

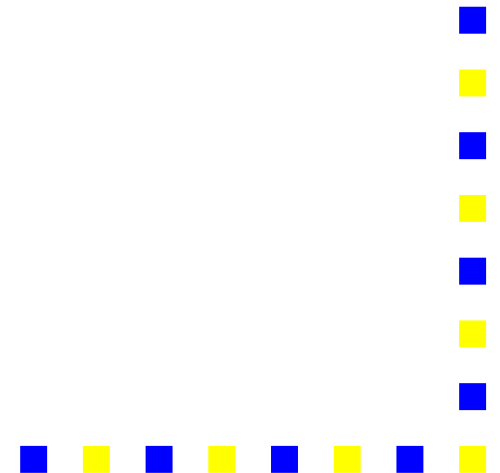
---

# *Specification language and WP calculus for Java Bytecode.*

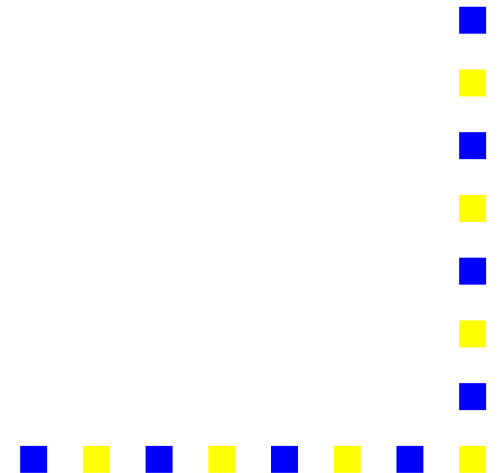
joint work in progress

Mariela Pavlova, Lilian Burdy

INRIA Sophia-Antipolis



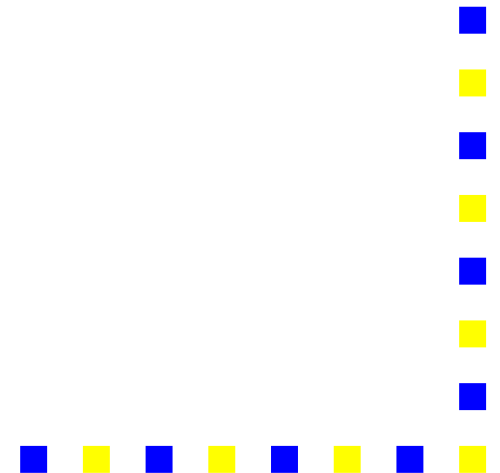
- Proof Carrying ByteCode
  - Proof obligations.
  - What is the language in which properties will be expressed ?



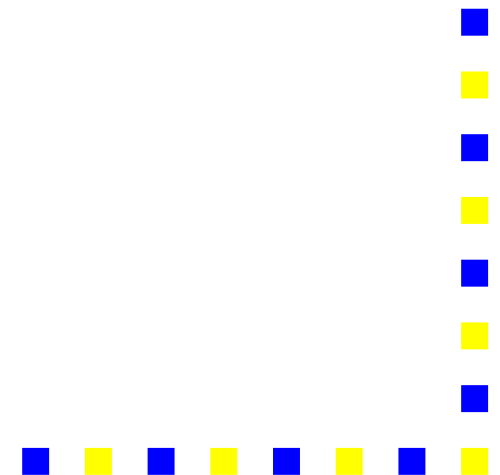
# Overview

---

JAVA source File	Java bytecode
JML specification	Specification language for bytecode



- Specifying java source files with the Java Modeling Language (JML). Examples
- Translation of JML into specification language for Java bytecode.
- Generation of class files containing specification information.
- WP for Java bytecode
- Example
- Conclusion

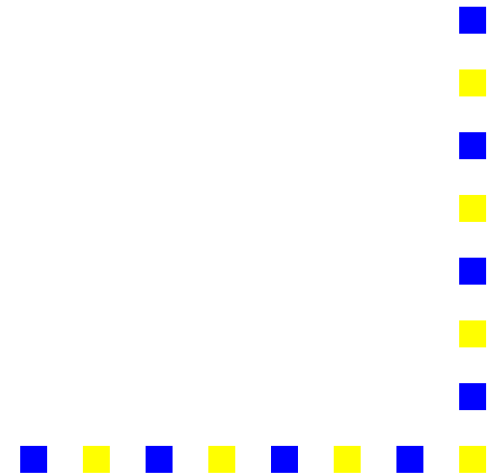


# Java Modeling Language

---

A specification language by which one can declare :

