

# La correttezza del Bytecode Verifier (versione semplificata)

Il Java Bytecode Verifier controlla la correttezza del bytecode Java prima che esso sia eseguito. Esso verifica che

il bytecode utilizza una *quantità finita di risorse*, cioè un numero finito di *variabili locali* ed una quantità finita di *pila* ;

i salti avvengono ad indirizzi legali del bytecode;

le operazioni sono sempre applicate ad argomenti di tipo opportuno.

L'algoritmo di verifica è di tipo "data flow". Quello che noi facciamo è di dimostrarne la correttezza con l'interpretazione astratta.

**Il linguaggio JVMML.** Un programma  $P$  è una funzione parziale da un insieme di indirizzi **ADDR** a *istruzioni*.

$$\begin{aligned} \textit{istruzioni} ::= & \text{ inc } \mid \text{ pop } \mid \text{ push0 } \mid \text{ load } x \mid \text{ store } x \mid \text{ if } L \\ & \mid \text{ putfield } \sigma.a \tau \mid \text{ getfield } \sigma.a \tau \\ & \mid \text{ return } \end{aligned}$$

dove

$x$  è una variabile in **VAR**

$L$  è un indirizzo in **ADDR**

$\sigma$  e  $\tau$  sono tipi in  $\mathcal{T}$ .

**I Tipi  $\mathcal{T}$ .** L'insieme dei tipi  $\mathcal{T}$  contiene i tipi  $\perp$ , **INT** e i "tipi oggetto".

Il simbolo  $\tau$  è usato per indicare un qualunque tipo di  $\mathcal{T}$ ; il simbolo  $\sigma$  indica un generico tipo oggetto.

Ad tipo oggetto  $\sigma$  è assegnato un **record**  $\underline{\sigma} = [ a_1: \tau_1 ; \dots ; a_k: \tau_k ]$  che rappresenta la sequenza dei *campi dell'oggetto*.

Si assume un insieme infinito di nomi di oggetti (puntatori) per ogni tipo  $\sigma$ .

Sui tipi  $\mathcal{T}$  è definita una relazione di sottotipo  $<:$ , che è riflessiva, antisimmetrica e transitiva, e contiene

*per ogni*  $\tau: \quad \tau <: \top$

( $\top$  è il **super-tipo** di ogni tipo).

**Ordinamento dei Valori.** Non c'è alcun ordinamento sui valori.

## Significato intuitivo delle istruzioni.

la *macchina virtuale di Java* (JVM) utilizza una pila  $s$  ed un insieme di variabili locali  $f$ , oltre ad un registro  $pc$  in cui è memorizzato l'indirizzo della istruzione da eseguire.

una generica configurazione è

$(pc, s, f)$

Il significato delle istruzioni della JVM è il seguente:

- `inc`, `pop`, `push0`, `load x` eseguono le ovvie operazioni sulla pila; `store x` memorizza l'elemento in testa alla pila nella variabile  $x$  (e lo elimina dalla pila).
- `if l` è l'istruzione di salto: se il valore sulla pila è diverso da 0 allora la prossima istruzione da eseguire è quella all'indirizzo  $l$ , altrimenti si procede con la prossima istruzione in sequenza (al solito, il valore sulla pila è eliminato).

- `putfield`  $\sigma.a \tau$  e `getfield`  $\sigma.a \tau$  inseriscono e prelevano il valore di un campo di un certo oggetto.
- `return` è l'istruzione che termina l'esecuzione.

**La semantica operativa.** La generica configurazione della JVM è:

$$\Vdash_H (pc, s, f)$$

dove

- $pc$  è un indirizzo di memoria (il program counter),
- $s$  è una sequenza di valori (possono essere interi oppure indirizzi di oggetti). La sequenza vuota sarà indicata con  $\varepsilon$ ,
- $f$  è una funzione parziale da `VAR` a valori.  $f_0$  è la funzione che associa un qualunque valore alle variabili;
- $H$  è l'`heap`, cioè il luogo che memorizza gli oggetti. Formalmente,  $H$  è una funzione parziale da nomi di oggetti (indirizzi) in `record values`  $[a_1 : v_1 ; \dots ; a_k : v_k]$ , dove  $v_i$  sono valori (*interi* oppure *record values*).  $H_0$  è l'heap vuoto.

**tipo di una classe:** una classe  $P$  ha un record di tipi. Ogni elemento ha una delle due forme:

$\tau$  (campi)                       $\tau_1 \dots \tau_n$       void      (metodi)

Sia  $\sigma$  il tipo dell'oggetto "self" di  $P$ . Lo stato iniziale di  $P$  è

$$\Vdash_{H'} (0, \varepsilon, f')$$

dove

$H'$  è un heap che contiene almeno tutti gli oggetti memorizzati in variabili di  $f'$ .

