Lightweight Type-Like Hoare-Separation Specs for Java

Tiago Vieira Correia dos Santos

Departamento de Informática
Faculdade de Ciências e Tecnologia
Universidade Nova de Lisboa

September 10, 2010

INFORUM
Simpósio de Informática

Goals

Type systems are effective but not very precise, while program logics tend to be very precise, but undecidable.
This work goals are:

- Develop a lightweight specification language for Java, based on propositional logic;
- Extend the Java compiler, complementing type analysis with program correctness verification according to its specification.

INFORUM
Simpósio de Informática

Outline

INFORUM
Simpósio de Informática

Formal Specification and Verification

The process of software validation is a difficult problem, benefiting from the use of rigorous methods, preferably based on formalized mathematical techniques (e.g. logic):

- Improve specifications quality, removing ambiguities;
- Detect errors in advance;
- Correctness, equivalence, termination.

It is a research area for over 40 years and has seen a new impetus recently.

Hoare marked the 40th anniversary of his article writing a new article [Hoare 2009], which discusses the developments that have occurred since that time, and what might still happen.

INFORUM
Simpósio de Informática

Formal Specification and Verification

The use of formal methods is not fully accepted by software developers:

- Complexity
- Cost

It is necessary to address formal methods using lightweight specifications.

Verification techniques grouped in:

- Static Verification
  - Theorem Proving
- Dynamic Verification
  - Code Coverage
  - Model Checking

This work focus is specification verification by static analysis, using theorem proving technique.

INFORUM
Simpósio de Informática

Verification Techniques

- Hoare Logic
  - Flo67, Hoa69, Lam80
- Separation Logic
  - Rey02, OHRY01, PB05
- Weakest Precondition Calculus
  - Dij75, Rey02
- Dynamic Logic
  - Bec01, BK07
- Higher-order Logic
  - HJ00, JP01

INFORUM
Simpósio de Informática

Verification Techniques

- Hoare Logic
  - Flo67, Hoa69, Lam80
- Separation Logic
  - Rey02, OHRY01, PB05
- Weakest Precondition Calculus
  - Dij75, Rey02
- Dynamic Logic
  - Bec01, BK07
- Higher-order Logic
  - HJ00, JP01

Our approach focus on Weakest Precondition Calculus and is inspired by Dual Intuitionistic Linear Logic and Separation Logic

=

Dual Hoare-Separation Logic

INFORUM Simpósio de Informática

Languages with verification support

There are lots of tools that provide support for program verification:

- ESC/Java2
- JACK
- KRAKATOA
- LOOP
- jStar
- KeY
- Forge

Languages have also been designed with support for program verification:

- Gypsy
- Euclid
- Eiffel
- Spec#
- SPARK
- D

INFORUM Simpósio de Informática

Languages with verification support

- Require user intervention (e.g KRAKATOA);
- Don't satisfy correctness and/or completeness (e.g ESC/Java2, Forge);
- Coverage language limitations;
- Higher specification language expressiveness, but too complex.

Programmers:

- Have little knowledge in logic areas;
- Intend automatic processes, less complex and integrated into program development.

INFORUM
Simpósio de Informática

Lightweight Type-Like Hoare-Separation Specs for Java

In our approach:

- Specification language similar to JML and Spec#'s but lightweight, based on monadic propositional logic and handles aliasing by separating pure from linear properties
  - No quantifiers
  - No reference to the value of an expression at its precondition (*old*)
- Verification based on weakest precondition calculus, executed directly from Java syntactic constructs
  - Verification Conditions proved using a SMT-Solver

INFORUM
Simpósio de Informática

Expected Results

- Cover as much as possible the features of Java
- Integrate verification in the compile process
  - Approach the checks carried out by type systems to the fully formal logical checks.
- Perform verification automatically without user interaction

