Interpretação e Compilação de Linguagens (de Programação)

21/22

Luís Caires (http://ctp.di.fct.unl.pt/~lcaires/)

Mestrado Integrado em Engenharia Informática

Departamento de Informática

Faculdade de Ciências e Tecnologia

Universidade Nova de Lisboa

The language CALCI (abstract syntax)

- CALCI AST constructors: num, add, mul, div, sub, id, def

num:	Integer → CALCI
id:	String → CALCI
add:	CALCI × CALCI → CALCI
mul:	CALCI × CALCI → CALCI
div:	CALCI × CALCI → CALCI
sub:	CALCI × CALCI → CALCI
def:	List(String × CALCI) × CALCI → CALCI

The language CALCI (concrete syntax)

- Sample CALCI program

```
def x = 2
      z = 2 * x
  in
     def y = def z = x+2 in z+z end
     in
        y + def y = 2+z in y end
    end
end
```

CALC Interpreter (environment based)

- Algorithm **eval( )** that computes the denotation (integer value) of any **open** CALCI expression:

$$\text{eval} : \text{CALCI} \times \text{ENV} \rightarrow \text{Integer}$$

eval(**num**(*n*) , *env*) ≜ *n*

eval(**id**(s) , *env*) ≜ *env*.Find(s)

eval(**add**(E1,E2) , *env*) ≜ **eval**(E1, *env*) + **eval**(E2, *env*)

…

```
eval( def(s, E1, E2), env) ≜ [ v1 = eval(E1, env);
                               env = env.BeginScope();
                               env =  env.Assoc(s, v1);
                               val = eval(E2, env);
                               env = env.EndScope();
                               return val ]
```

- Note: Case of id(s) implemented by lookup of the value of s in the current environment

CALC Compiler (environment based)

- Algorithm compile( ) that generates machine code for any **open** CALCI expression:

$$\text{eval} : \text{CALCI} \times \text{ENV} \rightarrow \text{CodeSeq}$$

Compilation Schemes

We will use compilation schemes to define our compiler

A compilation scheme defines the sequence of instructions generated for a programming language construct

[[E]]D = … code sequence ….

- E : program fragment
- D: environment (associates identifiers to "coordinates")

[[E_1+E_2]]D =
  [[E_1]]D
  [[E_2]]D
  iadd

Compilation Schemes

We will use compilation schemes to define our compiler

A compilation scheme defines the sequence of instructions generated for a programming language construct

$[[E_1 + E_2]]D =$
  $[[E_1]]D$
  $[[E_2]]D$
  iadd

$[[E_1 * E_2]]D =$
  $[[E_1]]D$
  $[[E_2]]D$
  imul

Compilation Schemes

We will use compilation schemes to define our compiler

A compilation scheme defines the sequence of instructions generated for a programming language construct

$[[E_1 - E_2]]D =$
- $[[E_1]]D$
- $[[E_2]]D$
- **isub**

$[[E_1 / E_2]]D =$
- $[[E_1]]D$
- $[[E_2]]D$
- **idiv**

Compilation Schemes

We will use compilation schemes to define our compiler

A compilation scheme defines the sequence of instructions generated for a programming language construct

[[n]]D =
    sipush n

[[- E]]D =
    sipush 0
    [[E]]D
    isub

Compilation Schemes

We will use compilation schemes to define our compiler

A compilation scheme defines the sequence of instructions generated for a programming language construct

[[n]]D =
    sipush n

[[- E]]D =
    [[E]]D
    ineg

Compilation Schemes

We will use compilation schemes to define our compiler

A compilation scheme defines the sequence of instructions generated for a programming language construct

[[x]]D =

???

[[**def** x = E **in** E **end**]]D =

???

Compilation Environment (D)

The compilation environment D maps each free name of the program to be compiled to its **coordinates**:

D(x) = (d, s) where

d: the depth of the topmost stack frame (from bottom of stack) where the identifier is declared.

s: the slot in the frame where the associated value is stored

At runtime, the value of an identifier s where D(x) = (d, s) is to be found by climbing the runtime environment N-d frames, and getting the value in the s slot, where N is the depth of the current environment D.

