

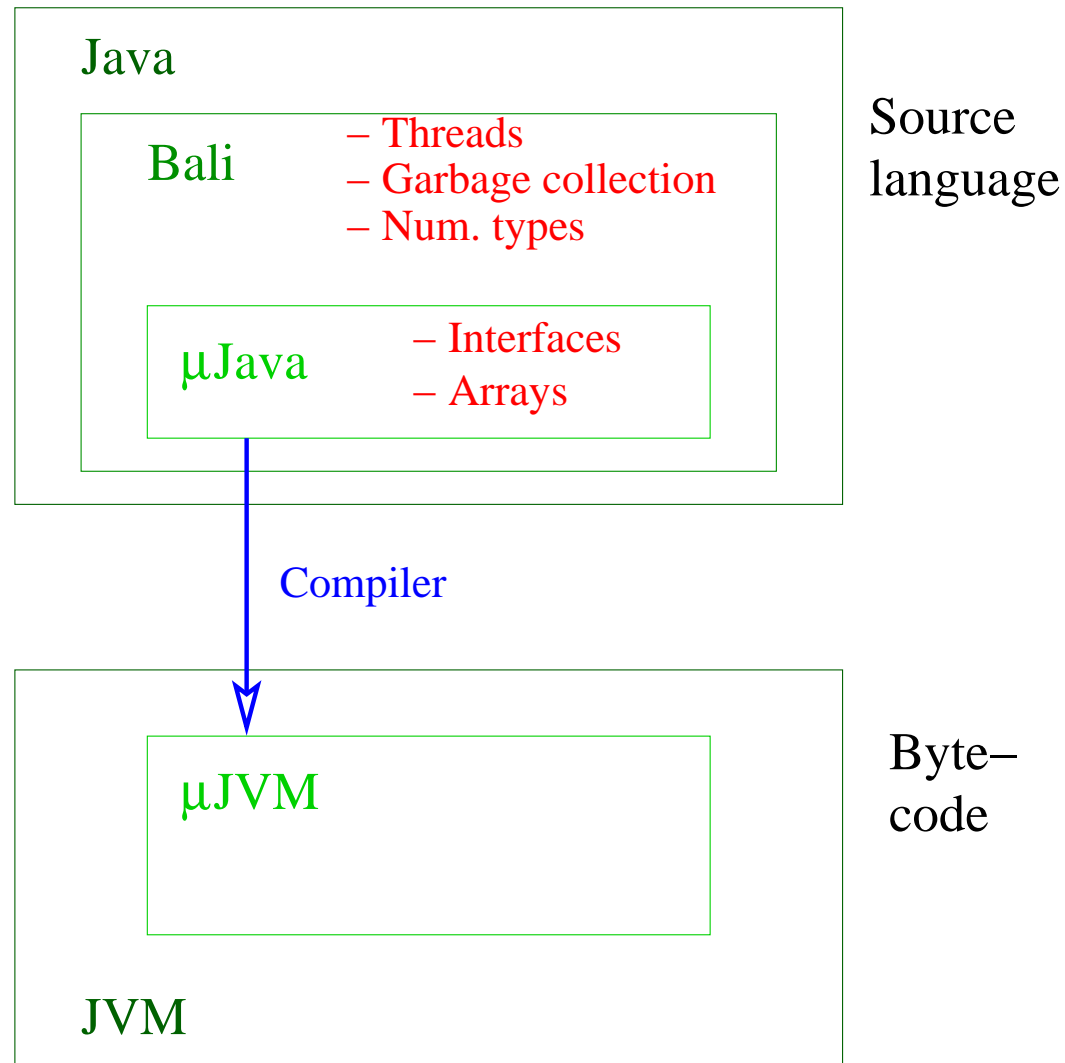
Verification of a Java Compiler in Isabelle

Martin Strecker

7.1.2002

- Java:
 - Types, values, terms, ...
 - Typing and operational semantics
- JVM
- Compiler
 - Definition
 - Proof Techniques
- Compilation and Bytecode Verification

Java and Isabelle-Java



Java: Types, Values, Terms (1)

