

Extending KeY to handle C programs

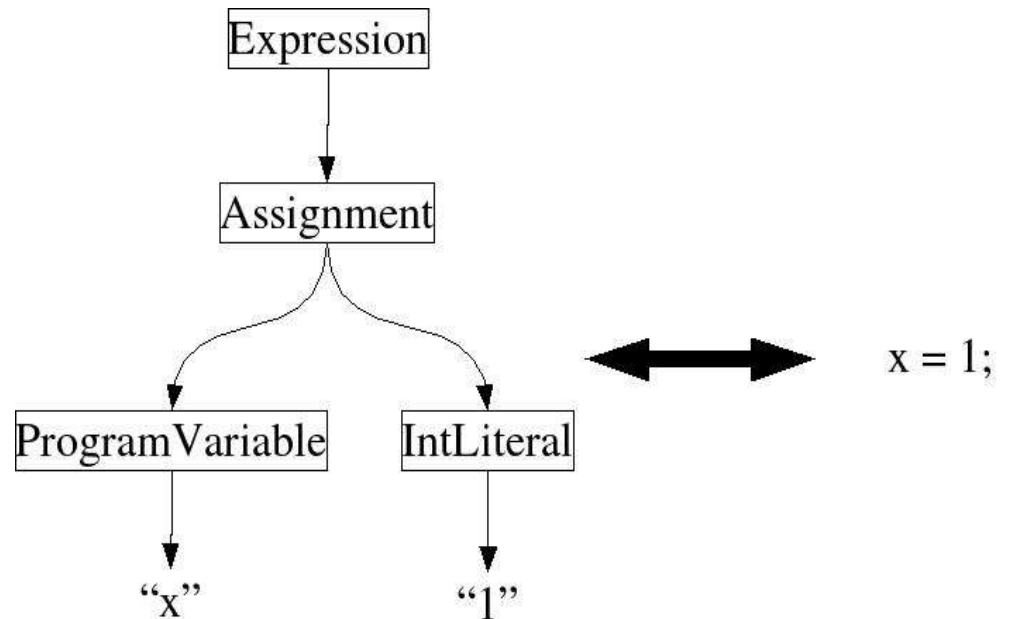
Sebastian Kekkonen
Göteborgs Universitet

What the work is about

- Find out which parts of KeY that are language dependent
- Changing the KeY-infrastructure for adding a new language
 - Data structures
 - Front-end
 - Language data types
 - Pretty printing