$f' = f_0 [0 : v_0, 1 : v_1, \dots, n : v_n]$ , tale che  $v_0 : \sigma, v_1 : \tau_1, \dots, v_n : \tau_n$ .

La forma delle transizioni sarà:

$$\Vdash_H (pc, s, f) \quad \Vdash_{H'} (pc', s', f')$$

(si osservi che il programma  $P$  è lasciato implicito)

$$\frac{P[pc] = \text{inc}}{\vdash_H (pc, n.s, f) \quad \vdash_H (pc+1, (n+1).s, f)}$$

$$\frac{P[pc] = \text{push0}}{\vdash_H (pc, s, f) \quad \vdash_H (pc+1, 0.s, f)}$$

$$\frac{P[pc] = \text{pop}}{\vdash_H (pc, v.s, f) \quad \vdash_H (pc+1, s, f)}$$

$$\frac{P[pc] = \text{load } x}{\vdash_H (pc, s, f) \quad \vdash_H (pc+1, f(x).s, f)}$$

$$\frac{P[pc] = \text{store } x}{\vdash_H (pc, v.s, f) \quad \vdash_H (pc+1, s, f[x : v])}$$

$$\frac{P[pc] = \text{if } L}{\vdash_H (pc, 0.s, f) \quad \vdash_H (pc+1, s, f)}$$

$$\frac{P[pc] = \text{if } L \quad n \quad 0}{\vdash_H (pc, n.s, f) \quad \vdash_H (L, s, f)}$$

$$\frac{P[pc] = \text{putfield } \sigma.a \tau \quad H' = H[o.a : v]}{\vdash_H (pc, v.o.s, f) \quad \vdash_{H'} (pc+1, s, f)}$$

$$\frac{P[pc] = \text{getfield } \sigma.a \tau \quad H(o.a) = v}{\vdash_H (pc, o.s, f) \quad \vdash_H (pc+1, v.s, f)}$$

- osservazione.** (1) L'unica istruzione che modifica l'heap è la `putfield`.  
 (2) Non c'è alcuna regola per `return`.

## Il dominio astratto delle configurazioni

Definiamo la **controparte astratta di una configurazione**  $\Vdash_H(p_C, s, f)$ .

la **pila astratta**  $S$  è una sequenza  $\tau_1. \dots .\tau_n$  di di tipi in  $\mathcal{T}$

$F$  è una funzione parziale a **dominio finito** da **VAR** a tipi.  $F_0$  è la funzione che associa  $\perp$  ad ogni variabile.

La **configurazione astratta** è  $\perp$ ,  $\top$  oppure una coppia:

$(S, F)$

La **relazione d'ordine** sulle configurazioni astratte è la seguente:

$\perp \leq (S, F) \leq \top$ , per ogni  $(S, F)$  ;

$(S, F) \leq (S', F')$  se e solamente se:

a.  $S = \tau_1. \dots .\tau_n$ ,  $S' = \tau_1'. \dots .\tau_n'$  e, per ogni  $i$ ,  $\tau_i' <: \tau_i$  ( $\tau_i$  è sottotipo di  $\tau_i'$ )  
(le lunghezze sono uguali!)

b. per ogni  $x \in \text{VAR}$ ,  $F(x) <: F'(x)$  ( $F'(x)$  è sottotipo di  $F(x)$ )

Il **dominio astratto** è l'insieme di tutte le configurazioni astratte con la relazione d'ordine  $\leq$ .

### osservazione.

il dominio astratto è un reticolo (ciò deriva dal fatto che " $<$ :" è un ordinamento parziale)

l'elemento  $\top$  rappresenta una situazione di errore: se le computazioni astratte avranno  $\top$  nella forma stabile, allora il programma non sarà considerato corretto

il dominio astratto ha un numero infinito di configurazioni. Il dominio è infinito

- ✓ in larghezza
- ✓ in altezza

essere infinito in altezza è problematico per l'interpretazione astratta (bisogna introdurre operatori di widening e narrowing).

se l'insieme VAR è finito: dominio astratto risulta finito in altezza

Il **dominio concreto** sono insiemi di configurazioni del tipo:

$$\{ \Vdash_{Hi} (pc_i, s_i, f_i) \mid i \in I \}$$

per qualche famiglia  $I$  (anche vuota).

l'ordinamento è quello insiemistico.