Compilation of declarations

The JVM code generated for an expression

def x1 = E1 ... xn = En **in** E **end**

1. creates and pushes a new stack frame (heap allocated) to store the value of the n identifiers x1 ... xn

2. initialises the slot for each xi with the value of Ei

3. pops off of the frame

At all times, a reference to the top of the runtime stack environment is stored in a JVM local variable SL

Compilation of declarations

The JVM code generated for an expression

def x1 = E1 ... xn = En **in** E **end**

1. creates and pushes a new stack frame (heap allocated) to store the value of the n identifiers x1 ... xn

2. initialises the slot for each xi with the value of Ei

3. pops off of the frame

The runtime stack is a runtime JVM representation of the interpreter environment.

Compilation of declarations

The JVM code generated for an expression

def x1 = E1 ... xn = En **in** E **end**

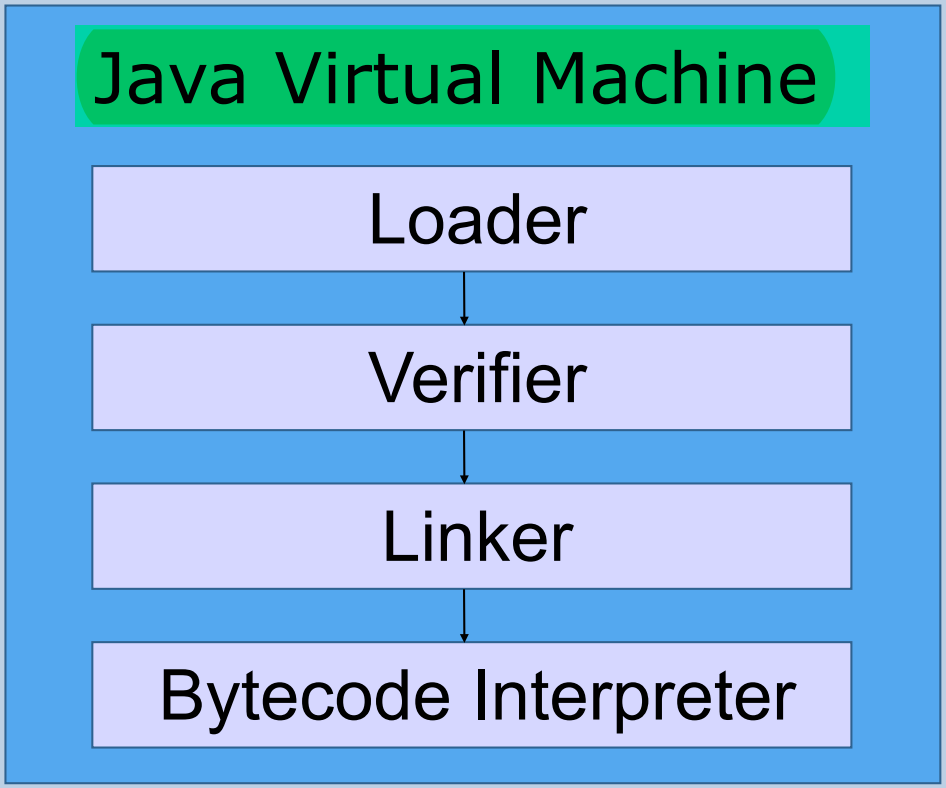
1. creates and pushes a new stack frame (heap allocated) to store the value of the n identifiers x1 ... xn

2. initialises the slot for each xi with the value of Ei

3. pops off of the frame

The generated code for an id does not need to dynamically search up in the stack, it knows its coordinates (from D(id))