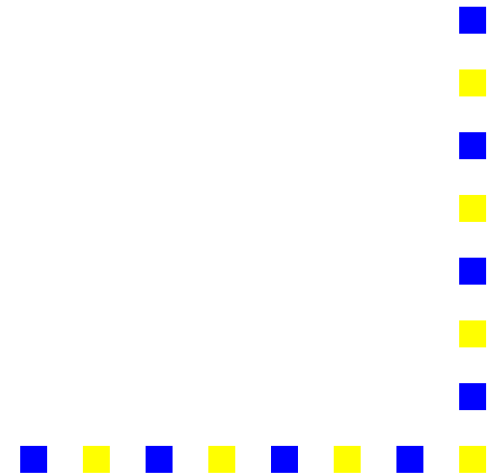
- method specification- preconditions, postconditions, loop invariants , frame conditions can be specified for a method.
- class specification- class invariants and history constraints can be specified for a class.



# *Java Modeling Language.*

---

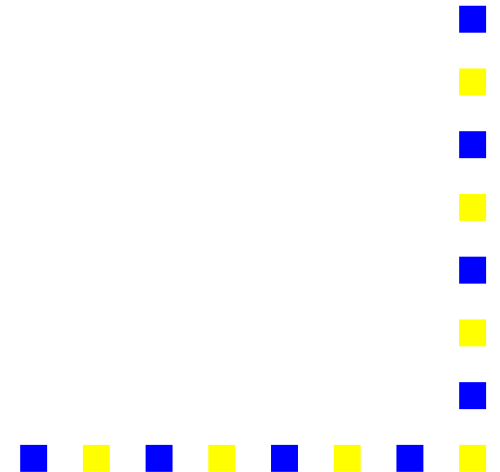
- Java expressions without side effects.
- JML model variables - in the JML specification variables that are discarded by the compiler can be used. These fields are used for specification purposes only.
- specific JML constants - `\result`, `\old`, etc.



# Java Modeling Language.Example

---

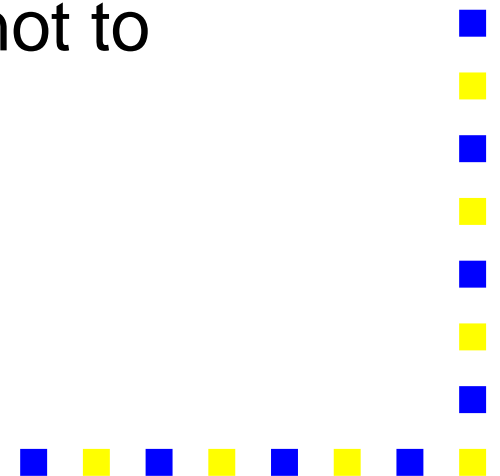
```
//@requires i != 0;
//@ensures \result == 1/i;
//@exsures ArithmeticException i==0;
int m(int i ) {
    int j;
    j = 1/i;
    return j;
}
```



# Translation of JML

---

- translation that should fit to bytecode - use of the same names, for example
- Integration of the specification in the class file
- The new class file format must respect the VM specification and not create problems at execution time.
- efficient coding - not too rich in order not to increase considerably the class file

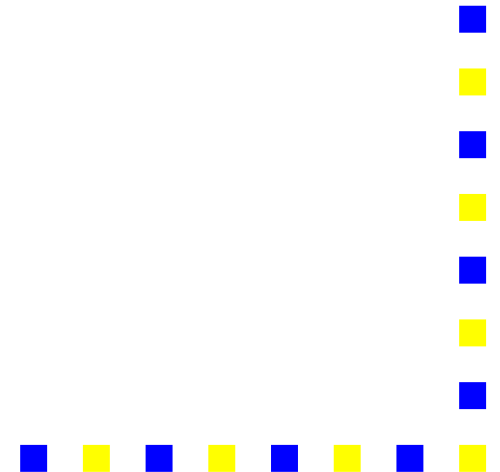




# Translation of JML

---

- for every JML unit - precondition, postcondition there will be a new attribute defined
- attribute\_info {  
    u2 attribute\_name\_index;  
    u4 attribute\_length;  
    u1 info[attribute\_length];  
}