Dual Hoare-Separation Logic

$$
\begin{array}{lcll}
\psi & ::= & \phi + \varphi & \text{(Dual Formula)} \\
\varphi & ::= & \emptyset \mid \phi \mid \phi * \varphi & \text{(Linear Formula)} \\
\phi & ::= & & \text{(Classic Formula)} \\
 & & \bot & \text{(Bottom)} \\
 & \mid & \phi \; lc \; \phi & \text{(Binary Formula)} \\
 & \mid & \neg\phi & \text{(Negation)} \\
 & \mid & (\phi) & \text{(Parenthesized Formula)} \\
 & \mid & P(t_1, t_2, \cdots, t_n) & \text{(Predicate Symbols)} \\
lc & ::= & \vee \mid \wedge \mid \Rightarrow \mid \Leftrightarrow & \text{(Logical Connectives)} \\
t & ::= & & \text{(Terms)} \\
 & & c & \text{(Constants)} \\
 & \mid & x & \text{(Variables)} \\
 & \mid & f(t_1, t_2, \cdots, t_n) & \text{(Function Symbols)}
\end{array}
$$

INFORUM Simpósio de Informática

Dual Hoare-Separation Logic

ψ	$::=$	$\phi + \varphi$	(Dual Formula)
φ	$::=$	$\emptyset \mid \phi \mid \phi * \varphi$	(Linear Formula)
ϕ	$::=$		(Classic Formula)
		$\bot$	(Bottom)
	$\mid$	$\phi \; lc \; \phi$	(Binary Formula)
	$\mid$	$\neg\phi$	(Negation)
	$\mid$	(ϕ)	(Parenthesized Formula)
	$\mid$	$P(t_1, t_2, \cdots, t_n)$	(Predicate Symbols)
lc	$::=$	$\vee \mid \wedge \mid \Rightarrow \mid \Leftrightarrow$	(Logical Connectives)
t	$::=$		(Terms)
		c	(Constants)
	$\mid$	x	(Variables)
	$\mid$	$f(t_1, t_2, \cdots, t_n)$	(Function Symbols)

INFORUM Simpósio de Informática

Dual Hoare-Separation Logic

ψ	$::=$	$\phi + \varphi$	(Dual Formula)
φ	$::=$	$\emptyset \mid \phi \mid \phi * \varphi$	(Linear Formula)
ϕ	$::=$		(Classic Formula)
		$\bot$	(Bottom)
	$\mid$	$\phi \; lc \; \phi$	(Binary Formula)
	$\mid$	$\neg\phi$	(Negation)
	$\mid$	(ϕ)	(Parenthesized Formula)
	$\mid$	$P(t_1, t_2, \cdots, t_n)$	(Predicate Symbols)
lc	$::=$	$\vee \mid \wedge \mid \Rightarrow \mid \Leftrightarrow$	(Logical Connectives)
t	$::=$		(Terms)
		c	(Constants)
	$\mid$	x	(Variables)
	$\mid$	$f(t_1, t_2, \cdots, t_n)$	(Function Symbols)

INFORUM Simpósio de Informática

Assertions

D, I	::=	$CF + SLF$	(Dual Formula)
SLF	::=	$CF \mid CF * SLF$	(Sep. Formula)
CF	::=	**true** | **false** | $CF\ bop\ CF$ | **!** CF | $b : S$	(Classic Formula)
bop	::=	&& | || | => | <=>	(Logical Connectives)
b	::=	fn | **this** | **return** | pn	(Properties/States – Target)
S	::=	**true** | **false** | **pos** | **neg** | **zero** | **null** | sn	(Properties/States)