JVM Architecture



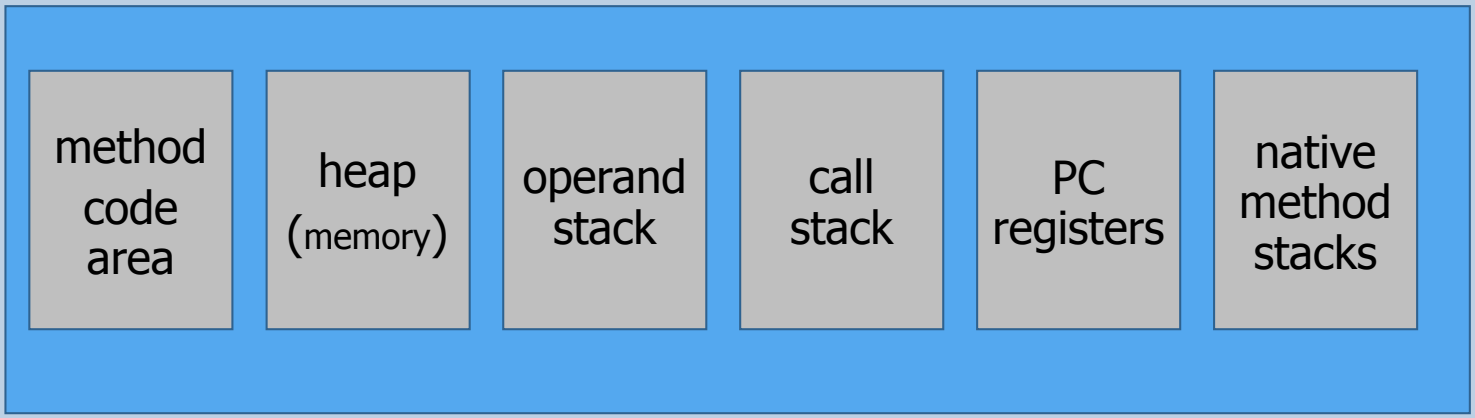
Operand Stack:

stores arguments and results of instructions

Heap:

stores dynamically allocated objects, arrays, strings, classes

....



Structure of Stack Frames (jasmin syntax)

```
class frame_id
.super java/lang/Object
.field public sl Lancestor_frame_id;
.field public s_1 I
.field public s_2 I
..
.field public s_n I
.end method
```

frame_id : unique type name generated by compiler

sl : holds reference to ancestor frame

ancestor_frame_id : name of the ancestor frame type

s_i : slot #i (stores value of id #i in this level)

Note: a stack frame is represented a JVM class with no methods (just public fields). This is akin to a C struct.

Structure of Stack Frames (jasmin syntax)

```
class frame_id
.super java/lang/Object
.field public sl Lancestor_frame_id;
.field public s_1 I
.field public s_2 I
..
.field public s_n I
.end method
```

frame_id : unique type name generated by compiler

sl : (reference also called the "static link")

ancestor_frame_id : name of the ancestor frame type

s_i: slot #i (stores value of id #i in this level)

Note: a stack frame is represented a JVM class with no methods (just public fields). This is akin to a C struct.

Compilation of declarations

[[**def** x1 = E1 … xn = En **in** E **end**]]D =

frame creation and linkage into environment stack (push)

```
new frame_id
dup
invokespecial frame_id/<init>()V
dup
aload SL
putfield frame_id/sl Lcurrframetype
astore SL
```

JVM instructions

new

Operation

Create new object

Format

```
new
indexbyte1
indexbyte2
```

Forms

new = 187 (0xbb)

Operand Stack

... →

..., *objectref*

Description

The unsigned *indexbyte1* and *indexbyte2* are used to construct an index into the run-time constant pool of the current class (§2.6), where the value of the index is (*indexbyte1* << 8) | *indexbyte2*. The run-time constant pool item at the index must be a symbolic reference to a class or interface type. The named class or interface type is resolved (§5.4.3.1) and should result in a class type. Memory for a new instance of that class is allocated from the garbage-collected heap, and the instance variables of the new object are initialized to their default initial values (§2.3, §2.4). The *objectref*, a `reference` to the instance, is pushed onto the operand stack.

On successful resolution of the class, it is initialized (§5.5) if it has not already been initialized.

JVM instructions

dup

Operation

Duplicate the top operand stack value

Format

dup

Forms

dup = 89 (0x59)

Operand Stack

..., *value* →

..., *value*, *value*

Description

Duplicate the top value on the operand stack and push the duplicated value onto the operand stack.

The *dup* instruction must not be used unless *value* is a value of a category 1 computational type (§2.11.1).

JVM instructions

aload

Operation

Load `reference` from local variable

Format

```
aload
index
```

Forms

aload = 25 (0x19)

Operand Stack

... →

..., *objectref*

Description

The *index* is an unsigned byte that must be an index into the local variable array of the current frame (§2.6). The local variable at *index* must contain a `reference`. The *objectref* in the local variable at *index* is pushed onto the operand stack.

