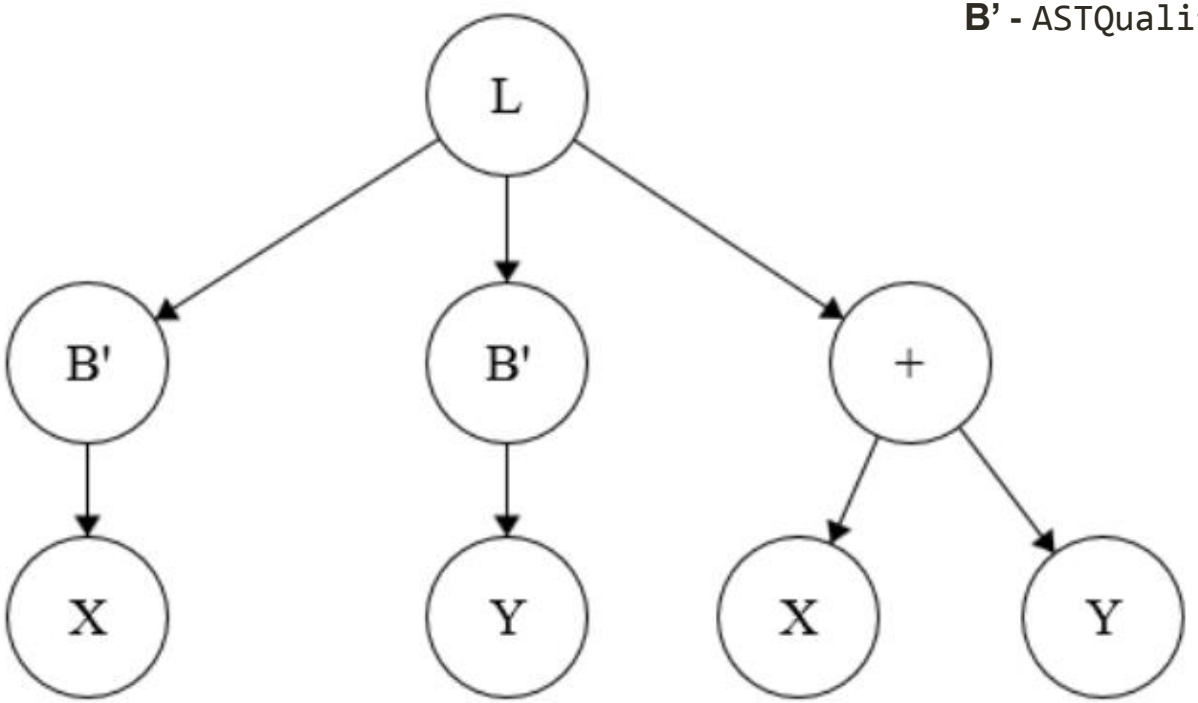
$$\lambda\,(x, y) = x + y$$

Fig. 5: *The old implementation. Note how the variables x and y are the children of L.*

ASTLambdaFunction

After …

Legend
L - `ASTLambdaFunctionNode`
B' - `ASTQualifierNode` (~ bvar)



$$\lambda\,(x, y) = x + y$$

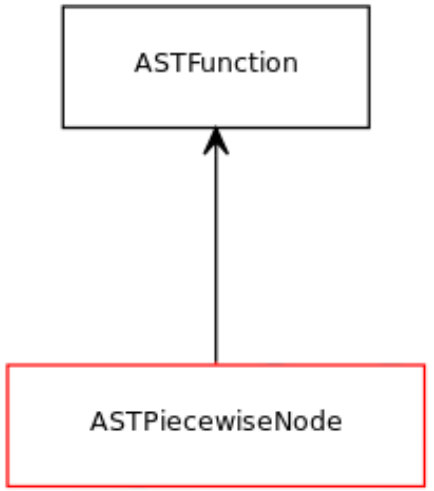
Fig. 6: *The new implementation. The bvar nodes reduce ambiguity and aid in the correct identification of x and y..*

ASTPiecewiseFunction

```
<piecewise>
      <piece>
            <cn> 1.55 </cn>
            <apply>
                  <leq/>
                  <ci> C </ci>
                  <cn> 3100 </cn>
             </apply>
      </piece>
      <otherwise>
            <cn> 2.5 </cn>
       <otherwise>
</piecewise
```

$$f(C) = \begin{cases} 1.5 & : \text{if } C \leq 3100 \\ 2.5 & : \text{otherwise} \end{cases}$$