# Generation of new class file format

---

## ClassFile

```
u2 constant_pool_count;  
cp_info constant_pool[constant_pool_count-1];  
:  
methodinfo[]  
:  
u2 attributes_count;  
attribute_info attributes[attributes_count];
```



# Translation of JML. Extension of the constant pool(CP)

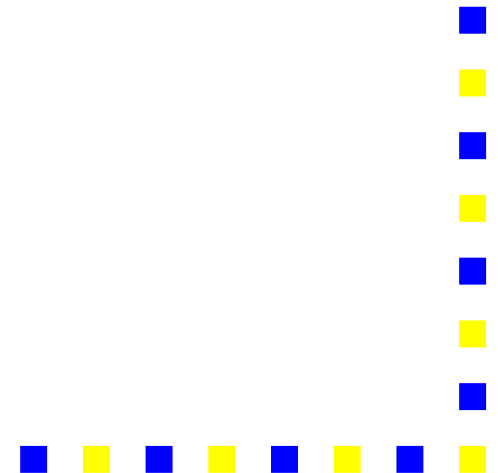
---

- Java virtual machine instructions do not rely on the runtime layout of classes, interfaces, class instances, or arrays. Instead, instructions refer to symbolic information in the class CP.
- Motivation  
Specification may involve fields that are not present in the class CP:
  - java fields that are not dereferenced in the code -
  - so there is no index for them in the constant pool
- Attribute - `ConstantPool_attribute` , that contains references that are added every time that they are not in the original constant pool, but are needed for the specification

# Translation of JML. Model variables

---

- Completely ignored by Java compilers. Define `Model_Field_Attribute`
- For every model variable in class `C`, an attribute added to the `attribute` array for the class file for `C`
- If a `model variable` is dereferenced at least once, add new index into the `ConstantPool_attribute`



# Translation of JML. Method specification translation

---

- Precondition, Postcondition, Loop Invariant, Assertions translated as new attributes for the `method_info` attribute
- Translation of any JML constant `c`- by its corresponding `code(c)`
- Translation of fields - by their corresponding index in the constant pool(the original or in its extension)
- Translation of local variables - by their indexes in the local variable array

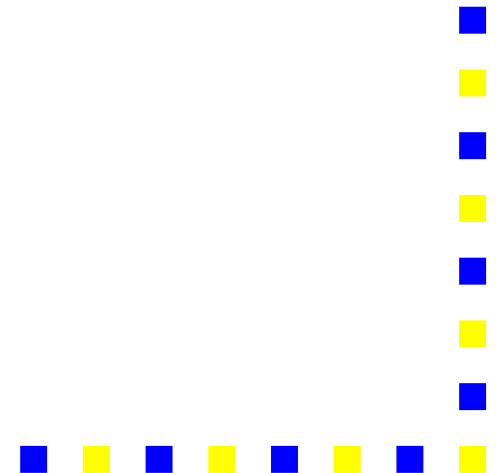


# Translation of JML

---

## ClassFile

```
u2 constant_pool_count;  
cp_info constant_pool[constant_pool_count-1];  
:  
methodinfo  
  code_attribute  
  Requires_Attribute  
  Ensures_Attribute  
  :  
  Class_Invariant_Attribute;  
  Constant_Pool_Attribute;
```

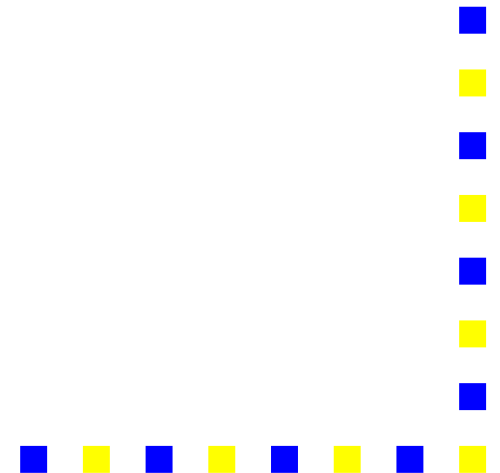


# *Translation of JML.Example*

---

Translation of method postcondition in bytecode format :