Notes

The *aload* instruction cannot be used to load a value of type `returnAddress` from a local variable onto the operand stack. This asymmetry with the *astore* instruction (§*astore*) is intentional.

The *aload* opcode can be used in conjunction with the *wide* instruction (§*wide*) to access a local variable using a two-byte unsigned index.

JVM instructions

astore

Operation

Store `reference` into local variable

Format

```
astore
index
```

Forms

astore = 58 (0x3a)

Operand Stack

..., *objectref* →

...

Description

The *index* is an unsigned byte that must be an index into the local variable array of the current frame (§2.6). The *objectref* on the top of the operand stack must be of type `returnAddress` or of type `reference`. It is popped from the operand stack, and the value of the local variable at *index* is set to *objectref*.

Notes

The *astore* instruction is used with an *objectref* of type `returnAddress` when implementing the `finally` clause of the Java programming language (§3.13).

The *aload* instruction (§*aload*) cannot be used to load a value of type `returnAddress` from a local variable onto the operand stack. This asymmetry with the *astore* instruction is intentional.

The *astore* opcode can be used in conjunction with the *wide* instruction (§*wide*) to access a local variable using a two-byte unsigned index.

JVM instructions

putfield

Operation

Set field in object

Format

```
putfield
indexbyte1
indexbyte2
```

Forms

putfield = 181 (0xb5)

Operand Stack

..., *objectref*, *value* →

...

Description

The unsigned *indexbyte1* and *indexbyte2* are used to construct an index into the run-time constant pool of the current class (§2.6), where the value of the index is (*indexbyte1* << 8) | *indexbyte2*. The run-time constant pool item at that index must be a symbolic reference to a field (§5.1), which gives the name and descriptor of the field as well as a symbolic reference to the class in which the field is to be found. The class of *objectref* must not be an array. If the field is `protected`, and it is a member of a superclass of the current class, and the field is not declared in the same run-time package (§5.3) as the current class, then the class of *objectref* must be either the current class or a subclass of the current class.

The referenced field is resolved (§5.4.3.2). The type of a *value* stored by a *putfield* instruction must be compatible with the descriptor of the referenced field (§4.3.2). If the field descriptor type is `boolean`, `byte`, `char`, `short`, or `int`, then the *value* must be an `int`. If the field descriptor type is `float`, `long`, or `double`, then the *value* must be a `float`, `long`, or `double`, respectively. If the field descriptor type is a reference type, then the *value* must be of a type that is assignment compatible (JLS §5.2) with the field descriptor type. If the field is `final`, it must be declared in the current class, and the instruction must occur in an instance initialization method (`<init>`) of the current class (§2.9).

The *value* and *objectref* are popped from the operand stack. The *objectref* must be of type `reference`. The *value* undergoes value set conversion (§2.8.3), resulting in *value*', and the referenced field in *objectref* is set to *value*'.

Compilation of declarations

[[**def** x1 = E1 ... xn = En **in** E **end**]]D =

initialization of identifier slots in frame

```
aload SL
[[E1]]D+{ x1|->(s_1,s'), xn|->(s_n,s') }      // s' = |D|+1
putfield frame_id/s_1 I
aload SL
[[E2]]D+{ x1|->(s_1,s'), xn|->(s_n,s') }
putfield frame_id/s_2 I
....
aload SL
[[En]]D+{ x1|->(s_1,s'), xn|->(s_n,s') }
putfield frame_id/s_n I
```

Compilation of declarations

[[**def** x1 = E1 ... xn = En **in** E **end**]]D =

code for definition body

[[E]]D+{ x1|->(s_1,s'), xn|->(s_n,s') }

Compilation of declarations

[[**def** x1 = E1 … xn = En **in** E **end**]]D =

frame pop off and update of local SL

```
aload SL
getfield frame_id/sl Lcurrframetype
astore SL
```

JVM instructions

getfield

Operation

Fetch field from object

Format

```
getfield
indexbyte1
indexbyte2
```

Forms

getfield = 180 (0xb4)

Operand Stack

..., *objectref* →

..., *value*

Description

The *objectref*, which must be of type `reference`, is popped from the operand stack. The unsigned *indexbyte1* and *indexbyte2* are used to construct an index into the run-time constant pool of the current class (§2.6), where the value of the index is (*indexbyte1* << 8) | *indexbyte2*. The run-time constant pool item at that index must be a symbolic reference to a field (§5.1), which gives the name and descriptor of the field as well as a symbolic reference to the class in which the field is to be found. The referenced field is resolved (§5.4.3.2). The *value* of the referenced field in *objectref* is fetched and pushed onto the operand stack.