$pn \in$ parameter names
$fn \in$ field names
$sn \in$ state names

INFORUM
Simpósio de Informática

Programs

program	::=	*classDecl**	(Program)
classDecl	::=	**class** *cn* { *classMember** }	(Class Declaration)
classMember	::=	...	(Class Member)
	|	*field*	(Instance Variable Declaration)
	|	*method*	(Method Declaration)
	|	*constructor*	(Constructor Declaration)
	|	*classSpec*	(Class Specification)
classSpec	::=		(Class Specification)
		define *sn*;	(Abstract Definition)
	|	**define** *sn* = *D*;	(Concrete Definition)
	|	**invariant** *D*;	(Class Invariant)
field	::=	*T fn* [= *E*]? ;	(Instance Variable Declaration)
method	::=	*modifier T mn*($\overline{arg}$) $\overline{spec}$ { *ST* }	(Method Declaration)
constructor	::=	*modifier cn*($\overline{arg}$) $\overline{spec}$ { *ST* }	(Constructor Declaration)
modifier	::=	**public** | **static** | ... | **pure**	(Modifiers)
spec	::=		(Procedure Specification)
		requires *D*	(Precondition)
	|	**ensures** *D*	(Postcondition)
ST	::=	...	(Statement)
	|	**assume** *D*	(Assume)
	|	**sassert** *D*	(Static Assert)

$cn \in$ class names
$sn \in$ state names
$fn \in$ field names

INFORUM Simpósio de Informática

Programs

program	::=	*classDecl**	(Program)
classDecl	::=	**class** *cn* { *classMember** }	(Class Declaration)
classMember	::=	...	(Class Member)
	|	*field*	(Instance Variable Declaration)
	|	*method*	(Method Declaration)
	|	*constructor*	(Constructor Declaration)
	|	*classSpec*	(Class Specification)
classSpec	::=		(Class Specification)
		define *sn*;	(Abstract Definition)
	|	**define** *sn* = *D*;	(Concrete Definition)
	|	**invariant** *D*;	(Class Invariant)
field	::=	*T fn* [= *E*]? ;	(Instance Variable Declaration)
method	::=	*modifier T mn*($\overline{arg}$) $\overline{spec}$ { *ST* }	(Method Declaration)
constructor	::=	*modifier cn*($\overline{arg}$) $\overline{spec}$ { *ST* }	(Constructor Declaration)
modifier	::=	**public** | **static** | ... | **pure**	(Modifiers)
spec	::=		(Procedure Specification)
		requires *D*	(Precondition)
	|	**ensures** *D*	(Postcondition)
ST	::=	...	(Statement)
	|	**assume** *D*	(Assume)
	|	**sassert** *D*	(Static Assert)

$cn \in$ class names
$sn \in$ state names
$fn \in$ field names

INFORUM Simpósio de Informática

Programs

program	::=	*classDecl**	(Program)
classDecl	::=	**class** *cn* { *classMember** }	(Class Declaration)
classMember	::=	...	(Class Member)
	|	*field*	(Instance Variable Declaration)
	|	*method*	(Method Declaration)
	|	*constructor*	(Constructor Declaration)
	|	*classSpec*	(Class Specification)
classSpec	::=		(Class Specification)
		define *sn*;	(Abstract Definition)
	|	**define** *sn* = *D*;	(Concrete Definition)
	|	**invariant** *D*;	(Class Invariant)
field	::=	*T fn* [= *E*]? ;	(Instance Variable Declaration)
method	::=	*modifier T mn*($\overline{arg}$) $\overline{spec}$ { *ST* }	(Method Declaration)
constructor	::=	*modifier cn*($\overline{arg}$) $\overline{spec}$ { *ST* }	(Constructor Declaration)
modifier	::=	**public** | **static** | ... | **pure**	(Modifiers)
spec	::=		(Procedure Specification)
		requires *D*	(Precondition)
	|	**ensures** *D*	(Postcondition)
ST	::=	...	(Statement)
	|	**assume** *D*	(Assume)
	|	**sassert** *D*	(Static Assert)