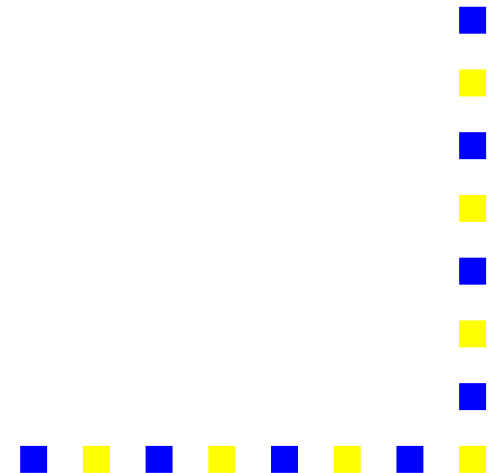
```
JMLEnsures_attribute {  
    u2 attribute_name_index;  
    u4 attribute_length;  
    un attribute_formula;  
}
```



# Example

---

```
//@requires i != 0;  
//@ensures \result == 1/i;  
int m(int i ) {  
    int j;  
    j = 1/i;  
    return j;  
}
```





# Example of translation of the postcondition predicate

---

For method  $m$  postcondition attribute will look be:

```
JMLEnsures_attribute {
```

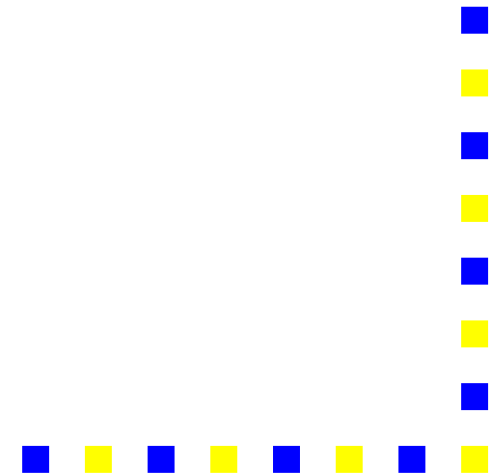
```
u2 attribute_name_index;
```

```
u4 attribute_length;
```

```
un attribute_formula =
```

```
 $code(\backslash result) \Vdash == \top \Vdash 1 \text{ div } local(1) \top$ 
```

```
}
```



# Example of translation of the postcondition predicate

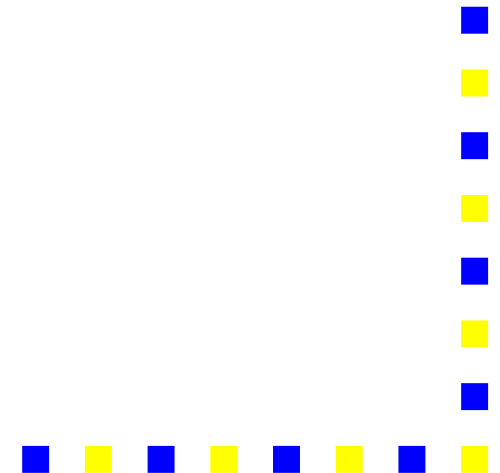
---

For method  $m$  precondition attribute will look be:  
JMLEnsures\_attribute {

u2 attribute\_name\_index;

u4 attribute\_length;

un attribute\_formula =  $\lceil local(1)! = 1 \rceil$   
}



# Translation of JML.Class specification

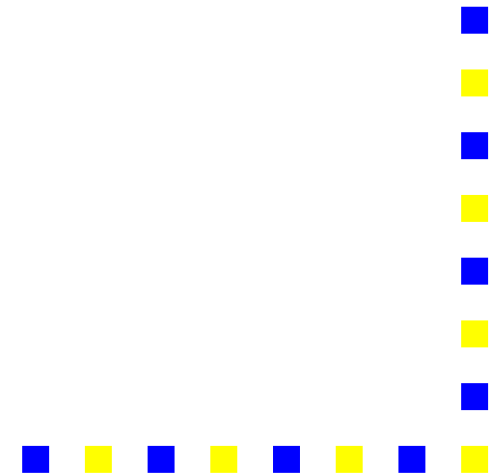
---

- Translation of class specification
  - Class invariant, History constraints : new attributes defined for the `class_info` data structure
  - JMLClassInvariant\_attribute {

u2 attribute\_name\_index;

u4 attribute\_length;