The type of *objectref* must not be an array type. If the field is `protected`, and it is a member of a superclass of the current class, and the field is not declared in the same run-time package (§5.3) as the current class, then the class of *objectref* must be either the current class or a subclass of the current class.

Compilation of identifiers (bound uses)

[[x]]D =

Assume D(x) = (**d**, **s**)

```
aload SL
getfield frame_id/sl Lancestor_frame_id
...
getfield frame_id/sl Lancestor_frame_id
getfield frame_id/s I
```

} N-d stack up dereferences

the number k of dereference (using getfield sl) is N-**d**

the coordinates stored in the compilation environment are used to generate code that fetches the identifier value from the appropriate frame at runtime.

Compilation of identifiers (bound uses)

[[x]]D =

Assume D(x) = (**d**, **s**)

```
aload SL
getfield frame_id/sl Lancestor_frame_id
…
getfield frame_id/sl Lancestor_frame_id
getfield frame_id/s I
```

} N-d stack up dereferences

D(x) = (**d**, **s**) where

d: the depth of the topmost stack frame (from bottom of stack) where the identifier is declared.

s: the slot in the frame where the associated value is stored

Compilation of CALCI (example code)

```
.class public frame_0
.super java/lang/Object
.field public sl Ljava/lang/Object;
.field public v0 I
.field public v1 I

.method public <init>()V
aload_0
invokenonvirtual java/lang/Object/<init>()V
return


.end method
```

default constructor (JVM requires it)

```
.class public frame_1
.super java/lang/Object
.field public sl Lframe_0;
.field public v0 I

.end method
```

```
def
  x  = 2
  y  = 3
in
 def
     k = x + y
 in
     x + y + k
 end
end;;
```

Compilation of CALCI (example code)

```
new frame_0
dup
invokespecial frame_0/<init>()V
dup
aload_3
putfield frame_0/sl Ljava/lang/Object;
astore_3
aload_3
sipush 2
putfield frame_0/v0 I
aload_3
sipush 3
putfield frame_0/v1 I
new frame_1
dup
invokespecial frame_1/<init>()V
dup
aload_3
putfield frame_1/sl Lframe_0;
astore_3
aload_3
getfield frame_1/sl Lframe_0;
getfield frame_0/v0 I
aload_3
```

```
aload_3
getfield frame_1/sl Lframe_0;
getfield frame_0/v1 I
iadd
putfield frame_1/v0 I
aload_3
getfield frame_1/sl Lframe_0;
getfield frame_0/v0 I
aload_3
getfield frame_1/sl Lframe_0;
getfield frame_0/v1 I
iadd
aload_3
getfield frame_1/v0 I
iadd
aload_3
getfield frame_1/sl Lframe_0;
astore_3
aload_3
getfield frame_0/sl Ljava/lang/Object;
astore_3
```

```
def
  x  = 2
  y  = 3
in
  def
    k = x + y
  in
      x + y + k
  end
end;;
```

Compilation of CALCI (example code)

```
new frame_0
dup
invokespecial frame_0/<init>()V
dup
aload_3
putfield frame_0/sl Ljava/lang/Object;
astore_3
aload_3
sipush 2
putfield frame_0/v0 I
aload_3
sipush 3
putfield frame_0/v1 I
new frame_1
dup
invokespecial frame_1/<init>()V
dup
aload_3
putfield frame_1/sl Lframe_0;
astore_3
aload_3
getfield frame_1/sl Lframe_0;
getfield frame_0/v0 I
aload_3
```

```
aload_3
getfield frame_1/sl Lframe_0;
getfield frame_0/v1 I
iadd
putfield frame_1/v0 I
aload_3
getfield frame_1/sl Lframe_0;
getfield frame_0/v0 I
aload_3
getfield frame_1/sl Lframe_0;
getfield frame_0/v1 I
iadd
aload_3
getfield frame_1/v0 I
iadd
aload_3
getfield frame_1/sl Lframe_0;
astore_3
aload_3
getfield frame_0/sl Ljava/lang/Object;
astore_3
```

```
def
  x  = 2
  y  = 3
in
  def
    k = x + y
  in
      x + y + k
  end
end;;
```