... defined as inductive data types:

Types:

```
datatype prim_ty = Void | Boolean | Integer
```

```
datatype ref_ty = NullT | ClassT cname
```

```
datatype ty = PrimT prim_ty | RefT ref_ty
```

Values:

```
datatype val = Unit | Bool bool | Intg int | Null | Addr loc
```

- No interfaces / arrays
- Only numeric type: Integer

Java: Types, Values, Terms (2)

Terms:

```
datatype expr = ...
  | NewC cname
  | vname ::= expr
  | {cname} expr . vname ::= expr
  | BinOp binop expr expr
  | {cname} expr . mname {(ty) list} (expr) list
```

```
datatype stmt = ...
  | Expr expr
  | If (expr) stmt Else stmt
  | Throw expr
```

Method Call: Annotation

$$a_2 . m(a_3, c) \implies \{A_2\} a_2 . m(\{ [A_1, C] \} [a_3, c])$$

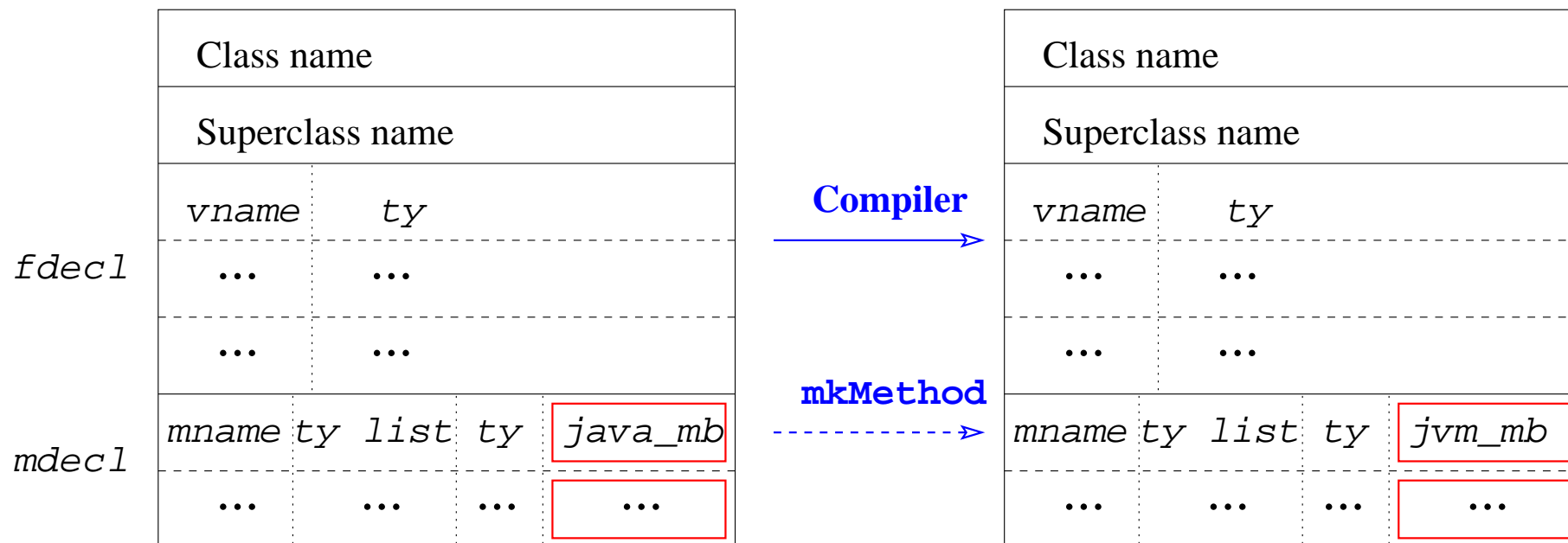
Java: Methods, Classes, Programs

	Class name			
	Superclass name			
<i>fdecl</i>	<i>vname</i>	<i>ty</i>		
		
		
<i>mdecl</i>	<i>mname</i>	<i>ty list</i>	<i>ty</i>	<i>java_mb</i>

java_mb =
 $vname\ list \times (vname \times ty)\ list$
 $\times stmt \times expr$