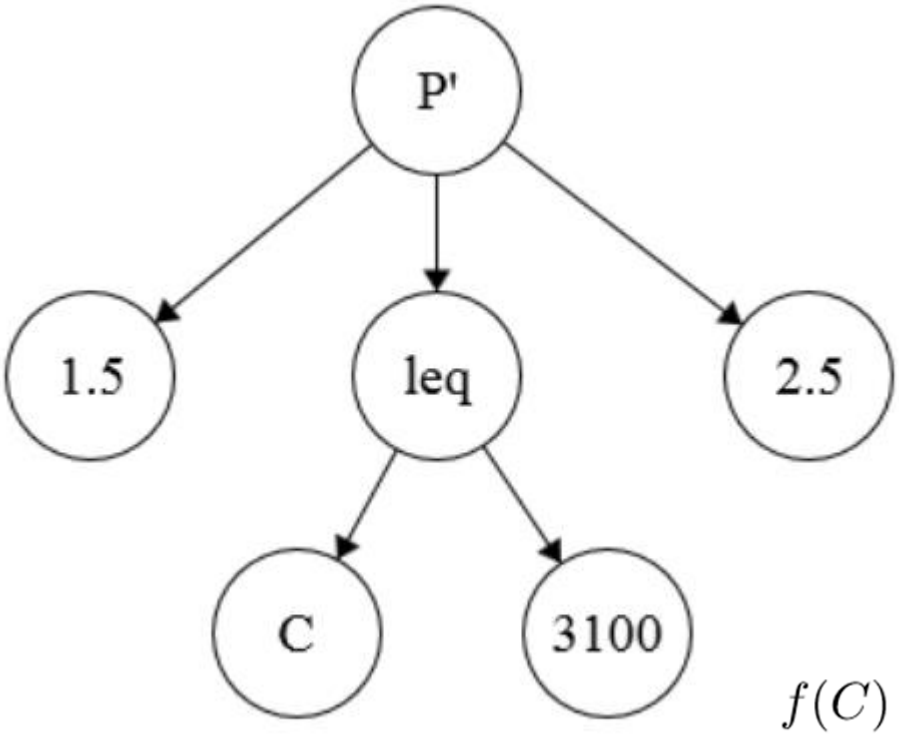
$$f : \mathbb{N} \to \mathbb{R} \quad C \in \mathbb{N}$$



ASTPiecewiseFunction

Before …

Legend
P' - `ASTPiecewiseFunctionNode`

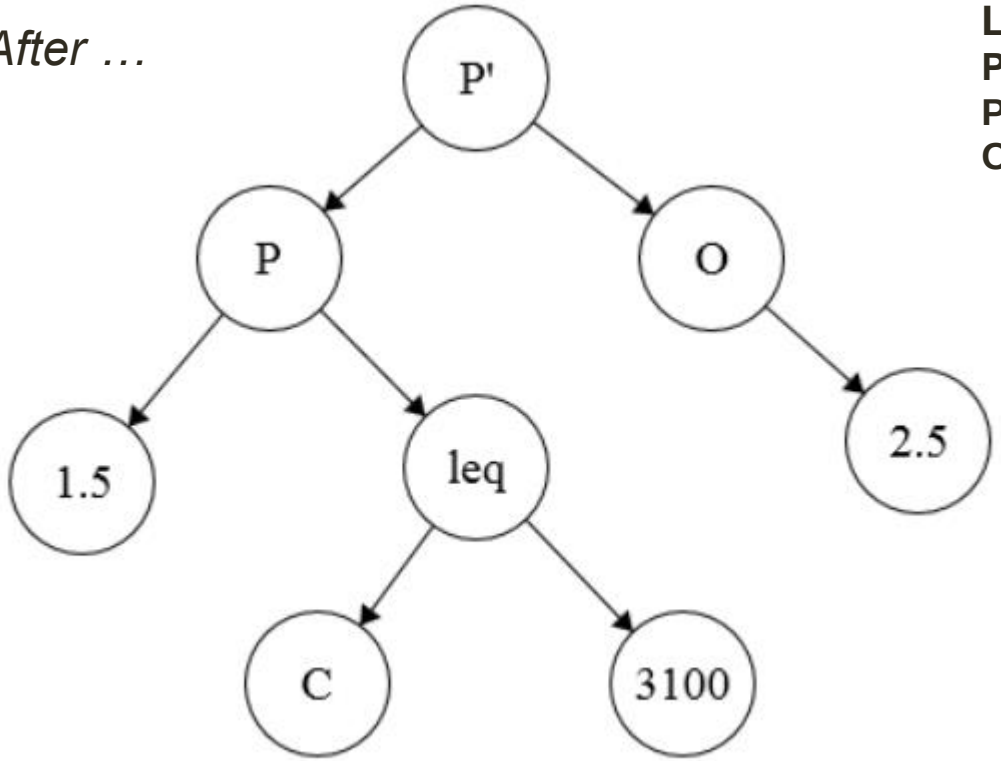


$$f(C) = \begin{cases} 1.5 & : \text{if } C \leq 3100 \\ 2.5 & : \text{otherwise} \end{cases}$$