Compilation of CALCI (example code)

```
new frame_0
dup
invokespecial frame_0/<init>()V
dup
aload_3
putfield frame_0/sl Ljava/lang/Object;
astore_3
aload_3
sipush 2
putfield frame_0/v0 I
aload_3
sipush 3
putfield frame_0/v1 I
new frame_1
dup
invokespecial frame_1/<init>()V
dup
aload_3
putfield frame_1/sl Lframe_0;
astore_3
aload_3
getfield frame_1/sl Lframe_0;
getfield frame_0/v0 I
aload_3
```

```
aload_3
getfield frame_1/sl Lframe_0;
getfield frame_0/v1 I
iadd
putfield frame_1/v0 I
aload_3
getfield frame_1/sl Lframe_0;
getfield frame_0/v0 I
aload_3
getfield frame_1/sl Lframe_0;
getfield frame_0/v1 I
iadd
aload_3
getfield frame_1/v0 I
iadd
aload_3
getfield frame_1/sl Lframe_0;
astore_3
aload_3
getfield frame_0/sl Ljava/lang/Object;
astore_3
```

```
def
  x  = 2
  y  = 3
in
  def
    k = x + y
  in
      x + y + k
  end
end;;
```

Compilation of CALCI (example code)

```
new frame_0
dup
invokespecial frame_0/<init>()V
dup
aload_3
putfield frame_0/sl Ljava/lang/Object;
astore_3
aload_3
sipush 2
putfield frame_0/v0 I
aload_3
sipush 3
putfield frame_0/v1 I
new frame_1
dup
invokespecial frame_1/<init>()V
dup
aload_3
putfield frame_1/sl Lframe_0;
astore_3
aload_3
getfield frame_1/sl Lframe_0;
getfield frame_0/v0 I
aload_3
```

```
aload_3
getfield frame_1/sl Lframe_0;
getfield frame_0/v1 I
iadd
putfield frame_1/v0 I
aload_3
getfield frame_1/sl Lframe_0;
getfield frame_0/v0 I
aload_3
getfield frame_1/sl Lframe_0;
getfield frame_0/v1 I
iadd
aload_3
getfield frame_1/v0 I
iadd
aload_3
getfield frame_1/sl Lframe_0;
astore_3
aload_3
getfield frame_0/sl Ljava/lang/Object;
astore_3
```

```
def
  x  = 2
  y  = 3
in
  def
    k = x + y
  in
      x + y + k
  end
end;;
```

Compilation of CALCI (example code)

```
new frame_0
dup
invokespecial frame_0/<init>()V
dup
aload_3
putfield frame_0/sl Ljava/lang/Object;
astore_3
aload_3
sipush 2
putfield frame_0/v0 I
aload_3
sipush 3
putfield frame_0/v1 I
new frame_1
dup
invokespecial frame_1/<init>()V
dup
aload_3
putfield frame_1/sl Lframe_0;
astore_3
aload_3
getfield frame_1/sl Lframe_0;
getfield frame_0/v0 I
aload_3
```

```
aload_3
getfield frame_1/sl Lframe_0;
getfield frame_0/v1 I
iadd
putfield frame_1/v0 I
aload_3
getfield frame_1/sl Lframe_0;
getfield frame_0/v0 I
aload_3
getfield frame_1/sl Lframe_0;
getfield frame_0/v1 I
iadd
aload_3
getfield frame_1/v0 I
iadd
aload_3
getfield frame_1/sl Lframe_0;
astore_3
aload_3
getfield frame_0/sl Ljava/lang/Object;
astore_3
```

```
def
  x  = 2
  y  = 3
in
  def
    k = x + y
  in
      x + y + k
  end
end;;
```