cn $\in$ class names
sn $\in$ state names
fn $\in$ field names

INFORUM Simpósio de Informática

Programs

program	::=	*classDecl**	(Program)
classDecl	::=	**class** *cn* { *classMember** }	(Class Declaration)
classMember	::=	...	(Class Member)
	|	*field*	(Instance Variable Declaration)
	|	*method*	(Method Declaration)
	|	*constructor*	(Constructor Declaration)
	|	*classSpec*	(Class Specification)
classSpec	::=		(Class Specification)
		define *sn*;	(Abstract Definition)
	|	**define** *sn* = *D*;	(Concrete Definition)
	|	**invariant** *D*;	(Class Invariant)
field	::=	*T fn* [= *E*]? ;	(Instance Variable Declaration)
method	::=	*modifier T mn*($\overline{arg}$) $\overline{spec}$ { *ST* }	(Method Declaration)
constructor	::=	*modifier cn*($\overline{arg}$) $\overline{spec}$ { *ST* }	(Constructor Declaration)
modifier	::=	**public** | **static** | ... | **pure**	(Modifiers)
spec	::=		(Procedure Specification)
		requires *D*	(Precondition)
	|	**ensures** *D*	(Postcondition)
ST	::=	...	(Statement)
	|	**assume** *D*	(Assume)
	|	**sassert** *D*	(Static Assert)

cn $\in$ class names
sn $\in$ state names
fn $\in$ field names

INFORUM
Simpósio de Informática

Programs

program	::=	*classDecl**	(Program)
classDecl	::=	**class** *cn* { *classMember** }	(Class Declaration)
classMember	::=	...	(Class Member)
	|	*field*	(Instance Variable Declaration)
	|	*method*	(Method Declaration)
	|	*constructor*	(Constructor Declaration)
	|	*classSpec*	(Class Specification)
classSpec	::=		(Class Specification)
		define *sn*;	(Abstract Definition)
	|	**define** *sn* = *D*;	(Concrete Definition)
	|	**invariant** *D*;	(Class Invariant)
field	::=	*T fn* [= *E*]? ;	(Instance Variable Declaration)
method	::=	*modifier T mn*($\overline{arg}$) $\overline{spec}$ { *ST* }	(Method Declaration)
constructor	::=	*modifier cn*($\overline{arg}$) $\overline{spec}$ { *ST* }	(Constructor Declaration)
modifier	::=	**public** | **static** | ... | **pure**	(Modifiers)
spec	::=		(Procedure Specification)
		requires *D*	(Precondition)
	|	**ensures** *D*	(Postcondition)
ST	::=	...	(Statement)
	|	**assume** *D*	(Assume)
	|	**sassert** *D*	(Static Assert)

cn $\in$ class names
sn $\in$ state names
fn $\in$ field names

INFORUM Simpósio de Informática

Example (1)

```
1   public class File {
2       define open;
3
4       public File()
5           ensures + !open
6       {
7           assume + !open;
8       }
9
10      public void open()
11          requires + !open
12          ensures + open
13      {
14          assume + open;
15      }
16
17      public void write(int b)
18          requires + open
19          ensures + open
20      { }
21
22      public int read()
23          requires + open
24          ensures + open
25      { return 0; }
```

```
25      ...
26
27      public void close()
28          requires + open
29          ensures + !open
30      {
31          assume + !open;
32      }
33
34      public pure void nothing() {}
35
36      public static void main(
37              String[] args) {
38          File f = new File();
39          f.nothing();
40          f.open();
41          f.nothing();
42          f.nothing();
43          f.close();
44          f.nothing();
45          f.open();
46          //f.open(); // Wrong
47          f.close();
48          //f.close(); // Wrong
49      }
50  }
```