mdecl =
 $mname \times ty\ list \times ty \times java_mb$

Java / JVM: Methods, Classes, Programs



java_mb =
vname list × (*vname* × *ty*) *list*
 × *stmt* × *expr*

jvm_mb =
nat × *nat* × *bytecode*

'*c* *mdecl* = *mname* × *ty list* × *ty* × '*c*

'*c* *cdecl* = *cname* × *cname* × *fdecl list* × '*c* *mdecl list*

Java: Typing

Inductively defined judgements

- Expressions: $E \vdash e ::_E T$
- Statements: $E \vdash s ::_S \checkmark$

with environment E : $java_mb\ env = java_mb\ prog \times lenv$

Java: Operational Semantics

State

$\sigma : xstate = xcpt\ option \times heap \times locals$

Evaluation

- of expressions: $\Gamma \vdash (\sigma, e) \longrightarrow_E (v, \sigma')$
- of statements: $\Gamma \vdash (\sigma, s) \longrightarrow_S \sigma'$

with program $\Gamma: java_mb\ prog$

\implies big step (“natural”) semantics

Type Safety

Evaluation transforms state σ conforming to E to state σ' again conforming to E .

JVM: Instructions

```

datatype instr
  = Load nat                | Store nat
  | LitPush val              | New cname
  | Getfield vname cname    | Putfield vname cname
  | Checkcast cname
  | Invoke cname mname (ty list)
  | Return
  | Pop
  | Dup          | Dup_x1      | Dup_x2          | Swap
  | IAdd
  | Goto int      | Ifcmpeq int

```

```

bytecode = instr list

```

JVM: State and Operational Semantics

State

datatype *jvm_state* = *xcpt option* × *heap* × *frame list*
frame = *opstack* × *locvars* × *cname* × *sig* × *nat*

Operational Semantics

- One-step execution relation:

`exec_instr (Load idx) G hp stk vars Cl sig pc frs =
 (None, hp, ((vars ! idx) # stk, vars, Cl, sig, pc+1)#frs)`

- Execution `exec_all` as transitive closure

Compiler: Definition (1)

`mkExpr` :: `java_mb` => `expr` => `bytecode`

`mkStmt` :: `java_mb` => `stmt` => `bytecode`

`mkExpr jmb (vn ::= e) = mkExpr jmb e @ [Dup , Store (index jmb vn)]`

`mkExpr jmb ({cn}e1.mn {Ts}(ps)) =
 mkExpr jmb e1 @ mkExprs jmb ps @ [Invoke cn mn Ts]`

`mkStmt jmb (c1;; c2) = (mkStmt jmb c1) @ (mkStmt jmb c2)`

Compiler: Definition (2)

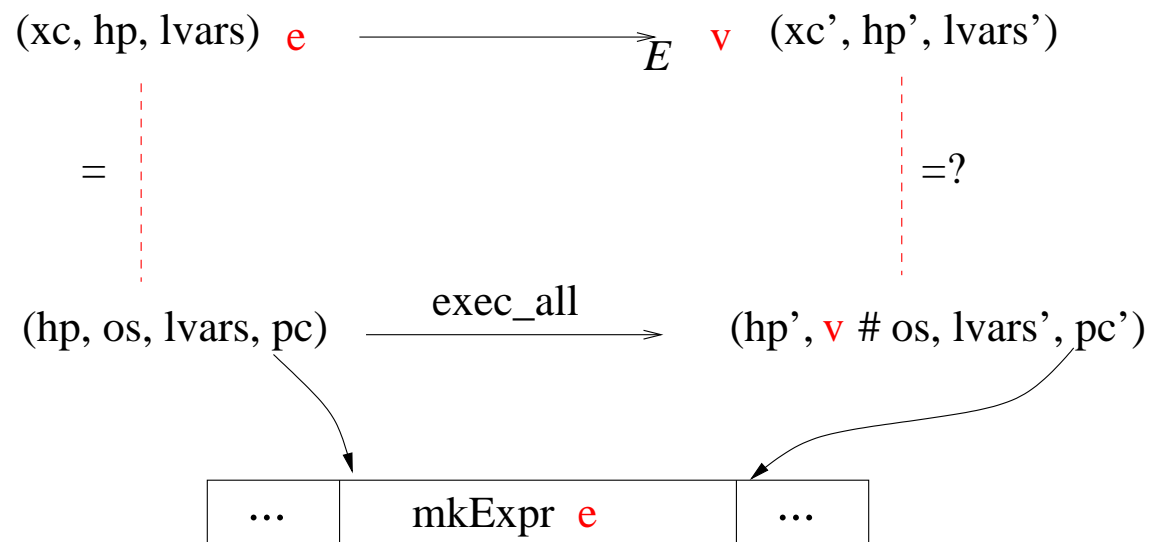
```
mkMethod :: java_mb => nat * nat * bytecode
```

```
mkMethod jmb == let (params,locals,blk,res) = jmb  
                in (max_ssize jmb blk res,  
                    length locals,  
                    concat (map (mkInit jmb) locals) @  
                    mkStmt jmb blk @ mkExpr jmb res @ [Return])
```