Compilation of CALCI (example code)

```
new frame_0
dup
invokespecial frame_0/<init>()V
dup
aload_3
putfield frame_0/sl Ljava/lang/Object;
astore_3
aload_3
sipush 2
putfield frame_0/v0 I
aload_3
sipush 3
putfield frame_0/v1 I
new frame_1
dup
invokespecial frame_1/<init>()V
dup
aload_3
putfield frame_1/sl Lframe_0;
astore_3
aload_3
getfield frame_1/sl Lframe_0;
getfield frame_0/v0 I
```

```
aload_3
getfield frame_1/sl Lframe_0;
getfield frame_0/v1 I
iadd
putfield frame_1/v0 I
aload_3
getfield frame_1/sl Lframe_0;
getfield frame_0/v0 I
aload_3
getfield frame_1/sl Lframe_0;
getfield frame_0/v1 I
iadd
aload_3
getfield frame_1/v0 I
iadd
aload_3
getfield frame_1/sl Lframe_0;
astore_3
aload_3
getfield frame_0/sl Ljava/lang/Object;
astore_3
```

```
def
  x = 2
  y = 3
in
  def
    k = x + y
  in
      x + y + k
  end
end;;
```

Compilation of CALCI (example code)

```
new frame_0
dup
invokespecial frame_0/<init>()V
dup
aload_3
putfield frame_0/sl Ljava/lang/Object;
astore_3
aload_3
sipush 2
putfield frame_0/v0 I
aload_3
sipush 3
putfield frame_0/v1 I
new frame_1
dup
invokespecial frame_1/<init>()V
dup
aload_3
putfield frame_1/sl Lframe_0;
astore_3
aload_3
getfield frame_1/sl Lframe_0;
getfield frame_0/v0 I
```

```
aload_3
getfield frame_1/sl Lframe_0;
getfield frame_0/v1 I
iadd
putfield frame_1/v0 I
aload_3
getfield frame_1/sl Lframe_0;
getfield frame_0/v0 I
aload_3
getfield frame_1/sl Lframe_0;
getfield frame_0/v1 I
iadd
aload_3
getfield frame_1/v0 I
iadd
aload_3
getfield frame_1/sl Lframe_0;
astore_3
aload_3
getfield frame_0/sl Ljava/lang/Object;
astore_3
```

```
def
  x  = 2
  y  = 3
in
  def
    k = x + y
  in
      x + y + k
  end
end;;
```

Compilation of CALCI (example code)

```
new frame_0
dup
invokespecial frame_0/<init>()V
dup
aload_3
putfield frame_0/sl Ljava/lang/Object;
astore_3
aload_3
sipush 2
putfield frame_0/v0 I
aload_3
sipush 3
putfield frame_0/v1 I
new frame_1
dup
invokespecial frame_1/<init>()V
dup
aload_3
putfield frame_1/sl Lframe_0;
astore_3
aload_3
getfield frame_1/sl Lframe_0;
getfield frame_0/v0 I
```

```
aload_3
getfield frame_1/sl Lframe_0;
getfield frame_0/v1 I
iadd
putfield frame_1/v0 I
aload_3
getfield frame_1/sl Lframe_0;
getfield frame_0/v0 I
aload_3
getfield frame_1/sl Lframe_0;
getfield frame_0/v1 I
iadd
aload_3
getfield frame_1/v0 I
iadd
aload_3
getfield frame_1/sl Lframe_0;
astore_3
aload_3
getfield frame_0/sl Ljava/lang/Object;
astore_3
```

```
def
  x  = 2
  y  = 3
in
  def
    k = x + y
  in
      x + y + k
  end
end;;
```

Compilation of CALCI (example code)

```
new frame_0
dup
invokespecial frame_0/<init>()V
dup
aload_3
putfield frame_0/sl Ljava/lang/Object;
astore_3
aload_3
sipush 2
putfield frame_0/v0 I
aload_3
sipush 3
putfield frame_0/v1 I
new frame_1
dup
invokespecial frame_1/<init>()V
dup
aload_3
putfield frame_1/sl Lframe_0;
astore_3
aload_3
getfield frame_1/sl Lframe_0;
getfield frame_0/v0 I
```

```
aload_3
getfield frame_1/sl Lframe_0;
getfield frame_0/v1 I
iadd
putfield frame_1/v0 I
aload_3
getfield frame_1/sl Lframe_0;
getfield frame_0/v0 I
aload_3
getfield frame_1/sl Lframe_0;
getfield frame_0/v1 I
iadd
aload_3
getfield frame_1/v0 I
iadd
aload_3
getfield frame_1/sl Lframe_0;
astore_3
aload_3
getfield frame_0/sl Ljava/lang/Object;
astore_3
```

```
def
  x  = 2
  y  = 3
in
  def
    k = x + y
  in
      x + y + k
  end
end;;
```