INFORUM Simpósio de Informática

Example (2)

```
1   public class Stack {
2
3       private Entry head;
4
5       class Entry {
6           public Entry(int x)
7               requires x:pos
8           { this(x, null); }
9
10          public Entry(int x, Entry n)
11              requires x:pos
12          {
13              this.x = x;
14              this.next = n;
15          }
16
17          private int x;
18          invariant x:pos;
19          private Entry next;
20
21          public Entry getNext()
22          { return next; }
23
24          public int getElement()
25              ensures return:pos
26          { return x; }
27      }
```

```
27      ...
28
29      public Stack()
30      { head = null; }
31
32      public void push(int i)
33          requires i:pos
34          ensures +!head:null
35      { head = new Entry(i, head); }
36
37      public int pop()
38          requires +!head:null
39          ensures return:pos
40      {
41          int res = head.getElement();
42          head = head.getNext();
43          return res;
44      }
45  }
```

INFORUM Simpósio de Informática

Verification Approach

Approach to program verification:

- Apply weakest precondition calculus to every method and constructor body;
- Rewrite properties and Java conditions in propositional logic predicates;
- Submit formulas to a SMT-Solver for validity proof.

WP Calculus

Weakest Precondition

Being P a program with postcondition R, then the weakest precondition is represented as:

$$wp(P, R)$$

$$[\texttt{while loop}]$$

$$\frac{(I \wedge \neg\varepsilon) \Rightarrow R \quad (I \wedge \varepsilon) \Rightarrow wp\,(ST, I)}{wp\left(\begin{array}{l}\textbf{while } (\varepsilon) \\ \quad \textbf{invariant } I \\ ST\end{array}, R\right) = I}$$

WP Calculus

$$wp\,(x = k.mn(\overline{y}, \overline{z}), C + S) = I_c[this/k] \quad \Rightarrow \quad [\texttt{pure call - pm/pr/po}]$$

$$\left(\begin{array}{c} k \neq \texttt{null} \wedge Q_{mn_A}[this/k, \overline{p_1}/\overline{y}] \\ \wedge \\ \left(\left(\begin{array}{c} k \neq \texttt{null} \\ \wedge \\ R_{mn_A}[this/k, \overline{p_1}/\overline{y}, return/f] \end{array} \right) \Rightarrow C[x/f] \right) \end{array} \right)$$

$$+$$

$$\left(\left(\begin{array}{c} Q_{mn_B}[this/k, \overline{p_1}/\overline{y}, \overline{p_2}/\overline{z}] \\ \wedge \\ \left(R_{mn_B}[this/k, \overline{p_1}/\overline{y}, \overline{p_2}/\overline{z}, return/f] \Rightarrow S \downarrow \{k, x, \overline{z}\}[x/f] \right) \end{array} \right) \right)$$

$$*$$

$$S - \{k, x, \overline{z}\}$$

INFORUM Simpósio de Informática

WP Calculus

$$[\texttt{method call - lr/lo}]$$

$$\frac{S \downarrow \{\overline{z} \backslash x\} = \emptyset \qquad \texttt{true} + (k \neq \texttt{null} \wedge R_{mn_B}[this/k, \overline{p_1}/\overline{y}, return/f]) \Rightarrow S \downarrow \{k, x, \overline{z}\}[x/f]}{wp\,(x = k.mn(\overline{y}, \overline{z}), C + S) = I_c[this/k] \Rightarrow \left(\begin{array}{c} \left(\begin{array}{c} Q_{mn_A}[this/k, \overline{p_1}/\overline{y}] \\ \wedge \\ (R_{mn_A}[this/k, \overline{p_1}/\overline{y}] \Rightarrow C[x/f]) \end{array} \right) \\ + \\ \left(\begin{array}{c} k \neq \texttt{null} \\ \wedge \\ Q_{mn_B}[this/k, \overline{p_1}/\overline{y}, \overline{p_2}/\overline{z}] \\ * \\ S - \{k, x, \overline{z}\} \end{array} \right) \end{array} \right)}$$

Example

```
1  class Math {
2      public static pure int abs(int x)
3          ensures !return:neg
4      {
5          if (x > 0) return x;
6          else return -x;
7      }
8  }
```