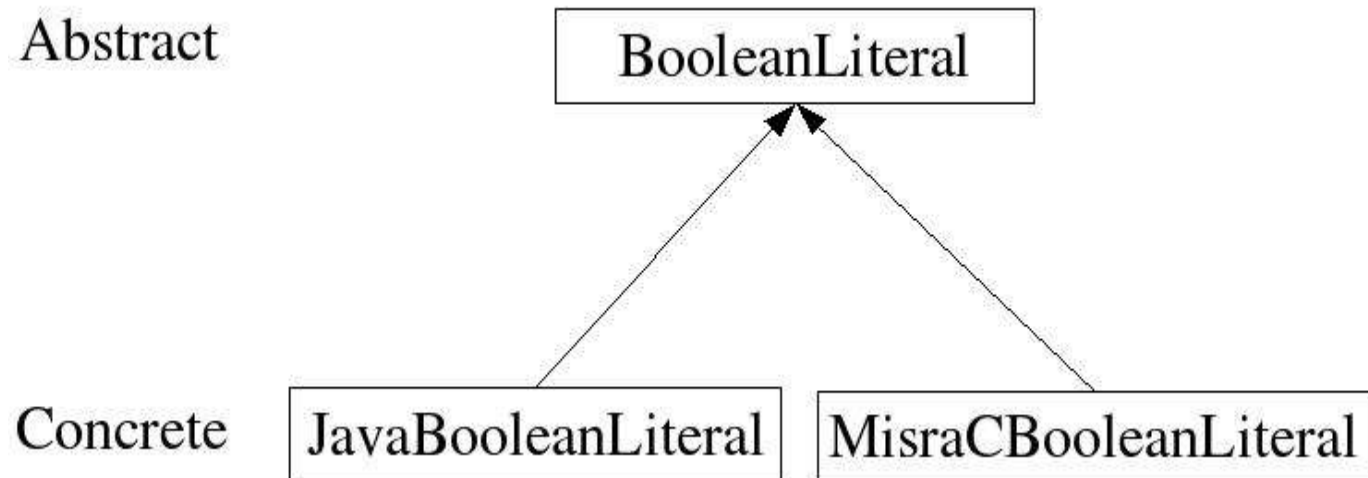
Data structures - description

- Program code is represented as an Abstract Syntax Tree (AST)
 - Indentation and other unimportant stuff is left out



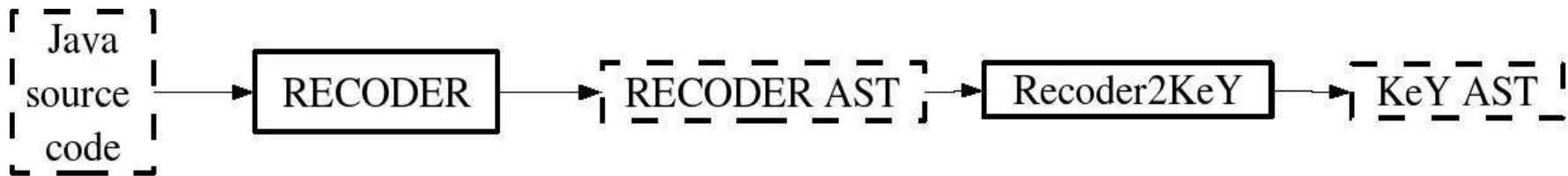
Data structures - changes

- New classes for each new language, just inherit the common base class.
 - `key.java.expression.literal.BooleanLiteral`
 - `key.java.expression.literal.java.JavaBooleanLiteral`
 - `key.java.expression.literal.misrac.MisraCBooleanLiteral`
- Visitor classes



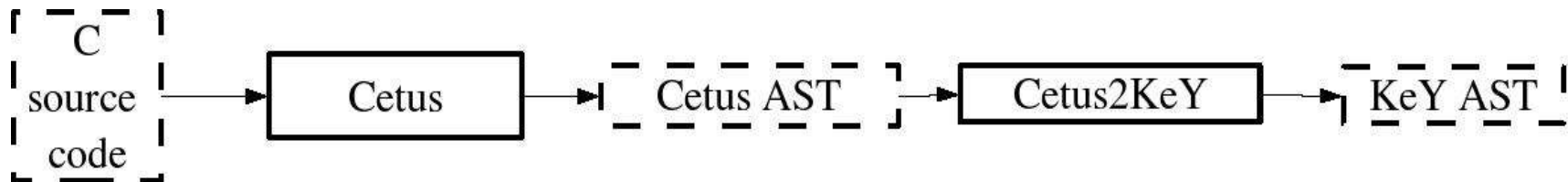
Front-end - description

- Takes the input and makes it usable by KeY
- Parser
 - RECODER
- Converter
 - Recoder2Key



Front-end - changes

- Parser
 - Cetus
- Converter
 - Cetus2KeY



Language data type - description

- Two sets of language data types (LDT's) in KeY
 - Logical terms
 - Java expression
- Conversion can happen back and forth

```
LessThanComparison {  
  find {.. #boolv = #s1 < #s2; ...}  
  replacewith ( lt(#s1,#s2) -> {.. #boolv = true; ...} &  
               !lt(#s1,#s2) -> {.. #boolv = false; ...}))}
```

Language data type - changes

- The original LDT class was made abstract and two new sub classes were created
- Important methods
 - Term translateLiteral(Literal lit)
 - Expression translateTerm(Term t, ExtList children)

Pretty printing - changes

- The class PrettyPrinter was made abstract and two new sub classes were created
 - JavaPrettyPrinter and MisraCPrettyPrinter only contains the constructors
- Some changes to the types

Summary

- Concluding example

```
\<{  
  bool b, b1, b2;  
  b1 = 0 < 1;  
  b2 = true && 4 >= 5;  
  b = b1 || b2;  
}\> b=TRUE
```