Compiler: Correctness Statement

Assumption (preliminary): No exceptions during evaluation

Correctness (for expressions):



Statements: similar

Verification: Preconditions

Remember: *Current environment* (Γ, Λ) given by:

- Current program Γ
- locals Λ of current method (identified by class C and signature S)

Preconditions:

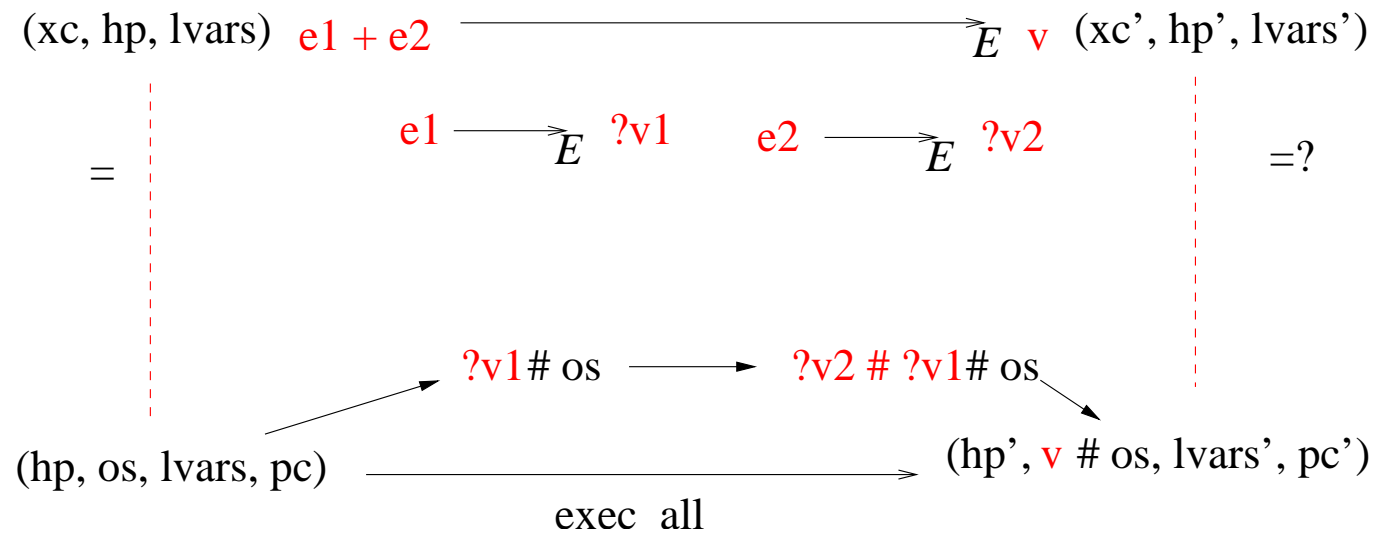
- Γ is well-formed
- C and S are defined in Γ
- State $(xc, hp, lvars)$ conforms to environment (Γ, Λ)
- Expression e is well-typed: $\exists T. (\Gamma, \Lambda) \vdash e ::_E T$