Example

$$wp\left(\begin{array}{l}\textbf{if } (x > 0) \textbf{ return } x;\\ \textbf{else return } -x\end{array}, !\textbf{return} : \textbf{neg}\right) =$$

Example

$$wp\left(\begin{array}{l}\textbf{if } (x > 0) \textbf{ return } x;\\ \textbf{else return } -x\end{array}, !\textbf{return} : \textbf{neg}\right) =$$

[conditional]

$$wp\left(\begin{array}{l}\textbf{if } (\varepsilon)\ ST_1\\ \textbf{else } ST_2\end{array}, C + S\right) = \varepsilon \Rightarrow wp\,(ST_1, C + S) \wedge \neg\varepsilon \Rightarrow wp\,(ST_2, C + S)$$

Example

$$wp\left(\begin{array}{l}\textbf{if } (x > 0) \textbf{ return } x;\\ \textbf{else return } -x\end{array}, !\textbf{return} : \textbf{neg}\right) =$$

[conditional]

$$wp\left(\begin{array}{l}\textbf{if } (\varepsilon)\ ST_1\\ \textbf{else } ST_2\end{array}, C + S\right) = \varepsilon \Rightarrow wp\,(ST_1, C + S) \wedge \neg\varepsilon \Rightarrow wp\,(ST_2, C + S)$$

$$= \begin{array}{c} >(x,0) \Rightarrow wp\Big(\textbf{return } x, \neg <(\textbf{return}, 0) + \emptyset\Big)\\ \wedge \\ \neg >(x,0) \Rightarrow wp\Big(\textbf{return } -x, \neg <(\textbf{return}, 0) + \emptyset\Big)\end{array} =$$

INFORUM Simpósio de Informática

Example

$$
\begin{array}{c}
> (x, 0) \Rightarrow wp\Big(\textbf{return } x, \neg < (\textbf{return}, 0) + \emptyset\Big) \\
= \qquad\qquad \wedge \qquad\qquad = \\
\neg > (x, 0) \Rightarrow wp\Big(\textbf{return } -x, \neg < (\textbf{return}, 0) + \emptyset\Big)
\end{array}
$$

Example

$$
\begin{array}{ccc}
 & > (x, 0) \Rightarrow wp\Big(\textbf{return } x, \neg < (\textbf{return}, 0) + \emptyset\Big) & \\
= & \wedge & = \\
 & \neg > (x, 0) \Rightarrow wp\Big(\textbf{return } -x, \neg < (\textbf{return}, 0) + \emptyset\Big) &
\end{array}
$$

[pure return]

$$wp\,(\textbf{return}\ \varepsilon, C + S) = R_{mn_{\boldsymbol{A}}}[\textbf{return}/\varepsilon] + R_{mn_{\boldsymbol{B}}}[\textbf{return}/\varepsilon]$$

INFORUM Simpósio de Informática

Example

$$
\begin{array}{c}
> (x, 0) \Rightarrow wp\Big(\textbf{return } x, \neg < (\textbf{return}, 0) + \emptyset\Big) \\
= \qquad\qquad \wedge \qquad\qquad = \\
\neg > (x, 0) \Rightarrow wp\Big(\textbf{return } -x, \neg < (\textbf{return}, 0) + \emptyset\Big)
\end{array}
$$

[pure return]

$$wp\,(\textbf{return}\ \varepsilon, C + S) = R_{mn_{\boldsymbol{A}}}[\textbf{return}/\varepsilon] + R_{mn_{\boldsymbol{B}}}[\textbf{return}/\varepsilon]$$

$$
\begin{array}{c}
> (x, 0) \Rightarrow \Big(\neg < (x, 0) + \emptyset\Big) \\
= \qquad\qquad \wedge \\
\neg > (x, 0) \Rightarrow \Big(\neg < (-x, 0) + \emptyset\Big)
\end{array}
\qquad \equiv \qquad
\begin{array}{c}
> (x, 0) \Rightarrow \neg < (x, 0) \\
\wedge \qquad\qquad + \emptyset \\
\neg > (x, 0) \Rightarrow \neg < (-x, 0)
\end{array}
$$