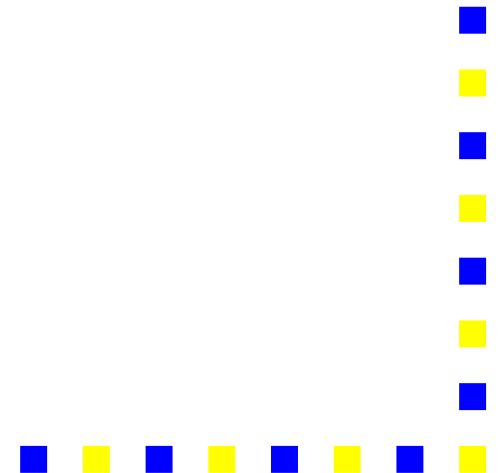
un attribute\_formula;  
}



# *Translation of JML. Limitations*

---

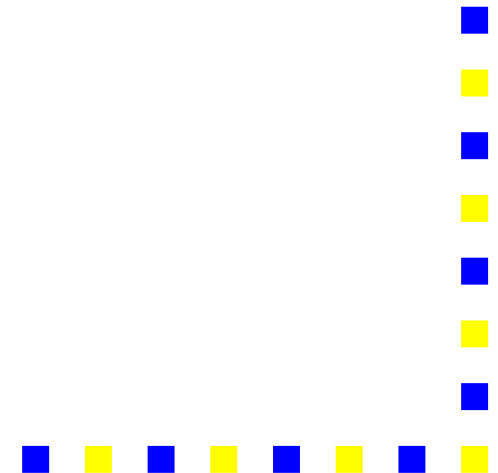
- Additional information that is not a must in the Java Virtual machine specification is required:  
`Linenumbertable` , `Local_variable_table` attributes might not be generated by certain Java language compilers.



# Weakest precondition for Java bytecode. Definitions

---

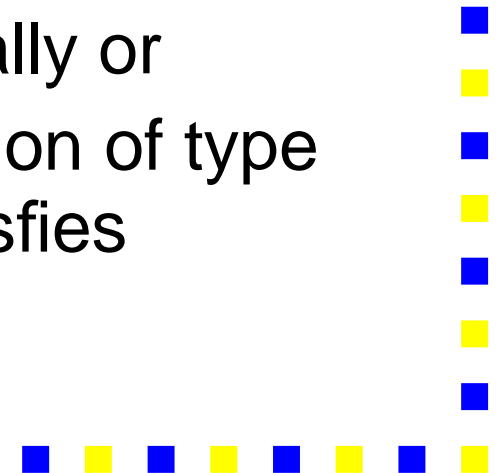
- defined over the execution graph of a bytecode
- Definition of a bytecode `block` : a subsequence of a bytecode  $B$  that
  - starts either with the initial instruction of  $B$ , either with a target of a jump instruction and
  - terminates either with a jump instruction or the last instruction of the bytecode  $B$ .



# Weakest precondition for Java bytecode

---

- $w_p : \text{Java\_instruction} \rightarrow \text{Predicate} \rightarrow (\text{Exception\_name} \rightarrow \text{Predicate}) \rightarrow \text{Predicate}$
- *Definition :*  
 $w_p(b, psi^n, psi^e)$  is a predicate that must hold in those initial states of the execution of the bytecode block  $b$  for which
  - it terminates in a state that satisfies the predicate  $psi^n$  if it terminates normally or
  - it terminates by throwing an exception of type *Exception\_Name* in a state that satisfies  $psi^e(\text{Exception\_Name})$



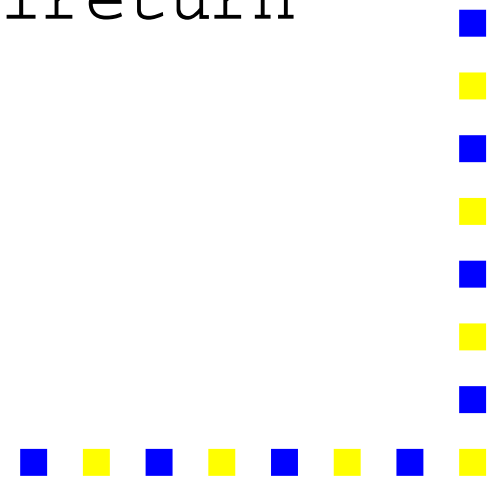
# WP for Java bytecode. Example

---

ByteCode for the method  $m$  :

```
//@ requires i != 0;
//@ ensures \result == 1/i;
int m(int i) {
    int j;
    j = 1/i;
    return j;
}
```