## le funzioni e

La funzione è definita su insiemi di configurazioni come segue:

- $( ) = \perp$  ;
- $\{ \Vdash_H(pC, s, f) \} = (S, F)$ , dove
  - o  $S = \tau_1 \dots \tau_n$ , se  $s = v_1 \dots v_n$  e, per ogni  $i$ ,  $v_i : \tau_i$  ;
  - o per ogni  $x_i \text{ VAR}$ ,  $F(x_i) = \tau_i$ , se  $f(x_i) = v_i$  e  $v_i : \tau_i$ .

Inoltre, sia per  $S$  che per  $F$ , non esiste  $\tau_i'$  tale che  $v_i : \tau_i'$  e  $\tau_i' < : \tau_i$  ( $\tau_i$  è il minimo sottotipo possibile);

- $(C \cup C') = (C) \cup (C')$  .

$$[(S, F)] = \{ \Vdash_H(\mathbf{n}, s, f) \mid \begin{array}{l} - \mathbf{n} \quad \text{ADDR} \\ - s = v_{\tau_1} \dots v_{\tau_n}, \text{ se } S = \tau_1 \dots \tau_n ; \\ - f(x) = v_{\tau}, \text{ se } F(x) = \tau ; \\ - H \text{ è un qualunque heap (che} \\ \text{definisce le variabili in } s, \text{ ed } f) \end{array} \right. \\ \left. \right\}$$

**esercizio.** Dimostrare che le funzioni  $\alpha$  e  $\gamma$  verificano le proprietà di astrazione e concretizzazione.

**questione.** Quali proprietà consente di verificare il dominio astratto con le funzioni  $\alpha$  e  $\gamma$  ?

1. se  $\models_H (p_C, s, f)$  e  $\models_{H'} (p_C, s', f')$  appartengono ad una configurazione concreta C e le lunghezze di  $s$  ed  $s'$  sono differenti, allora  $\alpha(C) = \perp$  (errore).

## Le interpretazioni astratte delle istruzioni

Istruzione per istruzione, vediamo come sono definite nel dominio astratto:

$$P[pc] = \mathbf{inc} \quad \left\{ \begin{array}{ll} (s, f) & (s, f) \quad \text{se } s = \mathbf{INT}.s', \quad pc+1 \quad \text{dom}(P) \\ (s, f) & \top \quad \text{altrimenti} \end{array} \right.$$

$$P[pc] = \mathbf{push0} \quad \left\{ \begin{array}{ll} (s, f) & (\mathbf{INT}.s, f) \quad \text{se } pc+1 \quad \text{dom}(P) \\ (s, f) & \top \quad \text{altrimenti} \end{array} \right.$$

$$P[pc] = \mathbf{pop} \quad \left\{ \begin{array}{ll} (s, f) & (s', f) \quad \text{se } s = \tau.s', \quad pc+1 \quad \text{dom}(P) \\ (s, f) & \top \quad \text{altrimenti} \end{array} \right.$$

$$P[pc] = \text{load } \mathbf{x} \quad \begin{cases} (s, f) & (f(\mathbf{x}).s, f) & \text{se } pc+1 \in \text{dom}(P) \\ (s, f) & \top & \text{altrimenti} \end{cases}$$

$$P[pc] = \text{store } \mathbf{x} \quad \begin{cases} (\tau.s, f) & (s, f[\mathbf{x} : \tau]) & \text{se } \mathbf{x} \in \text{dom}(f), pc+1 \in \text{dom}(P) \\ (s, f) & \top & \text{altrimenti} \end{cases}$$

$$P[pc] = \text{if } \mathbf{L} \quad \begin{cases} (\text{INT}.s, f) & (s, f) \cup (s_L, f_L) & \text{se } pc+1, \mathbf{L} \in \text{dom}(P) \\ (s, f) & \top & \text{altrimenti} \end{cases}$$

$$P[pc] = \text{putfield } \sigma.a \tau \quad \begin{cases} (\tau.s, f) & (s, f) & \text{se } pc+1 \in \text{dom}(P) \\ (s, f) & \top & \text{altrimenti} \end{cases}$$

$$P[pc] = \text{getfield } \sigma.a \tau \quad \begin{cases} (.s, f) & (\tau.s, f) & \text{se } pc+1 \in \text{dom}(P) \\ (s, f) & \top & \text{altrimenti} \end{cases}$$

**esercizio.** Dimostrare che tutte le funzioni di sopra sono monotone e continue.

**questione.** Quali proprietà consente di verificare l'interpretazione astratta delle istruzioni?

1. salti ad indirizzi non validi producono errore ( $\top$ );
2. le operazioni sulla pila richiedono una opportuna struttura dei tipi della pila, altrimenti producono errore ( $\top$ );
3. allo stesso modo per le operazioni sulle variabili.