INFORUM
Simpósio de Informática

Example

Recall that the absolute value of a number is well specified iff:

$$\{\ \}\ \textbf{int}\ \text{abs}(\textbf{int}\ x)\ \{\ !\textbf{return}{:}\text{neg}\ \}$$

Verification Condition Submitted to the SMT-Solver:

$$(\top + \emptyset) \Rightarrow \begin{pmatrix} > (x,0) \Rightarrow \neg < (x,0) \\ \wedge \\ \neg > (x,0) \Rightarrow \neg < (-x,0) \end{pmatrix} + \emptyset \equiv$$

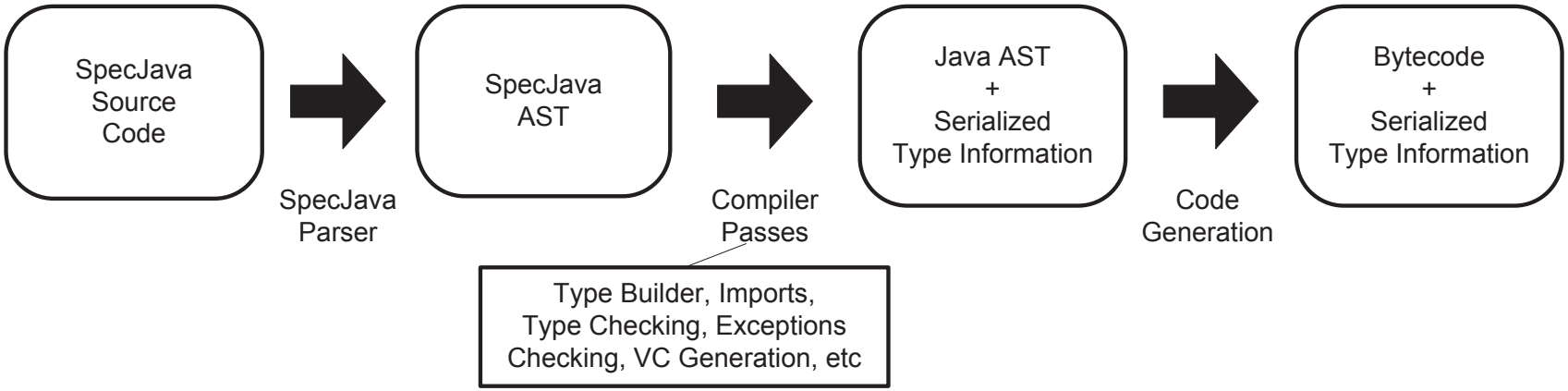
$$\equiv (\top + \emptyset) \Rightarrow (\top + \emptyset) =$$

$$(\top \Rightarrow \top) + \emptyset = \top + \emptyset$$

INFORUM Simpósio de Informática

Implementation

The extension of the Java language is implemented using the Polyglot framework, which implements an extensible compiler for Java. [Nathaniel Nystrom, Michael Clarkson, and Andrew Myers]



`http://www.cs.cornell.edu/Projects/polyglot/`

`http://pessoa.fct.unl.pt/tiago.santos/tools/specjava.zip`

INFORUM Simpósio de Informática

Challenges

Some challenges of this work:

- Extend WP calculus for the Java language
- Solve aliasing problem
- Completely define the specification language and its integration in Polyglot
- Integrate SMT-Solver in Polyglot

INFORUM
Simpósio de Informática

Future Work

- Dual Logic is new
  - Requires a theoretical study
- Extend WP Calculus
  - **break** statement
  - **continue** statement
  - Inheritance
- Extend Specification Support
  - Interface Specification
  - Core Java Specification
- Concurrency