Compilation of CALCI (example code)

```
new frame_0
dup
invokespecial frame_0/<init>()V
dup
aload_3
putfield frame_0/sl Ljava/lang/Object;
astore_3
aload_3
sipush 2
putfield frame_0/v0 I
aload_3
sipush 3
putfield frame_0/v1 I
new frame_1
dup
invokespecial frame_1/<init>()V
dup
aload_3
putfield frame_1/sl Lframe_0;
astore_3
aload_3
getfield frame_1/sl Lframe_0;
getfield frame_0/v0 I
```

```
aload_3
getfield frame_1/sl Lframe_0;
getfield frame_0/v1 I
iadd
putfield frame_1/v0 I
aload_3
getfield frame_1/sl Lframe_0;
getfield frame_0/v0 I
aload_3
getfield frame_1/sl Lframe_0;
getfield frame_0/v1 I
iadd
aload_3
getfield frame_1/v0 I
iadd
aload_3
getfield frame_1/sl Lframe_0;
astore_3
aload_3
getfield frame_0/sl Ljava/lang/Object;
astore_3
```

```
def
  x  = 2
  y  = 3
in
  def
    k = x + y
  in
      x + y + k
  end
end;;
```

Compilation of CALCI (example code)

```
new frame_0
dup
invokespecial frame_0/<init>()V
dup
aload_3
putfield frame_0/sl Ljava/lang/Object;
astore_3
aload_3
sipush 2
putfield frame_0/v0 I
aload_3
sipush 3
putfield frame_0/v1 I
new frame_1
dup
invokespecial frame_1/<init>()V
dup
aload_3
putfield frame_1/sl Lframe_0;
astore_3
aload_3
getfield frame_1/sl Lframe_0;
getfield frame_0/v0 I
```

```
aload_3
getfield frame_1/sl Lframe_0;
getfield frame_0/v1 I
iadd
putfield frame_1/v0 I
aload_3
getfield frame_1/sl Lframe_0;
getfield frame_0/v0 I
aload_3
getfield frame_1/sl Lframe_0;
getfield frame_0/v1 I
iadd
aload_3
getfield frame_1/v0 I
iadd
aload_3
getfield frame_1/sl Lframe_0;
astore_3
aload_3
getfield frame_0/sl Ljava/lang/Object;
astore_3
```

```
def
  x  = 2
  y  = 3
in
 def
   k = x + y
 in
      x + y + k
 end
end;;
```

Compilation of CALCI (example code)

```
new frame_0
dup
invokespecial frame_0/<init>()V
dup
aload_3
putfield frame_0/sl Ljava/lang/Object;
astore_3
aload_3
sipush 2
putfield frame_0/v0 I
aload_3
sipush 3
putfield frame_0/v1 I
new frame_1
dup
invokespecial frame_1/<init>()V
dup
aload_3
putfield frame_1/sl Lframe_0;
astore_3
aload_3
getfield frame_1/sl Lframe_0;
getfield frame_0/v0 I
aload_3
getfield frame_1/sl Lframe_0;
getfield frame_0/v1 I
iadd
putfield frame_1/v0 I
aload_3
getfield frame_1/sl Lframe_0;
getfield frame_0/v0 I
aload_3
getfield frame_1/sl Lframe_0;
getfield frame_0/v1 I
iadd
aload_3
getfield frame_1/v0 I
iadd
aload_3
getfield frame_1/sl Lframe_0;
astore_3
aload_3
getfield frame_0/sl Ljava/lang/Object;
astore_3
```

```
def
  x  = 2
  y  = 3
in
  def
    k = x + y
  in
      x + y + k
  end
end;;
```

Interpretação e Compilação de Linguagens (de Programação)

21/22

Luís Caires (http://ctp.di.fct.unl.pt/~lcaires/)

Mestrado Integrado em Engenharia Informática

Departamento de Informática

Faculdade de Ciências e Tecnologia

Universidade Nova de Lisboa