0	iconst_1
1	iload_1
2	idiv
3	istore_2
4	iload_2
5	ireturn



# *WP for Java bytecode. Example*

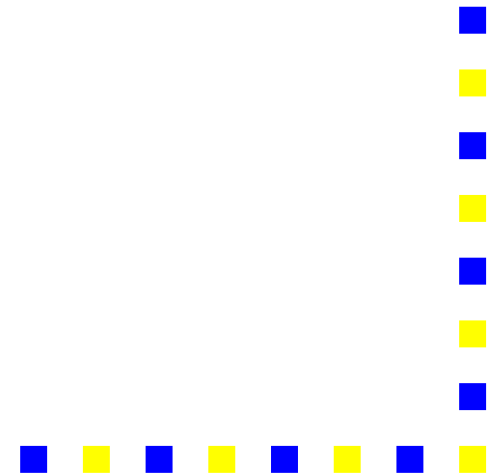
---

Some namings :

s - the stack

t - the stack top

head - a function that returns the subbytecode of a bytecode except for the last instruction



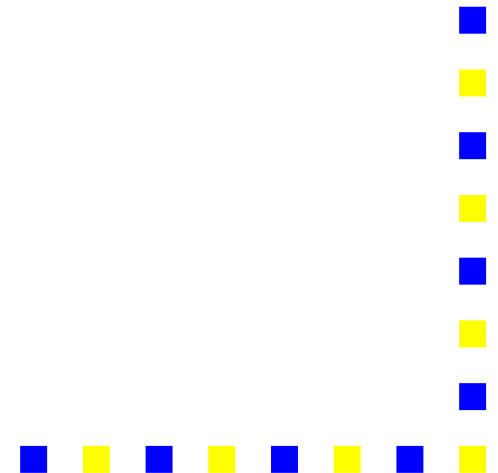


# *WP for Java bytecode. Example*

---

Calculating the Weakest precondition for the method  $m$  over its bytecode:

•  $wp(B, \text{postcondition}(m)) =$



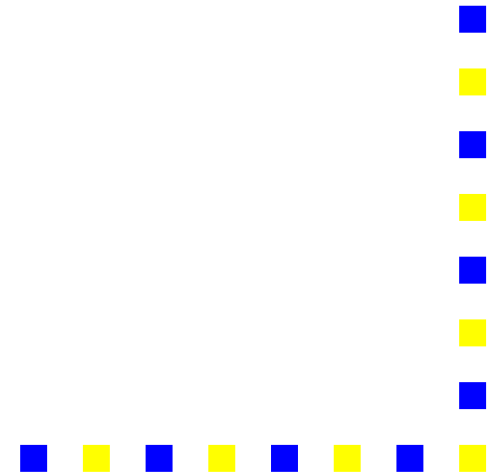
# *WP for Java bytecode. Example*

---

Calculating the Weakest precondition for the method  $m$  over its bytecode:

- $wp(B, \text{postcondition}(m)) =$

- $wp(B, \text{code}(\backslash\text{result})^{\ulcorner} == \ulcorner 1 \text{ div local}(1)^{\urcorner})$

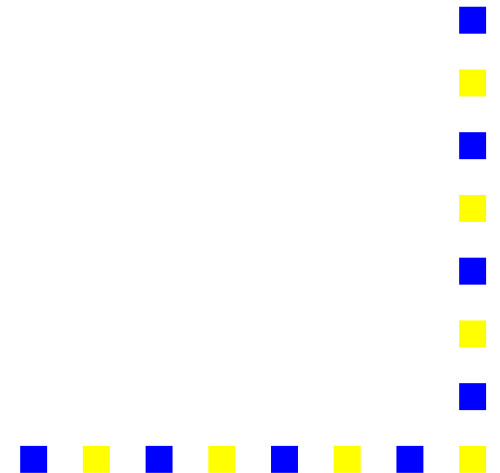


# WP for Java bytecode. Example

---

Calculating the Weakest precondition for the method  $m$  over its bytecode:

- $wp(B, \text{postcondition}(m)) =$
- $wp(B, \text{code}(\backslash\text{result})^{\ulcorner} == \ulcorner 1 \text{ div local}(1)^{\urcorner})$
- $wp(\text{head}(B), wp(\text{ireturn}, \text{code}(\backslash\text{result})^{\ulcorner} == \ulcorner 1 \text{ div local}(1)^{\urcorner}))$