Fig. 7: *The old implementation. The condition (leq) and value (1.5) are both directly linked to the piecewise node.*

ASTPiecewiseFunction

After …



Legend
P' - `ASTPiecewiseFunctionNode`
P - `ASTQualifierNode` (~ piece)
O - `ASTQualifierNode` (~ otherwise)

$$f(C) = \begin{cases} 1.5 & : \text{if } C \leq 3100 \\ 2.5 & : \text{otherwise} \end{cases}$$

Fig. 8: *The new implementation. The piece (P) node pairs up each value / condition pair.*

ASTFactory

- A collection of static methods for:
  - Creating new abstract syntax trees
  - Altering the topology of existing trees
- Creation of a new class necessary as no other class in the math package can accommodate static methods

ASTFactory

- Changes to a variety of methods:
  - `sum(…)`
  - `product(…)`
  - `diff(…)`
- New additions
  - `and(…)`
  - `or(…)`
  - `xor(…)`
- Resulting trees will not be automatically reduced to binary form. This will be done at user's discretion. More flexible.

Compilers

- MathML / LaTeX / C-like Infix
  - Generics
  - `ASTNode2Value<T>`
  - T - String, Double, Integer, Element

- Testing
  - MathML files in org.jsbml.math.compilers.resources
  - Updating / modifying the tests will require you to modify test files (next slide)

Compilers (MathML)

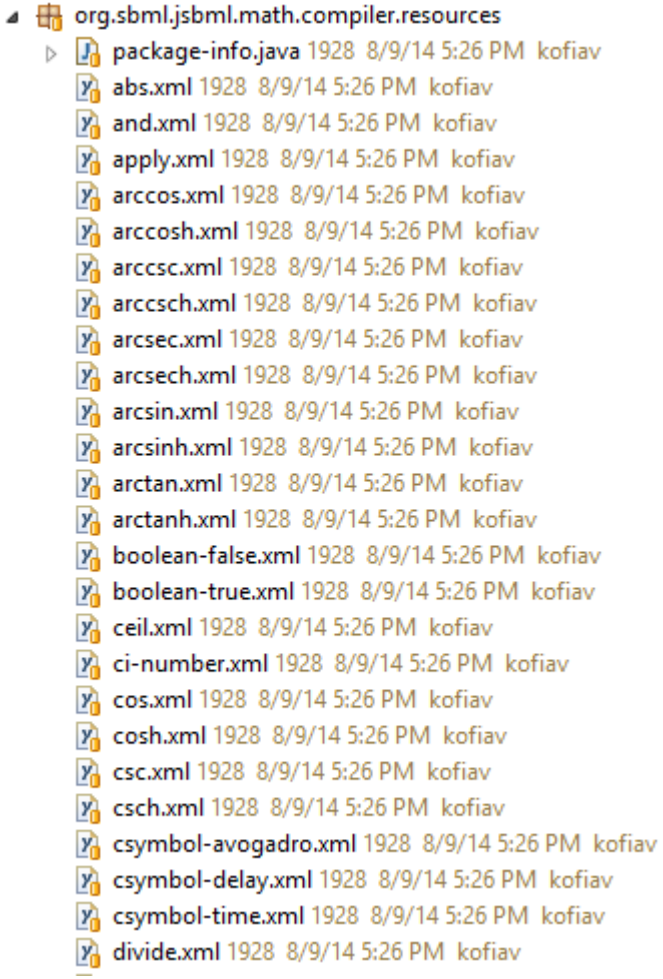


Fig. 9: Contents of abs.xml

```
<?xml version='1.0' encoding='UTF-8'?>
<math xmlns="http://www.w3.org/1998/Math/MathML">
  <apply>
    <abs/>
    <cn type="integer"> -5 </cn>
  </apply>
</math>
```

- syntax / semantics

Acknowledgements

- Google™
- OBF
- JSBML
- libSBML