\rightsquigarrow Correctness statement for “reasonable” programs only

Proof Techniques

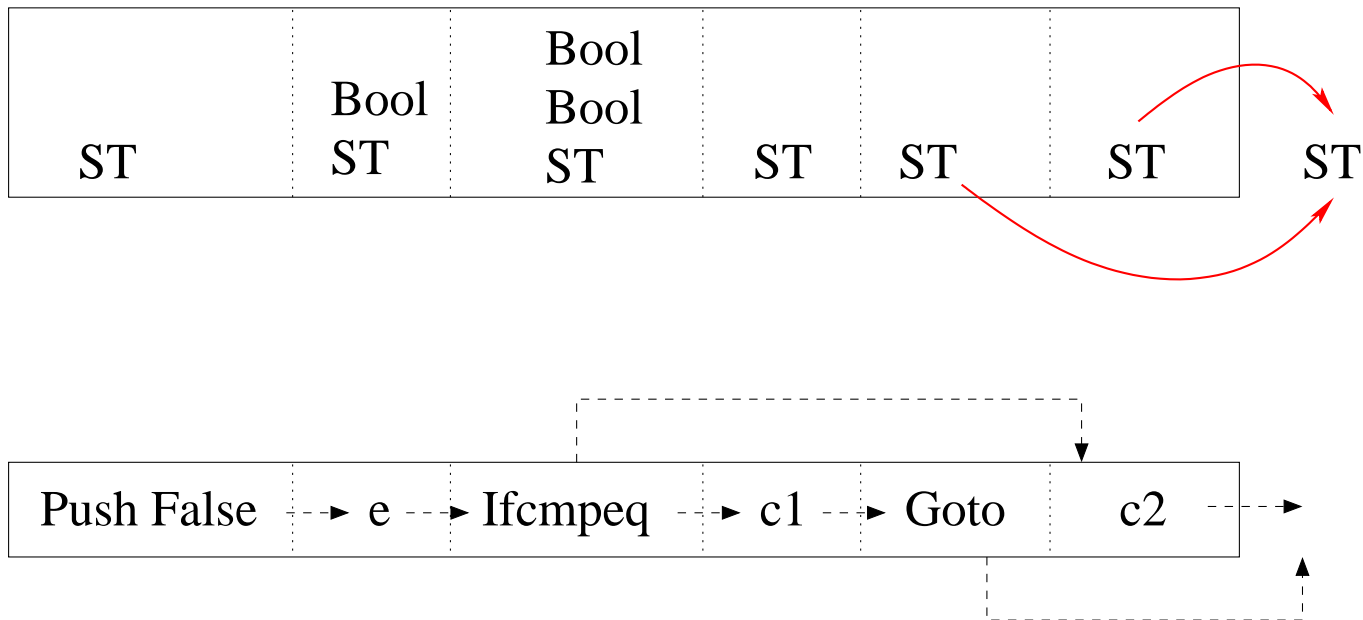
Proof: Induction over evaluation relation:

- Propagate assumptions, e.g.: conformance (requires Type Safety)
- Apply induction hypotheses
- Evaluate symbolically



Compilation and Bytecode Verification

mkStTpStmt (If (e) c1 Else c2)



mkStmt (If (e) c1 Else c2)

```
mkStTpStmt  :: java_mb => stmt => statetype list
mkStmt      :: java_mb => stmt => bytecode
```

Summary

Formalization

- Contains most essentials, some details missing
- Few Isabelle specifics \rightsquigarrow transferable to other environments

Compiler

- Translation of method bodies; no data refinements, no optimizations (which?)
- Object-orientation of minor importance
- Executable (extraction of ML code), easy to produce “real” class files
- Big step semantics leads to concise correctness statement

To do

- Integrate exceptions
- Compilation and bytecode verification
- Streamline proofs
- Tackle larger language fragments