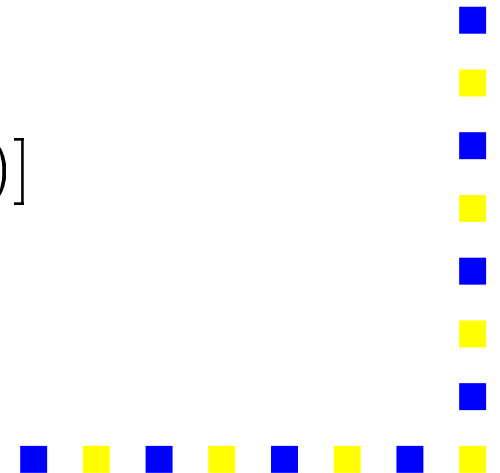


# WP for Java bytecode. Example

---

Calculating the Weakest precondition for the method  $m$  over its bytecode:

- $wp(B, \text{postcondition}(m)) =$
- $wp(B, \text{code}(\backslash\text{result})^{\ulcorner} == \ulcorner 1 \text{ div local}(1) \urcorner)$
- $wp(\text{head}(B), wp(\text{ireturn}, \text{code}(\backslash\text{result})^{\ulcorner} == \ulcorner 1 \text{ div local}(1) \urcorner))$
- $wp(\text{head}(B), \text{code}(\backslash\text{result})^{\ulcorner} == \ulcorner 1 \text{ div local}(1) \urcorner [\text{code}(\backslash\text{result})] \leftarrow S(t))$



# WP for Java bytecode. Example

Calculating the Weakest precondition for the method  $m$  over its bytecode:

- $wp(B, \text{postcondition}(m)) =$
- $wp(B, \text{code}(\backslash\text{result})^{\ulcorner} == \ulcorner 1 \text{ div local}(1) \urcorner)$
- $wp(\text{head}(B), wp(\text{ireturn}, \text{code}(\backslash\text{result})^{\ulcorner} == \ulcorner 1 \text{ div local}(1) \urcorner))$
- $wp(\text{head}(B), \text{code}(\backslash\text{result})^{\ulcorner} == \ulcorner 1 \text{ div local}(1) \urcorner [\text{code}(\backslash\text{result}) \leftarrow S(t)])$
- $wp(\text{head}(\text{head}(B)), wp(\text{iload}_1, \text{code}(\backslash\text{result})^{\ulcorner} == \ulcorner 1 \text{ div local}(1) \urcorner [\text{code}(\backslash\text{result}) \leftarrow S(t)]))$

.....



# WP for Java bytecode. Example

---

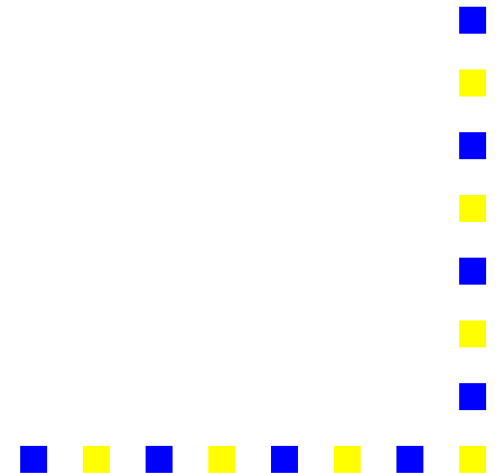
What is obtained is :

$$\begin{aligned} (code(\backslash result) \Vdash \neg \Vdash 1 \text{ div } local(1) \neg) \\ [code(\backslash result) \leftarrow S(t)] \\ [S(t) \leftarrow local(1)] \\ [t \leftarrow t + 1] \end{aligned}$$

....

doing all the substitutions the weakest precondition for  $m$  is :

$$(\Vdash 1 \text{ div } local(1) \neg \Vdash \Vdash \neg \Vdash 1 \text{ div } local(1) \neg)$$



- Results
  - class file format extension containing specification information - doesnot violate the VM specification and will not create conflicts on execution
  - calculus for extracting proof obligations from the java bytecode and the added specification
- Possible shortcomings : the size of the file increases, tests needed