**Per verificare un programma**, occorre:

- scrivere le equazioni su  $(S, F)$  per ogni istruzione, come stabilito dalla semantica astratta delle istruzioni

**conseguenza:** il sistema di equazioni ha soluzione e tale soluzione è determinabile in tempo finito.

**Esempio.** Sia una classe con un campo "a" che contiene interi ed un metodo che modifica "a" come segue: se il valore in "a" è 0 allora ci memorizza 1, altrimenti lo lascia inalterato.

Il bytecode che implementa questo metodo è il seguente:

```
0   load 0
1   getfield   .a   INT
2   store 1
3   load 1
4   if   11
5   push 0
6   inc
7   store 1
8   load 0
9   load 1
10  putfield   .a   INT
11  return
```

ed il sistema di equazioni è

0	load 0			$(S_0, f_0) = (, [0: , 1: \top])$
1	getfield	.a	INT	$(S_1, f_1) = (, [0: , 1: \top])$
2	store 1			$(S_2, f_2) = (\text{INT}, [0: , 1: \top])$
3	load 1			$(S_3, f_3) = (, [0: , 1: \text{INT}])$
4	if	11		$(S_4, f_4) = (\text{INT}, [0: , 1: \text{INT}])$
5	push 0			$(S_5, f_5) = (, [0: , 1: \text{INT}]) \cup (S_{11}, f_{11})$
6	inc			$(S_6, f_6) = (S_5, f_5) + (\text{INT}, [])$
7	store 1			$(S_7, f_7) = (S_6, f_6) \oplus (\text{INT}, [])$
8	load 0			$(S_8, f_8) = ((S_7, f_7) - (, [])) + (, [1: ])$
9	load 1			$(S_9, f_9) = (S_8, f_8) + (f_8(0), [])$
10	putfield	.a	INT	$(S_{10}, f_{10}) = (S_9, f_9) + (f_9(1), [])$
11	return			$(S_{11}, f_{11}) = (S_{10}, f_{10}) - (\text{INT.}, [])$

dove le operazioni "+", "⊕" e "-" sono definite così:

$$(s,f) + (\tau, [n_i : v_i^{i \ I}]) = (\tau.s, f[n_i : v_i^{i \ I}])$$

$$(s,f) - (s',[]) = \begin{cases} (s'',f) & \text{se } s = s'.s'' \\ \top & \text{altrimenti} \end{cases}$$

$$(s,f) \oplus (\tau,[]) = \begin{cases} (s,f) & \text{se } s = \tau.s' \\ \top & \text{altrimenti} \end{cases}$$

L'esecuzione astratta è illustrata dalla tabella seguente.

	iterazione 0	iterazione 1	...	iterazione 8,9,...
$(S_0, F_0)$	$(\perp, [0: \perp, 1: \top])$	$(\perp, [0: \perp, 1: \top])$		$(\perp, [0: \perp, 1: \top])$
$(S_1, F_1)$	$\perp$	$(\perp, [0: \perp, 1: \top])$		$(\perp, [0: \perp, 1: \top])$
$(S_2, F_2)$	$\perp$	$(\text{INT}, [0: \perp, 1: \top])$		$(\text{INT}, [0: \perp, 1: \top])$
$(S_3, F_3)$	$\perp$	$(\perp, [0: \perp, 1: \text{INT}])$		$(\perp, [0: \perp, 1: \text{INT}])$
$(S_4, F_4)$	$\perp$	$(\text{INT}, [0: \perp, 1: \text{INT}])$		$(\text{INT}, [0: \perp, 1: \text{INT}])$
$(S_5, F_5)$	$\perp$	$(\perp, [0: \perp, 1: \text{INT}])$		$(\perp, [0: \perp, 1: \text{INT}])$
$(S_6, F_6)$	$\perp$	$\perp$		$(\text{INT}, [0: \perp, 1: \text{INT}])$
$(S_7, F_7)$	$\perp$	$\perp$		$(\text{INT}, [0: \perp, 1: \text{INT}])$
$(S_8, F_8)$	$\perp$	$\perp$		$(\perp, [0: \perp, 1: \text{INT}])$
$(S_9, F_9)$	$\perp$	$\perp$		$(\perp, [0: \perp, 1: \text{INT}])$
$(S_{10}, F_{10})$	$\perp$	$\perp$		$(\text{INT}, [0: \perp, 1: \text{INT}])$
$(S_{11}, F_{11})$	$\perp$	$\perp$		$(\perp, [0: \perp, 1: \text{INT}])$

Il programma è corretto perchè nessuno degli stati della forma stabile è etichettato con  $\top$ .